МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное
образовательное учреждение высшего образования
«Южно-Уральский государственный университет
(национальный исследовательский университет)»

Высшая школа электроники и компьютерных наук
Кафедра «Электронные вычислительные машины»

ДОПУСТИТЬ К ЗАЩИТЕ
Заведующий кафедрой ЭВМ
_____ Д.В. Топольский
«___»_____ 2025 г.

Разработка приложения для создания визуального дизайна

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ
ЮУРГУ-090301.2025.405 ПЗ ВКР

Руководитель работы,
к.т.н., доцент каф. ЭВМ
_____Ю.Б. Кухта
«___»_____2025 г.

Автор работы,
студентка группы КЭ-405
_____А.Р. Юзликеева
«___»_____2025 г.

Нормоконтролёр,
ст. преподаватель каф. ЭВМ
_____ С.В. Сяськов
«___»_____2025 г.

Челябинск-2025

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное
образовательное учреждение высшего образования
«Южно-Уральский государственный университет
(национальный исследовательский университет)»
Высшая школа электроники и компьютерных наук
Кафедра «Электронные вычислительные машины»

УТВЕРЖДАЮ
Заведующий кафедрой ЭВМ
_____ Д.В. Топольский
«___»_____2025 г.

ЗАДАНИЕ

на выпускную квалификационную работу бакалавра

студентке группы КЭ-405
Юзликеевой Алине Рифовне
обучающемуся по направлению
09.03.01 «Информатика и вычислительная техника»

1. **Тема работы: «**Разработка приложения для создания визуального дизайна» утверждена приказом по университету от «21» апреля 2025 г. №648-13/12.

2. **Срок сдачи студентом законченной работы:** 01 июня 2025 г.

3. **Исходные данные к работе:**

3.1. Структура дизайн-проекта согласно концепции брендбука «Ridex», включающая фирменные цвета, логотип, элементы фирменного стиля.

3.2. Коллекция методических материалов по курсу «Дизайнер в ИТ», интегрированная в приложение для самостоятельного изучения пользователями.

3.3. Коллекция готовых шаблонов, интегрированная в приложение для облегчения процесса создания первых дизайн-проектов.

3.4. Серверная часть приложения должна разрабатываться на языке TypeScript и работать в среде Node.js.

3.5. Приложение должно быть доступно через браузер без необходимости установки программного обеспечения на устройства пользователя.

3.6. Система должна обеспечивать безопасную аутентификацию и хранение паролей с использованием методов шифрования.

3.7. Приложение должно обеспечивать возможность одновременной работы нескольких пользователей над проектом с сохранением изменений в реальном времени.

4. **Перечень подлежащих разработке вопросов:**

1. Анализ теоретических и практических основ дизайна, включая обзор существующих решений для дизайна.
2. Проектирование архитектурного решения.
3. Разработка приложения.
4. Тестирование разработанного приложения.

5. **Дата выдачи задания:** 2 декабря 2024 г.

Руководитель работы _____/Ю.Б. Кухта /

Студентка _____/А.Р. Юзликеева /

КАЛЕНДАРНЫЙ ПЛАН

Этап	Срок сдачи	Подпись руководителя
Анализ теоретических и практических основ дизайна, включая обзор существующих решений для дизайна	03.03.2025	
Проектирование архитектурного решения	22.03.2025	
Разработка приложения	12.04.2025	
Тестирование разработанного приложения	26.04.2025	
Компоновка текста работы и сдача на нормоконтроль	22.05.2025	
Подготовка презентации и доклада	30.05.2025	

Руководитель работы _____/Ю.Б. Кухта /

Студентка _____/А.Р. Юзликеева /

Аннотация

А.Р. Юзликеева. Разработка приложения для создания визуального дизайна. – Челябинск: ФГАОУ ВО «ЮУрГУ (НИУ)», ВШ ЭКН; 2025, 201 с., 39 ил., библиогр. список – 55 наим.

Выпускная квалификационная работа посвящена разработке современного веб-приложения для создания и редактирования визуального дизайна с поддержкой коллаборативной работы в реальном времени

В первой главе проведён анализ предметной области и обзор существующих решений, выявлены основные ограничения и недостатки популярных приложений.

Во второй главе сформулированы функциональные и нефункциональные требования, выбран современный технологический стек и разработана архитектура приложения. Описаны ключевые компоненты системы, структура базы данных и интерфейсные прототипы.

Третья глава посвящена практической реализации приложения. Рассмотрены этапы настройки проекта, создание основных модулей, включая графический редактор с поддержкой различных слоёв, систему аутентификации и модуль совместной работы в реальном времени.

В четвёртой главе описан процесс тестирования веб-приложения, включая функциональное и нефункциональное тестирование.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Современный мир насыщен визуальной информацией, что делает профессию дизайнера особенно востребованной. Ежедневно мы взаимодействуем с огромным количеством графики, изображений, интерфейсов и рекламы, которые формируют наше восприятие и влияют на принимаемые нами решения. Визуальное оформление имеет решающее значение в коммуникации, позволяя передавать важную информацию за считанные секунды. Согласно исследованию McKinsey [32], компании с высоким индексом дизайна (McKinsey Design Index) в среднем получают на 32% больше прибыли и на 56% больше дивидендов для акционеров. Это подчеркивает прямую связь между качественным дизайном и бизнес-успехом.

Актуальность исследования обусловлена ростом спроса на специалистов в области визуального дизайна, стремлением компаний выделиться на рынке и необходимостью адаптации к новым технологическим и экономическим условиям. По данным исследования, проведенного DNA Team и HeadHunter [49], количество вакансий дизайнеров с начала 2024 года выросло на 19%. В дальнейшем потребность в этих специалистах будет только расти. Несмотря на развитие автоматизации и замену многих профессий роботами, творческая деятельность останется сферой, требующей человеческого участия. Технологический прогресс напрямую связан с дизайном: даже в условиях нестабильности сохраняется устойчивый спрос на эстетичные и удобные продукты.

Несмотря на растущий спрос, существует ряд факторов, затрудняющих реализацию потенциала начинающих специалистов. Из-за экономических и политических факторов многие популярные инструменты стали недоступны для

российских специалистов. Это вынудило их осваивать альтернативные решения и адаптироваться к новым условиям.

Визуальный дизайн как способ коммуникации широко изучен и объединяет в себе эстетическую привлекательность и функциональность для формирования положительного пользовательского опыта [45]. Его цель — передача информации, формирование эмоционального отклика и создание запоминающегося визуального образа.

Основу дизайна составляют такие принципы, как композиция, колористика и типографика. Они лежат в основе визуальных и функциональных решений. Грамотное применение этих элементов позволяет создавать качественные визуальные продукты. Однако для начинающих дизайнеров освоение этих дисциплин может оказаться сложным: они требуют теоретической подготовки и практических навыков, что делает процесс обучения без соответствующих инструментов затруднительным.

В настоящее время существует множество программных инструментов для дизайна, включая Figma, Sketch, Adobe XD, Adobe Photoshop и Illustrator. Каждая из них предлагает различные уникальные функции, однако универсального решения, которое бы подошло для всех случаев, особенно для начинающих дизайнеров, не существует.

Проблема исследования заключается в противоречии между растущим спросом на дизайнеров и ограниченной доступностью удобных инструментов для обучения и практики начинающих дизайнеров — особенно в условиях санкций, ограничивших использование зарубежного программного обеспечения. Это создает барьеры для входа в профессию и замедляет развитие отечественного дизайн-сообщества.

Целью данной выпускной квалификационной работы является разработка приложения, облегчающего процесс создания визуального дизайна для начинающих пользователей.

Для достижения поставленной цели, необходимо решить следующие поставленные задачи:

1. Проанализировать теоретические и практические основы дизайна, включая обзор существующих решений для дизайна.
2. Спроектировать архитектурное решение.
3. Разработать приложение.
4. Протестировать разработанное приложение.

Новизна работы заключается в создании приложения, ориентированного на образовательные и практические потребности начинающих дизайнеров, с учётом текущих рыночных условий и ограниченного доступа к зарубежному ПО. Это придаёт исследованию как теоретическую значимость (вклад в разработку подходов к обучающим цифровым инструментам), так и практическую (создание доступного и удобного продукта).

1. АНАЛИЗ ТЕОРЕТИЧЕСКИХ И ПРАКТИЧЕСКИХ ОСНОВ ДИЗАЙНА, ВКЛЮЧАЯ ОБЗОР СУЩЕСТВУЮЩИХ РЕШЕНИЙ ДЛЯ ДИЗАЙНА

Современная цифровая среда предъявляет всё более высокие требования к визуальной коммуникации и качеству пользовательского интерфейса. С ростом числа цифровых продуктов и сервисов увеличивается спрос на дизайнеров, которые умеют не только работать с визуальными элементами, но и создавать продуманный пользовательский опыт. В этой связи особенно важно использование специализированного программного обеспечения для дизайна. Оно не только помогает воплощать креативные идеи, но и значительно повышает продуктивность работы.

Ключевым трендом последних лет стало внедрение искусственного интеллекта и генеративных инструментов в процессы проектирования. Такие решения, как Midjourney и DALL·E, позволяют автоматически создавать визуальные материалы. Однако, несмотря на впечатляющие возможности, они часто выдают шаблонные изображения, не учитывающие контекст и специфику задачи. В результате создаваемый дизайн может не соответствовать требованиям проекта и быть недостаточно эффективным. Кроме того, такие технологии пока не способны заменить человеческое восприятие контекста, культурных особенностей и эмоциональных нюансов, играющих решающую роль в создании действительно действенного дизайна.

Современный дизайн — это не просто работа с графикой. Это комплексный процесс, включающий разработку концепции, проектирование пользовательского взаимодействия и адаптацию под различные устройства и сценарии использования. Он требует не только владения инструментами, но и глубоких знаний основ композиции, типографики и колористики. Без этого

невозможно создать по-настоящему удобное, эффективное и визуально привлекательное решение.

Композиция — это согласованное размещение элементов [54]. Её основная задача заключается в создании целостного восприятия, чтобы все элементы воспринимались как единое целое. Композиция способствует быстрому пониманию того, что хотел сказать дизайнер, и выделению ключевых моментов. Именно она управляет вниманием и воздействует на эмоции.

Существуют основные принципы, делающие визуальные материалы более понятными и доступными для пользователя [44].

Визуальная иерархия — это способ организации информации, при котором пользователь может быстро и легко ориентироваться, и различать первостепенное от второстепенного. Она определяет порядок восприятия информации. Наиболее важный элемент в композиции называют визуальной доминантой. Для её выделения и создания иерархии используют такие средства, как масштаб, контраст, цвет и негативное пространство.

Ощущение порядка в композиции создаёт принцип выравнивания. Он гласит, что элементы не должны располагаться случайным образом. Выровненные элементы композиции создают целостность и воспринимаются как единое целое, даже если между ними есть расстояние. Целостность является ключевым аспектом в дизайне. Для создания порядка активно используется сетка. Существуют четыре основных вида сеток: манускриптная (или блочная), колоночная, модульная и пиксельная. Однако иногда дизайнеры намеренно нарушают порядок размещения объектов по сетке, чтобы усилить визуальный акцент на объекте.

В дизайне элементы могут восприниматься по-разному: некоторые выглядят тяжёлыми, тем самым привлекая к себе внимание, другие кажутся лёгкими и менее значимыми. Такое расположение объектов называют

асимметричным, и оно требует более тщательной проработки, поскольку у элементов разный вес, и они имеют разную визуальную значимость [51, 54].

Симметричное расположение объектов создаётся равномерным распределением элементов — это формирует ощущение порядка и стабильности. Такие композиции воспринимаются нами как гармоничные и привлекают внимание, так как человеческий мозг стремится к простоте. Однако чрезмерная симметрия может казаться статичной и предсказуемой, поэтому её дополняют яркими акцентами или асимметричными элементами.

Золотым сечением называется такое соотношение объектов, которое считается наиболее приятным для восприятия по своим пропорциям. Для этого соотношения существует универсальное математическое число, равное 1,618. Этот коэффициент используется для создания гармоничных структур, изменяя размеры элементов относительно друг друга [55].

Правило третей представляет собой упрощённую версию золотого сечения. Согласно этому правилу, ключевой объект размещается в одной из боковых третей композиции, в то время как остальные две трети остаются пустыми [54, 55].

Негативное, или свободное, пространство между элементами помогает расставить акценты и создать баланс в композиции. Пустое пространство вокруг объектов способствует привлечению к ним внимания. Более того, с помощью негативного пространства можно создавать оптические иллюзии.

Ритм представляет собой повторение определённых элементов, которое связывает основные объекты, направляет внимание пользователя и способствует единообразию [41, 51]. Это связано с тем, что восприятие одинаковых элементов требует меньше когнитивных усилий. Если в дизайне отсутствует контраст, но присутствует ритм, то это можно назвать паттерном.

Паттерны активно используются в оформлении фирменного стиля, а также в полиграфии для создания упаковок, обложек и этикеток.

В основном дизайн ориентирован на человека, поэтому важно учитывать психологические аспекты восприятия и воздействия. Гештальт-принципы представляют собой набор правил, которые описывают, как люди воспринимают образы и структурируют их [35]. Человеческий мозг, как уже было сказано ранее, стремится к упорядоченности и везде ищет закономерности. Если учитывать гештальт-принципы при проектировании дизайна, то можно более эффективно управлять вниманием, снижать когнитивную нагрузку и подталкивать пользователей к целевому действию. Выделяют семь ключевых принципов: близости, замыкания, подобия, непрерывности, восприятия, организации и симметрии. Рассмотрим принципы, на основе которых элементы объединяются в композицию [46].

Согласно теории близости, объекты, находящиеся рядом, в непосредственной близости друг от друга, воспринимаются как связанные между собой [46, 53]. Пользователи не должны тратить время на размышления о том, есть ли между элементами связь — это должно быть интуитивно понятно. При этом данный принцип настолько силён, что преодолевает различия в цвете, форме и других характеристиках, которые могут отличать группу.

Принцип подобия напоминает принцип близости, но, в отличие от него, объединяет объекты, обладающие общими признаками, такими как форма, цвет или размер. Принцип непрерывности утверждает, что объекты, размещённые вдоль одной линии, воспринимаются как связанные между собой. Эта линия может быть как реальной, так и воображаемой, как прямой, так и кривой.

Целостность подразумевает гармоничное взаимодействие элементов, которые вместе создают единую композицию. Причём каждый элемент должен быть визуально связан с другими, чтобы максимально эффективно и точно

передавать задуманное сообщение. Именно эта согласованность даёт ощущение, что все составляющие работают в унисон.

Гармония композиции формируется не только благодаря расположению элементов, но и через цветовые акценты, которые помогают упорядочивать визуальный ряд. Цвет не только объединяет элементы, но и акцентирует их значимость, создавая эмоциональное восприятие. Колористика — наука, которая занимается исследованием природы цвета, его свойств и характеристик, а также различных цветов, включая основные, составные и дополнительные [40].

В 1676 году английский математик Исаак Ньютон использовал призму для разложения солнечного света на спектр цветов [42]. На основе этого эксперимента учёный разработал цветовой круг (Рисунок 1), включающий в себя семь оттенков: красный, оранжевый, жёлтый, зелёный, голубой, синий и фиолетовый.

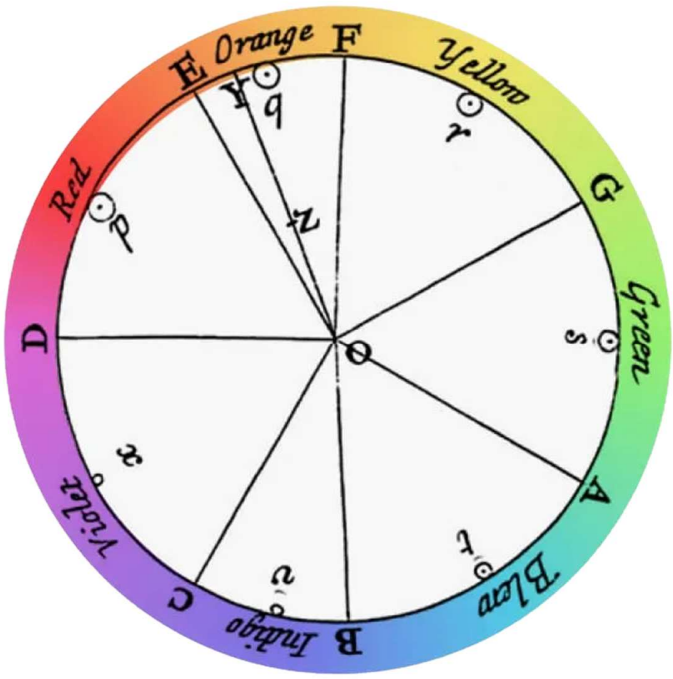

Рисунок 1 – Цветовой круг Исаака Ньютона

В наши дни дизайнеры используют цветовой круг, разработанный швейцарским художником Иоганнесом Иттеном в начале XX века. Этот круг является визуальным отображением двенадцати цветов, организованных по определённому логическому порядку [42].

Цветовой круг — полезный инструмент для формирования цветовых палитр [48, 51]. Палитра представляет собой тщательно подобранный набор цветов, который иллюстрирует взаимосвязь между первичными, вторичными и третичными цветами. Первичные цвета — это те оттенки, которые нельзя получить путём смешивания других цветов. Их всего три: красный, синий и жёлтый. На круге эти цвета расположены на равном расстоянии друг от друга. Вторичные цвета образуются путём смешивания каждого из основных цветов с равным количеством следующего за ним цвета. Так получаются фиолетовый, оранжевый и зелёный цвета. Третичные цвета формируются при смешивании равных количеств цветов, находящихся по соседству. К ним относятся: красно-фиолетовый, красно-жёлтый, жёлто-оранжевый, жёлто-зелёный, сине-зелёный и сине-фиолетовый. Цветовой круг Иоганнеса Иттена (Рисунок 2).

Рисунок 2 – Цветовой круг Иоганнеса Иттена

В процессе исследований было выявлено пять основных цветовых комбинаций, которые воспринимаются зрителем наиболее гармонично [47]. Эти сочетания включают в себя два и более цвета, которые, в свою очередь, связаны определёнными пропорциями.

Монохромная схема основывается на использовании исключительно разных оттенков одного и того же цвета. Такая палитра создаёт спокойное восприятие за счёт отсутствия резкого контраста.

Аналоговая схема формируется из цветов, расположенных в соседних секторах цветового круга. Она обеспечивает мягкие переходы между оттенками.

Схема, основанная на двух оттенках, находящихся на противоположных концах цветового круга, называется дополнительной. Такое сочетание обеспечивает максимальный цветовой контраст.

Триада представляет собой комбинацию из трёх цветов, равномерно распределённых по кругу, образуя равносторонний треугольник. В сплит-комплементарной схеме также используется треугольник, но в этом случае применяется его равнобедренная форма.

Тетрада включает в себя четыре цвета, которые образуют геометрическую фигуру в виде прямоугольника или квадрата. Данная схема является самой сложной, так как она требует использования тщательно сбалансированных цветов.

Для создания цветовой палитры крайне важно учитывать характеристики цвета, которые позволяют описывать и классифицировать цвета. К таким характеристикам относятся тон, насыщенность и светлота [48].

Цветовой тон определяет семейство, к которому относится тот или иной цвет. Например, пурпурный и бордовый принадлежат к красному тону. Тон фактически выступает синонимом слова «цвет», поэтому, говоря о цвете, мы в первую очередь имеем в виду его тон.

Изменения тона, насыщенности и светлоты (Рисунок 3).

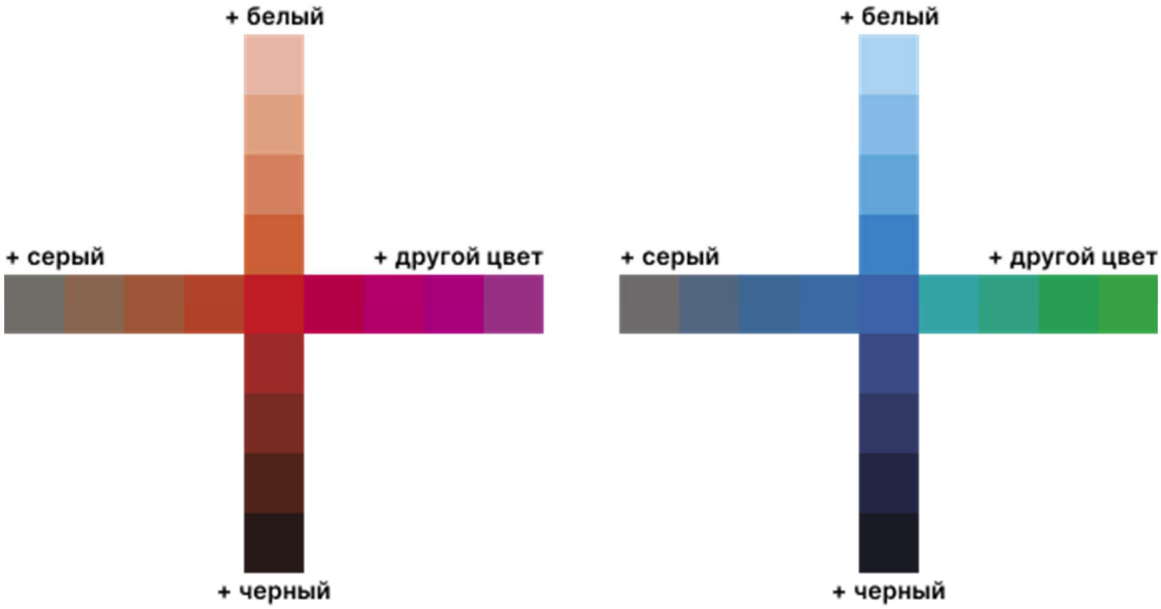

Рисунок 3 – Изменение тона, насыщенности и светлоты

Насыщенность отражает степень яркости или тусклости цвета. Если насыщенность высока, цвет выглядит энергичным и контрастным. Однако при большом количестве насыщенных оттенков они могут начать конкурировать и подавлять друг друга.

Цветовой круг, о котором говорилось ранее, состоит исключительно из чистых цветов. Его можно расширить, если добавить чёрный или белый цвет. Светлота определяет степень светлоты или темноты цвета. Она влияет на то, как визуальную работу будет воспринимать наш глаз.

Цвет способен оказывать воздействие на человеческое сознание, пробуждая различные ассоциации и эмоции. Большинство людей не скрывают свои предпочтения в отношении определённых цветовых сочетаний. Цвет способен вызывать такие чувства, как страх, восхищение, спокойствие, умиление, а также надежду и даже любовь. Воздействие цвета во многом

определяется насыщенностью и тоном: чем интенсивнее цвет, тем сильнее его эффект, вне зависимости от тона [39].

Для описания цветов применяются различные математические модели, известные как цветовые модели. Рассмотрим две основные: CMYK и RGB [38].

CMYK представляет собой модель, основанную на четырёх цветах: Cyan (голубой), Magenta (пурпурный), Yellow (жёлтый) и Key (ключевой), который представляет собой чёрный цвет. В этой модели цвета создаются путём вычитания света, отражённого от поверхности, что по сути является процессом смешивания красок. CMYK широко используется в полиграфии, в частности при печати на различных носителях, таких как книги, журналы, плакаты и этикетки.

RGB представляет собой модель, основанную на трёх цветах: Red (красный), Green (зелёный), Blue (синий). Мы можем наблюдать эту модель на дисплеях различных электронных устройств, таких как компьютеры, смартфоны и телевизоры. Интенсивность каждого цвета представлена числом в диапазоне от 0 до 255. Например, значение RGB (100, 255, 0) соответствует лаймовому цвету, а RGB (255, 150, 0) — оранжевому.

HEX-код определяет яркость каждого цвета с помощью двухзначных чисел в шестнадцатеричной системе счисления, что позволяет получить 256 различных комбинаций. Например, #FFFFFF обозначает белый цвет, #64FF00 — лаймовый, #FF9600 — оранжевый. Главным преимуществом такого формата является удобство копирования и унификация значений.

Из-за специфики цветовой модели CMYK не все оттенки, представленные в RGB, могут быть точно воспроизведены. Для достижения максимально точной передачи цветов в процессе печати требуется предварительная обработка.

Цветовой охват различных устройств (Рисунок 4). Чёрная линия обозначает область цветовой модели RGB. Красная линия указывает на диапазон цветов, доступный при высококачественной офсетной печати, а синий пунктир — цветовой охват стандартного принтера.

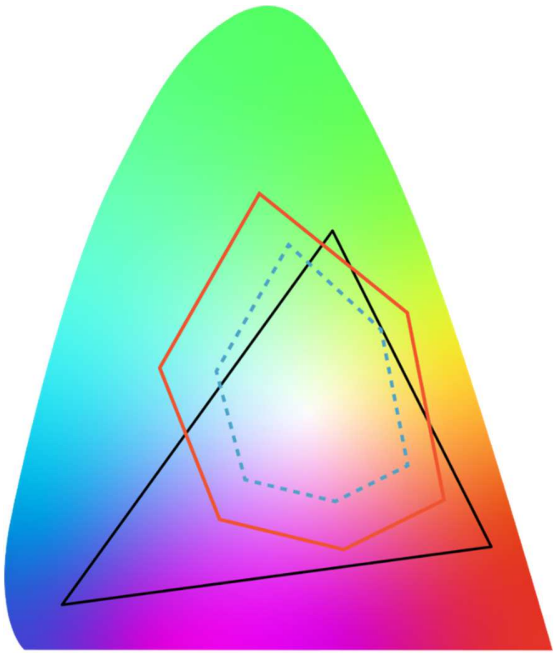

Рисунок 4 – Цветовой охват различных устройств

Так как модель CMYK не способна воспроизводить некоторые оттенки, была разработана дополнительная палитра — Pantone. Она включает в себя заранее подготовленные цвета, которые смешиваются до печати и затем загружаются в печатную машину, что обеспечивает единообразие цвета. Модели CMYK и Pantone можно комбинировать, что позволяет значительно расширить палитру доступных цветов.

Ранее мы рассматривали контраст в контексте композиции, который характеризуется такими параметрами, как размер, форма, положение, цвет. Особое внимание стоит уделять цветовому контрасту, который играет ключевую роль в обеспечении удобства чтения. Хороший контраст приносит

пользу всем, но особенно важен для людей с нарушениями зрения, численность которых в мире составляет 285 миллионов человек.

WCAG (Web Content Accessibility Guidelines) — руководство, содержащее рекомендации по доступности контента, включая разработку технологических стандартов. В руководстве представлена формула уровня контрастности, которая определяет значение уровня и соотношения [50]. Формула выводит число от 1 до 21, где 21 соответствует максимальному уровню контраста. Для оценки контрастности определены два уровня успеха: AA и AAA.

Уровень AA требует, чтобы контраст между фоном и текстом был достаточно выраженным, чтобы обеспечить хорошую читаемость: контрастность текста не менее 4,5:1. Для текста крупного размера (от 24 px) соотношение должно быть не менее 3:1.

Уровень AAA обеспечивает более высокий уровень доступности для определённых групп пользователей. Требования минимально составляют 7:1 для обычного текста, а для крупного — 4,5:1. Кроме того, минимальное значение контрастности для элементов, не являющихся текстом, должно быть не ниже 3:1.

Цвет играет важную роль в привлечении внимания, создании определённого настроения и выделении ключевых элементов. Однако для того, чтобы пользователь смог уловить основную идею, требуется текстовое сопровождение. Текст служит эффективным инструментом для передачи информации в наиболее ясной и точной форме.

Типографика представляет собой искусство работы с текстом: от подбора шрифта до размещения элементов [43, 52]. Грамотно оформленный текст способствует лучшему восприятию информации.

Восприятие текста определяется не только его содержанием, но и его формой. Мы привыкли, что текст, предназначенный для чтения, должен быть

максимально незаметным. Наш мозг распознаёт смысл словосочетаний быстрее, чем различает отдельные буквы, благодаря привычной форме знаков и расстояниям между ними. В рекламе, напротив, шрифт должен вызывать у читателя определённые эмоции через визуальные образы. Таким образом, шрифты можно классифицировать на две категории: текстовые и акцидентные [43].

В качественном шрифте форма всех символов построена в соответствии с определённой логикой, которая формирует его индивидуальные особенности. Именно эта логика определяет характер шрифта, что, в свою очередь, играет важную роль в создании «атмосферы» и подборе гармоничных сочетаний шрифтов.

Антиква (от англ. serif) представляет собой контрастный шрифт с засечками. Например, Times New Roman является одновременно антиквой и текстовым шрифтом. Гротеском (от англ. sans-serif) называют шрифт без засечек. Самым распространённым примером гротеска является шрифт Helvetica.

Шрифты могут иметь различные начертания, которые отличаются наклоном и насыщенностью. Прямые начертания включают в себя light, regular и bold, а курсивные — italic light, italic regular, italic bold. Совокупность начертаний, объединённых общей концепцией, называется гарнитурой [52].

В процессе работы часто возникает необходимость использовать несколько шрифтов одновременно. Этот вопрос является довольно сложным и неоднозначным. При выборе шрифтов важно обращать внимание на то, насколько гармонично они будут сочетаться между собой [51].

Пара шрифтов считается гармоничной, если они имеют минимальное количество различий в стиле, размере и насыщенности. Шрифты, которые схожи по этим характеристикам, но не идентичны, называют конфликтными.

Такие визуальные сходства нарушают порядок и эстетическую привлекательность. Контрастной считается такая пара шрифтов, которые заметно отличаются друг от друга. В дизайне, направленном на привлечение внимания, часто присутствует множество контрастных элементов.

Визуальная иерархия формируется, в том числе, благодаря иерархии шрифтов. Применение разных шрифтов и их размеров способствует созданию чёткой структуры в заголовках, подзаголовках и основном тексте.

Понимание теоретических основ дизайна служит фундаментом для разработки качественных визуальных решений. Тем не менее, эти знания теряют свою значимость без практической составляющей. Насмотренность представляет собой визуальный опыт человека — своего рода внутреннюю шкалу, по которой он оценивает всё, что его окружает. Регулярный анализ успешных работ помогает соединить теоретическую и практическую части, развить чувство стиля и находить новые креативные идеи в дизайне.

Чтобы успешно применять знания, важно использовать инструменты, которые позволяют эффективно реализовывать идеи и способствуют развитию профессиональных навыков.

1.1. ОБЗОР СУЩЕСТВУЮЩИЙ ПРОГРАММ И СЕРВИСОВ, ПРЕДНАЗНАЧЕННЫХ ДЛЯ РАЗРАБОТКИ ДИЗАЙНА

Существует множество программ и онлайн-сервисов, предназначенных для создания дизайна. Рассмотрим те, которые непосредственно связаны с темой данной работы.

Рисунок 5 представляет интерфейс Figma [37], разработанной американской компанией Figma Inc.

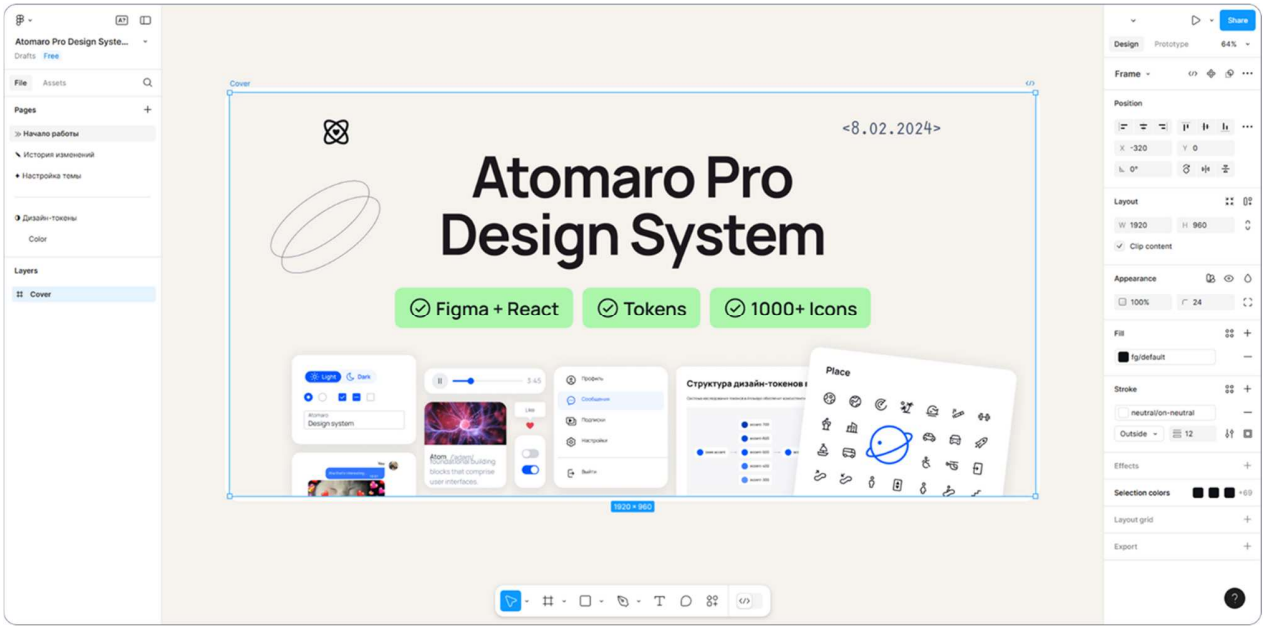

Рисунок 5 – Графический редактор Figma

Figma — графический редактор для проектирования пользовательских интерфейсов. Он был представлен в 2016 году как альтернатива Sketch и сегодня занимает лидирующие позиции среди дизайнерских инструментов. Ключевыми преимуществами Figma являются бесплатный доступ, широкий функционал и поддержка всех основных операционных систем. Программа работает в браузере, поэтому не требует установки и загрузки файлов — весь процесс происходит онлайн. Пользователи могут управлять правами доступа, что позволяет работать над проектом совместно или ограничивать возможность редактирования. Это особенно удобно при командной работе. Интерфейс Figma достаточно интуитивен, но новичкам он может показаться сложным — в частности, из-за отсутствия готовых шаблонов. Хотя в сообществе Figma доступно множество проектов, созданных другими пользователями, они не встроены по умолчанию и требуют поиска. В базовой версии редактора отсутствуют встроенные инструменты для создания диаграмм, таблиц, графиков, цветовых палитр и тестирования на соответствие стандартам WCAG.

Однако функциональность можно расширить с помощью плагинов — дополнительных модулей, которые упрощают и ускоряют работу. В России Figma по-прежнему доступна, однако могут возникнуть сложности с оплатой из-за санкционных ограничений.

Рисунок 6 показывает интерфейс онлайн-сервиса Canva [5], разработанного австралийской компанией Canva Pty Ltd.

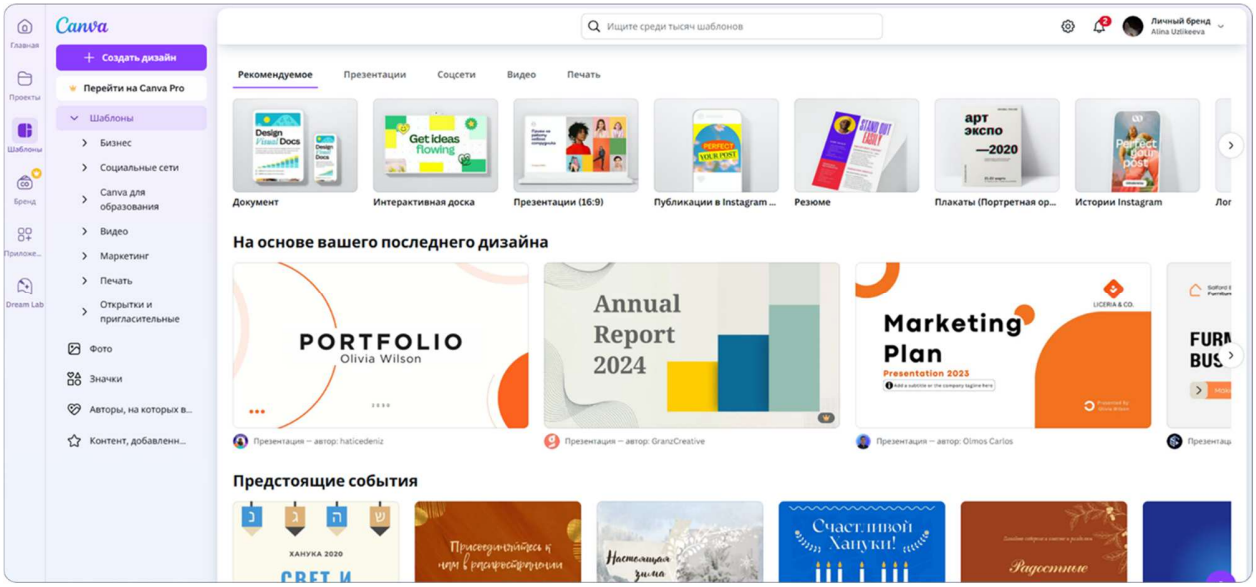

Рисунок 6 – Онлайн-инструмент Canva

Canva — веб-платформа для графического дизайна, не требующая установки. Благодаря интуитивно понятному интерфейсу она подходит как профессионалам, так и начинающим пользователям. Сервис предлагает большую библиотеку шаблонов для социальных сетей, презентаций, визиток и других проектов, что особенно полезно новичкам. Canva включает базовые инструменты для создания диаграмм и графиков, а также функции работы с цветовыми палитрами. Интерфейс полностью переведён на русский язык. Однако в 2022 году сервис приостановил деятельность в России, что ограничивает его доступность. Совместная работа в реальном времени

возможна, но редактирование с полным доступом предоставляется только на платных тарифах.

Рисунок 7 представляет графический редактор Pixso [22], разработанный китайской компанией Wanxing Technology.

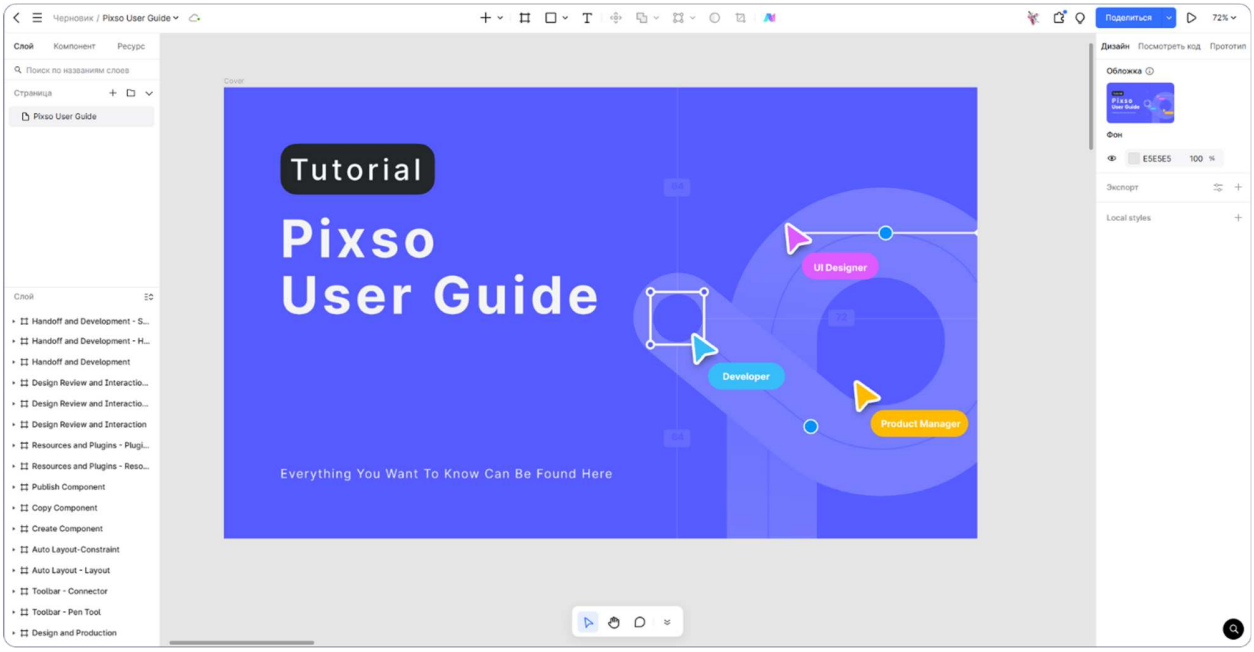

Рисунок 7 – Графический редактор Pixso

Pixso позиционируется как альтернатива Figma. Он доступен как в браузере, так и в виде десктопного приложения. Сервис поддерживает совместное редактирование, а схожий с Figma интерфейс облегчает адаптацию. Однако отсутствие встроенных шаблонов может затруднить освоение программы для новичков. Сообщество пользователей пока сравнительно небольшое: к началу 2025 года насчитывается около 300 проектов, тогда как в Figma — более 1500. Инструменты для работы с данными, диаграммами и цветовыми палитрами отсутствуют. Pixso доступен в России и имеет русскоязычный интерфейс. Бесплатная версия предоставляет широкие возможности, однако для командной работы, как и в случае с Figma, может потребоваться платная подписка.

Рисунок 8 представляет интерфейс векторного редактора Sketch [28], разработанного голландской компанией Bohemian Coding.

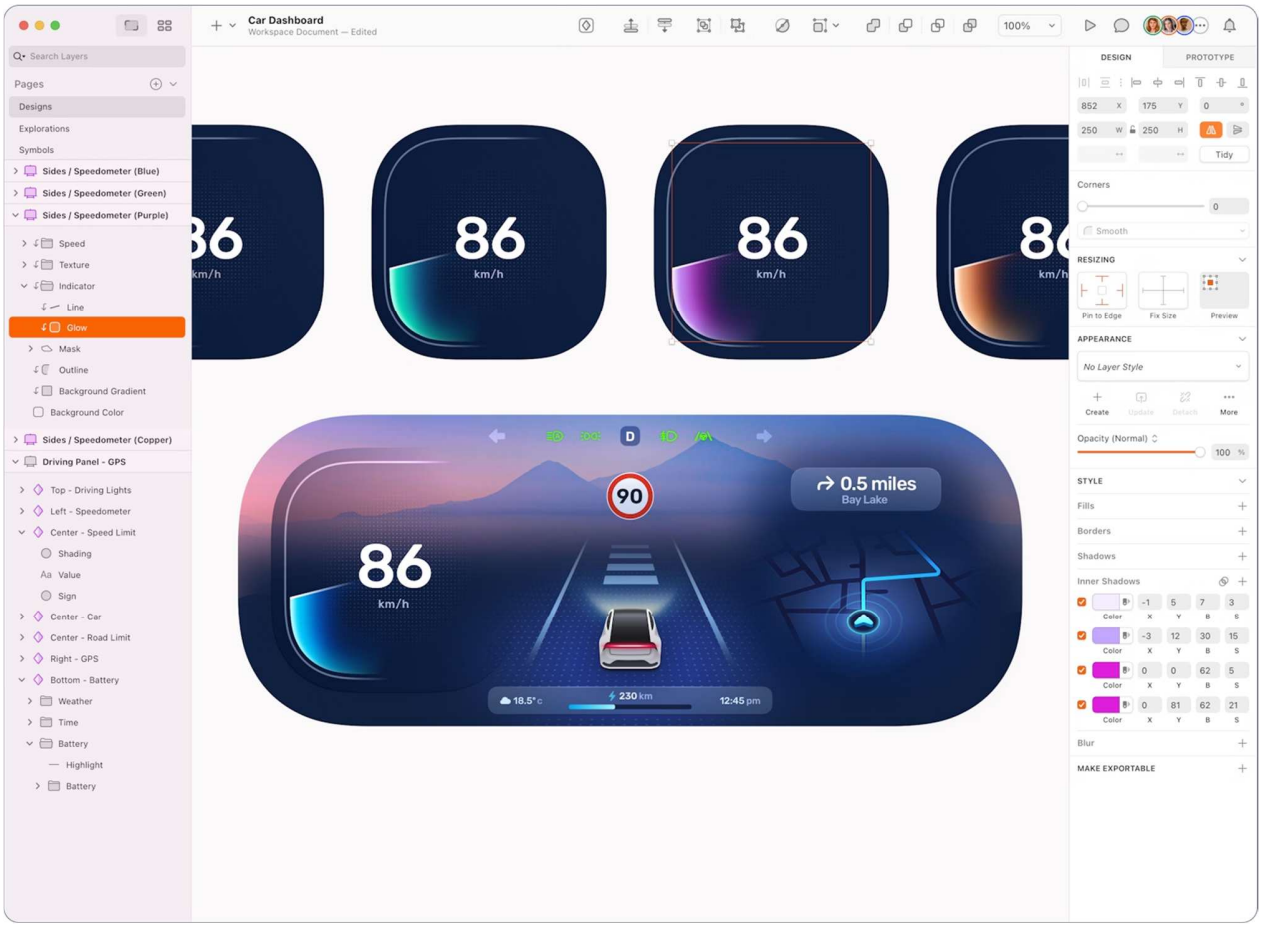

Рисунок 8 – Графический редактор Sketch

Инструмент был выпущен в 2010 году и работает только на macOS, требуя установки — браузерной версии не предусмотрено. При этом программа занимает всего 38 мегабайт, что значительно меньше по сравнению с Photoshop, размер которого может достигать гигабайт. Sketch ориентирован на профессиональных дизайнеров, что может затруднить его освоение для новичков. Как и в Pixso, здесь ограничено количество встроенных шаблонов. Также отсутствуют инструменты для работы с данными и цветовыми палитрами. Функция совместного редактирования реализована через облачный

сервис Sketch Cloud, однако по уровню функциональности он уступает таким конкурентам, как Figma. В России сервис доступен, но возможны сложности с оплатой подписки из-за санкционных ограничений. Интерфейс не переведён на русский язык. Бесплатной версии нет — предлагается только пробный период.

Рисунок 9 представляет приложение Lunacy [16], разработанное компанией Icons8 LLC. Icons8 — это веб-платформа, предоставляющая стоковые графические ресурсы: иконки, иллюстрации и фотографии.

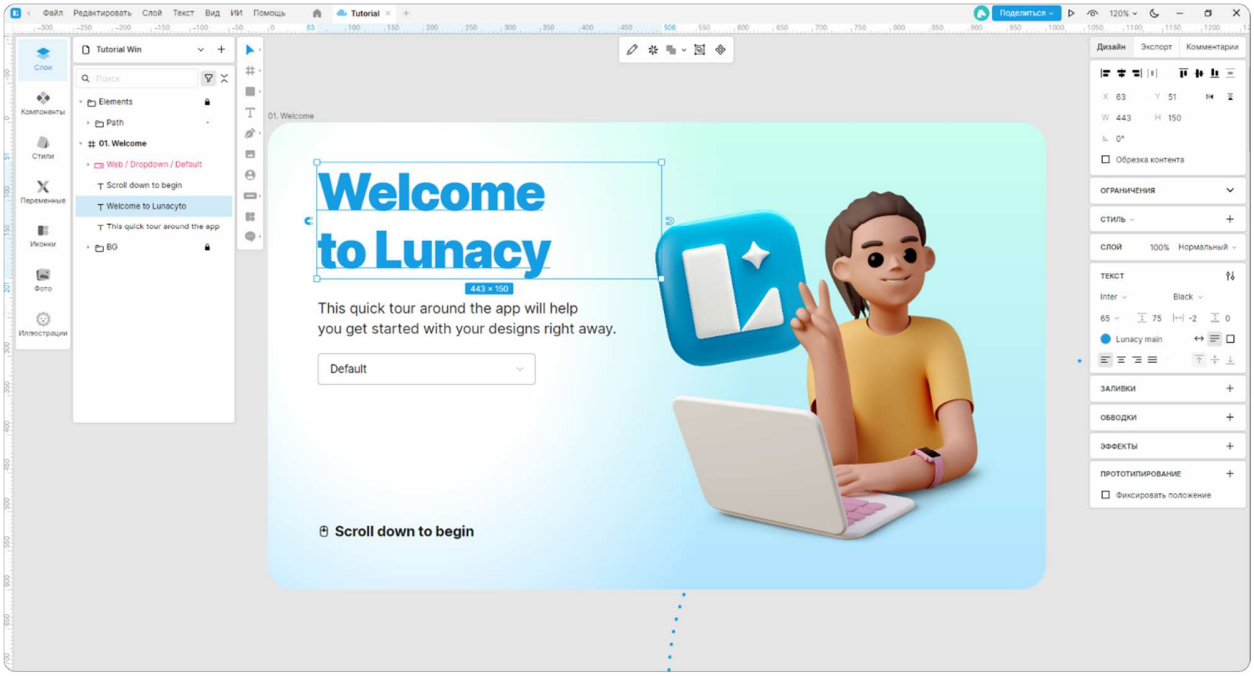

Рисунок 9 – Графический редактор Lunacy

Lunacy был создан в 2016 году как средство для просмотра файлов Sketch, но со временем превратился в самостоятельный графический редактор. Программа доступна для разных операционных систем, однако требует установки. Интерфейс достаточно удобен, но может показаться непривычным для некоторых пользователей. В редактор встроены ресурсы от Icons8, однако готовые шаблоны и инструменты для работы с палитрами и данными отсутствуют. Совместная работа поддерживается, но с ограничениями:

полноценное редактирование возможно только через облачное хранилище. В России Lunacy доступен без ограничений. Интерфейс полностью переведён на русский язык. Программа распространяется бесплатно, однако количество облачных документов ограничено. Так как Lunacy интегрирован с экосистемой Icons8, стоковые изображения доступны в базовом качестве. Для скачивания файлов в высоком разрешении требуется платная подписка.

Для сравнения инструментов было выбрано девять критериев. Работа в браузере удобна тем, что не требует установки и обеспечивает быстрый доступ. Интуитивно понятный интерфейс важен, поскольку пользователи чаще выбирают простые и быстро осваиваемые решения. Доступность для новичков делает платформу привлекательной для широкой аудитории. Шаблоны ускоряют работу, палитры помогают с выбором цветов, а функции работы с данными — с таблицами и графиками. Учитывались доступность сервиса в России, локализация и возможность бесплатного использования. Таблица 1 содержит результаты сравнения.

Таблица 1 – Сравнение инструментов по созданию визуального контента

Инструмент / **Критерий**	Figma	Canva	Pixso	Sketch	Lunacy
Работа в браузере	+	+	+	-	-
Удобство интерфейса	+	+	+	+	+
Доступность для новичков	-	+	-	-	-
Готовые шаблоны	±	+	±	±	-
Инструменты для работы с палитрой	-	+	-	-	-
Инструменты для работы с данными	-	+	-	-	-
Доступность сервиса на территории РФ	±	-	+	-	+
Локализация интерфейса	-	+	+	-	+
Бесплатный доступ	+	+	+	-	+

1.2. ОБЗОР СУЩЕСТВУЮЩИХ ПРОГРАММ И СЕРВИСОВ, ПРЕДНАЗНАЧЕННЫХ ДЛЯ ТЕСТИРОВАНИЯ ЦВЕТОВЫХ ПАЛИТР

Рассмотрим инструменты для создания и тестирования цветовых палитр.

Рисунок 10 показывает онлайн-сервис Coolors [7], разработанный Фабио Фантаччио. Этот инструмент генерирует случайные цветовые палитры.

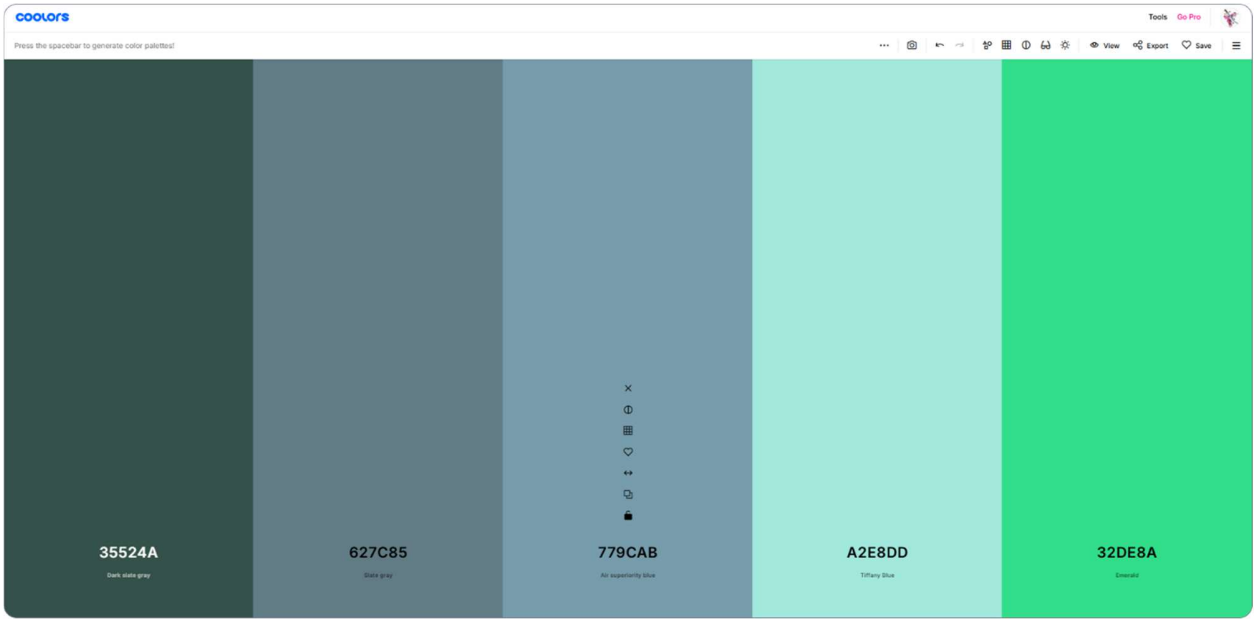

Рисунок 10 – Онлайн-сервис Coolors

Coolors отличается простотой использования: новую палитру можно получить нажатием пробела. Интерфейс интуитивен и предлагает широкий функционал. Среди возможностей — генератор палитр, большая библиотека популярных сочетаний и инструмент для извлечения цветовых схем из изображений. Пользователи могут тестировать палитры на различных макетах и проверять контрастность по стандартам WCAG. Также сервис интегрируется с Adobe и Figma через плагины и предлагает мобильные приложения для iOS и Android. Однако бесплатная версия имеет ограничения: детальная настройка палитр отсутствует, максимум — 5 цветов в палитре, работа только с одним проектом и одной коллекцией. Официальный сайт сервиса содержит рекламу.

Рисунок 11 представляет онлайн-инструмент Adobe Color [1], входящий в состав Adobe Creative Cloud и предназначенный для создания и настройки цветовых палитр.

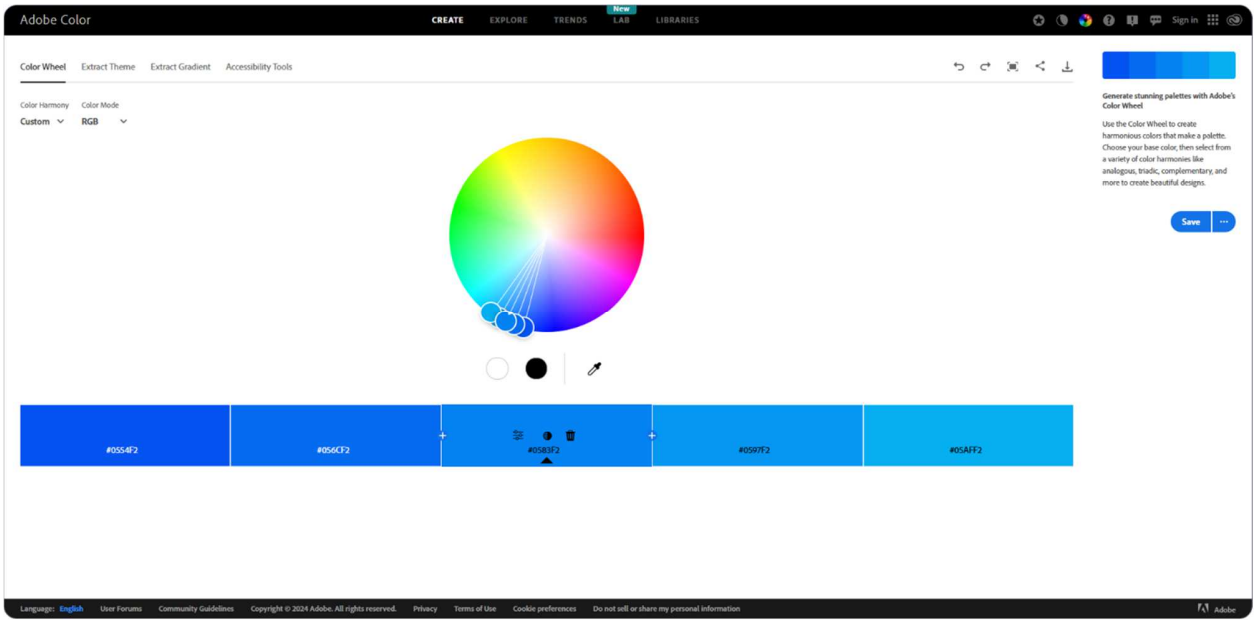

Рисунок 11 – Онлайн-инструмент Adobe Color

Платформа позволяет создавать и редактировать палитры с помощью цветового круга и различных схем. В сервисе есть большая коллекция готовых цветовых сочетаний, возможность извлекать палитры из изображений и проверять контрастность по стандарту WCAG. Благодаря синхронизации с Creative Cloud, палитры можно использовать в других продуктах Adobe. Основной недостаток — отсутствие функции тестирования палитр на реальных макетах, что затрудняет оценку результата. Кроме того, интерфейс может показаться сложным для новичков, создавая дополнительные трудности.

Рисунок 12 представляет онлайн-сервис Paletton [21], разработанный Хельмутом Вайнбергом. Инструмент предназначен для генерации цветовых палитр на основе классических цветовых схем.

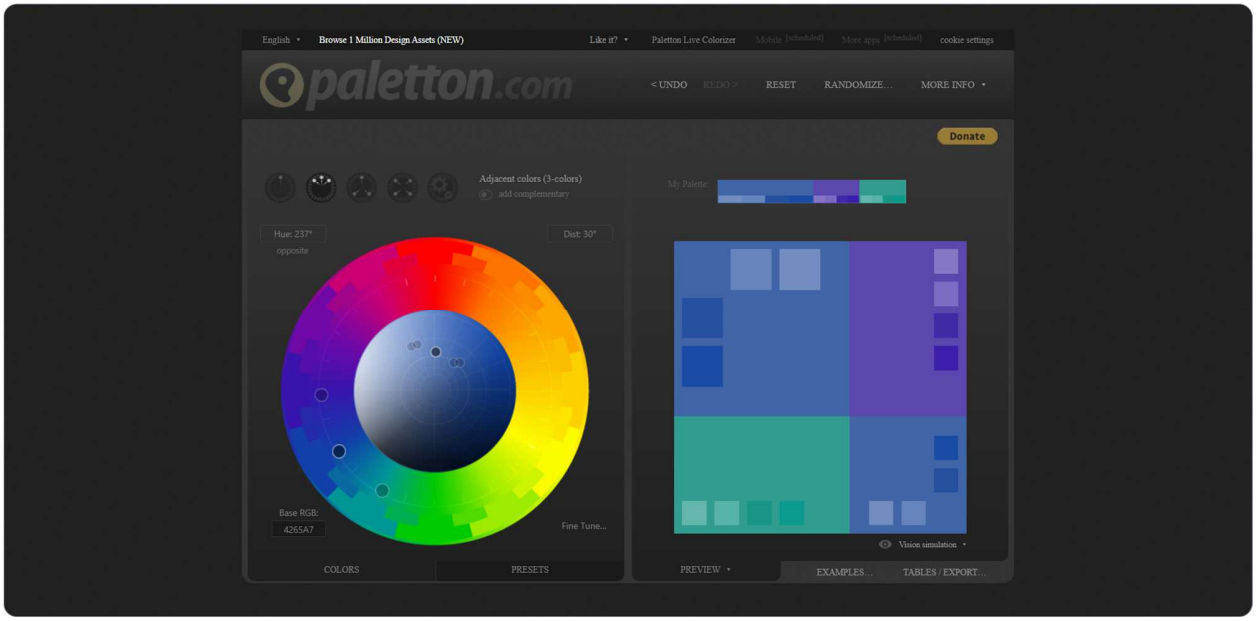

Рисунок 12 – Онлайн-инструмент Paletton

Преимущество Paletton — удобное управление и возможность создавать палитры, настраивая их по выбранным цветовым сочетаниям. Сервис визуализирует палитры, демонстрируя их в примерах пользовательских интерфейсов. Однако интерфейс Paletton выглядит устаревшим по сравнению с аналогичными инструментами. В сервисе отсутствует библиотека готовых палитр и функция проверки контраста по стандарту WCAG. Кроме того, поддерживается только RGB-формат, без возможности использовать другие цветовые коды, включая HEX.

Для сравнения инструментов были выбраны пять критериев. Простота использования влияет на эффективность работы и сокращает время освоения сервиса. Библиотека готовых палитр облегчает выбор и экономит время. Гибкость настройки позволяет адаптировать палитру под индивидуальные предпочтения. Возможность тестирования палитр помогает увидеть, как цвета будут выглядеть на макетах. Проверка контраста по стандарту WCAG важна для обеспечения доступности и удобства пользователей с нарушениями зрения.

Таблица 2 содержит результаты сравнения.

Таблица 2 – Сравнение сервисов по создания цветовых палитр

Инструмент / **Критерий**	Coolors	Adobe Color	Paletton
Простота использования сервиса	+	-	+
Наличие библиотеки готовых палитр	+	+	-
Возможность гибкой настройки палитры	-	+	+
Возможность тестирования палитр на макетах	+	-	+
Наличие функции проверки контраста по системе WCAG	+	+	-

1.3. ВЫВОД

В первой части проведён анализ предметной области и методической литературы по тематике работы, что позволило определить основные аспекты, влияющие на создание качественного дизайна. Обзор современных инструментов позволил оценить их возможности, преимущества и недостатки.

Сравнительный анализ показал, что большинство программ для дизайна сложны в использовании и не ориентированы на новичков. Исключением является Canva, предлагающая широкий выбор готовых шаблонов, но недоступная в России. Кроме того, во многих инструментах отсутствуют встроенные функции для работы с палитрами и данными, что что ограничивает пользователей и вынуждает дизайнеров использовать плагины, которые могут быть платными. Среди сервисов для работы с цветовыми палитрами наиболее функционален Coolors, однако его бесплатная версия имеет существенные ограничения. Таким образом, существует необходимость в создании универсального программного решения, ориентированного на новичков.

2. ПРОЕКТИРОВАНИЕ АРХИТЕКТУРНОГО РЕШЕНИЯ

2.1. ОПРЕДЕЛЕНИЕ ТРЕБОВАНИЙ

Требования к создаваемому продукту служат основой для определения его функциональности, оценки объёма работы и расстановки приоритетов. Чётко сформулированные требования ускоряют процесс разработки.

В рамках проекта были выделены три категории требований:

- общие или бизнес-правила;
- функциональные;
- нефункциональные.

Общие требования описывают ключевые правила и ограничения, которые определяют работу системы в соответствии с целями бизнеса. Функциональные требования определяют, какие задачи система должна выполнять для достижения пользовательских целей. Нефункциональные требования устанавливают стандарты и критерии, по которым система должна выполнять свои функции — включая производительность, безопасность, удобство использования и масштабируемость.

2.1.1 ОБЩИЕ ТРЕБОВАНИЯ

1. Интерфейс приложения должен соответствовать концепции брендбука компании «Ridex», включая использование фирменной цветовой палитры, логотипа, типографики и других элементов фирменного стиля.
2. В приложение должна быть интегрирована коллекция методических материалов по курсу «Дизайнер в ИТ», с возможностью их самостоятельного изучения пользователями.
3. Приложение должно содержать коллекцию готовых шаблонов дизайн-проектов, упрощающих старт для новых пользователей. Шаблоны

должны быть сгруппированы по категориям и быть доступными для копирования и редактирования.

4. Приложение должно быть доступно на русском языке, включая все элементы интерфейса, сообщения об ошибках и уведомления

2.1.2 ФУНКЦИОНАЛЬНЫЕ ТРЕБОВАНИЯ

1. Приложение должно предоставлять пользователям возможность регистрации и авторизации. Реализация должна включать безопасное хранение паролей и возможность авторизации через популярные сервисы.
2. Приложение должно обеспечивать создание и редактирование макетов. В редакторе должны быть инструменты для работы с текстом, формам, рисованием, а также возможность настройки свойства элементов дизайна, включая положение, размер, непрозрачность, цвет и обводку.
3. Приложение должно поддерживать функциональность совместной работы над проектами. Несколько пользователей могут одновременно просматривать и редактировать один и тот же проект с отображением изменений в реальном времени и системой управления доступом.
4. Клиентское веб-приложение должно включать графический интерфейс, обеспечивающий навигацию по материалам, доступ к шаблонам, настройку параметров проекта, предпросмотр и сохранение результатов работы.
5. Приложение должно быть адаптировано и для профессиональных дизайнеров, предоставляя инструменты для цветовых палитр с учетом требований WCAG для проверки контраста.

2.1.3 НЕФУНКЦИОНАЛЬНЫЕ ТРЕБОВАНИЯ

1. Приложение должно быть доступно через современный браузер без необходимости установки дополнительного программного обеспечения или плагинов на устройства пользователей.
2. Интерфейс приложения должен быть интуитивно понятным и удобным для пользователей, обеспечивая быструю и комфортную работу.
3. Серверная часть приложения должна разрабатываться на языке TypeScript и работать в среде Node.js.
4. Приложение должно обеспечивать защиту данных пользователей, включая безопасную аутентификацию и шифрование пароля.

2.2. АРХИТЕКТУРА ПРЕДЛАГАЕМОГО РЕШЕНИЯ

Проект будет реализован с использованием трехуровневой архитектуры, традиционной для веб-приложений [36]. Она обладает рядом ключевых преимуществ. Прежде всего, такая архитектура обеспечивает масштабируемость: каждый уровень можно независимо расширять в зависимости от потребностей, что позволяет эффективно распределять ресурсы и оптимизировать работу приложения. Кроме того, она повышает безопасность, поскольку клиент не имеет прямого доступа к данным, что снижает риск утечки информации и несанкционированного доступа к базе данных. Наконец, архитектура отличается высокой гибкостью — изменения или замена отдельных компонентов возможны без влияния на остальные части системы.

Структура включает три основных уровня:

- уровень представления — отвечает за взаимодействие с пользователем и отображение интерфейса;
- уровень бизнес-логики — обрабатывает запросы и управляет данными;

- уровень хранения данных — обеспечивает сохранение и доступ к информации.

Архитектура веб-приложения (Рисунок 13).

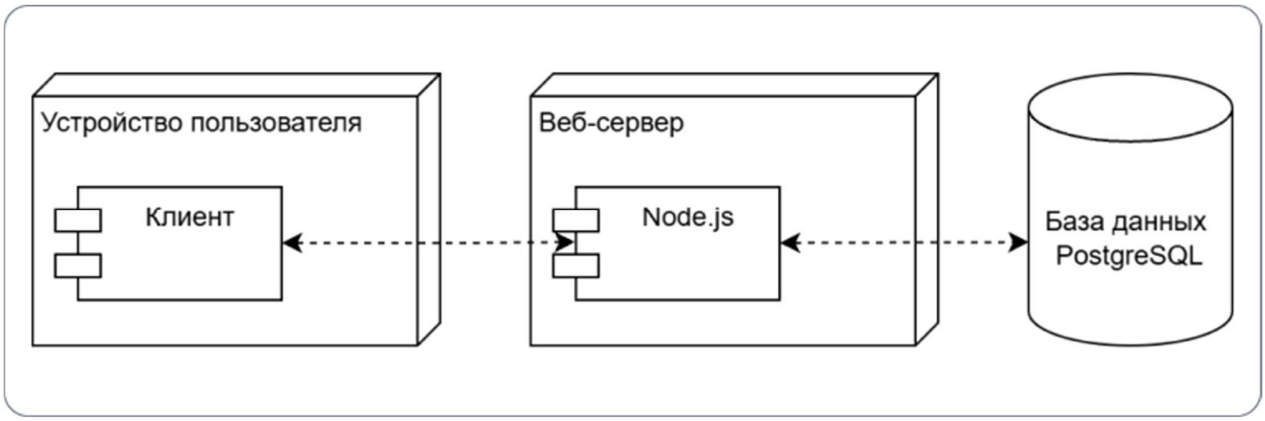

Рисунок 13 – Архитектура веб-приложения

Разрабатываемое веб-приложение будет написано на TypeScript [33], который, в отличие от JavaScript, использует статическую типизацию. Это позволяет выявлять ошибки на ранних этапах, улучшает читаемость кода и упрощает сопровождение проекта, повышая надёжность программного обеспечения.

Клиентская часть будет реализована с помощью фреймворка Next.js [18], который обеспечивает серверный рендеринг, статическую генерацию страниц, быструю загрузку и адаптивный интерфейс. Кроме того, он упрощает маршрутизацию через файловую структуру и облегчает взаимодействие с API.

Серверная часть будет работать на платформе Node.js [20], которая обеспечивает высокую производительность при асинхронных операциях и реализует событийно-ориентированную архитектуру. Это делает её подходящей для обработки большого количества одновременных подключений. Богатая экосистема библиотек и модулей упрощает интеграцию сторонних сервисов.

2.3. СРЕДСТВА РЕАЛИЗАЦИИ ПРИЛОЖЕНИЯ

В рамках проекта будет использован современный технологический стек T3 Stack [30], ориентированный на простоту, модульность и полную типизацию на всех уровнях разработки.

Клиентская часть будет реализована с использованием HTML [13], а стилизация интерфейса — с помощью CSS-фреймворка Tailwind CSS [31]. Его преимущество — возможность быстрой адаптивной вёрстки с применением утилитарных классов, что позволит сократить время на написание стилей CSS [8]. UI-компоненты будут взаимодействовать с сервером через HTTP/REST-запросы с использованием API-интерфейса fetch [12]. Страницы будут обращаться к серверу через API-маршруты, получая доступ к серверным действиям.

Аутентификация и управление сессиями пользователей будут реализованы с применением библиотеки NextAuth.js [19] которая обеспечит простую интеграцию с провайдерами и упростит управление пользователями.

Взаимодействие между клиентом и сервером будет организовано с помощью сервиса Liveblocks [15], предоставляющего API для совместной работы в реальном времени. Это позволит пользователям одновременно редактировать графические элементы и мгновенно видеть изменения.

Серверная часть будет также включать API-маршруты Next.js [17] и серверные действия. API-маршруты будут служить точками входа для HTTP-запросов и будут связаны с системой аутентификации через middleware, обеспечивающий проверку прав доступа. Серверные действия будут использоваться для выполнения операций на стороне сервера.

Структура данных приложения будет описана с помощью схемы базы данных, а взаимодействие с данными — реализовано через ORM Prisma [25].

Это обеспечит удобный синтаксис для типизированных запросов, снизит вероятность ошибок и упростит управление данными. В качестве СУБД будет использоваться PostgreSQL [23] — реляционная система, отличающаяся надёжностью, кроссплатформенностью и способностью обрабатывать большие объёмы информации.

С помощью EDraw Network Diagrammer [10] была спроектирована UML-диаграмма компонентов (Рисунок 14) для отображения взаимодействия компонентов веб-приложения.

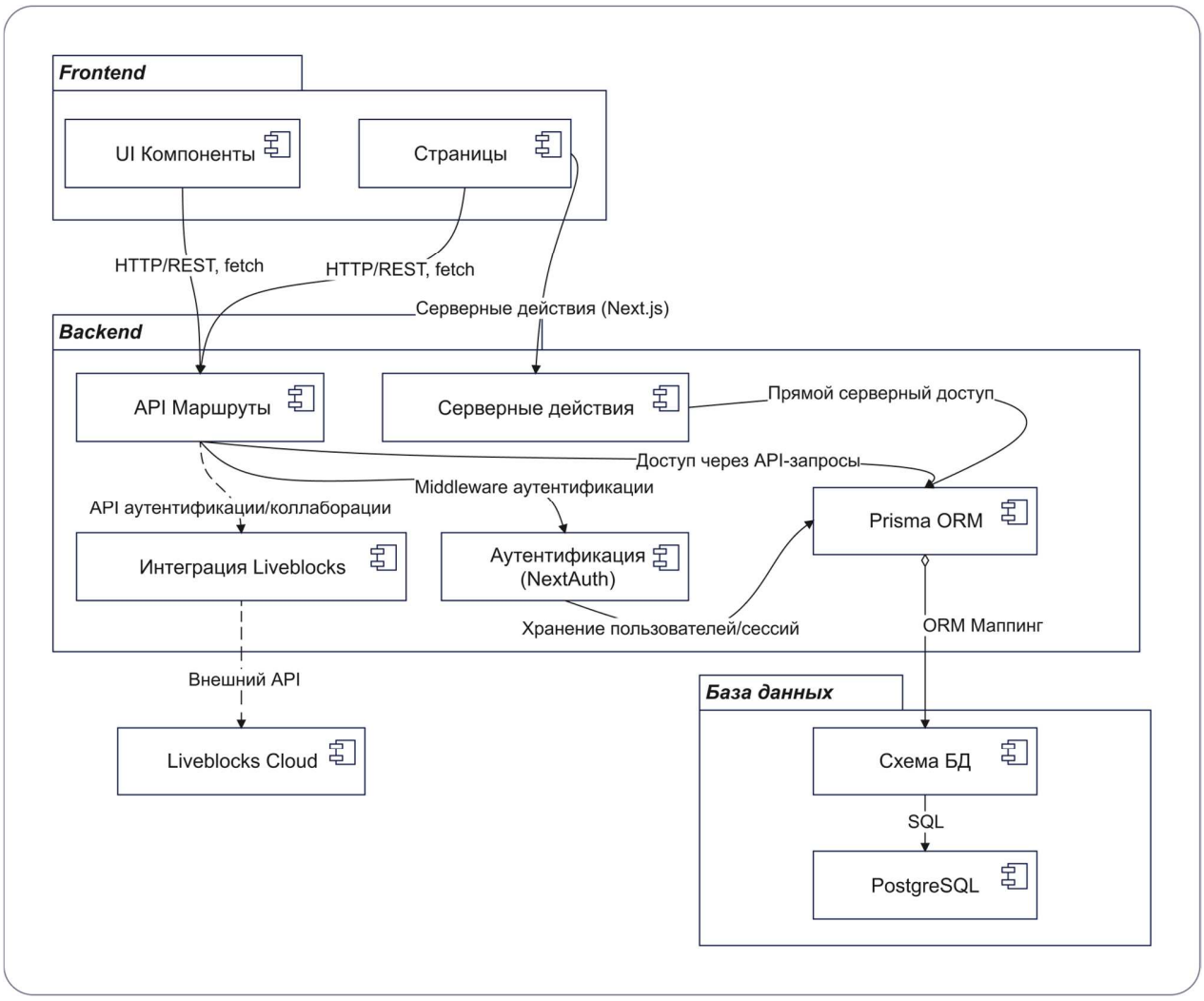

Рисунок 14 – UML-диаграмма компонентов

2.4. ОПИСАНИЕ БАЗЫ ДАННЫХ

База данных приложения включает шесть основных таблиц, каждая из которых соответствует отдельной сущности и хранит определённый тип информации. Схема хранения данных приложения (Рисунок 15).

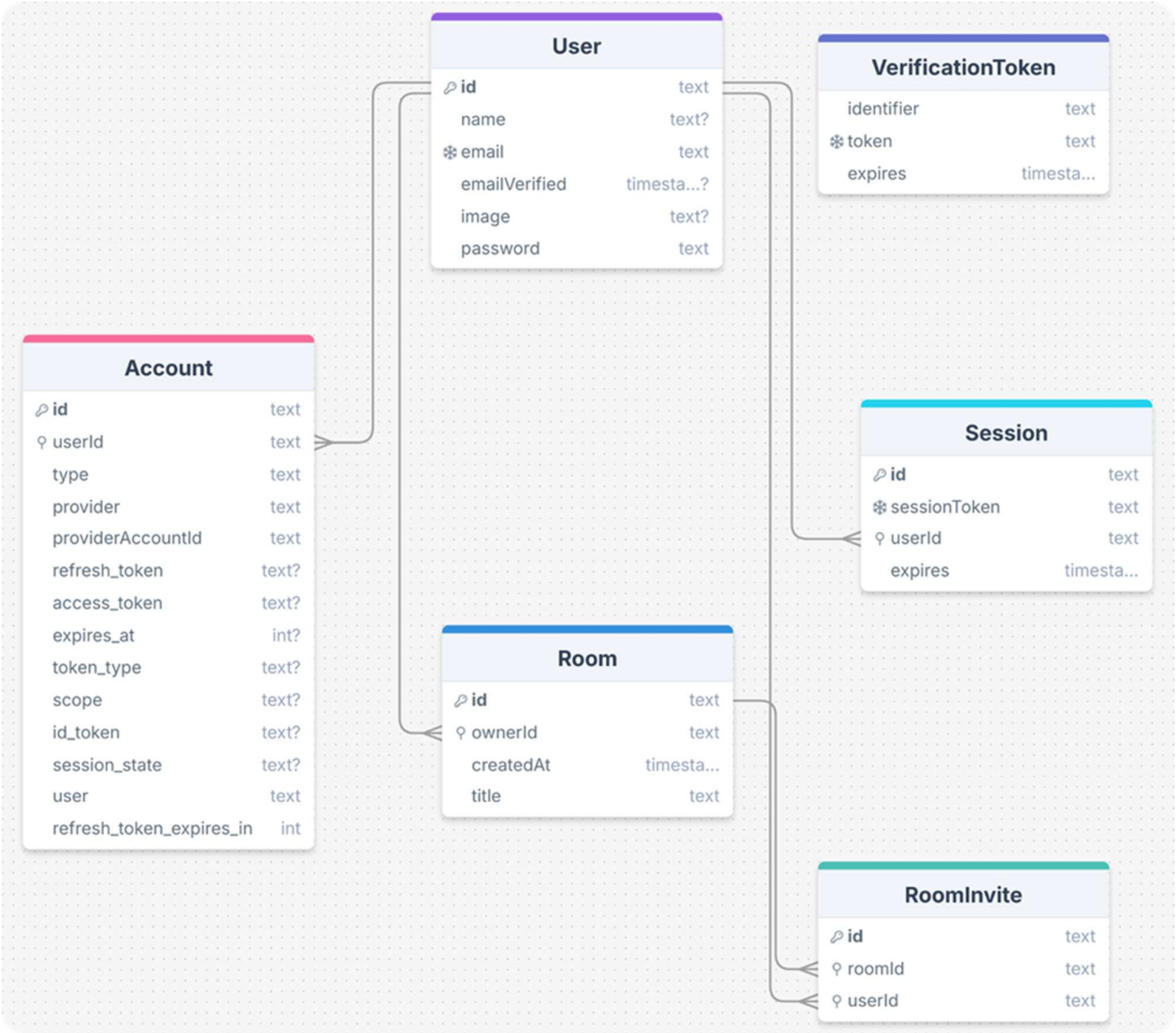

Рисунок 15 – Структура базы данных

Все данные приложения хранятся в PostgreSQL. Каждая таблица представляет сущность, связанную с пользователями, их сессиями, рабочими комнатами и другими ключевыми элементами системы. PostgreSQL

поддерживает широкий набор типов данных, обеспечивающих точное хранение и обработку информации. Тип данных определяет допустимые значения поля, способ их интерпретации и доступные операции.

Таблица User содержит основную информацию о пользователях:

- id (string) – уникальный идентификатор учетной записи, первичный ключ;
- name (string, опционально) – имя пользователя;
- email (string) – адрес электронной почты, уникальный индекс;
- emailVerified (datatime, опционально) – дата и время подтверждения электронной почты;
- image (string, опционально) – ссылка на изображение профиля;
- password (string, опционально) – хэшированный пароль пользователя.

Связи таблицы: один пользователь может иметь несколько учетных записей (Account), несколько сессий (Session), быть владельцем нескольких комнат (Room) и иметь несколько приглашений в комнаты (RoomInvite).

Таблица Session используется для хранения информации об активных сессиях пользователей в системе:

- id (string) – уникальный идентификатор сессии, первичный ключ;
- sessionToken (string) – уникальный токен сессии, уникальный индекс;
- userId (string) – идентификатор пользователя, внешний ключ (User.id);
- expires (datatime) – время истечения сессии.

Таблица VerificationToken служит для хранения временных токенов для подтверждения электронной почты:

- identifier (string) – идентификатор пользователя, часть составного ключа;
- token (string) – уникальный токен подтверждения, уникальный индекс;
- expires (datatime) – время истечения токена.

В таблице ограничение на уникальное комбинацию identifier и token.

Таблица Account хранит информацию об учетных записях пользователей и их привязке к провайдерам аутентификации:

- id (string) – идентификатор учетной записи, первичный ключ;
- userId (string) – идентификатор пользователя, внешний ключ (User.id);
- type (string) – тип учетной записи;
- provider (string) – название провайдера аутентификации;
- providerAccountId (string) – идентификатор учетной записи у провайдера;
- refresh_token (string, опционально) – токен обновления;
- access_token (string, опционально) – токен доступа;
- expires_at (int, опционально) – время истечения токена;
- token_type (string, опционально) – тип токена;
- scope (string, опционально) – область действия токена;
- id_token (string, опционально) – идентификационный токен;
- session_state (string, опционально) – состояние сессии;
- refresh_token_expires_in (int, опционально) – срок действия токена.

В таблице ограничение на уникальное сочетание полей provider и providerAccountId.

Таблица Room хранит информацию о комнатах для взаимодействия пользователей в рамках приложения:

- id (string) – уникальный идентификатор учетной записи, первичный ключ;
- ownerId (string) – идентификатор владельца комнаты, внешний ключ (User.id);
- createdAt (datatime) – дата и время создания комнаты;
- title (string) – название комнаты.

Связи таблицы: каждая комната имеет одного владельца (User) и может иметь несколько приглашений пользователей (RoomInvite).

Таблица RoomInvite содержит информацию о приглашениях пользователей в комнаты для совместной работы:

- id (string) – уникальный идентификатор учетной записи, первичный ключ;
- roomId (string) – идентификатор комнаты, внешний ключ (Room.id);
- userId (string) – идентификатор приглашенного пользователя, внешний ключ (User.id).

Связи таблицы: каждое приглашение связано с одной комнатой (Room) и относится к одному пользователю (User).

2.5. ЛОГИЧЕСКАЯ СТРУКТУРА ПРОЕКТА

Для наглядного представления приложения была разработана диаграмма классов (Рисунок 16), содержащая основные классы системы, их свойства, методы и взаимосвязи.

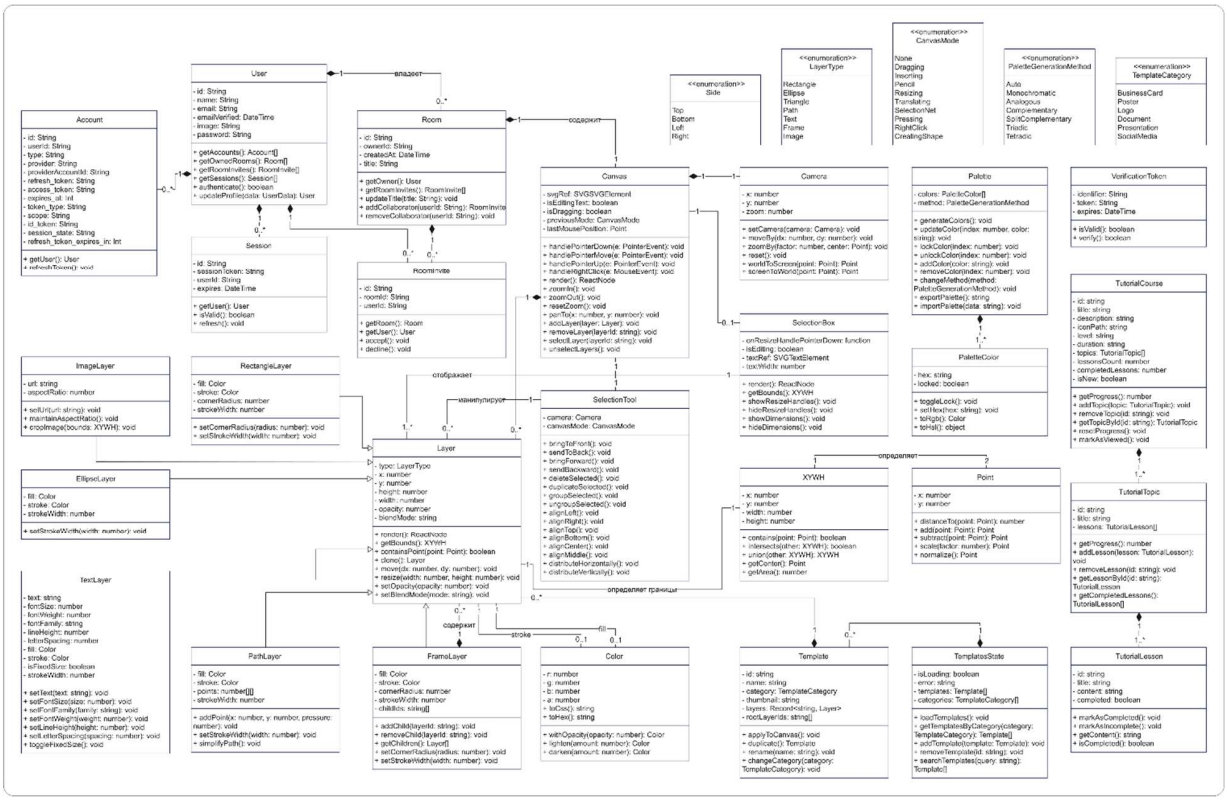

Рисунок 16 – UML-диаграмма классов

Подсистема управления пользователями включает классы User, Account, Session и VerificationToken. Эти классы обеспечивают хранение информации о пользователях, их сессиях, учётных записях и токенах подтверждения.

Подсистема совместной работы будет реализована с помощью классов Room и RoomInvite. Room представляет собой пространство для взаимодействия пользователей, а RoomInvite отвечает за приглашения в такие пространства.

Редактирование будет реализовано на основе класса Canvas, который управляет слоями дизайна. Визуализация холста будет обеспечиваться с помощью класса Camera. Абстрактный класс Layer служит базой для всех типов графических элементов: RectangleLayer, EllipseLayer, PathLayer, TextLayer, FrameLayer и ImageLayer. Каждый из этих классов будет представлять отдельный тип слоя с собственным набором свойств. Цветовая модель будет реализована через класс Color, поддерживающий систему RGB и прозрачность.

Инструменты выделения представлены классами SelectionTool и SelectionBox. SelectionTool предоставляет функциональность для управления выделенными элементами: изменения порядка слоёв, выравнивания и равномерного распределения. Класс SelectionBox будет визуализировать границы выделения и позволит изменять размеры выбранных объектов.

Вспомогательные сущности будут реализованы через классы Point и XYWH. Класс Point будет использоваться для описания координат, а XYWH — для представления прямоугольных областей по координатам, ширине и высоте.

Цветовая палитра будет формироваться с помощью классов Palette и PaletteColor. Palette управляет набором цветов и реализует методы генерации гармоничных цветовых схем. PaletteColor хранит информацию об отдельном цвете палитры и его текущем состоянии.

Обучающая подсистема будет включать классы TutorialCourse, TutorialTopic и TutorialLesson. Курс, представленный в виде класса

TutorialCourse, будет состоять из тем (TutorialTopic), каждая из которых будет включать несколько отдельных уроков (TutorialLesson).

Система будет использовать перечисления (enum) для описания различных состояний и категорий. LayerType — для определения типов слоёв, CanvasMode — для режимов работы холста, Side — для сторон выделения, TemplateCategory — для категорий шаблонов, а PaletteGenerationMethod — для методов генерации палитр.

Диаграмма отражает разнообразные типы связей между классами: композицию, агрегацию и ассоциации с указанием направлений и кратности. Так, пользователь (User) может иметь несколько учётных записей (Account) и сессий (Session); каждая комната (Room) содержит один холст (Canvas); а слой типа FrameLayer может включать в себя несколько других слоёв (Layer).

2.6. ПРОЦЕССЫ ВЗАИМОДЕЙСТВИЯ

Для моделирования взаимодействия пользователей с системой были разработаны диаграммы последовательности и состояний.

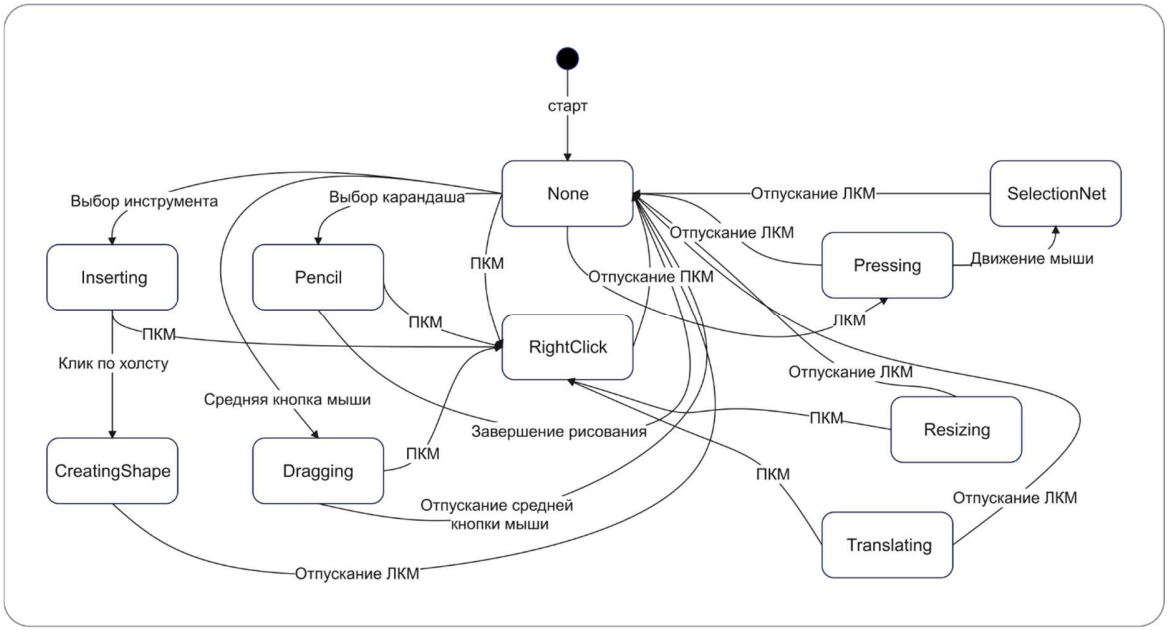

Рисунок 17 – UML-диаграмма состояний CanvasMode

Диаграмма состояний (Рисунок 17) иллюстрирует жизненный цикл пользовательского взаимодействия с редактором на основе модели «состояние — действие». Переходы между состояниями инициируются действиями пользователя.

Изначально система находится в нейтральном состоянии (None). При нажатии левой кнопки мыши она переходит в состояние Pressing, которое при движении мыши трансформируется в состояние SelectionNet — режим выделения области. При выборе инструментов создания графических примитивов активируется режим Inserting. В этом режиме доступны два способа создания фигур: одиночный клик добавляет элемент стандартного размера 100×100 пикселей, а удержание и перемещение курсора переводят систему в состояние CreatingShape, позволяя формировать произвольные размеры.

Выбор инструмента «Карандаш» активирует режим свободного рисования (Pencil). Нажатие средней кнопки мыши переводит систему в режим панорамирования холста (Dragging). Состояния Translating и Resizing, отвечающие за перемещение и масштабирование объектов, завершаются либо отпусканием левой кнопки мыши, либо вызовом контекстного меню.

Контекстное меню доступно из любого состояния при нажатии правой кнопки мыши, что обеспечивает гибкий и интуитивно понятный доступ к дополнительным функциям на любом этапе работы.

Каждое состояние завершается действием, возвращающим систему в нейтральный режим. Так, отпускание левой кнопки мыши после выделения завершает процесс и переводит систему обратно в состояние None.

Диаграмма последовательности (Рисунок 18) отражает ключевые этапы взаимодействия пользователей с системой в процессе редактирования дизайна. Диаграмма охватывает весь жизненный цикл проекта — от аутентификации и загрузки до финального сохранения.

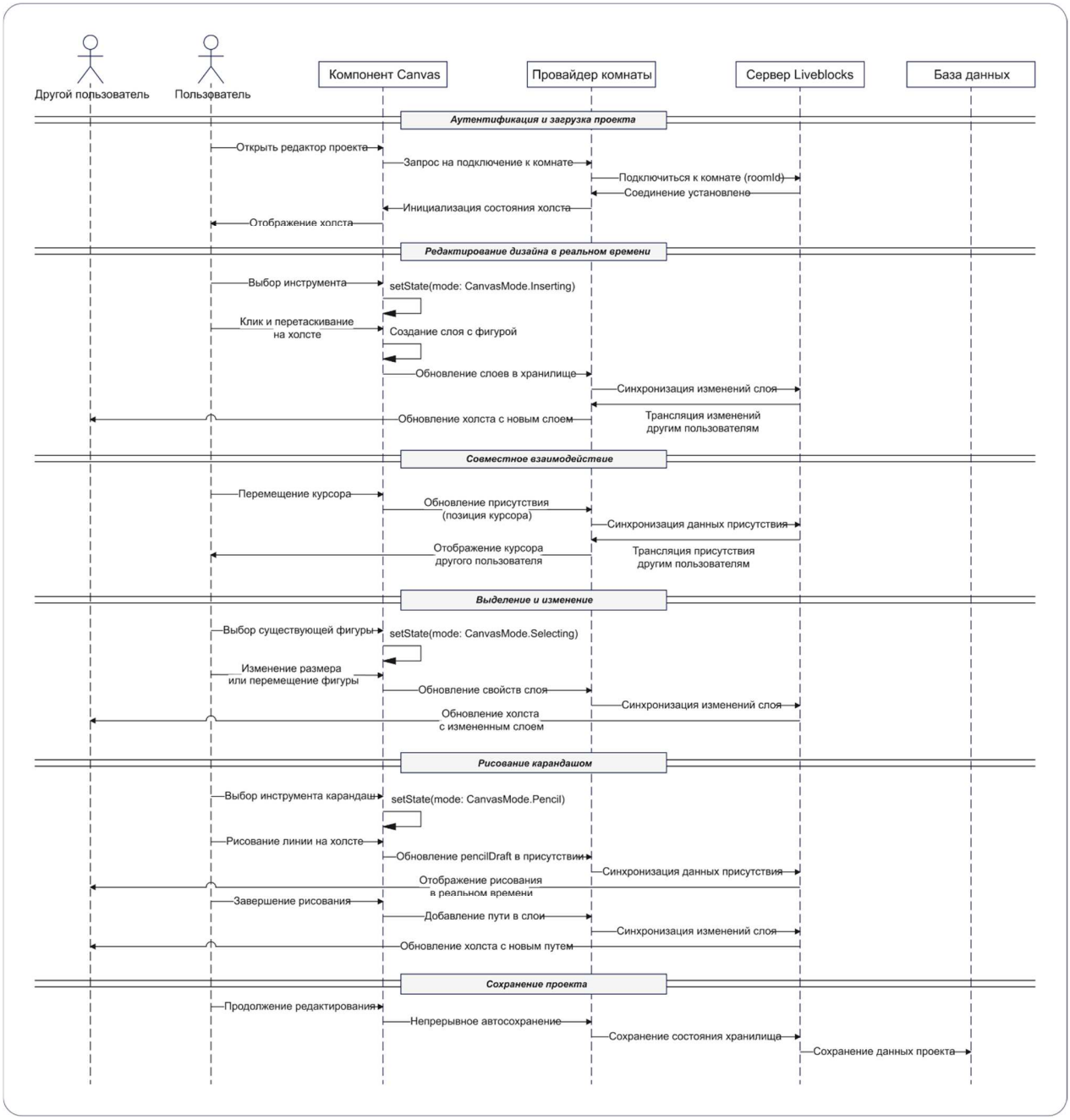

Рисунок 18 – UML-диаграмма последовательности

В процессе участвуют основные компоненты: текущий пользователь (инициатор редактирования), другой пользователь (соавтор), холст (canvas), провайдер комнаты (связующее звено между клиентом и сервером), сервер Liveblocks (обеспечивает синхронизацию) и база данных (хранит проекты).

Процесс условно разделён на несколько логических блоков.

На первом этапе пользователь открывает редактор, подключается к комнате через провайдера и сервер Liveblocks, после чего инициализируется состояние холста и загружается интерфейс.

Далее следуют базовые действия: выбор инструмента, создание графических элементов кликом или перетаскиванием, обновление слоёв и синхронизация изменений между участниками сессии.

При совместной работе система отслеживает действия других пользователей в реальном времени, включая перемещения курсора, благодаря механизму обновления данных присутствия.

Блок «Выделение и изменение» описывает действия по выбору объектов, их перемещению и масштабированию с последующей синхронизацией изменений между пользователями.

Отдельный раздел посвящён инструменту «Карандаш»: от его активации до отображения результатов в реальном времени и сохранения нарисованного пути как нового слоя.

Заключительный этап включает автоматическое сохранение проекта, передачу состояния в хранилище Liveblocks и запись окончательных данных в базу данных.

Реализация данного подхода обеспечивает высокую интерактивность системы и комфортные условия для совместной работы над дизайн-проектами, что является ключевым преимуществом разрабатываемого решения.

2.7. ФИЗИЧЕСКАЯ СТРУКТУРА ПРОЕКТА

Диаграмма развертывания (Рисунок 19) иллюстрирует физическую архитектуру приложения и демонстрирует распределение программных компонентов по аппаратным узлам.

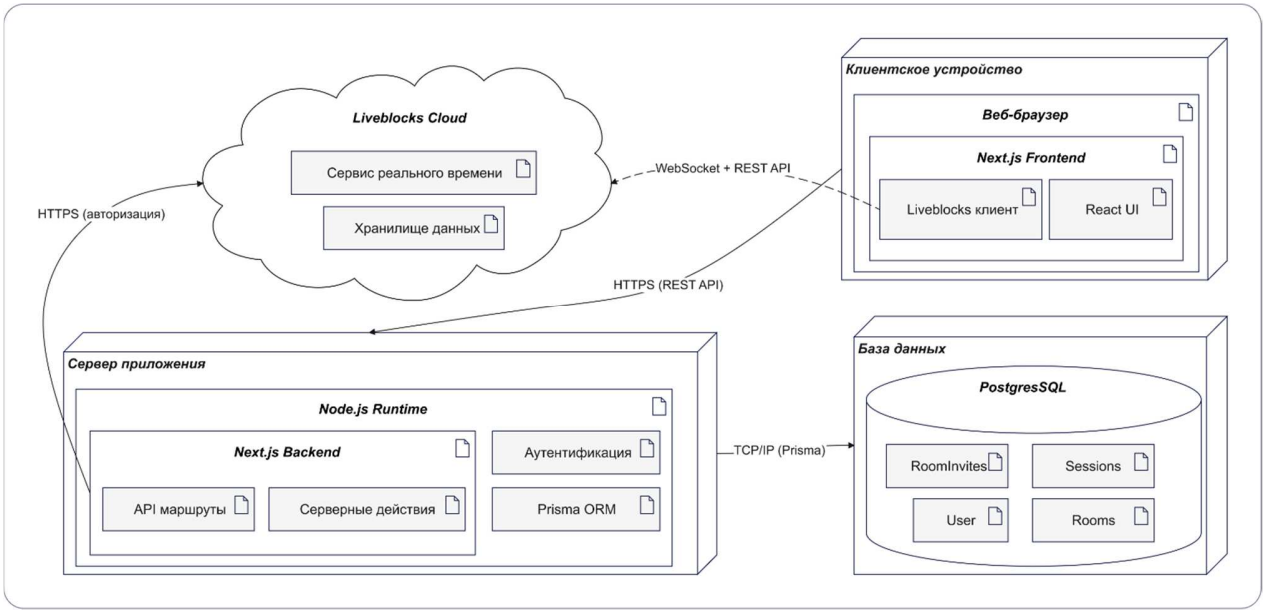

Рисунок 19 – Диаграмма развертывания

На клиентской стороне система представлена веб-приложением, где работает фронтенд на Next.js с React UI [27] и клиентской частью Liveblocks, отвечающую за синхронизацию в реальном времени и совместную работу.

На серверной части функционирует среда Node.js, запускающая серверную логику Next.js. Серверная архитектура включает маршруты API для обработки HTTP-запросов, функции серверного рендеринга, аутентификацию через NextAuth.js и взаимодействие с базой данных посредством Prisma ORM.

Компоненты связаны между собой следующим образом: клиент взаимодействует с сервером по протоколу HTTPS, используя REST API. Сервер обращается к базе данных PostgreSQL по протоколу TCP/IP через Prisma. Клиент Liveblocks подключается к облачному сервису Liveblocks по WebSocket и REST API, обеспечивая передачу и синхронизацию данных в реальном времени. Серверная часть Next.js также обращается к Liveblocks Cloud по HTTPS для выполнения авторизации. NextAuth.js использует PostgreSQL для хранения информации о сессиях и учётных записях.

2.8. ПРОЕКТИРОВАНИЕ ИНТЕРФЕЙСА

Для разрабатываемого веб-приложения с помощью Balsamiq Wireframe [3] спроектированы ключевые страницы: регистрация, авторизация, главная и редактор. Вайрфреймы — низкоуровневые схемы интерфейса, которые помогают определить структуру, расположение элементов и навигацию.

При первом запуске пользователю показывается страница регистрации (Рисунок 20).

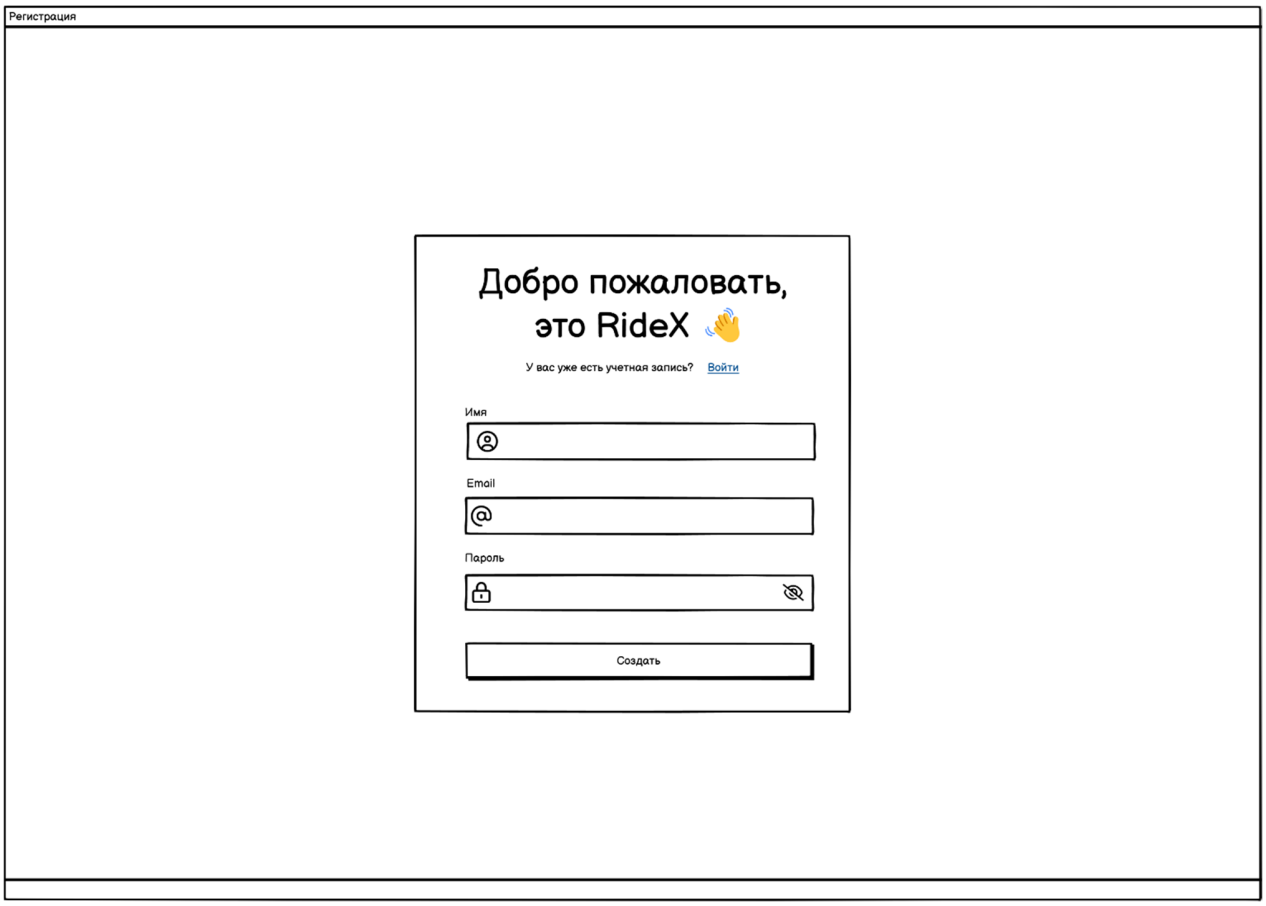

Рисунок 20 – Страница регистрации

На странице есть поле для имени, электронная почта, пароль с возможностью скрывать или показывать символы, кнопка создания учетной записи и ссылка на страницу входа для уже зарегистрированных пользователей.

Рисунок 21 показывает страницу авторизации.

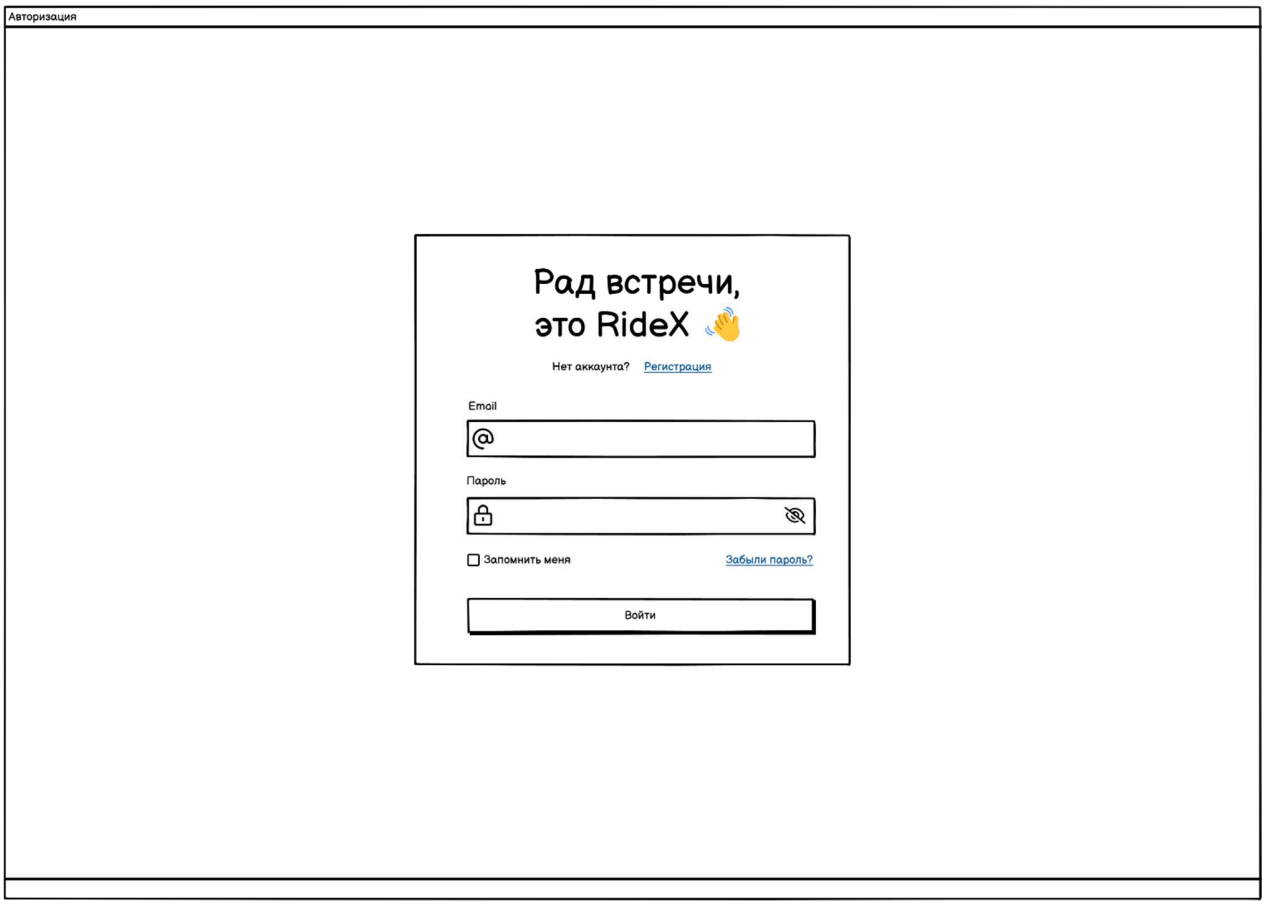

Рисунок 21 – Страница авторизации

Функциональность страницы авторизации схожа со страницей регистрации: поля для электронной почты и пароля с возможностью скрытия/показа, кнопка входа и ссылка на регистрацию для новых пользователей. Дополнительно предусмотрен чекбокс «Запомнить меня», сохраняющий данные с помощью cookie [6] для удобства последующих входов. Также есть ссылка на восстановление пароля.

После успешной авторизации пользователь перенаправляется на главную страницу приложения (Рисунок 22).

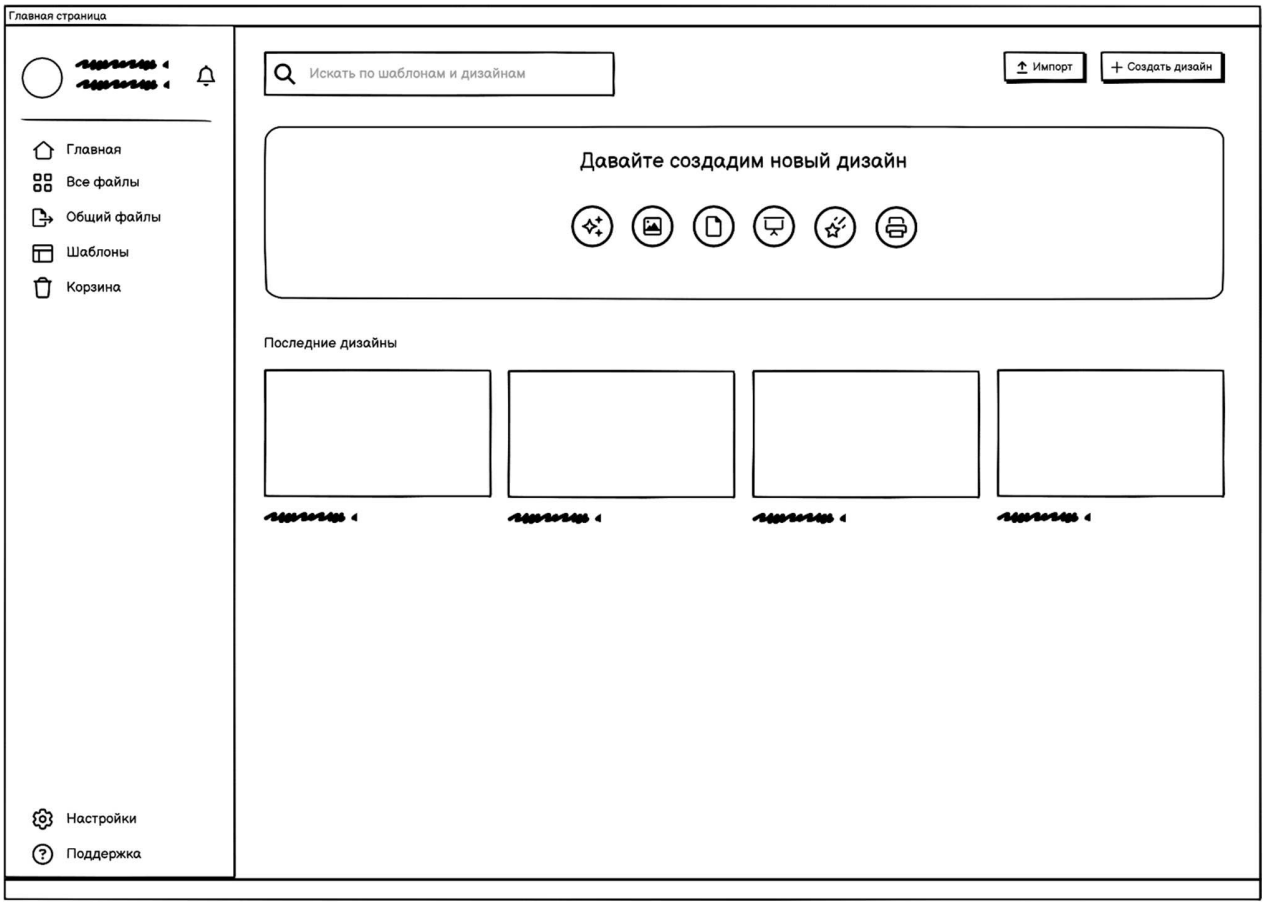

Рисунок 22 – Главная страница

Слева отображается информация о пользователе: аватар и адрес электронной почты, а также панель навигации по основным разделам приложения. Вверху размещена строка поиска по шаблонам и дизайнам, кнопки импорта файлов из внешних редакторов (например, Figma) и кнопка создания нового дизайна.

В центральной части — баннер с предложением создать дизайн по предустановленным шаблонам. Ниже находится блок недавно созданных файлов с названиями и датами создания.

Раздел «Все файлы» содержит все проекты пользователя, «Общие файлы» — проекты, к которым у пользователя есть доступ. В разделе «Шаблоны» собраны готовые макеты для работы.

При выборе файла пользователь попадает в редактор (Рисунок 23).

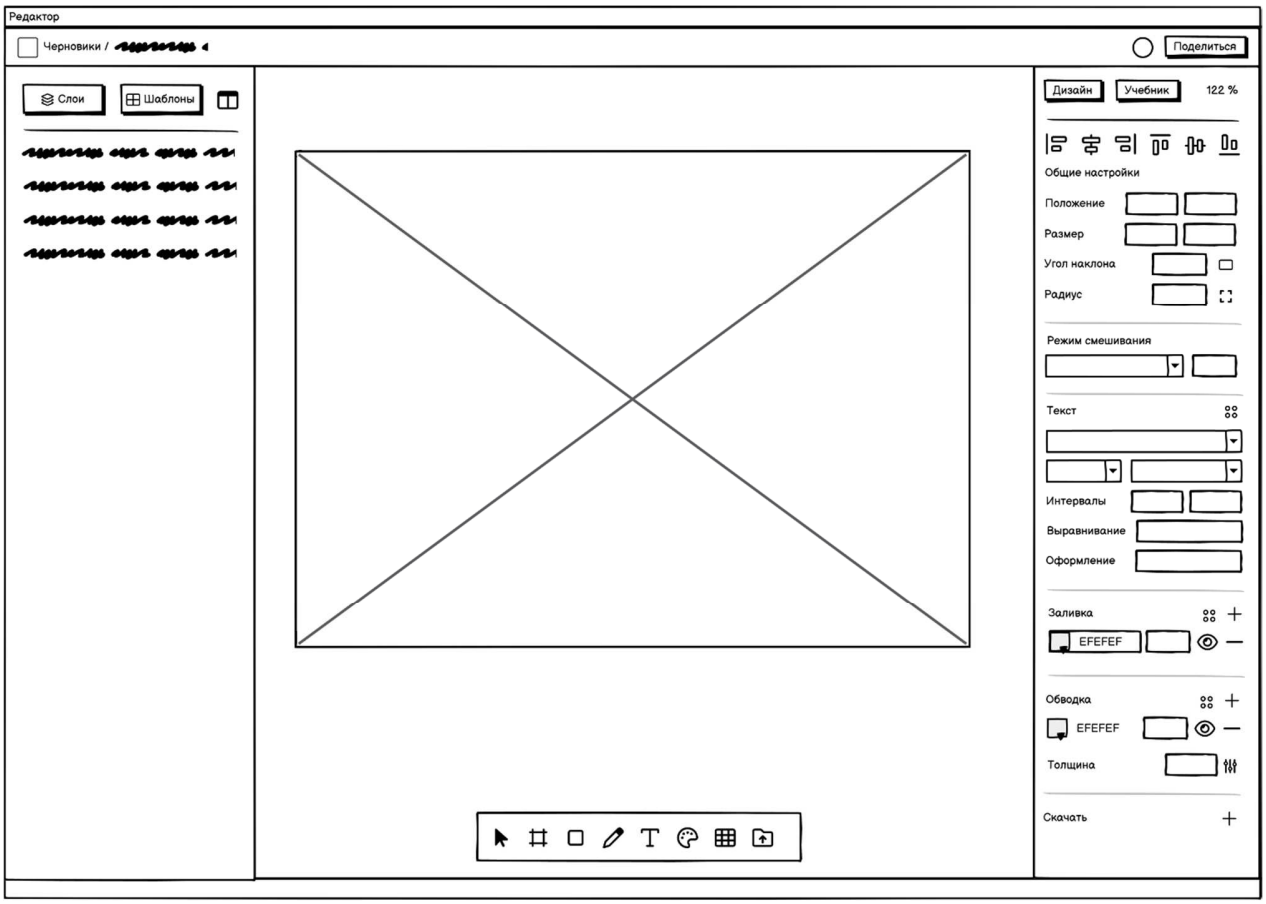

Рисунок 23 – Редактор

Редактор — ключевая часть приложения для создания и редактирования дизайнов. В верхней части отображаются название файла, список пользователей, работающих с ним в реальном времени, и кнопка «Поделиться», позволяющая скачивать файл или предоставлять доступ другим для просмотра и редактирования.

Слева находится боковая панель с двумя переключаемыми разделами: список слоев и библиотека шаблонов. Кнопка минимизации скрывает все панели, кроме панели инструментов.

Раздел слоев показывает все элементы на рабочем пространстве, где можно выделять, скрывать или закреплять отдельные слои.

Центральная часть — рабочее пространство в виде холста для редактирования дизайна. Внизу расположена панель инструментов с функциями перемещения, создания фигур, рисования, добавления текста, работы с цветами, создания таблиц и загрузки изображений.

Справа — панель свойств с настройками для выбранных элементов. Параметры зависят от объекта: например, для текста доступны настройки шрифта, заливки, обводки, режимы смешивания и другие визуальные параметры.

Также в панели свойств реализовано переключение между вкладками «Дизайн» и «Учебник по дизайну», что позволяет быстро получать информацию о функционале редактора и основах дизайна.

2.9. ВЫВОД

Во втором разделе выполнено проектирование разрабатываемого веб-приложения. Определены требования к системе, которые заложили основу для дальнейшей разработки. Разработана трехуровневая архитектура приложения, обеспечивающая масштабируемость, безопасность и гибкость системы. Выбранный технологический стек обеспечивает полную типизацию на всех уровнях разработки и высокую производительность. Спроектирована база данных PostgreSQL, состоящая из шести основных таблиц, описывающих сущности пользователей, сессий, рабочих комнат и других ключевых элементов системы. Построены диаграммы классов, последовательностей, состояний и развертывания, охватывающие ключевые подсистемы и отражающие жизненный цикл взаимодействия с редактором и распределение компонентов по узлам. Разработаны вайрфреймы ключевых страниц приложения: регистрации, авторизации, главной страницы и редактора, что позволило определить структуру интерфейса, расположение элементов и навигацию.

3. РАЗРАБОТКА ПРИЛОЖЕНИЯ

3.1. СОЗДАНИЕ ПРОЕКТА

Инициализация нового проекта включает несколько последовательных шагов. Сначала создаётся изолированное виртуальное окружение, затем устанавливаются необходимые инструменты, включая ключевой — Node.js. Зависимости добавляются через пакетный менеджер npm. Во время инициализации система выбираем параметры конфигурации. После выполнения этих шагов в файловой системе автоматически создаются директории и файлы:

- prisma/ – директория содержит файлы, описывающие изменения в структуре базы данных;
- schema.prisma – конфигурационный файл Prisma, в котором задаются параметры подключения к базе данных и выполняются миграции;
- public/ – изначально пустая директория для хранения общих файлов;
- src/ – основная директория с исходным кодом приложения.

Внутри src/ располагаются:

- app/ – управляет маршрутами и страницами приложения;
- [...nextauth]/ – динамический маршрут для обработки запросов аутентификации через NextAuth.js;
- route.ts – логика обработки маршрута авторизации;
- layout.tsx – задаёт общий макет страниц;
- server/ – содержит серверную логику;
- config.ts – настройки авторизации;
- db.ts – подключение к базе данных PostgreSQL;
- styles/ – стили оформления;
- globals.css – глобальные стили для всего приложения;

- env.js – работа с переменными окружения.

  Структура также включает ключевые конфигурационные файлы:

- .env – переменные окружения (например, данные БД и секретные ключи);
- .eslintrc.cjs – конфигурация линтера ESLint;
- next-env.d.ts – корректная интеграция TypeScript с Next.js;
- next.config.js – настройки фреймворка Next.js;
- package.json – метаданные проекта и список зависимостей;
- postcss.config.js – конфигурация PostCSS (используется с TailwindCSS);
- prettier.config.js – параметры форматирования кода с помощью Prettier;
- tailwind.config.ts – настройки TailwindCSS (цвета, типографика и др.);
- tsconfig.json – параметры компиляции TypeScript.

  Рисунок 24 отображает полученную структуру.

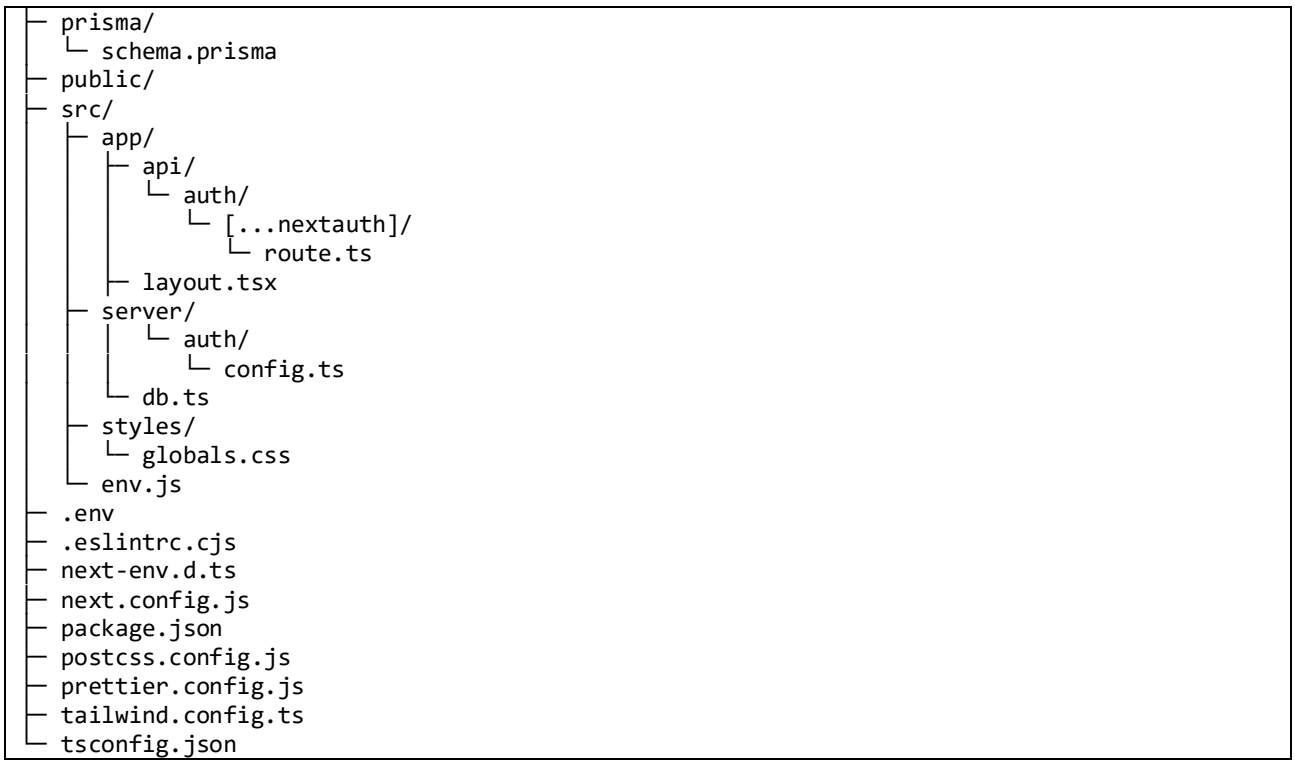

Рисунок 24 – Начальная структура проекта

В стандартной конфигурации T3 Stack по умолчанию включена интеграция с провайдером аутентификации Discord [9]. Это решение предназначено для демонстрации возможностей OAuth и упрощения начальной настройки, но в текущем проекте не используется. Для корректной инициализации необходимо удалить переменные окружения AUTH_DISCORD_ID и AUTH_DISCORD_SECRET из файлов .env, env.js и config.ts.

Для обеспечения единого стиля кода используется Prettier [24], автоматически форматирующий исходные файлы согласно заданным правилам. Для анализа и предотвращения ошибок применяется линтер ESLint [11], поддерживающий высокое качество и читаемость кода.

3.2. СОЗДАНИЕ БАЗЫ ДАННЫХ

T3 Stack автоматически генерирует файлы для работы с базой данных, включая конфигурационный файл schema.prisma с параметрами подключения к СУБД. Исходный код схемы (Листинг А.1 приложения А).

В проекте используется облачная платформа Supabase [29], предоставляющая бессерверные решения на базе PostgreSQL. Она обеспечивает автоматическое масштабирование в зависимости от нагрузки и защищает данные с помощью SSL-шифрования.

Фреймворк T3 Stack автоматически создаёт таблицы User, Account, Session и VerificationToken. Дополнительно проект расширен моделями Room и RoomInvite для поддержки функциональности совместной работы.

3.3. ИНТЕГРАЦИЯ БИБЛИОТЕКИ ДЛЯ ОПТИМИЗАЦИИ РАЗРАБОТКИ

Для ускорения разработки UI используется библиотека компонентов AlignUI [2]. Она предоставляет доступные и настраиваемые компоненты, поставляя их исходный код напрямую в проект. Это даёт полный контроль над внутренней логикой и внешним видом, устраняя зависимость от сторонних «чёрных ящиков». Компоненты AlignUI основаны на Radix UI [26], стилизованы с помощью Tailwind CSS и написаны на TypeScript.

Интеграция начинается с добавления базовых стилей командой npx @alignui/cli tailwind. После выбора параметров конфигурации интегрируются следующие вспомогательные инструменты:

- cn – утилита для динамического объединения классов Tailwind CSS;
- tv – расширение библиотеки tailwind-variants, адаптированное под AlignUI;
- recursiveCloneChildren – функция для рекурсивного клонирования дочерних элементов React;
- polymorphic – типизация для создания строго типизированных полиморфных компонентов.

3.4. РАЗРАБОТКА МОДУЛЯ АВТОРИЗАЦИИ И РЕГИСТРАЦИИ

Модуль авторизации обеспечивает безопасный доступ пользователей к функционалу веб-приложения. Конфигурация NextAuth включает провайдеры авторизации через GitHub, Google и с использованием email/пароля, настройку сессий и PrismaAdapter для работы с базой данных. Исходный код конфигурации (Листинг Б.1 приложения Б).

Валидация данных реализована с помощью библиотеки Zod [34], позволяющей создавать типизированные схемы. Исходный код схем валидации (Листинг Б.2 приложения Б).

В текущей реализации используются две основные схемы:

- signUpSchema для проверки данных при регистрации нового пользователя, включающая валидацию email, пароля и имени;
- signInSchema для проверки данных при входе пользователя, включающая валидацию email и наличия пароля.

На стороне сервера разработаны функции для аутентификации и регистрации. Исходный код серверных функций (Листинг Б.3 приложения Б).

Функция signout() отвечает за выход пользователя из системы, вызывая стандартный метод NextAuth. Функция authenticate(formData) получает email и пароль, проверяет наличие пользователя в базе данных, сравнивает введённый пароль с хранящимся хэшем с использованием bcrypt [4], при успешной проверке инициирует вход и возвращает сообщения об ошибках при необходимости. Функция register(formData) валидирует данные формы, проверяет уникальность email, хэширует пароль, создаёт запись нового пользователя и обрабатывает возможные ошибки валидации или выполнения.

Для защиты маршрутов реализован middleware-обработчик (Листинг Б.4 приложения Б), проверяющий наличие JWT-токена [14] в запросе. Если токен отсутствует, пользователь перенаправляется на страницу входа. Защите подлежат маршрут /dashboard и все его подпути.

Аутентификация работает следующим образом: клиент отправляет форму с данными, сервер валидирует их. При регистрации пароль хэшируется и сохраняется в базе данных. При входе сервер сравнивает хэш введённого пароля с сохранённым. В случае успеха создаётся JWT-токен, который используется в последующих запросах для доступа к защищённым ресурсам.

3.5. РЕАЛИЗАЦИЯ СОВМЕСТНОЙ РАБОТЫ

Взаимодействие между клиентской и серверной частями реализовано с использованием сервиса Liveblocks, обеспечивающего передачу и синхронизацию данных в реальном времени через WebSocket-соединения. Это позволяет пользователям совместно редактировать данные и мгновенно реагировать на действия друг друга.

Центральным понятием являются «комнаты» — изолированные виртуальные пространства для совместной работы. По умолчанию доступ к комнатам открыт, без обязательной аутентификации. В рамках проекта настроена авторизация с использованием токенов доступа: клиент идентифицируется через публичный ключ, а сервер генерирует токены с помощью секретного ключа.

Основной компонент, отвечающий за подключение к Liveblocks — Room (Листинг В.1 приложения В). Он инициализирует соединение, настраивает начальное состояние пользователя (выделенные объекты, положение курсора, цвет пера, черновик) и создает хранилище с параметрами (цвет фона, карта слоёв, список ID слоёв, буфер обмена). Компонент RoomContent отвечает за отображение загрузки в зависимости от статуса подключения.

Для управления доступом к комнатам разработаны серверные функции (Листинг В.2 приложения В).

В текущей реализации используются две основные схемы:

- createRoom() — создание комнаты и связывание с владельцем;
- updateRoomTitle(title, id) — изменение названия комнаты;
- deleteRoom(id) — удаление комнаты;
- shareRoom(id, inviteEmail) — отправка приглашения по email;
- deleteInvitation(id, inviteEmail) — отмена приглашения.

Все функции включают валидацию ID пользователя и проверку прав доступа.

API-маршрут для интеграции с Liveblocks (Листинг В.3 приложения В) инициализирует объект Liveblocks с использованием секретного ключа, получает текущую сессию через NextAuth, извлекает данные пользователя из базы и создаёт сессию Liveblocks с соответствующими правами. Этот маршрут играет ключевую роль в обеспечении безопасности, контролируя, к каким комнатам и с каким уровнем доступа может подключаться пользователь.

Интерфейс для совместной работы использует динамическую типизацию и реализован на странице редактора проекта (Продолжение приложения В

Окончание листинга В.3

```
// Инициализация Liveblocks с секретным ключом из переменных окружения
const liveblocks = new Liveblocks({ secret: env.LIVEBLOCKS_SECRET_KEY });

/**
 * Обработчик POST-запросов для аутентификации в Liveblocks
 * @param {Request} req - Объект запроса
 * @returns {Response} Ответ с данными сессии Liveblocks
 */
export async function POST(_req: Request) {
  // Получение текущей сессии пользователя из NextAuth
  const userSession = await auth();

  // Получение полных данных пользователя из базы данных
  // с информацией о его комнатах и приглашениях
  const user = await db.user.findUniqueOrThrow({
    where: { id: userSession?.user.id },
    include: {
      // Комнаты, созданные пользователем (владелец)
      ownedRooms: true,
      // Приглашения в комнаты, полученные пользователем
      roomInvites: {
        include: {
          room: true,
        },
      },
    },
  });

  // Подготовка сессии Liveblocks для пользователя с его профилем
  const session = liveblocks.prepareSession(user.id, {
    userInfo: {
      name: user.email ?? "Незнакомец",
      image: user.image ?? null,
    },
```

```
  });

  // Предоставляем полный доступ к комнатам, которыми владеет пользователь
  user.ownedRooms.forEach((room: { id: string }) => {
    session.allow(`room:${room.id}`, session.FULL_ACCESS);
  });

  // Предоставляем полный доступ к комнатам, в которые пользователь был приглашен
  user.roomInvites.forEach((invite: { room: { id: string } }) => {
    session.allow(`room:${invite.room.id}`, session.FULL_ACCESS);
  });

  // Авторизация сессии и получение данных для клиента
  const { status, body } = await session.authorize();

  // Возвращаем ответ с результатами авторизации
  return new Response(body, { status });
}
```


Листинг В.4 приложения В). Страница получает ID проекта из URL, извлекает сессию пользователя, запрашивает данные проекта из базы и отображает холст с инструментами редактирования. Компонент Canvas обёрнут в провайдеры Room и CanvasProvider для обеспечения синхронизации и контекста редактирования.

3.6. РЕАЛИЗАЦИЯ ИНТЕРАКТИВНОЙ ДОСКИ

Canvas — HTML-элемент для работы с растровой графикой, предоставляющий доступ к API Canvas и WebGL, позволяя управлять пикселями и создавать интерактивные визуальные элементы. Модуль Canvas (Листинг Г.1 приложения Г) реализует интерактивный редактор визуализации данных с модульной архитектурой. Компонент использует React-хуки для управления состоянием и обработки взаимодействия пользователя с холстом. Реализованы обработчики событий для рисования, выбора, трансформации объектов и перемещения. Ключевой элемент — система управления режимами (рисование, выделение, перемещение).

Для управления глобальным состоянием и доступа к функциям холста из разных компонентов используется React-контекст (Листинг Г.2 приложения Г). Он содержит данные о состоянии холста, положении камеры, слоях и инструментах. Это избавляет от «prop drilling» и упрощает архитектуру.

Масштабирование реализовано в хуке (Листинг Г.3 приложения Г). Он обрабатывает прокрутку колёсика мыши, вычисляя новое положение и масштаб холста. Поддерживаются панорамирование, масштабирование относительно курсора при зажатом Ctrl и программные функции zoomIn/zoomOut. Для предотвращения конфликтов с масштабированием браузера используется компонент (Листинг Г.4 приложения Г), блокирующий нежелательные жесты.

Выделенные элементы обрабатываются компонентом SelectionTool (Листинг Г.5 приложения Г), отображающим контекстное меню для управления порядком слоёв. Через мутации Liveblocks реализованы действия: перемещение на передний/задний план или на один слой вперёд/назад. Меню динамически рассчитывает своё положение с учётом масштаба и камеры.

Каждый компонент выполняет определённую функцию, а взаимодействие происходит через контекст и события. Поддерживается многопользовательский режим с отображением курсоров других участников и синхронизацией состояния. Для работы с большим числом элементов используются оптимизированные алгоритмы рендеринга и обработки событий, requestAnimationFrame для плавных обновлений и мемоизация компонентов для снижения перерисовок. Сложные графические преобразования инкапсулированы в функции вроде pointerEventToCanvasPoint и calculateBoundingBox, что упрощает поддержку кода и повышает его устойчивость к ошибкам.

3.7. РЕАЛИЗАЦИЯ УПРАВЛЕНИЯ СЛОЯМИ

Управление слоями построено на принципах паттерна «Компоновщик», обеспечивающего работу с графическими элементами и их группами через единый интерфейс.

Ключевой элемент — универсальный компонент (Листинг Д.1 приложения Д), определяющий тип слоя и выбирающий соответствующий компонент для отображения. Он получает данные слоя по ID и визуализирует элемент в зависимости от типа. В случае неопознанного типа возвращается null — это позволяет избежать ошибок при отрисовке. Для повышения производительности используется мемоизация, снижающая нагрузку при частичных изменениях.

Особое внимание уделено фреймам — контейнерам для других элементов. При их отрисовке создаётся область обрезки (clipPath), ограничивающая видимость дочерних элементов. Компонент рекурсивно вызывает себя для отображения вложенных слоёв.

Создание слоёв реализовано по паттерну «Фабричный метод» (Листинг Д.2 приложения Д). Интерфейс фабрики позволяет создавать прямоугольники, эллипсы, текстовые блоки, фреймы и изображения с набором базовых свойств — координатами, размерами, прозрачностью и режимом наложения. Для специфичных типов предусмотрены дополнительные параметры: для текста — шрифт, размер, межстрочный интервал; для фреймов — список дочерних элементов.

Компонент Rectangle (Листинг Д.3 приложения Д) отвечает за отрисовку прямоугольников. Он получает идентификатор слоя и объект с параметрами RectangleLayer. При визуализации учитываются координаты, размеры, цвет заливки и обводки, прозрачность и скругление углов. Реализована визуальная

обратная связь при наведении и выделении, а также масштабируемая толщина обводки. Для этого используются динамически изменяемые CSS-классы, зависящие от текущего масштаба холста.

Функции добавления новых слоёв определены в ShapeDrawingFunctions (Листинг Д.4 приложения Д). Хук useCreateLayerFunctions позволяет создавать слои по клику или при перетаскивании. Для вставки изображений используется событие imageDataReady, передающее URL, размеры и пропорции.

Логика изменения размеров и перемещения реализована в LayerManipulation (Листинг Д.5 приложения Д). Хук useLayerManipulation обрабатывает действия с маркерами изменения размеров, перемещением слоёв и взаимодействием с фреймами. При перемещении слоя внутрь фрейма он добавляется в список дочерних элементов. При выходе за пределы — перемещается в общий список. При перемещении фрейма все вложенные элементы следуют за ним.

Компоненты взаимодействуют через контекст холста и мутации Liveblocks. Для генерации уникальных идентификаторов используется nanoid, предотвращающий конфликты при одновременном создании слоёв разными пользователями.

3.8. РЕАЛИЗАЦИЯ МОДУЛЯ ИНТЕРФЕЙСА

Модуль интерфейса в редакторе представляет собой систему панелей и элементов управления для взаимодействия с холстом и его содержимым.

Центральный компонент UICanvas (Листинг Е.1 приложения Е) выступает контейнером для всех панелей и координирует их работу. Он управляет состоянием свернутых и развернутых панелей, передаёт данные о выбранных слоях и синхронизирует интерфейс с состоянием Liveblocks для

многопользовательского взаимодействия. Компонент также получает информацию о текущей комнате, пользователях и владельце проекта.

Верхняя панель (Листинг Е.2 приложения Е) отображает логотип, название проекта, список активных пользователей и меню управления доступом. Информация о пользователях получается с помощью хуков useSelf и useOthers из Liveblocks, позволяя в реальном времени отслеживать присутствие коллег. При большом числе участников реализован механизм показа максимум двух аватаров, а остальных — под счетчиком «+N».

Левая боковая панель (Листинг Е.3 приложения Е) предоставляет инструменты для работы со структурой слоёв и шаблонами. Она поддерживает два режима: «Слои» и «Шаблоны», переключаемых через вкладки.

Компонент LayerTree (Листинг Е.4 приложения Е) визуализирует иерархическое дерево слоёв с поддержкой вложенности, выделения и управления видимостью. Реализован рекурсивный подход для корректного отображения фреймов и их содержимого. Для каждого типа слоя показывается соответствующая иконка и имя. Для текстовых слоёв используется часть текста с ограничением по длине.

Правая боковая панель (Продолжение приложения Е

Листинг Е.5 приложения Е) служит для отображения и редактирования свойств выбранного слоя. Она динамически адаптируется к типу элемента, показывая только релевантные настройки. Для упрощения поддержки кода реализован паттерн «Адаптер», согласующий интерфейсы компонентов настроек с системой обновления слоёв. Это позволяет изменять структуру данных слоя, модифицируя только адаптеры, а не все компоненты настройки.

Модуль поддерживает разные состояния и режимы работы. При свернутой левой панели правая отображается полноценно только при выбранном слое, иначе также сворачивается.

3.9. РЕАЛИЗАЦИЯ МОДУЛЯ ШАБЛОНОВ

Модуль шаблонов предоставляет быстрый доступ к готовым элементам дизайна, повышая продуктивность пользователей и упрощает создание сложных композиций, особенно для пользователей без глубоких дизайнерских навыков.

Компонент TemplatesTab (Листинг Ж.1 приложения Ж) выступает контейнером для системы шаблонов в левой боковой панели. Он структурирует шаблоны по категориям, управляет состоянием активной категории и обеспечивает навигацию между ними.

Компонент TemplatesList (Листинг Ж.2 приложения Ж) отображает шаблоны выбранной категории в сеточной раскладке. Добавление шаблона на холст реализовано через drag-and-drop. При перетаскивании создаётся preview-изображение, следующее за курсором. При отпускании на холсте шаблон преобразуется в соответствующие слои с сохранением всех свойств и взаимосвязей. Для сложных шаблонов обеспечивается корректное создание иерархии слоёв — группировка и позиционирование.

3.10. РЕАЛИЗАЦИЯ МОДУЛЯ ОБУЧЕНИЯ

Модуль обучения предоставляет доступ к учебным материалам с поэтапным усложнением и сохранением индивидуального прогресса.

Компонент Tutorial (Листинг З.1 приложения З) отображает список доступных курсов, отслеживает и сохраняет прогресс пользователя в локальном хранилище браузера. Функции loadCoursesWithProgress и saveProgressToLocalStorage загружают и сохраняют состояние обучения соответственно. Структура данных представляет собой объект, где каждому курсу соответствует массив пройденных уроков.

Компонент CourseCard (Листинг З.2 приложения З) отображает название курса, иконку, описание, уровень сложности, продолжительность и индикатор прогресса. Состояние прохождения курса определяется автоматически: если все уроки завершены — курс считается пройденным, если хотя бы один урок завершён — он помечается как «в процессе», иначе — как «не начат». Система легко расширяется добавлением описания нового курса в массив tutorialCourses, где каждый курс представлен в виде объекта с метаданными.

Компонент TutorialModal (Листинг З.3Листинг З.2 приложения З) отображает содержимое курса с иерархической навигацией по темам и урокам слева и контентом активного урока справа. Доступ к следующему уроку регулируется функцией canAccessLesson, требующей завершения предыдущего. Переход между уроками осуществляется через функцию goToNextLesson, автоматически фиксирующую прогресс. Содержимое форматируется функцией formatContent, преобразующей Markdown-подобный синтаксис в стилизованный HTML.

3.11. РЕАЛИЗАЦИЯ МОДУЛЯ КОНСТРАСТНОСТИ

Модуль анализирует и отображает уровень контрастности между элементами интерфейса, что критично для читаемости контента.

Основная логика (Листинг И.1 приложения И) основана на стандарте APCA (Advanced Perceptual Contrast Algorithm), предлагающем более точную оценку по сравнению с WCAG 2.0 и рекомендованном для WCAG 3.0.

Функция calculateAPCAContrast рассчитывает контрастность с учетом особенностей восприятия цвета и яркости. Она принимает два HEX-цвета и возвращает значение от -108 до 106. Знак зависит от полярности (тёмный текст на светлом фоне или наоборот). Алгоритм включает преобразование цветов в

линейное пространство, вычисление яркости с учетом чувствительности к различным длинам волн и применение специальных коэффициентов APCA.

Функция evaluateAPCAContrast интерпретирует численное значение и присваивает качественную оценку с буквенным обозначением (AAA, AA) и описанием применимости (для основного текста или заголовков).

Визуальный компонент (Листинг И.2 приложения И) интегрирован в правую панель редактора. Он автоматически определяет фактический фон, вычисляя координаты слоя, находя центральную точку и проверяя, находится ли она внутри другого слоя с более низким z-index. Если да — он используется как фон, если нет — приоритет отдается родительскому фрейму или холсту. Благодаря функциям isLiveObject и универсальному геттеру getLayerProperty обеспечивается корректная работа как с обычными JavaScript-объектами, так и с реактивными LiveObject, что снижает вероятность ошибок и облегчает поддержку кода. Результаты анализа визуализируются через всплывающую подсказку с численным значением, соответствием WCAG и краткой рекомендацией.

3.12. РЕАЛИЗАЦИЯ ГРАФИЧЕСКОГО ИНТЕРФЕЙСА

Для удобства взаимодействия разработан интуитивно понятный интерфейс. Модули регистрации (Рисунок 25) и авторизации (Рисунок 25Рисунок 26) содержат поля для ввода электронной почты и пароля с функциями отображения/скрытия введённого пароля. Модуль регистрации дополнительно включает поле для ввода имени.

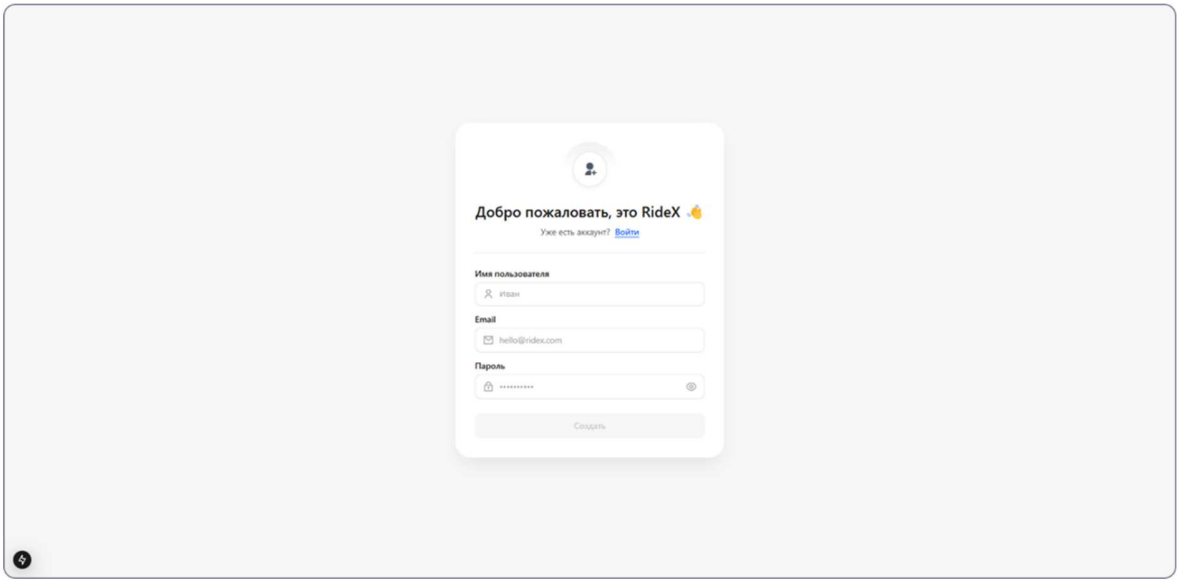

Рисунок 25 – Форма регистрации нового пользователя

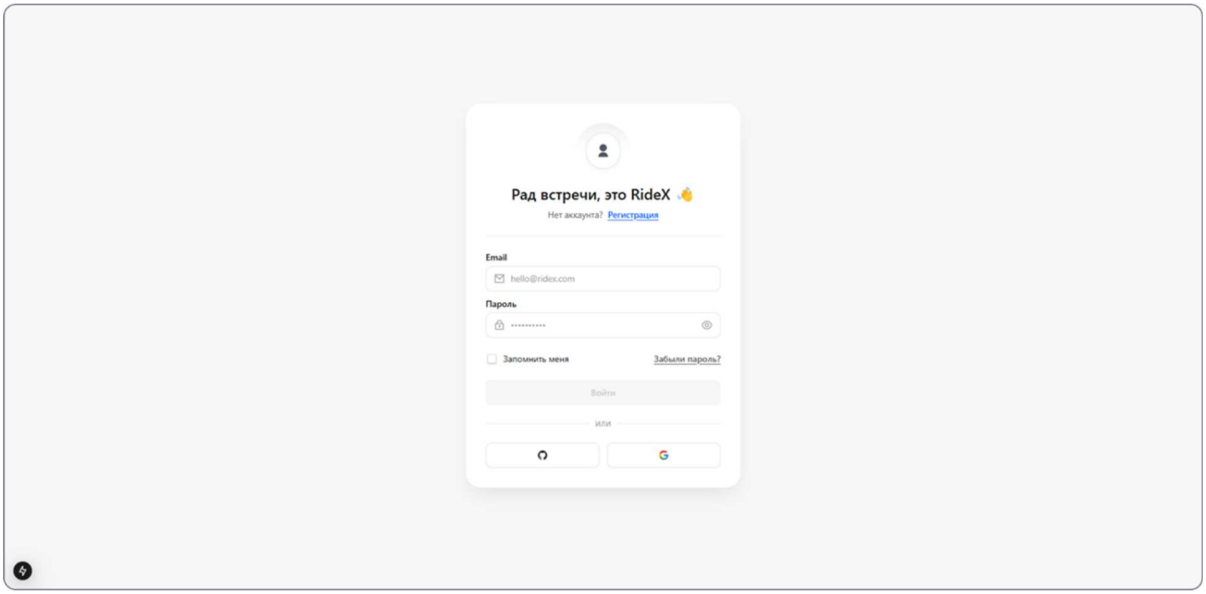

Рисунок 26 – Форма авторизации пользователя

После входа пользователь попадает на главную страницу (Рисунок 27) с баннером для создания нового дизайна и ссылками на ключевые функции. Ниже расположен блок «Последние дизайны», который изначально пуст. Слева находится навигационное меню с данными о пользователе.

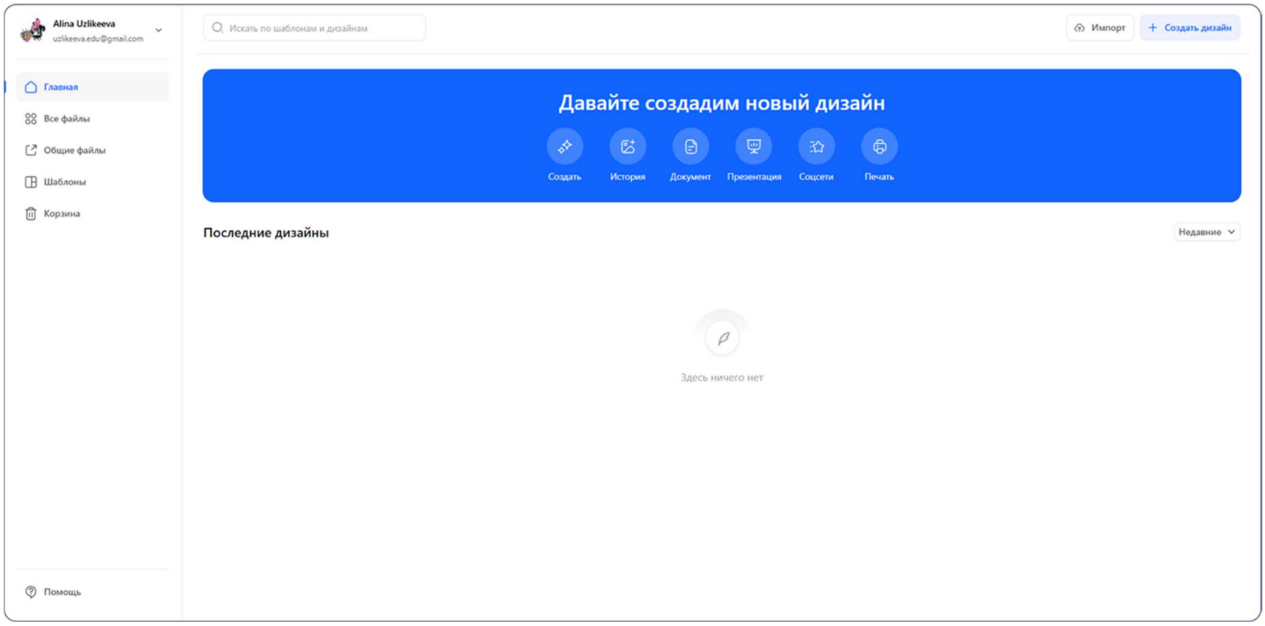

Рисунок 27 – Главная страница

На странице «Все файлы» (Рисунок 28) отображаются все проекты пользователя. При нажатии правой кнопкой мыши открывается контекстное меню с опциями переименования или удаления.

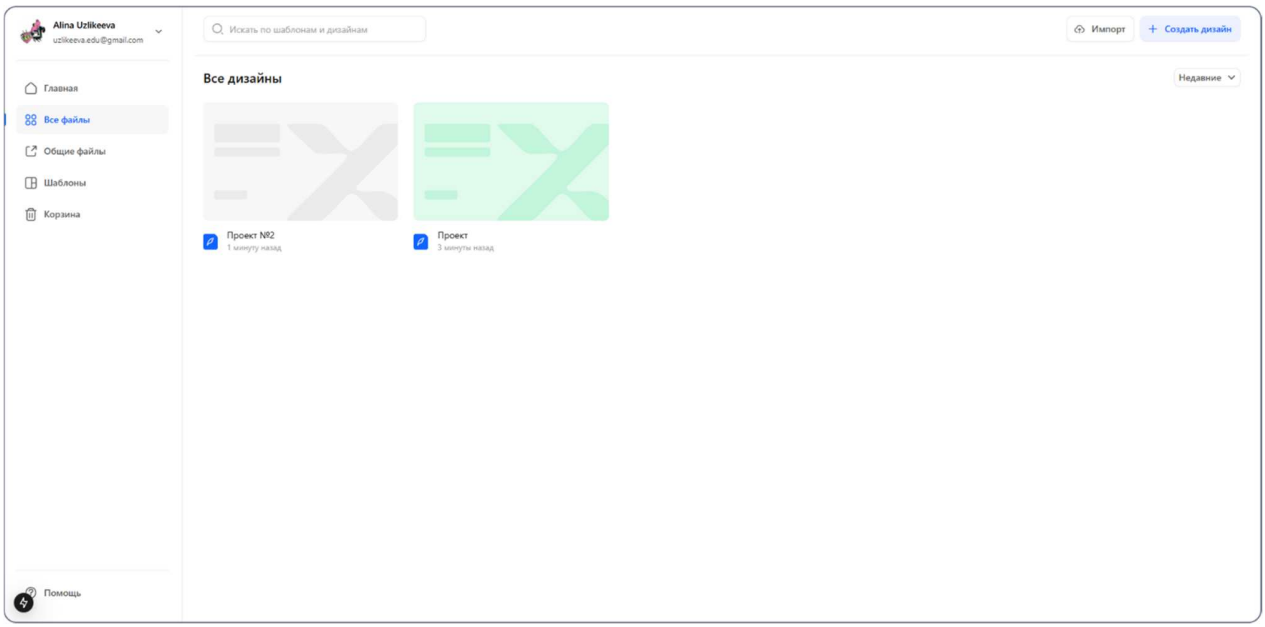

Рисунок 28 – Раздел «Все файлы»

Раздел «Общие файлы» (Рисунок 29) содержит проекты других пользователей с открытым доступом. На странице «Шаблоны» представлены все доступные дизайн-шаблоны для использования в работе. Вкладка «Корзина» хранит удалённые файлы; по истечении 90 дней они автоматически удаляются.

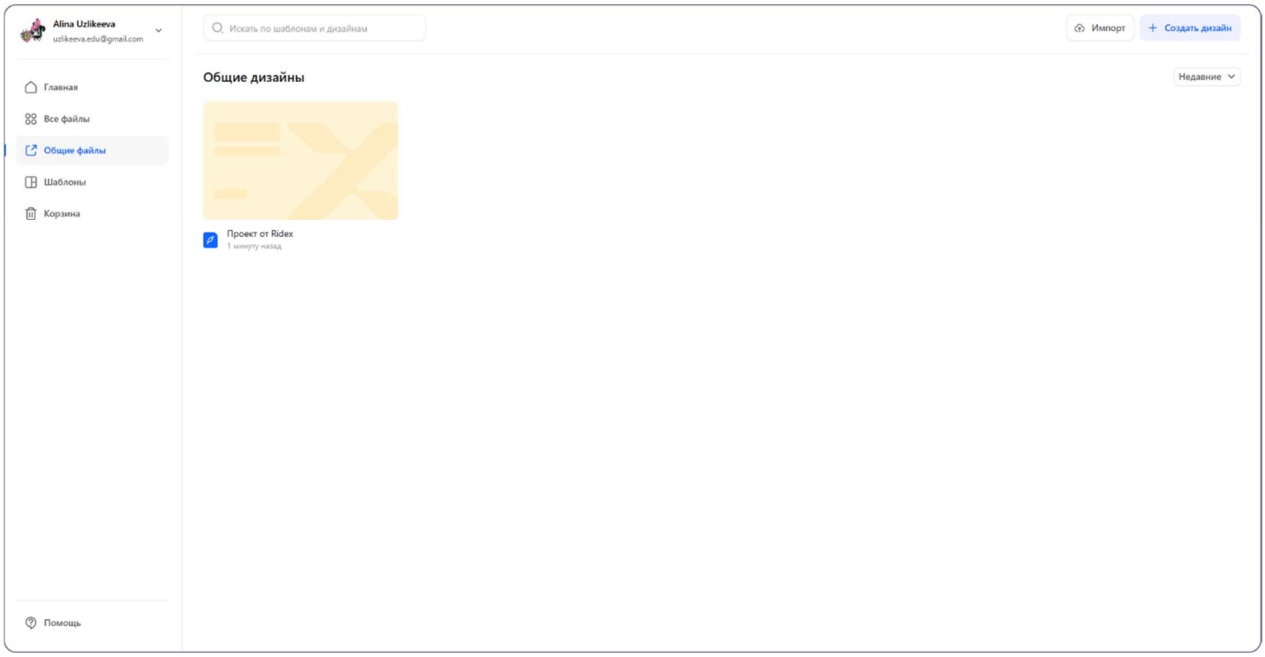

Рисунок 29 – Раздел «Общие файлы»

Рисунок 30 и Рисунок 31 представляют компактный интерфейс редактора. Центральное пространство занимает рабочая область. Слева расположена панель с названием текущего проекта, справа — панель свойств. Внизу находится основная панель инструментов. Такой подход максимизирует полезное пространство холста и помогает пользователю сосредоточиться на задаче.

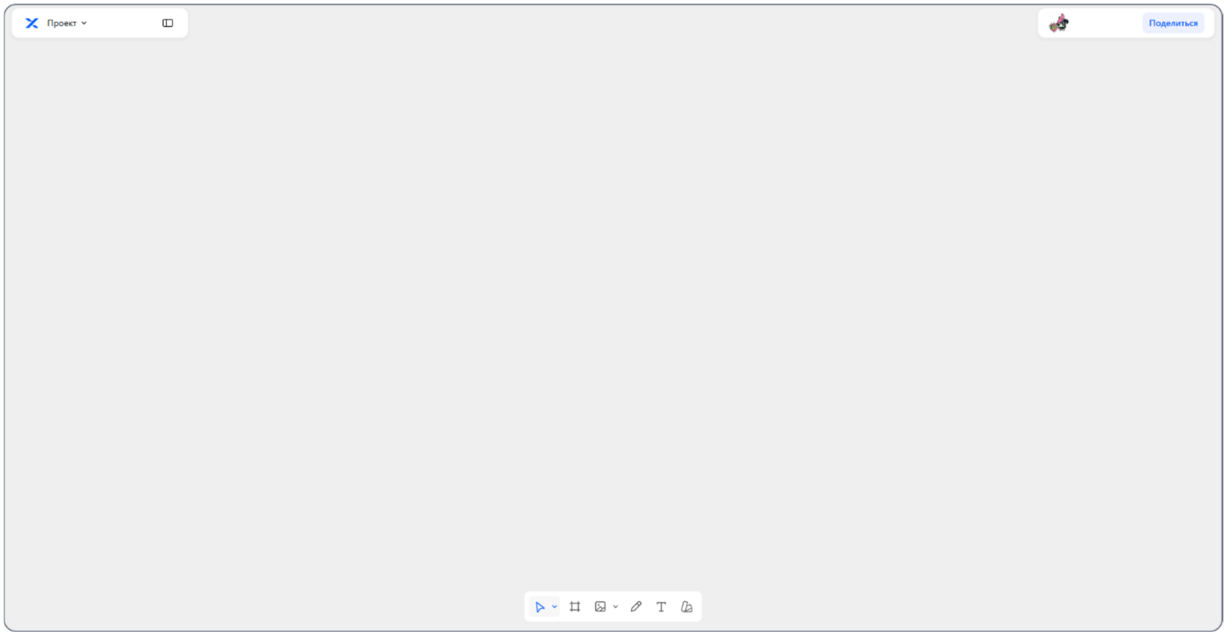

Рисунок 30 – Интерфейс редактора в скрытом виде

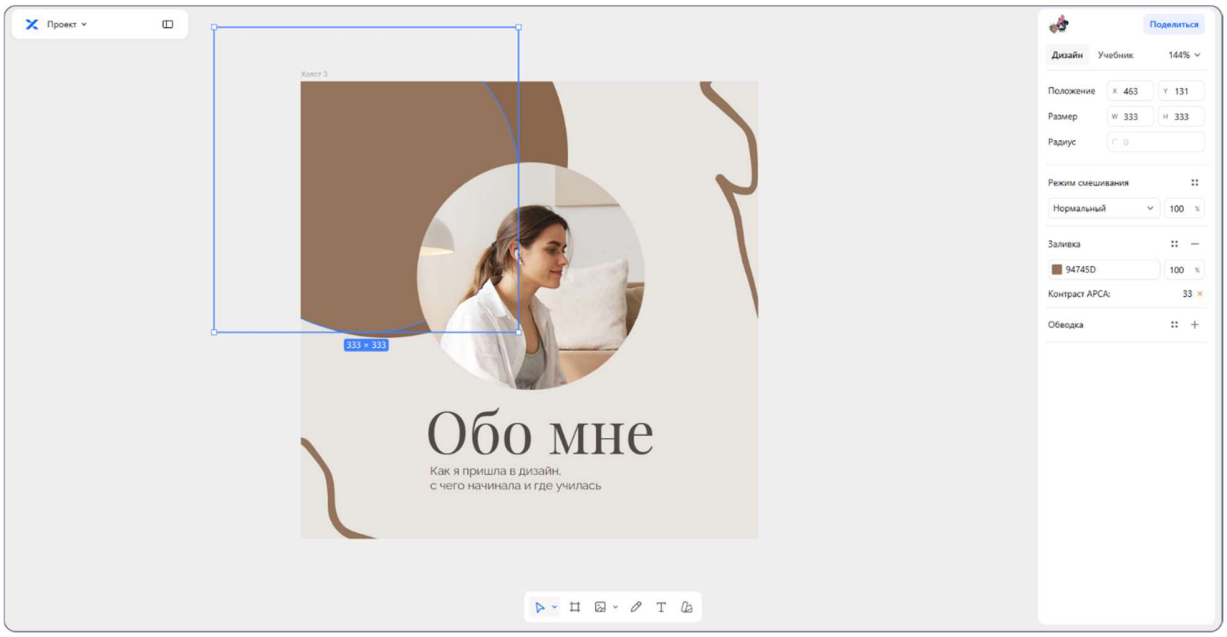

Рисунок 31 – Интерфейс редактора в скрытом виде с дизайном

Рисунок 32 и Рисунок 33 показывают полную версия интерфейса редактора. Слева отображается иерархический список элементов. Справа

расположена панель свойств для редактирования параметров выделенных элементов. Например, для текстовых слоёв доступны параметры шрифта, размера и межстрочного интервала, для графических — заливка, обводка и прозрачность.

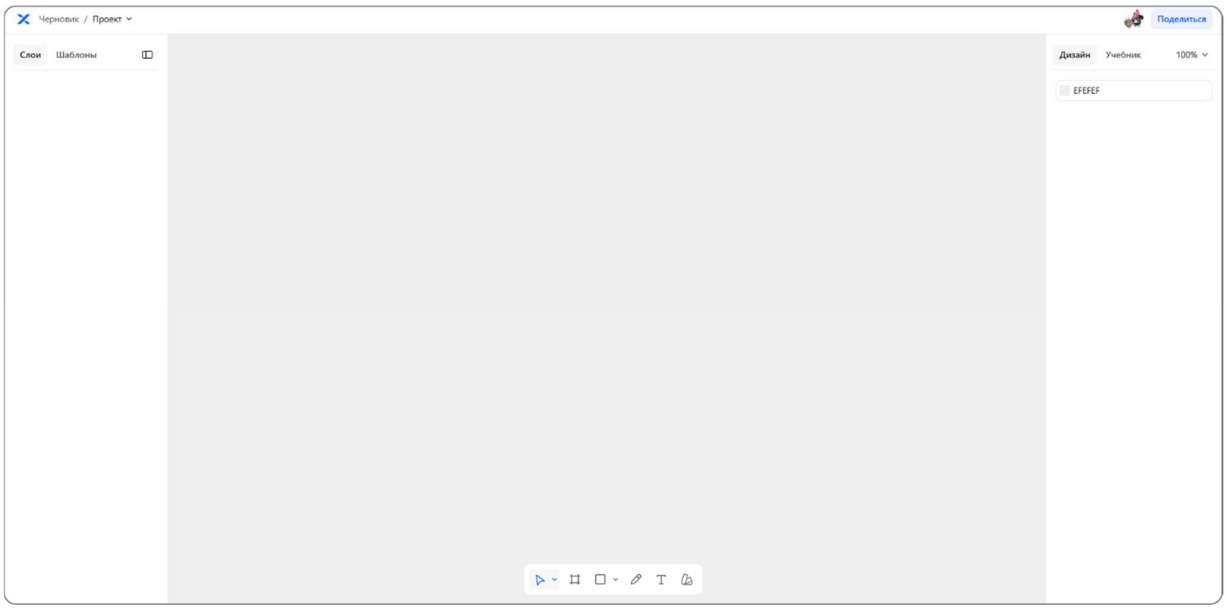

Рисунок 32 – Интерфейс редактора в полном виде

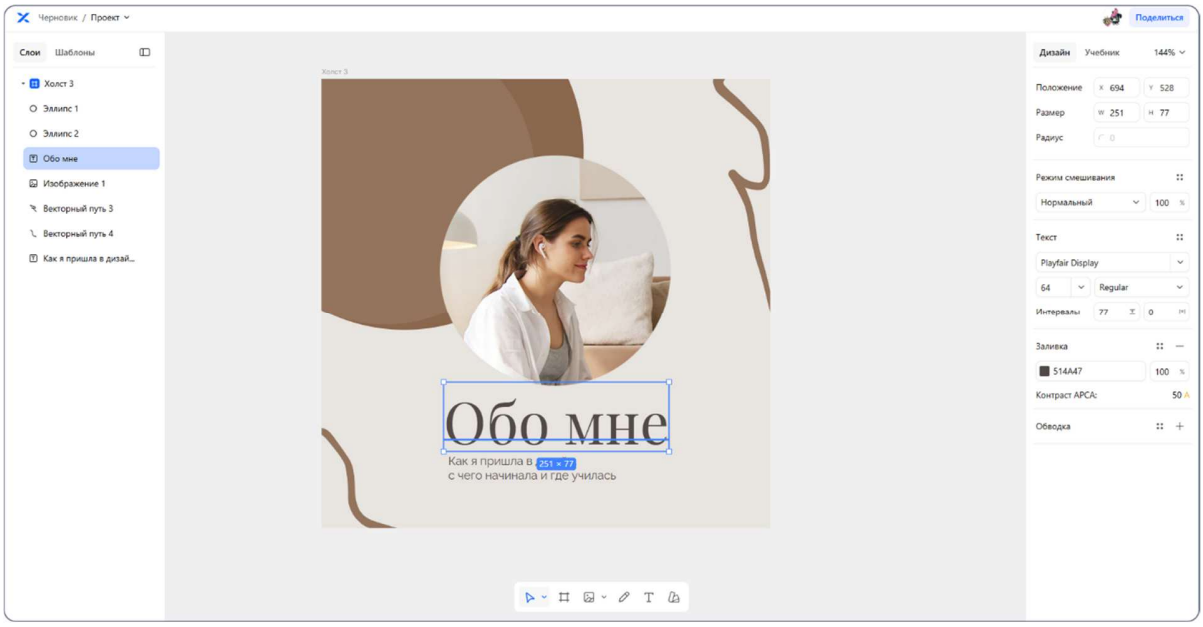

Рисунок 33 – Интерфейс редактора в полном виде с дизайном

Рисунок 34 отображает другие категории. Во вкладке «Шаблоны» отображается набор предустановленных элементов для быстрого добавления на холст. Во вкладке «Учебник» размещены обучающие курсы по дизайну.

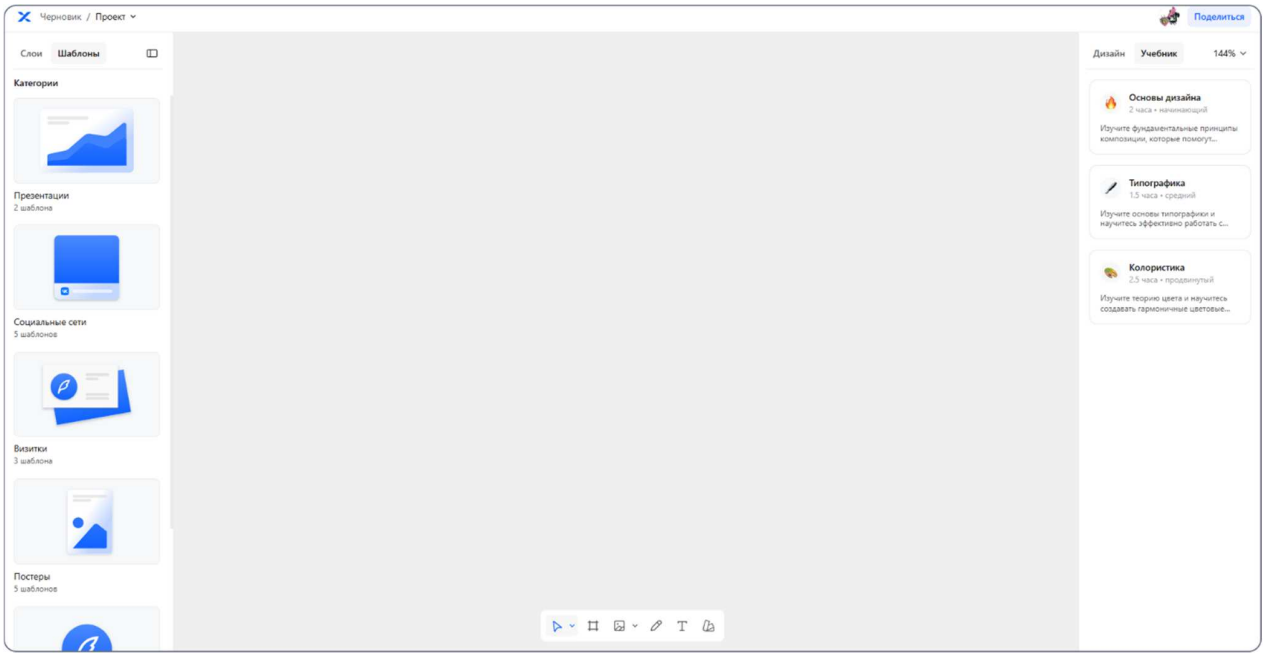

Рисунок 34 – Панель «Шаблоны» и «Учебник»

Рисунок 35 иллюстрирует процесс обучения в разделе «Основы дизайна». Пользователю предлагается изучить такие понятия, как композиция, визуальная иерархия и правило третей. Переход между уроками осуществляется кнопкой «Далее». Структура построена с акцентом на удобство использования, поэтапное усложнение материала и сохранение индивидуального прогресса

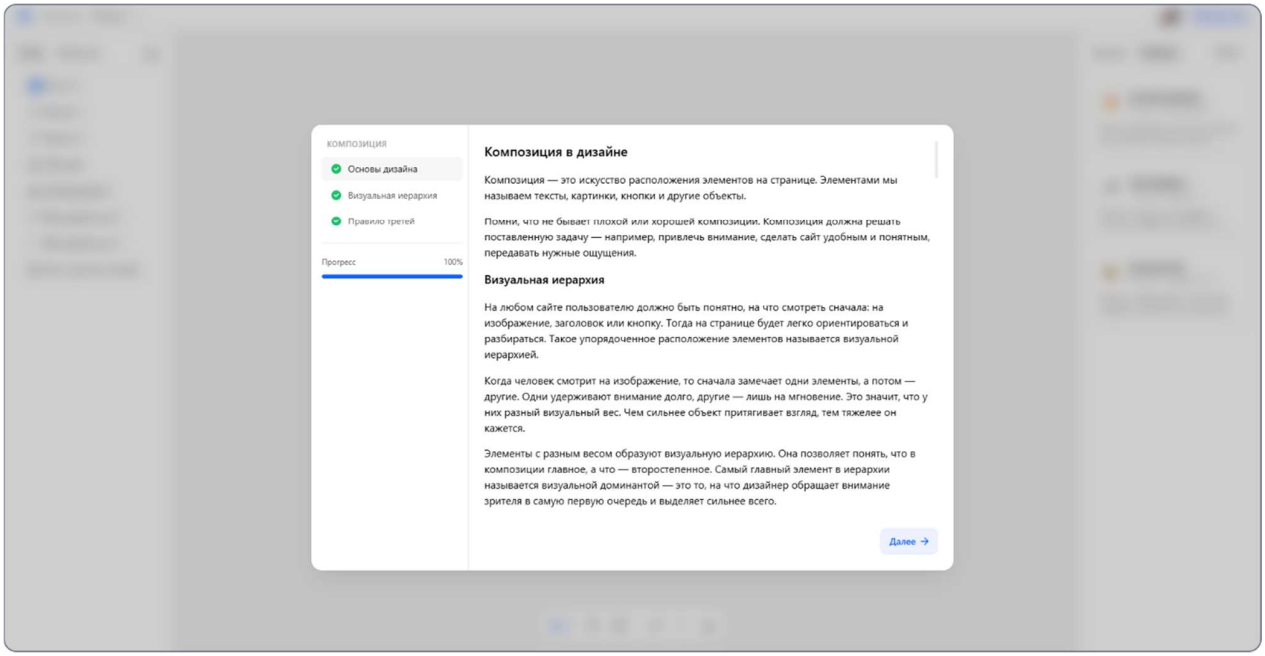

Рисунок 35 – Прохождение курса из раздела «Учебник»

Инструмент «Генератор палитры» (Рисунок 36), который автоматически подбирает цветовые сочетания и отображает их в виде блоков с HEX-кодами.

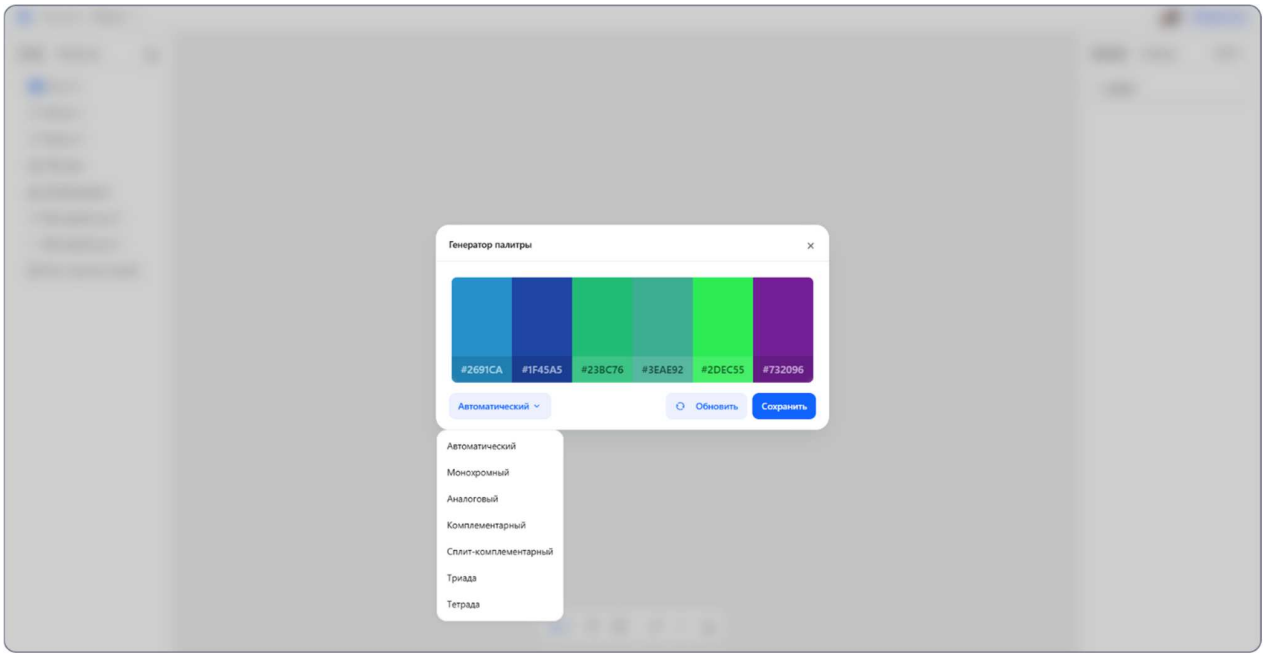

Рисунок 36 – Инструмент «Генератор палитры»

Пользователь может выбрать способ генерации, скопировать, закрепить, удалить или добавить цвет, обновить палитру и сохранить выбранный вариант.

Для удобного управления доступом к файлам разработан модуль настройки прав пользователей. Интерфейс позволяет добавлять участников по электронной почте и копировать ссылку для совместного доступа. В экранной форме отображается список участников с их ролями — владелец и редактор. Внешний вид панели управления доступом (Рисунок 37).

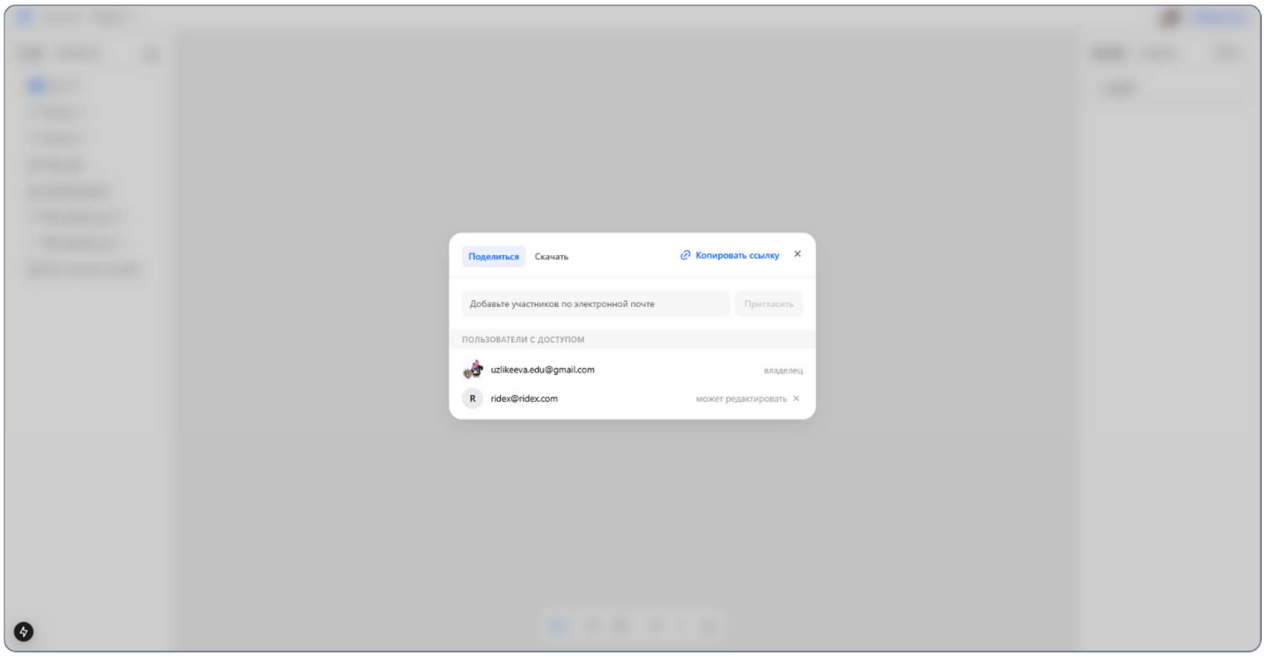

Рисунок 37 – Панель «Поделиться»

3.13. ВЫВОД

В третьей главе была описана практическая реализация веб-приложения для создания визуального дизайна. Рассмотрены этапы инициализации проекта, выбор и настройка подключения к базе данных, организация файловой структуры и настройка базовых компонентов приложения. Для быстрого создания интерфейса интегрирована библиотека AlignUI с настраиваемыми

компонентами. Реализован модуль авторизации и регистрации на NextAuth с безопасной проверкой, хешированием паролей и генерацией JWT.

Были реализованы:

В текущей реализации используются две основные схемы:

- модуль авторизации и регистрации;
- компонент для совместной работы;
- интерактивная доска на основе Canvas;
- модуль управления слоями с иерархической структурой;
- компоненты пользовательского интерфейса;
- библиотека готовых шаблонов с механизмом drag-and-drop;
- модуль обучения с отслеживанием прогресса пользователя;
- инструмент анализа контрастности на основе стандарта APCA.

Разработан интуитивно понятный графический интерфейс, обеспечивающий удобную навигацию по приложению и эффективную работу с дизайн-проектами. Интерфейс включает формы регистрации и авторизации, главную страницу с различными разделами для управления файлами, редактор с компактным и расширенным режимами отображения и генератор цветовых палитр и система управления доступом к файлам.

4. ТЕСТИРОВАНИЕ РАЗРАБОТАННОГО ПРИЛОЖЕНИЯ

Тестирование — важнейший этап в разработке программного обеспечения. Оно позволяет не только выявлять и устранять ошибки, но и проверять соответствие системы заданным требованиям.

4.1. ПРОВЕДЕНИЕ ФУНКЦИОНАЛЬНОГО ТЕСТИРОВАНИЯ

Таблица 3 отображает результаты функционального тестирования модуля регистрации.

Таблица 3 – Функциональное тестирование модуля регистрации

Название теста	**Действия**	**Результат**	**Тест пройден?**
Регистрация нового пользователя	Заполнить форму с корректными данными (имя, email, пароль) и отправить	Пользователь создаётся в базе, пароль сохраняется в хэшированном виде. Возвращается сообщение об успешной регистрации.	Да
Email уже зарегистрирован	Попробовать зарегистрировать пользователя с email, уже существующим в базе	Возвращается ошибка: «Пользователь с таким email уже существует»	Да
Пароль короче 8 символов	Ввести пароль короче 8 символов	Возвращается ошибка: «Пароль не соответствует требованиям»	Да
Пароль длиннее 32 символов	Ввести пароль более 32 символов	Возвращается ошибка: «Пароль не соответствует требованиям»	Да
Некорректный формат email	Ввести email в некорректном формате	Возвращается ошибка: «Некорректный email»	Да
Поле имени не заполнено	Оставить поле имени пустым	Возвращается ошибка: «Требуется указать имя»	Да

Таблица 4 отображает результаты функционального тестирования модуля авторизации.

Таблица 4 – Функциональное тестирование модуля авторизации

Название теста	**Действия**	**Результат**	**Тест пройден?**
Авторизация с корректными данными	Ввести действительные email и пароль зарегистрированного пользователя	Пользователь успешно аутентифицируется, создаётся сессия	Да
Авторизация через Google	Нажать кнопку входа Google, пройти авторизацию у провайдера	Пользователь успешно аутентифицируется через Google, создаётся сессия	Да
Авторизация через GitHub	Нажать кнопку входа GitHub, пройти авторизацию у провайдера	Пользователь успешно аутентифицируется через GitHub, создаётся сессия	Да
Неверный пароль	Ввести существующий email и неверный пароль	Возвращается ошибка: «Неверный пароль»	Да
Несуществующий пользователь	Ввести email, отсутствующий в базе данных	Возвращается ошибка: «Пользователь не найден»	Да

Таблица 5 отображает результаты функционального тестирования контроля доступа.

Таблица 5 – Функциональное тестирование контроля доступа

Название теста	**Действия**	**Результат**	**Тест пройден?**
Доступ к проектам без авторизации	Неавторизованный пользователь открывает страницу проектов	Пользователь перенаправляется на страницу входа	Да
Выход из системы	Отправить запрос на выход из аккаунта	Сессия пользователя удаляется	Да

Таблица 6 отображает результаты функционального тестирования модуля управления проектами.

Таблица 6 – Функциональное тестирование модуля управления проектами

Название теста	**Действия**	**Результат**	**Тест пройден?**
Создание проекта	Авторизованный пользователь создаёт новый проект	В базе данных создаётся запись в таблице Room, связанная с пользователем	Да
Создание проекта без авторизации	Неавторизованный пользователь пытается создать проект	Возвращается ошибка: «Идентификатор пользователя не найден»»	Да
Обновление названия проекта	Владелец проекта изменяет его название	Название успешно обновляется в базе данных	Да
Обновление названия чужого проекта	Пользователь пытается изменить название проекта, которым не владеет	Возвращается ошибка доступа	Да
Удаление проекта владельцем	Владелец проекта удаляет его	Запись о проекте и связанные приглашения удаляются из таблицы Room и RoomInvite	Да
Сохранение изменений в проекте	Внести изменения, обновить страницу	После обновления страницы все внесенные изменения сохраняются и корректно отображаются	Да

Таблица 7 отображает результаты функционального тестирования управления приглашениями.

Таблица 7 – Функциональное тестирование управления приглашениями

Название теста	**Действия**	**Результат**	**Тест пройден?**
Приглашение существующего пользователя в проект	Владелец вводит email зарегистрированного пользователя	В таблице RoomInvite создаётся запись, связывающая пользователя с проектом	Да
Приглашение несуществующего пользователя	Владелец вводит несуществующий email	Система выдаёт ошибку: «Пользователь не найден»	Да
Удаление приглашения в проект	Владелец удаляет приглашение	Запись в таблице RoomInvite удаляется	Да
Попытка удаления чужого приглашения не владельцем проекта	Пользователь, не являющийся владельцем, пытается удалить чужое приглашение	Система возвращает ошибку о недостатке прав	Да
Отказ приглашённого пользователя от доступа к проекту	Приглашённый пользователь удаляет своё приглашение	Запись в таблице RoomInvite для пользователя удаляется	Да

Таблица 8 отображает результаты функционального тестирования модуля совместного редактирования.

Таблица 8 – Функциональное тестирование модуля совместного редактирования

Название теста	**Действия**	**Результат**	**Тест пройден?**
Проверка совместного редактирования	Открыть проект на двух устройствах, внести изменения на одном из них	Изменения синхронизируются и отображаются на втором устройстве	Да
Проверка доступа к чужому проекту	Пользователь открывает чужой проект	Возвращается ошибка доступа	Да

Продолжение таблицы 8

Проверка отображения курсоров других пользователей	Подключиться к проекту с нескольких устройств, перемещать курсор	Курсоры с именами видны на других устройствах	Да
Проверка одновременного доступа нескольких пользователей	Пригласить нескольких пользователей к проекту и авторизоваться под каждым из них	Все приглашенные пользователи имеют доступ к проекту и видят его в своем списке	Да

Таблица 9 отображает результаты функционального тестирования модуля инструментов.

Таблица 9 – Функциональное тестирование модуля инструментов

Название теста	**Действия**	**Результат**	**Тест пройден?**
Переключение между инструментами	Выбрать разные инструменты на панели	Каждый инструмент активируется, курсор и интерфейс меняют вид	Да
Создание рамки	Выбрать «Рамка», щёлкнуть на холст	Создаётся рамка размером 200×200 px	Да
Создание рамки произвольного размера	Выбрать «Рамка», зажать левую кнопку мыши и тянуть по холсту	Создаётся рамка с размерами, по движению мыши	Да
Создание прямоугольника	Выбрать «Прямоугольник», щелкнуть на холст	Создаётся прямоугольник размером 100×100 px	Да
Создание прямоугольника произвольного размера	Выбрать «Прямоугольник», зажать левую кнопку мыши и тянуть по холсту	Создаётся прямоугольник с размерами по движению мыши	Да
Создание эллипса	Выбрать «Эллипс», щелкнуть на холст	Создаётся эллипс размером 100×100 px	Да

Продолжение таблицы 9

Создание эллипса произвольного размера	Выбрать «Эллипс», зажать левую кнопку мыши и тянуть по холсту	Создаётся эллипс с размерами, по движению мыши	Да
Вставка изображения	Выбрать «Изображение», выбрать файл на компьютере	Изображение появляется на холсте	Да
Создание текста	Выбрать «Текст», щёлкнуть на холсте, ввести текст	Появляется текстовый элемент с введённым текстом	Да
Рисование	Выбрать «Карандаш», зажать левую кнопку мыши и тянуть по холсту	Появляется линия, соответствующая траектории движения мыши	Да

Таблица 10 отображает результаты функционального тестирования управления объектами.

Таблица 10 – Функциональное тестирование управления объектами

Название теста	**Действия**	**Результат**	**Тест пройден?**
Проверка выбора только одного объекта	Выбрать объект, затем другой без зажатого Shift	Выделяется только последний объект, предыдущий снимается	Да
Проверка выделения области	Выбрать инструмент выделения, зажать левую кнопку и тянуть по холсту, создавая рамку выделения	Все объекты внутри рамки выделяются	Да
Проверка изменения размера фигуры	Выбрать фигуру, тянуть за точки изменения размера	Фигура масштабируется по движению указателя	Да

Продолжение таблицы 10

Проверка пропорционального изменения с Shift	Выбрать фигуру, зажать Shift, тянуть за угловую точку	Размер меняется пропорционально	Да
Проверка перемещения объекта	Выбрать фигуру, перетащить в новое место	Фигура перемещается на новое место	Да
Проверка перемещения объекта в рамку	Выбрать фигуру, перетащить в рамку	Фигура помещается внутрь рамки и становится её частью	Да
Проверка удаления объекта	Выбрать объект, нажать Delete	Объект удаляется с холста	Да

Таблица 11 отображает результаты функционального тестирования панели свойств объектов.

Таблица 11 – Функциональное тестирование панели свойств объектов

Название теста	**Действия**	**Результат**	**Тест пройден?**
Изменение радиуса скругления у прямоугольника	Выбрать прямоугольник, изменить радиус скругления в панели свойств	Углы прямоугольника скругляются согласно заданному значению	Да
Изменение радиуса скругления у эллипса	Выбрать эллипс, изменить радиус скругления в панели свойств	Радиус скругления не изменяется	Да
Изменение режима смешивания	Выбрать объект, изменить режим смешивания в панели свойств	Применяется выбранный режим, внешний вид объекта изменяется	Да
Изменение непрозрачности	Выбрать объект, изменить непрозрачность в панели свойств	Прозрачность объекта изменяется согласно значению	Да

Продолжение таблицы 11

Изменение цвета заливки и/или обводки	Выбрать объект, изменить цвет заливки и/или обводки в панели свойств	Цвет заливки и/или обводки меняется	Да
Изменение толщины обводки	Выбрать объект, изменить толщину обводки в панели свойств	Толщина обводки меняется	Да
Удаление цвета и/или обводки	Выбрать объект, удалить заливку и/или обводку	У объекта отсутствуют цвет заливки и/или обводки	Да
Добавление цвета и/или обводки	Выбрать объект, задать заливку и/или обводку	У объекта появляется заливка и/или обводка	Да
Изменение непрозрачности заливки и/или обводки	Выбрать объект, изменить прозрачность заливки и/или обводки	Уровень прозрачности цвета меняется	Да
Изменение шрифта	Выбрать текст, выбрать другой шрифт в панели свойств	Шрифт текста изменяется	Да
Изменение размера шрифта	Выбрать текст, изменить размер в панели свойств	Размер текста изменяется	Да
Изменение начертания шрифта	Выбрать текст, изменить начертание (например, полужирный)	Начертание изменяется	Да
Изменение межстрочного интервала	Выбрать текст, задать межстрочный интервал в панели свойств	Интервал изменяется согласно значению	Да

Продолжение таблицы 11

Изменение межбуквенного интервала	Выбрать текст, задать межбуквенный интервал в панели свойств	Интервал изменяется согласно значению	Да
Изменение цвета фона холста	В панели свойств изменить цвет холста	Цвет холста меняется	Да

Таблица 12 отображает результаты функционального тестирования дополнительных функций.

Таблица 12 – Функциональное тестирование дополнительных функций

Название теста	**Действия**	**Результат**	**Тест пройден?**
Генератор палитр	Открыть генератор, сгенерировать новую палитру	Новая палитра отображается и доступна для выбора цветов	Да
Выбор шаблона	Открыть галерею, выбрать шаблон и применить его	Шаблон добавляется на холст	Да
Прохождение урока	Открыть раздел «Уроки», выбрать урок и выполнить	Все шаги выполнены, отображается сообщение о завершении	Да
Проверка контраста текста и фона	Разместить текст на фоне	Отображается значение контраста и информация о соответствии стандарту	Да

Таблица 13 отображает результаты функционального тестирования управления холстом.

Таблица 13 – Функциональное тестирование управления холстом

Название теста	**Действия**	**Результат**	**Тест пройден?**
Масштабирование холста	Использовать кнопки масштабирования или колесо мыши	Холст увеличивается или уменьшается, сохраняя центр просмотра	Да

Продолжение таблицы 13

Перемещение холста	Зажать среднюю кнопку мыши и перетащить курсор	Холст перемещается в направлении движения курсора	Да
Отмена действия	Создать фигуру, нажать Ctrl + Z	Фигура удаляется с холста	Да
Повтор действия	После отмены действия нажать Ctrl + Shift + Z	Фигура возвращается на холст	Да
Выбор всех элементов	Нажать Ctrl + A	Все объекты на холсте выделяются	Да

4.2. ПРОВЕДЕНИЕ НЕФУНКЦИОНАЛЬНОГО ТЕСТИРОВАНИЯ

Нефункциональное тестирование направлено на оценку характеристик системы, не связанных напрямую с её функциональностью. Цель такого тестирования — убедиться в соответствии приложения требованиям надёжности, безопасности, удобству использования и совместимости.

Для проверки требования о доступности приложения через современный браузер без дополнительного программного обеспечения было проведено мануальное тестирование. Приложение протестировано в Google Chrome версии 124. Результаты показали корректное отображение интерфейса, стабильную работу всех функций и целостность вёрстки. Дополнительно проверена совместимость с другими популярными браузерами (Mozilla Firefox, Microsoft Edge, Safari). Во всех случаях подтверждена полная работоспособность приложения.

Для оценки удобства пользовательского интерфейса проведено юзабилити-тестирование с привлечением независимых пользователей. Тестирование проходило в формате выполнения типовых сценариев использования приложения без предварительного обучения. Результаты

показали, что интерфейс интуитивно понятен и удобен. Пользователи успешно выполнили все предложенные сценарии без существенных затруднений, что подтверждает соответствие приложения требованиям к удобству использования.

Проверка требования о защите пользовательских данных проводилась с использованием инструментов разработчика браузера (DevTools) для анализа передачи данных между клиентом и сервером.

Все взаимодействия (Рисунок 38) происходят по защищённому протоколу HTTPS, что обеспечивает шифрование передаваемой информации. Аутентификационные данные передаются методом POST, исключающим их отображение в URL и истории браузера. Запросы авторизации направляются на защищённый эндпоинт /signin. В ответах сервера присутствует заголовок Cache-Control: no-store, must-revalidate, предотвращающий кэширование конфиденциальной информации.

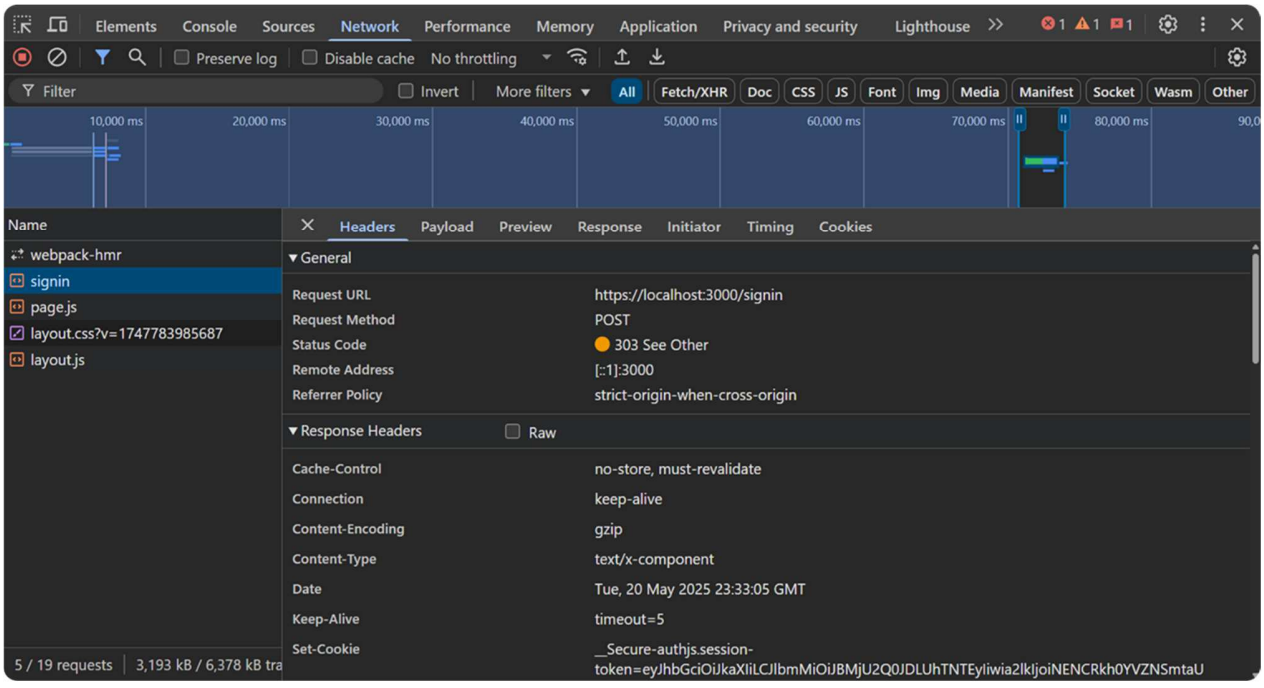

Рисунок 38 – Панель Network с детализацией запроса авторизации на сайте

После успешной аутентификации (Рисунок 39) сервер устанавливает сессионный токен в виде защищённой куки _Secure-authjs.session-token, снабжённой маркерами безопасности: Secure, гарантирующим передачу только через HTTPS, и HttpOnly, блокирующим доступ к токену со стороны JavaScript и обеспечивающим защиту от XSS-атак.

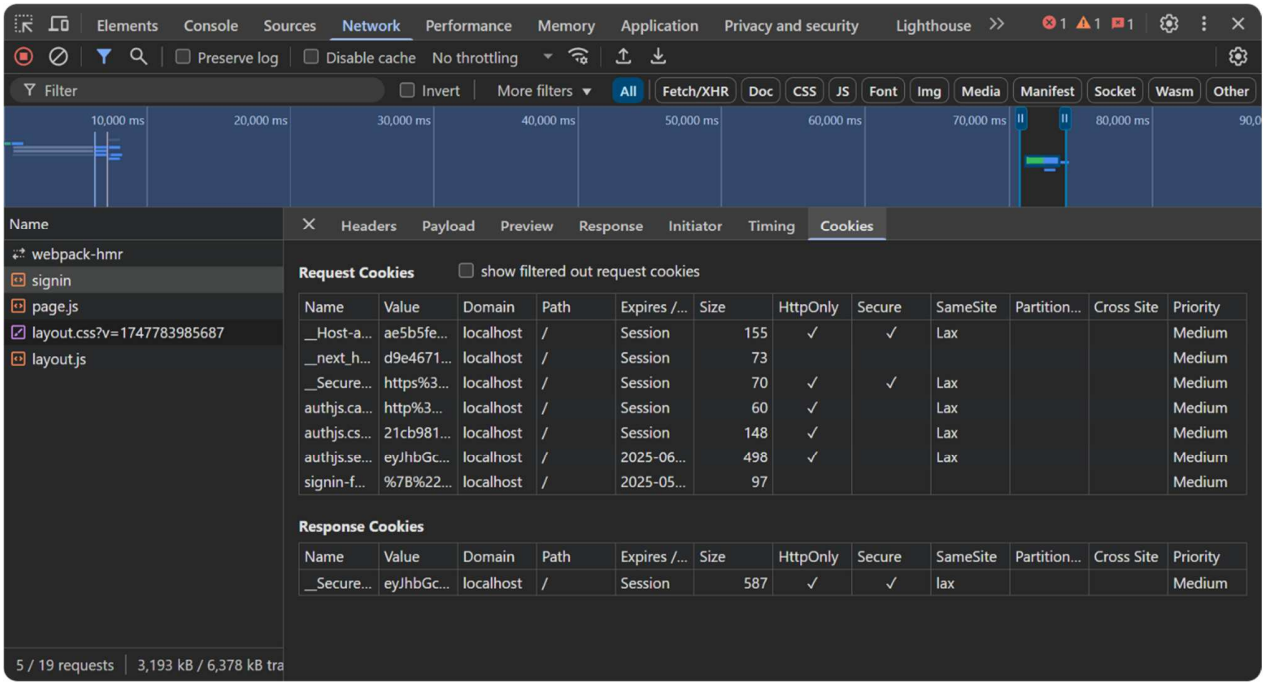

Рисунок 39 – Cookies, связанные с процессом авторизации пользователя

Проведённый анализ подтверждает надлежащий уровень защиты данных в соответствии с современными стандартами безопасности веб-приложений.

Таблица 14 отображает результаты нефункционального тестирования веб-приложения.

Таблица 14 – Нефункциональное тестирование веб-приложения

Название теста	**Результат**	**Тест пройден?**
Совместимость с браузерами	Приложение корректно работает во всех современных браузерах без необходимости установки дополнительного ПО	Да

Продолжение таблицы 14

Юзабилити-тестирование	Интерфейс признан интуитивно понятным и удобным. Пользователи успешно выполнили все сценарии без затруднений	Да
Безопасность данных	Реализованы все необходимые меры защиты пользовательских данных: HTTPS, защищённые куки, предотвращение кэширования чувствительной информации	Да

4.3. ВЫВОД

В этой главе подробно рассмотрен процесс тестирования приложения, охватывающий как проверку функциональных, так и нефункциональных характеристик. Функциональное тестирование подтвердило стабильную работу всех ключевых возможностей веб-приложения. Нефункциональное тестирование позволило оценить надёжность, безопасность и удобство использования. Полученные результаты показали полное соответствие всем установленным требованиям к совместимости, удобству использования и безопасности. Комплексное тестирование существенно повысило надёжность системы и подтвердило её готовность к практическому использованию.

6. ЗАКЛЮЧЕНИЕ

Аналитический обзор предметной области и методической литературы позволил выявить ключевые теоретические и практические аспекты, влияющие на качество дизайна. Особое внимание уделено трудностям, с которыми сталкиваются новички. Сравнительный анализ существующих решений показал, что большинство аналогов либо имеют высокий порог входа, либо недоступны на территории России, многие лишены встроенных инструментов для работы с цветовыми палитрами.

В рамках проектирования сформулированы функциональные и нефункциональные требования, разработана архитектура приложения, обеспечивающая масштабируемость и гибкость. Создана структура базы данных на PostgreSQL и подготовлены UML-диаграммы. Разработанные вайрфреймы интерфейсов позволили заранее проработать логику взаимодействия с пользователем.

В процессе реализации разработан полнофункциональный графический редактор с поддержкой совместной работы. Внедрены системы аутентификации, управления проектами и приглашения соавторов. Интерфейс адаптирован для пользователей разного уровня подготовки благодаря интуитивной навигации и быстрому доступу к инструментам. Основная проблема заключалась в обеспечении плавной и отзывчивой работы редактора при одновременной работе нескольких пользователей и манипуляции сложными графическими элементами. Эта проблема была решена за счет оптимизации алгоритмов рендеринга, внедрения системы кэширования состояний и использования эффективных структур данных для хранения и синхронизации изменений между пользователями.

Комплексное тестирование подтвердило соответствие приложения всем требованиям. Пользовательский интерфейс получил положительные отзывы на предварительных демонстрациях.

Разработанное приложение включает более 150 компонентов и свыше 30 000 строк кода. Приложение имеет высокую практическую ценность для образовательной среды, малых дизайнерских команд и начинающих специалистов. Гибкая архитектура позволяет масштабировать функциональность под различные задачи.

В перспективе возможно расширение проекта за счёт инструментов для работы с векторной графикой, интеграции AI-технологий генерации, системы плагинов и адаптация для мобильных устройств.

Таким образом, поставленные цель и задачи были полностью выполнены. Разработанное приложение представляет собой современное, конкурентоспособное и масштабируемое решение в области визуального дизайна. Оно может успешно применяться на практике и развиваться в соответствии с потребностями пользователей и рынка. Проект был представлен на конкурсе «Альфа-Шанс» от компании «Альфа-Банк».

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Adobe Color: Инструмент для работы с цветами [Электронный ресурс]. – URL: https://color.adobe.com. – Дата обращения: 10.03.2025.
2. AlignUI: Библиотека UI-компонентов [Электронный ресурс]. – URL: https://www.alignui.com. – Дата обращения: 14.04.2025.
3. Balsamiq Wireframe: Инструмент для прототипирования интерфейсов [Электронный ресурс]. – URL: https://balsamiq.com. – Дата обращения: 10.04.2025.
4. Bcrypt: Библиотека хэширования паролей [Электронный ресурс]. – URL: https://github.com/kelektiv/node.bcrypt.js. – Дата обращения: 14.04.2025.
5. Canva: Платформа для графического дизайна [Электронный ресурс]. – URL: https://www.canva.com. – Дата обращения: 10.03.2025.
6. Cookie: Маленький фрагмент данных, хранимый в браузере для сохранения информации о пользователе [Электронный ресурс]. – URL: https://developer.mozilla.org/en-US/docs/Web/HTTP/Cookies. – Дата обращения: 10.04.2025.
7. Coolors: Генератор цветовых схем [Электронный ресурс]. – URL: https://coolors.co. – Дата обращения: 10.03.2025.
8. CSS: Каскадные таблицы стилей, используемые для оформления веб-страниц [Электронный ресурс]. – URL: https://developer.mozilla.org/en-US/docs/Web/CSS. – Дата обращения: 25.03.2025.
9. Discord: Платформа для голосового и текстового общения [Электронный ресурс]. – URL: https://discord.com. – Дата обращения: 14.04.2025.
10. EDraw Network Diagrammer: Программа для создания сетевых диаграмм [Электронный ресурс]. – URL: https://www.edrawsoft.com/network-diagram.html. – Дата обращения: 10.04.2025.

11. ESLint: Инструмент анализа и исправления проблем в JavaScript/TypeScript коде [Электронный ресурс]. – URL: https://eslint.org. – Дата обращения: 14.04.2025.
12. Fetch API: Работа с сетевыми запросами [Электронный ресурс]. – URL: https://learn.javascript.ru/fetch-api. – Дата обращения: 10.04.2025.
13. HTML: Язык разметки гипертекста для создания веб-страниц [Электронный ресурс]. – URL: https://developer.mozilla.org/en-US/docs/Web/HTML. – Дата обращения: 25.03.2025.
14. JWT: JSON Web Tokens – стандарт для безопасной передачи информации [Электронный ресурс]. – URL: https://jwt.io. – Дата обращения: 14.04.2025.
15. Liveblocks: Инструмент для создания в реальном времени совместных приложений [Электронный ресурс]. – URL: https://liveblocks.io. – Дата обращения: 10.04.2025.
16. Lunacy: Бесплатный графический редактор от Icons8 [Электронный ресурс]. – URL: https://icons8.com/lunacy. – Дата обращения: 10.03.2025.
17. Next.js API Routes: Функционал Next.js для создания API внутри приложения [Электронный ресурс]. – URL: https://nextjs.org/docs/api-routes/introduction. – Дата обращения: 10.04.2025.
18. Next.js: React-фреймворк для создания веб-приложений с серверным рендерингом и генерацией статических сайтов [Электронный ресурс]. – URL: https://nextjs.org. – Дата обращения: 25.03.2025.
19. NextAuth.js: Библиотека для аутентификации в приложениях на базе Next.js [Электронный ресурс]. – URL: https://next-auth.js.org. – Дата обращения: 10.04.2025.
20. Node.js: Среда выполнения JavaScript, основанная на движке V8, для серверного программирования [Электронный ресурс]. – URL: https://nodejs.org. – Дата обращения: 25.03.2025.

21. Paletton: Инструмент для подбора цветовых палитр [Электронный ресурс]. – URL: https://paletton.com. – Дата обращения: 10.03.2025.
22. Pixso: Онлайн-платформа для дизайна и прототипирования [Электронный ресурс]. – URL: https://pixso.net. – Дата обращения: 10.03.2025.
23. PostgreSQL: Объектно-реляционная система управления базами данных [Электронный ресурс]. – URL: https://www.postgresql.org. – Дата обращения: 10.04.2025.
24. Prettier: Инструмент автоформатирования кода [Электронный ресурс]. – URL: https://prettier.io. – Дата обращения: 14.04.2025.
25. Prisma: ORM (Object-Relational Mapping) для работы с базами данных в TypeScript и JavaScript [Электронный ресурс]. – URL: https://www.prisma.io. – Дата обращения: 10.04.2025.
26. Radix UI: Набор примитивов интерфейса для React [Электронный ресурс]. – URL: https://www.radix-ui.com. – Дата обращения: 14.04.2025.
27. React UI: Пользовательские интерфейсы на базе React [Электронный ресурс]. – URL: https://react.dev. – Дата обращения: 10.04.2025.
28. Sketch: Векторный графический редактор для macOS [Электронный ресурс]. – URL: https://www.sketch.com. – Дата обращения: 10.03.2025.
29. Supabase: Open Source альтернатива Firebase [Электронный ресурс]. – URL: https://supabase.com. – Дата обращения: 14.04.2025.
30. T3 Stack: Набор технологий для создания современных веб-приложений [Электронный ресурс]. – URL: https://create.t3.gg/en/introduction. – Дата обращения: 25.03.2025.
31. TailwindCSS: Утилитарный CSS-фреймворк для быстрой разработки интерфейсов [Электронный ресурс]. – URL: https://tailwindcss.com. – Дата обращения: 25.03.2025.

32. The business value of design. McKinsey Design // McKinsey & Company [Электронный ресурс]. – URL: https://mckinsey.com/valueofdesign. – Дата обращения: 19.02.2025.
33. TypeScript: Язык программирования, расширяющий JavaScript за счёт статической типизации [Электронный ресурс]. – URL: https://www.typescriptlang.org. – Дата обращения: 25.03.2025.
34. Zod: TypeScript-first схема валидации [Электронный ресурс]. – URL: https://zod.dev. – Дата обращения: 14.04.2025.
35. Арнхейм, Р. Искусство и визуальное восприятие / Р. Арнхейм. – М.: Прогресс, 1974. – 392 с.
36. Волков, А.С. Обзор архитектурных компонентов современного веб-приложения / А.С. Волков, К.А. Волкова // Аллея науки. – 2019. – Т. 3. – № 1. – С. 958–961.
37. Графический редактор Figma [Электронный ресурс]. – URL: https://www.figma.com. – Дата обращения: 19.02.2025.
38. Гущина, О.М. Компьютерная графика и мультимедиатехнологии: учебно-методическое пособие / О.М. Гущина, Н.Н. Казаченок. – Тольятти: ТГУ, 2018. – 364 с. – ISBN 978-5-8259-1185-4. – Текст: электронный // Лань: электронно-библиотечная система. – URL: https://e.lanbook.com/book/139890. – Режим доступа: для авториз. пользователей. – Дата обращения: 19.02.2025.
39. Епифанова, А.Г. Особенности преподавания дисциплины «Цветоведение и колористика» будущим графическим дизайнерам: учебное пособие / А.Г. Епифанова, Е.Э. Савочкина. – Челябинск: ОУ ВО «Южно-Уральский технологический университет», 2021. – 158 с. – URL: https://www.inueco.ru/rio/2021/978-5-6046573-4-8.pdf. – Дата обращения: 19.02.2025.

40. Исаев, А.А. Философия цвета: феномен цвета в мышлении и творчестве: монография / А.А. Исаев, Д.А. Теплых. – 2-е изд. – Магнитогорск: МаГУ, 2011. – 180 с.
41. Иттен, И. Искусство формы. Мой форкурс в Баухаузе и других школах / И. Иттен. – 5-е изд. – М.: Д. Аронов, 2013. – 136 с.
42. Иттен, И. Искусство цвета / И. Иттен; пер. с нем. Л. Монаховой. – М.: Д. Аронов, 2000. – 96 с.
43. Королькова, А. Живая типографика / А. Королькова. – М.: IndexMarket, 2012. – 224 с.: ил.
44. Круг, С. Веб-дизайн: книга Стива Круга или «не заставляйте меня думать!» / С. Круг; пер. с англ. – СПб.: Символ-Плюс, 2005. – 200 с.
45. Кухта, М.С. Дизайн и технологии / М.С. Кухта. – Томск: STT, 2016. – 170 с.
46. Лебедев, А. Ководство: Руководство по дизайну [Электронный ресурс] / А. Лебедев. – URL: http://www.artlebedev.ru/kovodstvo. – Дата обращения: 19.02.2025.
47. Месяц, С.В. Иоганн Вольфганг Гёте и его учение о цвете (Часть первая) / С.В. Месяц. – М.: Кругъ, 2012. – 464 с.
48. Моллика, П. Теория цвета: Настольный путеводитель: от базовых принципов до практических решений / П. Моллика; пер. с англ. Д.А. Прокофьева. – М.: КоЛибри, Азбука-Аттикус, 2021. – 128 с. – ISBN 978-5-389-20528-4.
49. Мы изучили рынок труда дизайнеров: спрос и зарплаты растут. DNA Team // vc.ru [Электронный ресурс]. – URL: https://vc.ru/design/1553462-my-izuchili-rynok-truda-dizainerov-spros-i-zarplaty-rastut. – Дата обращения: 19.02.2025.

50. Руководство по обеспечению доступности веб-контента (WCAG) 2.0 [Электронный ресурс]. – URL: https://www.w3.org/TR/WCAG20. – Дата обращения: 19.02.2025.
51. Уильямс, Р. Дизайн. Книга для недизайнеров / Р. Уильямс. – 4-е изд. – СПб.: Питер, 2016. – 240 с.
52. Феличи, Дж. Типографика: шрифт, верстка, дизайн / Дж. Феличи; пер. с англ., коммент. С.И. Пономаренко. – СПб.: БХВ-Петербург, 2004. – 496 с.
53. Цой, В.В. Гештальт-принципы в развитии креативности будущих дизайнеров / В.В. Цой, О.П. Тарасова // Вестник Оренбургского государственного университета. – 2022. – № 234. – С. 103–109.
54. Шокова, Е.В. Информационный дизайн: учебное пособие / Е.В. Шокова, И.В. Ахматова. – Самара: Самарский университет, 2023. – 84 с. – ISBN 978-5-7883-1911-7. – Текст: электронный // Лань: электронно-библиотечная система. – URL: https://e.lanbook.com/book/406739. – Режим доступа: для авториз. пользователей. – Дата обращения: 19.02.2025.
55. Элам, К. Геометрия дизайна. Пропорции и композиция / К. Элам. – СПб.: Питер, 2011. – 112 с.: ил.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД СХЕМЫ БАЗЫ ДАННЫХ

Листинг А.1 — Исходный код схемы базы данных schema.prisma

```
generator client {
  provider = "prisma-client-js"
  output   = "../node_modules/.prisma/client"
}

// Настройки подключения к базе данных PostgreSQL
datasource db {
  provider  = "postgresql"
  url       = env("DATABASE_URL")
  directUrl = env("DIRECT_URL")
}

// Модель аккаунта пользователя для OAuth аутентификации
model Account {
  id                       String  @id @default(cuid())
  userId                   String
  type                     String
  provider                 String
  providerAccountId        String
  refresh_token            String?
  access_token             String?
  expires_at               Int?
  token_type               String?
  scope                    String?
  id_token                 String?
  session_state            String?
  refresh_token_expires_in Int?
  user                       User    @relation(fields: [userId], references: [id], onDelete:
Cascade)

  @@unique([provider, providerAccountId])
}

// Модель сессии пользователя
model Session {
  id           String   @id @default(cuid())
  sessionToken String   @unique
  userId       String
  expires      DateTime
  user         User     @relation(fields: [userId], references: [id], onDelete: Cascade)
}

// Основная модель пользователя
model User {
  id            String       @id @default(cuid())
  name          String?
  email         String       @unique
  emailVerified DateTime?
  image         String?
  password      String?
```

Окончание листинга А.1

```
  accounts      Account[]
  ownedRooms    Room[]       @relation("RoomOwner")
  roomInvites   RoomInvite[]
  sessions      Session[]
}

// Модель токена верификации для подтверждения email
model VerificationToken {
  identifier String
  token      String   @unique
  expires    DateTime

  @@unique([identifier, token])
}

// Модель комнаты (проекта)
model Room {
  id          String       @id @default(cuid())
  ownerId     String
  createdAt   DateTime     @default(now())
  title       String       @default("Проект")
  owner       User         @relation("RoomOwner", fields: [ownerId], references: [id])
  roomInvites RoomInvite[]
}

// Модель приглашения в комнату
model RoomInvite {
  id     String @id @default(cuid())
  roomId String
  userId String
  room   Room   @relation(fields: [roomId], references: [id])
  user   User   @relation(fields: [userId], references: [id])
}
```

ПРИЛОЖЕНИЕ Б
ИСХОДНЫЙ КОД МОДУЛЯ РЕГИСТРАЦИИ И АВТОРИЗАЦИИ

Листинг Б.1 – Исходный код конфигурации config.ts

```
/**
 * Модуль конфигурации NextAuth для системы аутентификации
 * Определяет настройки провайдеров, сессий и адаптера базы данных
 */

import { PrismaAdapter } from "@auth/prisma-adapter";
import { type DefaultSession, type NextAuthConfig } from "next-auth";
import Credentials from "next-auth/providers/credentials";
import GitHub from "next-auth/providers/github";
import Google from "next-auth/providers/google";
import bcrypt from "bcryptjs";
import { db } from "~/server/db";
import { signInSchema } from "~/lib/validations/auth";
import { env } from "~/env";

/**
 * Расширение типа сессии NextAuth
*/
declare module "next-auth" {
  interface Session extends DefaultSession {
    user: {
      id: string;
    } & DefaultSession["user"];
  }
}

/**
 * Основная конфигурация NextAuth
*/
export const authConfig = {
  // Настройка провайдеров аутентификации
  providers: [
    // Аутентификация через GitHub
    GitHub({
      clientId: env.GITHUB_ID,
      clientSecret: env.GITHUB_SECRET,
    }),

    // Аутентификация через Google
    Google({
      clientId: env.GOOGLE_ID,
      clientSecret: env.GOOGLE_SECRET,
    }),

    // Аутентификация через учетные данные (email/пароль)
    Credentials({
      credentials: {
        email: {}, // Email пользователя
        password: {}, // Пароль пользователя
      },
```

Окончание листинга Б.1

```
      // Функция проверки учетных данных
      authorize: async (credentials) => {
        try {
          // Валидация учетных данных через схему Zod
          const { email, password } = await signInSchema.parseAsync(credentials);

          // Поиск пользователя в базе данных
          const user = await db.user.findUnique({
            where: {
              email: email,
            },
          });

          // Проверка соответствия пароля
          const passwordMatch = await bcrypt.compare(
            password,
            user?.password ?? "",
          );

          // Если пароль не совпадает, возвращаем null
          if (!passwordMatch) {
            return null;
          }

          // Возвращаем данные пользователя при успешной аутентификации
          return user;
        } catch (error) {
          return null;
        }
      },
    }),
  ],

  // Настройка типа сессии
  session: {
    strategy: "jwt", // Используем JWT для сессий
  },

  // Настройка адаптера для хранения данных в базе через Prisma
  adapter: PrismaAdapter(db),

  // Функции обратного вызова для различных событий
  callbacks: {
    // Модификация данных сессии
    session: ({ session, token }) => ({
      ...session,
      user: {
        ...session.user,
        id: token.sub, // Добавляем ID пользователя из токена в объект сессии
      },
    }),
  },
} satisfies NextAuthConfig;
```

Листинг Б.2 – Исходный код схемы валидации auth.ts

```
/**
 * Модуль с валидационными схемами для форм аутентификации
 * Использует библиотеку Zod для типизированной валидации данных
 */

import { object, string } from "zod";

/**
 * Схема валидации для регистрации нового пользователя
 * Проверяет корректность email, сложность пароля и наличие имени
 */
export const signUpSchema = object({
  email: string({ required_error: "Требуется email" }).email("Некорректный email"),
  password: string({ required_error: "Требуется пароль" })
    .min(8, "Пароль должен содержать не менее 8 символов")
    .max(32, "Пароль должен содержать не более 32 символов"),
  name: string({ required_error: "Требуется указать имя" }),
});

/**
 * Схема валидации для входа пользователя
 * Проверяет корректность email и наличие пароля
 */
export const signInSchema = object({
  email: string({ required_error: "Требуется email" }).email("Неверный email"),
  password: string({ required_error: "Требуется пароль" }),
});
```

Листинг Б.3 – Исходный код функции аутентификации и регистрации auth.ts

```
"use server";

/**
 * Модуль содержит функции для аутентификации и регистрации пользователей
 * Включает методы для входа, выхода и создания новой учетной записи
 */

import { ZodError } from "zod";
import { db } from "~/server/db";
import bcrypt from "bcryptjs";
import { signIn, signOut } from "~/server/auth";
import { AuthError } from "next-auth";
import { signUpSchema } from "~/lib/validations";

/**
 * Функция для выхода пользователя из системы
 * Вызывает стандартный метод signOut из NextAuth
 */
export async function signout() {
  await signOut();
}
```

Продолжение листинга Б.3

```
/**
 * Функция для аутентификации пользователя
 * @param {FormData} formData - Данные формы с email и паролем
 * @returns {Object|undefined} - Объект с ошибкой или undefined при успешной аутентификации
 */
export async function authenticate(formData: FormData) {
  try {
    const email = formData.get("email") as string;
    const password = formData.get("password") as string;

    // Проверка существования пользователя в базе данных
    const user = await db.user.findUnique({ where: { email } });
    if (!user?.password) {
      return { email: "Пользователь не найден" };
    }

    // Проверка правильности пароля
    const isPasswordValid = await bcrypt.compare(password, user.password);
    if (!isPasswordValid) {
      return { password: "Неверный пароль" };
    }

    // Выполнение входа в систему
    await signIn("credentials", formData);
  } catch (error) {
    if (error instanceof AuthError) {
      const errorMessage =
        error.type === "CredentialsSignin"
          ? "Недействительные учетные данные"
          : "Что-то пошло не так";
      return { error: errorMessage };
    }
    throw error; // Пробрасываем неизвестные ошибки дальше
  }
}
/**
 * Функция для регистрации нового пользователя
 * @param {FormData} formData - Данные формы с именем, email и паролем
 * @returns {Object} - Объект с результатом регистрации или ошибками
 */
export async function register(formData: FormData) {
  try {
    // Валидация данных формы с помощью Zod
    const { name, email, password } = await signUpSchema.parseAsync({
      name: formData.get("name"),
      email: formData.get("email"),
      password: formData.get("password"),
    });

    // Проверка, существует ли уже пользователь с таким email
    const user = await db.user.findUnique({ where: { email } });
    if (user) {
      return { email: "Пользователь с таким email уже существует" };
    }
```

Окончание листинга Б.3

```
    // Хеширование пароля для безопасного хранения
    const hash = await bcrypt.hash(password, 10);

    // Создание нового пользователя в базе данных
    await db.user.create({
      data: { name, email, password: hash },
    });

    return { success: true };
  } catch (error) {
    // Обработка ошибок валидации
    if (error instanceof ZodError) {
      return error.flatten().fieldErrors;
    }
    // Логирование непредвиденных ошибок
    console.error(error);
    return "Произошла ошибка при регистрации";
  }
}
```

Листинг Б.4 – Исходный код middleware-защиты middleware.ts

```
/**
 * Middleware для защиты маршрутов приложения
 * Проверяет аутентификацию пользователя и перенаправляет неавторизованных пользователей
 */

import { NextResponse } from 'next/server';
import type { NextRequest } from 'next/server';
import { getToken } from 'next-auth/jwt';

// Основной обработчик middleware
export async function middleware(request: NextRequest) {
  const token = await getToken({
    req: request,
    secret: process.env.NEXTAUTH_SECRET
  });

  // Если пользователь не аутентифицирован, перенаправляем на страницу входа
  if (!token) {
    const signinUrl = new URL('/signin', request.url);
    return NextResponse.redirect(signinUrl);
  }
  return NextResponse.next();
}

/**
 * Конфигурация маршрутов для применения middleware
 * Определяет какие пути будут защищены проверкой аутентификации
 */
export const config = {
  matcher: ["/dashboard", "/dashboard/:path*"],
};
```

ПРИЛОЖЕНИЕ В
ИСХОДНЫЙ КОД СОВМЕСТНОЙ РАБОТЫ

Листинг В.1 – Исходный код компонента управления комнатой Room.tsx

```
/**
 * Компонент создания комнаты в Liveblocks
 * Предоставляет контекст совместной работы для дочерних компонентов
 * Инициализирует начальное состояние и присутствие пользователя в комнате
 */
"use client";

import { LiveList, LiveMap } from "@liveblocks/client";
import type { LiveObject } from "@liveblocks/client";
import {
  ClientSideSuspense,
  LiveblocksProvider,
  RoomProvider,
  useStatus,
} from "@liveblocks/react";
import type { ReactNode } from "react";
import type { Layer } from "~/types";
import { PageSkeleton } from "~/components/ui/skeleton";

function RoomContent({ children }: { children: ReactNode }) {
  const status = useStatus();

  // Показываем скелетон, пока не установлено соединение
  if (status !== "connected") {
    return <PageSkeleton />;
  }

  return <>{children}</>;
}

/**
 * Компонент для создания и подключения к комнате Liveblocks
 *
 * @param {Object} props - Параметры компонента
 * @param {ReactNode} props.children - Дочерние компоненты, которые будут иметь доступ к
 * @param {string} props.roomId - Идентификатор комнаты для подключения
 */
export function Room({
  children,
  roomId,
}: {
  children: ReactNode;
  roomId: string;
}) {
  return (
    <LiveblocksProvider authEndpoint="/api/liveblocks-auth">
      <RoomProvider
        id={roomId}
        // Начальное состояние присутствия пользователя в комнате
        initialPresence={{
```

Окончание листинга В.1

```
          selection: [], // Выделенные объекты
          cursor: null, // Положение курсора
          penColor: null, // Цвет пера
          pencilDraft: null, // Черновик карандаша
        }}
        // Начальное состояние хранилища данных комнаты
        initialStorage={{
          roomColor: { r: 239, g: 239, b: 239 }, // Цвет фона комнаты
          layers: new LiveMap<string, LiveObject<Layer>>(), // Карта слоев
          layerIds: new LiveList([]), // Упорядоченный список идентификаторов слоев
          clipboard: [], // Буфер обмена
        }}
      >
        {/* Компонент для отображения состояния загрузки */}
        <ClientSideSuspense fallback={<PageSkeleton />}>
          <RoomContent>{children}</RoomContent>
        </ClientSideSuspense>
      </RoomProvider>
    </LiveblocksProvider>
  );
}
```

Листинг В.2 – Исходный код серверных действий для управления доступом к комнате room.ts

```
"use server";

/**
 * Модуль содержит серверные функции для управления комнатами проекта
 * Включает методы создания, обновления, удаления комнат и управления доступом пользователей
 */

import { revalidatePath } from "next/cache";
import { redirect } from "next/navigation";
import { auth } from "~/server/auth";
import { db } from "~/server/db";

/**
 * Функция для создания новой комнаты
 * После создания перенаправляет пользователя на страницу комнаты
 * @throws {Error} Если идентификатор пользователя не найден
 */
export async function createRoom() {
  const session = await auth(); // Получение сессии пользователя

  // Проверка наличия идентификатора пользователя
  if (!session?.user.id) {
    throw new Error("Идентификатор пользователя не найден.");
  }

  // Создание комнаты и связывание с владельцем
  const room = await db.room.create({
```

Продолжение листинга В.2

```
    data: {
      owner: {
        connect: {
          id: session.user.id,
        },
      },
    },
    select: {
      id: true,
    },
  });

  // Перенаправление на страницу созданной комнаты
  redirect("/dashboard/" + room.id);
}

/**
 * Функция для обновления названия комнаты
 * @param {string} title - Новое название комнаты
 * @param {string} id - Идентификатор комнаты
 * @throws {Error} Если идентификатор пользователя не найден или комната не найдена
 */
export async function updateRoomTitle(title: string, id: string) {
  const session = await auth();

  // Проверка наличия идентификатора пользователя
  if (!session?.user.id) {
    throw new Error("Идентификатор пользователя не найден.");
  }

  // Проверка, что комната существует и принадлежит текущему пользователю
  await db.room.findUniqueOrThrow({
    where: {
      id: id,
      ownerId: session.user.id,
    },
  });

  // Обновление названия комнаты
  await db.room.update({
    where: {
      id: id,
    },
    data: {
      title: title,
    },
  });

  // Обновление данных на клиенте
  revalidatePath("dashboard");
}
/**
 * Функция для удаления доступа пользователя к комнате
 * @param {string} id - Идентификатор комнаты
```

Продолжение листинга В.2

```
 * @throws {Error} Если идентификатор пользователя не найден
 */
export async function removeRoomAccess(id: string) {
  const session = await auth();

  // Проверка наличия идентификатора пользователя
  if (!session?.user.id) {
    throw new Error("Идентификатор пользователя не найден.");
  }

  // Удаляем приглашение для текущего пользователя
  await db.roomInvite.deleteMany({
    where: {
      roomId: id,
      userId: session.user.id,
    },
  });

  // Обновление данных на клиенте
  revalidatePath("dashboard");
}

/**
 * Функция для удаления комнаты
 * Если пользователь является владельцем - удаляет комнату полностью
 * Если нет - только удаляет доступ текущего пользователя
 * @param {string} id - Идентификатор комнаты
 * @throws {Error} Если идентификатор пользователя не найден
 */
export async function deleteRoom(id: string) {
  const session = await auth();

  // Проверка наличия идентификатора пользователя
  if (!session?.user.id) {
    throw new Error("Идентификатор пользователя не найден.");
  }

  // Проверяем, является ли пользователь владельцем комнаты
  const room = await db.room.findUnique({
    where: {
      id: id,
      ownerId: session.user.id,
    },
  });

  if (room) {
    // Если пользователь владелец - удаляем комнату полностью
    await db.room.delete({
      where: {
        id: id,
      },
    });
  } else {
    // Если пользователь не владелец - удаляем только его доступ
```

Продолжение листинга В.2

```
    await removeRoomAccess(id);
  }
  // Обновление данных на клиенте
  revalidatePath("dashboard");
}

/**
 * Функция для отправки приглашения в комнату по email
 * @param {string} id - Идентификатор комнаты
 * @param {string} inviteEmail - Email приглашаемого пользователя
 * @returns {string|undefined} - Сообщение об ошибке или undefined при успехе
 * @throws {Error} Если идентификатор пользователя не найден
 */
export async function shareRoom(id: string, inviteEmail: string) {
  const session = await auth();

  // Проверка наличия идентификатора пользователя
  if (!session?.user.id) {
    throw new Error("Идентификатор пользователя не найден.");
  }

  // Проверка, что комната существует и принадлежит текущему пользователю
  const room = await db.room.findUnique({
    where: {
      id: id,
      ownerId: session.user.id,
    },
  });

  if (!room) {
    return "У вас нет прав для приглашения пользователей в эту комнату.";
  }

  // Поиск пользователя по email для приглашения
  const invitedUser = await db.user.findUnique({
    where: { email: inviteEmail },
    select: { id: true },
  });

  if (!invitedUser) {
    return "Пользователь не найден.";
  }

  // Создание приглашения в комнату
  await db.roomInvite.create({
    data: {
      roomId: id,
      userId: invitedUser.id,
    },
  });

  // Обновление данных на клиенте
  revalidatePath("dashboard");
}
```

Окончание листинга В.2

```
/**
 * Функция для удаления приглашения в комнату
 * @param {string} id - Идентификатор комнаты
 * @param {string} inviteEmail - Email пользователя, приглашение которого удаляется
 * @returns {string|undefined} - Сообщение об ошибке или undefined при успехе
 * @throws {Error} Если идентификатор пользователя не найден
 */
export async function deleteInvitation(id: string, inviteEmail: string) {
  const session = await auth();

  // Проверка наличия идентификатора пользователя
  if (!session?.user.id) {
    throw new Error("Идентификатор пользователя не найден.");
  }

  // Проверка, что комната существует и принадлежит текущему пользователю
  const room = await db.room.findUnique({
    where: {
      id: id,
      ownerId: session.user.id,
    },
  });

  if (!room) {
    return "У вас нет прав для удаления пользователей из этой комнаты.";
  }

  // Удаление приглашений, связанных с указанным email
  await db.roomInvite.deleteMany({
    where: {
      roomId: id,
      user: {
        email: inviteEmail,
      },
    },
  });

  // Обновление данных на клиенте
  revalidatePath("dashboard");
}
```

Листинг В.3 – Исходный код серверного маршрута для работы с комнатами route.ts

```
/**
 * API-маршрут для аутентификации и управления сессиями Liveblocks
 * Позволяет настраивать права доступа к комнатам для совместной работы пользователей
 */
import { Liveblocks } from "@liveblocks/node";
import { env } from "~/env";
import { auth } from "~/server/auth";
import { db } from "~/server/db";
```

Окончание листинга В.3

```
// Инициализация Liveblocks с секретным ключом из переменных окружения
const liveblocks = new Liveblocks({ secret: env.LIVEBLOCKS_SECRET_KEY });

/**
 * Обработчик POST-запросов для аутентификации в Liveblocks
 * @param {Request} req - Объект запроса
 * @returns {Response} Ответ с данными сессии Liveblocks
 */
export async function POST(_req: Request) {
  // Получение текущей сессии пользователя из NextAuth
  const userSession = await auth();

  // Получение полных данных пользователя из базы данных
  // с информацией о его комнатах и приглашениях
  const user = await db.user.findUniqueOrThrow({
    where: { id: userSession?.user.id },
    include: {
      // Комнаты, созданные пользователем (владелец)
      ownedRooms: true,
      // Приглашения в комнаты, полученные пользователем
      roomInvites: {
        include: {
          room: true,
        },
      },
    },
  });

  // Подготовка сессии Liveblocks для пользователя с его профилем
  const session = liveblocks.prepareSession(user.id, {
    userInfo: {
      name: user.email ?? "Незнакомец",
      image: user.image ?? null,
    },
  });

  // Предоставляем полный доступ к комнатам, которыми владеет пользователь
  user.ownedRooms.forEach((room: { id: string }) => {
    session.allow(`room:${room.id}`, session.FULL_ACCESS);
  });

  // Предоставляем полный доступ к комнатам, в которые пользователь был приглашен
  user.roomInvites.forEach((invite: { room: { id: string } }) => {
    session.allow(`room:${invite.room.id}`, session.FULL_ACCESS);
  });

  // Авторизация сессии и получение данных для клиента
  const { status, body } = await session.authorize();

  // Возвращаем ответ с результатами авторизации
  return new Response(body, { status });
}
```

Листинг В.4 – Исходный код страницы комнаты page.tsx

```
"use server";

/**
 * Страница редактора проекта
 * Отображает холст для редактирования проекта с инструментами дизайна
 */

import Canvas from "~/components/canvas/Canvas";
import { Room } from "~/components/liveblocks/Room";
import { auth } from "~/server/auth";
import { CanvasProvider } from "~/components/canvas/helper/CanvasContext";
import { redirect } from "next/navigation";
import { db } from "~/server/db";

// Тип параметров, которые ожидаются от страницы
type ParamsType = Promise<{ id: string }>;

/**
 * Компонент страницы редактора конкретного проекта
 * Предоставляет canvas для работы с дизайном в режиме реального времени
 *
 * @param {Object} props - Свойства компонента
 * @param {ParamsType} props.params - Параметры из URL, включающие ID проекта
 */
export default async function EditorPage({ params }: { params: ParamsType }) {
  // Извлекаем ID проекта из параметров
  const { id } = await params;

  // Получаем сессию текущего пользователя
  const session = await auth();

  // Извлекаем информацию о проекте из базы данных
  const room = await db.room.findUnique({
    where: { id: id },
    select: {
      title: true,
      ownerId: true,
      owner: true,
      roomInvites: {
        select: {
          user: true,
        },
      },
    },
  });

  // Если проект не найден, перенаправляем на страницу 404
  if (!room) {
    redirect("/404");
  }

  // Получаем список ID пользователей, приглашенных в проект
  const inviteeUserIds = room.roomInvites.map((invite) => invite.user.id);
```

Окончание листинга В.4

```
  // Проверяем, что пользователь имеет право доступа к проекту
  // (либо владелец, либо приглашен)
  const userHasAccess =
    inviteeUserIds.includes(session?.user.id ?? "") ||
    session?.user.id === room.ownerId;

  if (!userHasAccess) {
    redirect("/404");
  }

  return (
    // Оборачиваем в компонент Room для синхронизации через Liveblocks
    <Room roomId={`room:${id}`}>
      {/* Провайдер контекста для холста */}
      <CanvasProvider>
        {/* Сам компонент холста */}
        <Canvas
          roomName={room.title}
          roomId={id}
          othersWithAccessToRoom={room.roomInvites.map((invite) => invite.user)}
          owner={room.owner}
        />
      </CanvasProvider>
    </Room>
  );
}
```

ПРИЛОЖЕНИЕ Г
ИСХОДНЫЙ КОД ИНТЕРАКТИВНОЙ ДОСКИ

Листинг Г.1 – Исходный код основного компонента Canvas.tsx

```
"use client";

/**
 * Основной компонент редактора дизайна
 * Отвечает за отрисовку холста, обработку взаимодействий пользователя с холстом,
 * и управление слоями и объектами на холсте
 */

import { useMutation } from "@liveblocks/react";
import {
  calculateBoundingBox,
  colorToCss,
  pointerEventToCanvasPoint,
} from "~/utils";
import { CanvasMode, LayerType, type Point } from "~/types";
import { useState, useRef, useEffect } from "react";
import { useCanvas } from "~/components/canvas/helper/CanvasContext";
import { useDrawingFunctions } from "~/components/canvas/helper/DrawingFunctions";
import { useLayerManipulation } from "~/components/canvas/helper/LayerManipulation";
import { useSelectionFunctions } from "~/components/canvas/helper/SelectionFunctions";
import { useCreateLayerFunctions } from "~/components/canvas/helper/ShapeDrawingFunctions";

import LayerComponent from "./LayerComponent";
import Path from "./shapes/Path";
import SelectionBox from "./SelectionBox";
import SelectionTools from "~/components/canvas/SelectionTools";
import ToolsBar from "~/components/panels/ToolsBar";
import { useHotkeys } from "~/hooks/use-hotkeys";
import RenderPreviewLayer from "~/components/canvas/helper/RenderPreviewLayer";
import Sidebars from "~/components/panels/UICanvas";
import { useZoom } from "~/hooks/use-zoom";
import MultiplayerGuides from "./MultiplayerGuides";
import type { User } from "@prisma/client";

export default function Canvas({
  roomName,
  roomId,
  othersWithAccessToRoom,
  owner,
}: {
  roomName: string;
  roomId: string;
  othersWithAccessToRoom: User[];
  owner: User;
}) {
  const svgRef = useRef<SVGSVGElement>(null); // Ссылка на SVG элемент

  const {
    canvasState,
    setState,
```

Продолжение листинга Г.1

```
    setCamera,
    leftIsMinimized,
    setLeftIsMinimized,
    roomColor,
    layerIds,
    pencilDraft,
    history,
  } = useCanvas(); // Получаем состояние холста из контекста

  useZoom(camera, setCamera, svgRef as React.RefObject<SVGSVGElement>);
  useHotkeys(setState, setCamera, leftIsMinimized, setLeftIsMinimized); // Подключаем хук
для горячих клавиш

  const { startDrawing, continueDrawing, insertPath } = useDrawingFunctions(); // Функции
для рисования
  const { insertLayerByClick, insertLayerByDragging } =
    useCreateLayerFunctions(); // Функции для создания слоев
  const {
    onResizeHandlePointerDown,
    resizeSelectedLayer,
    translateSelectedLayers,
  } = useLayerManipulation(); // Функции для изменения слоев
  const { unselectLayers, startMultiSelection, updateSelectionNet } =
    useSelectionFunctions(); // Функции для выбора

  const [isEditingText, setIsEditingText] = useState(false); // Состояние редактирования
текста

  // Состояния для перемещения холста
  const [isDragging, setIsDragging] = useState(false);
  const [previousMode, setPreviousMode] = useState<CanvasMode | null>(null);
  const [lastMousePosition, setLastMousePosition] = useState<Point | null>(
    null,
  );

  // Добавляем обработчики событий для клавиши Shift при создании или изменении размеров
фигур
  useEffect(() => {
    const handleKeyDown = (e: KeyboardEvent) => {
      if (
        (canvasState.mode === CanvasMode.CreatingShape ||
          canvasState.mode === CanvasMode.Resizing) &&
        e.key === "Shift"
      ) {
        setState((prev) => ({
          ...prev,
          isShiftPressed: true,
        }));
      }
    };

    const handleKeyUp = (e: KeyboardEvent) => {
      if (
        (canvasState.mode === CanvasMode.CreatingShape ||
```

Продолжение листинга Г.1

```
        canvasState.mode === CanvasMode.Resizing) &&
      e.key === "Shift"
    ) {
      setState((prev) => ({
        ...prev,
        isShiftPressed: false,
      }));
    }
  };

  window.addEventListener("keydown", handleKeyDown);
  window.addEventListener("keyup", handleKeyUp);

  return () => {
    window.removeEventListener("keydown", handleKeyDown);
    window.removeEventListener("keyup", handleKeyUp);
  };
}, [canvasState.mode, setState]);

// Обработчики для перемещения холста с помощью средней кнопки мыши
useEffect(() => {
  const svg = svgRef.current;
  if (!svg) return;

  const handleMouseDown = (e: MouseEvent) => {
    if (e.button === 1) {
      // Средняя кнопка мыши
      e.preventDefault();
      setIsDragging(true);
      setLastMousePosition({ x: e.clientX, y: e.clientY });
      setPreviousMode(canvasState.mode);
      setState({ mode: CanvasMode.Dragging, origin: null });
    }
  };

  const handleMouseMove = (e: MouseEvent) => {
    if (isDragging && lastMousePosition) {
      e.preventDefault();

      const deltaX = e.clientX - lastMousePosition.x;
      const deltaY = e.clientY - lastMousePosition.y;

      // Используем requestAnimationFrame для плавного обновления
      requestAnimationFrame(() => {
        setCamera((camera) => ({
          x: camera.x + deltaX,
          y: camera.y + deltaY,
          zoom: camera.zoom,
        }));
      });

      setLastMousePosition({ x: e.clientX, y: e.clientY });
    }
  };
```

Продолжение листинга Г.1

```
  const handleMouseUp = (e: MouseEvent) => {
    if (e.button === 1) {
      setIsDragging(false);
      setLastMousePosition(null);
      if (previousMode !== null) {
        switch (previousMode) {
          case CanvasMode.None:
            setState({ mode: CanvasMode.None });
            break;
          case CanvasMode.Dragging:
            setState({ mode: CanvasMode.Dragging, origin: null });
            break;
          case CanvasMode.Pencil:
            setState({ mode: CanvasMode.Pencil });
            break;
          case CanvasMode.Inserting:
            setState({
              mode: CanvasMode.Inserting,
              layerType: LayerType.Rectangle,
            });
            break;
          default:
            setState({ mode: CanvasMode.None });
        }
        setPreviousMode(null);
      }
    }
  };

  svg.addEventListener("mousedown", handleMouseDown);
  window.addEventListener("mousemove", handleMouseMove);
  window.addEventListener("mouseup", handleMouseUp);

  return () => {
    svg.removeEventListener("mousedown", handleMouseDown);
    window.removeEventListener("mousemove", handleMouseMove);
    window.removeEventListener("mouseup", handleMouseUp);
  };
}, [
  camera,
  setCamera,
  canvasState.mode,
  setState,
  previousMode,
  isDragging,
  lastMousePosition,
]);

// Обработчик выхода курсора за пределы холста
const onPointerLeave = useMutation(({ setMyPresence }) => {
  setMyPresence({ cursor: null });
}, []);

// Обработчик нажатия указателя
```

Продолжение листинга Г.1

```
  const onPointerDown = useMutation(
    ({}, e: React.PointerEvent) => {
      const point = pointerEventToCanvasPoint(e, camera); // Получаем координаты точки на
холсте

      // Проверяем правый клик
      if (e.nativeEvent.button === 2) {
        setState({ mode: CanvasMode.RightClick });
        return;
      }

      // В режиме перемещения сохраняем начальную точку, от которой будет происходить
движение
      if (canvasState.mode === CanvasMode.Dragging) {
        setState({ mode: CanvasMode.Dragging, origin: point });
        return;
      }

      // Режим вставки объекта
      if (canvasState.mode === CanvasMode.Inserting) {
        unselectLayers();

        // Для всех типов слоев, включая изображения, используем одинаковый подход
        setState({
          mode: CanvasMode.CreatingShape,
          origin: point,
          current: point,
          layerType: canvasState.layerType,
          isClick: true, // Пока считаем, что это просто клик
          isShiftPressed: e.shiftKey,
          position: { x: point.x, y: point.y, width: 0, height: 0 },
          // Передаем данные изображения, если они есть
          ...("imageData" in canvasState
            ? { imageData: canvasState.imageData }
            : {}),
        });
        return;
      }

      // В режиме рисования вызываем функцию начала рисования
      if (canvasState.mode === CanvasMode.Pencil) {
        startDrawing(point, e.pressure);
        return;
      }

      setState({ origin: point, mode: CanvasMode.Pressing });
    },
    [camera, canvasState.mode, setState, startDrawing, unselectLayers],
  );

  // Обработчик движения указателя
  const onPointerMove = useMutation(
    ({ setMyPresence }, e: React.PointerEvent) => {
      const point = pointerEventToCanvasPoint(e, camera);
```

Продолжение листинга Г.1

```
if (canvasState.mode === CanvasMode.Pressing) {
  // Начало множественного выделения
  startMultiSelection(point, canvasState.origin);
} else if (canvasState.mode === CanvasMode.SelectionNet) {
  // Обновление сети выделения
  updateSelectionNet(point, canvasState.origin);
} else if (
  canvasState.mode === CanvasMode.Dragging &&
  canvasState.origin !== null
) {
  // Перемещение холста
  const deltaX = e.movementX;
  const deltaY = e.movementY;

  // Используем requestAnimationFrame для плавного обновления
  requestAnimationFrame(() => {
    setCamera((camera) => ({
      x: camera.x + deltaX,
      y: camera.y + deltaY,
      zoom: camera.zoom,
    }));
  });
} else if (canvasState.mode === CanvasMode.Translating) {
  // Перемещение выбранных слоев
  translateSelectedLayers(point);
} else if (canvasState.mode === CanvasMode.Pencil) {
  // Продолжение рисования
  continueDrawing(point, e);
} else if (canvasState.mode === CanvasMode.Resizing) {
  // Изменение размера выбранного слоя
  resizeSelectedLayer(point);
} else if (
  canvasState.mode === CanvasMode.CreatingShape &&
  canvasState.origin
) {
  // Создание новой фигуры
  const { x, y, width, height } = calculateBoundingBox(
    canvasState.origin,
    point,
    canvasState.isShiftPressed,
  );

  // Проверяем, что это не простой клик
  if ((width > 5 || height > 5) && canvasState.isClick) {
    setState((prevState) => ({
      ...prevState,
      isClick: false,
    }));
  }

  setState((prevState) => ({
    ...prevState,
    current: point,
    position: { x, y, width, height },
```

Продолжение листинга Г.1

```
      }));
    }

    // Обновляем позицию курсора для других пользователей
    setMyPresence({ cursor: point });
  },
  [
    camera,
    canvasState,
    continueDrawing,
    resizeSelectedLayer,
    translateSelectedLayers,
    startMultiSelection,
    updateSelectionNet,
  ],
);

// Обработчик отпускания указателя
const onPointerUp = useMutation(
  ({}, _e: React.PointerEvent) => {
    if (canvasState.mode === CanvasMode.RightClick) return;

    if ([CanvasMode.None, CanvasMode.Pressing].includes(canvasState.mode)) {
      // Отменяем выбор слоев и сбрасываем режим
      unselectLayers();
      setState({ mode: CanvasMode.None });
    } else if (
      canvasState.mode === CanvasMode.CreatingShape &&
      canvasState.origin
    ) {
      // Создание нового слоя
      if (
        canvasState.isClick ||
        ("layerType" in canvasState &&
          canvasState.layerType === LayerType.Image)
      ) {
        // Для простого клика или изображения
        insertLayerByClick(canvasState.origin, canvasState.layerType);
      } else if (
        canvasState.position &&
        "layerType" in canvasState &&
        canvasState.layerType !== LayerType.Image
      ) {
        // Для создания фигуры перетаскиванием
        insertLayerByDragging(canvasState.position);
      }
    } else if (canvasState.mode === CanvasMode.Dragging) {
      // Сброс начальной точки при окончании перетаскивания
      setState({ mode: CanvasMode.Dragging, origin: null });
    } else if (canvasState.mode === CanvasMode.Pencil) {
      // Вставка нарисованного пути
      insertPath();
    } else {
      // Сброс режима по умолчанию
```

Продолжение листинга Г.1

```
      setState({ mode: CanvasMode.None });
    }

    // Возобновляем запись истории
    history.resume();
  },
  [
    canvasState,
    setState,
    unselectLayers,
    history,
    insertLayerByClick,
    insertLayerByDragging,
    insertPath,
  ],
);

// Обработчик нажатия на слой
const onLayerPointerDown = useMutation(
  ({ self, setMyPresence }, e: React.PointerEvent, layerId: string) => {
    // Игнорирование действий в режиме рисования или вставки объекта
    if (
      canvasState.mode === CanvasMode.Pencil ||
      canvasState.mode === CanvasMode.Inserting
    ) {
      return;
    }

    history.pause(); // Приостанавливаем историю
    e.stopPropagation(); // Останавливаем всплытие клика

    // Проверяем правый клик
    if (e.nativeEvent.button === 2) {
      if (!self.presence.selection.includes(layerId)) {
        setMyPresence(
          {
            selection: [layerId],
          },
          { addToHistory: true },
        );
      }
      setState({ mode: CanvasMode.RightClick });
      return;
    }

    // Выбираем слой, если он еще не выбран
    if (!self.presence.selection.includes(layerId)) {
      setMyPresence(
        {
          selection: [layerId],
        },
        { addToHistory: true },
      );
    }
```

Продолжение листинга Г.1

```
    const point = pointerEventToCanvasPoint(e, camera);
    setState({ mode: CanvasMode.Translating, current: point });
  },
  [camera, canvasState.mode, history, setState],
);

return (
  <div className="flex h-screen w-full">
    <main className="fixed left-0 right-0 h-screen overflow-y-auto">
      {/* Контейнер с цветом фона, основанным на roomColor или дефолтном значении */}
      <div
        style={{backgroundColor: roomColor ? colorToCss(roomColor) : "#efefef"}}
        className="h-full w-full touch-none select-none"
      >
        {/* Контекстное меню по правому клику */}
        <SelectionTools camera={camera} canvasMode={canvasState.mode} />

        {/* SVG-элемент для рендеринга графики */}
        <svg
          ref={svgRef}
          onPointerUp={onPointerUp}
          onPointerDown={onPointerDown}
          onPointerMove={onPointerMove}
          onPointerLeave={onPointerLeave}
          className="h-full w-full touch-none select-none"
          onContextMenu={(e) => e.preventDefault()}
          style={{ touchAction: "none" }}
        >
          {/* Группа для рендеринга слоев с трансформацией для камеры */}
          <g
            style={{
              transform: `translate(${camera.x}px, ${camera.y}px) scale(${camera.zoom})`,
            }}
          >
            {/* Отображаем все слои */}
            {layerIds?.map((layerId) => (
              <LayerComponent
                key={layerId}
                id={layerId}
                onLayerPointerDown={onLayerPointerDown}
                canvasMode={canvasState.mode}
                setIsEditingText={setIsEditingText}
              />
            ))}

            {/* Предпросмотр создаваемого слоя */}
            <RenderPreviewLayer canvasState={canvasState} />

            {/* Бокс с размерами для выделения слоев */}
            <SelectionBox
              onResizeHandlePointerDown={onResizeHandlePointerDown}
              isEditing={isEditingText}
            />
```

Окончание листинга Г.1

```
            {canvasState.mode === CanvasMode.SelectionNet &&
              canvasState.current != null && (
                <rect
                  className="fill-primary-alpha-10 stroke-primary-light stroke-[0.5]"
                  x={Math.min(canvasState.origin.x, canvasState.current.x)}
                  y={Math.min(canvasState.origin.y, canvasState.current.y)}
                  width={Math.abs(
                    canvasState.origin.x - canvasState.current.x,
                  )}
                  height={Math.abs(
                    canvasState.origin.y - canvasState.current.y,
                  )}
                />
              )}

            {/* Отображение указателей других пользователей */}
            <MultiplayerGuides />

            {/* Отображение черновика при рисовании карандашом */}
            {pencilDraft !== null && pencilDraft.length > 0 && (
              <Path
                x={0}
                y={0}
                opacity={100}
                fill={colorToCss({ r: 0, g: 0, b: 0 })}
                points={pencilDraft}
                canvasMode={canvasState.mode}
              />
            )}
          </g>
        </svg>
      </div>
    </main>

    {/* Панель инструментов */}
    <ToolsBar
      canvasState={canvasState}
      setCanvasState={(newState) => setState(newState)}
    />

    {/* Боковые панели */}
    <Sidebars
      roomName={roomName}
      roomId={roomId}
      othersWithAccessToRoom={othersWithAccessToRoom}
      leftIsMinimized={leftIsMinimized}
      setLeftIsMinimized={setLeftIsMinimized}
      owner={owner}
    />
  </div>
 );
}
```

Листинг Г.2 – Исходный код контекста холста CanvasContext.tsx

```
"use client";

/**
 * Контекст для состояния холста и общего доступа к функциям холста
 * Предоставляет глобальное состояние и функции для всех компонентов холста
 */

import React, { createContext, useContext, useState } from "react";
import { type Camera, CanvasMode, type CanvasState } from "~/types";
import { useHistory, useSelf, useStorage } from "@liveblocks/react";

// Интерфейс типов данных, предоставляемых контекстом холста
interface CanvasContextType {
  canvasState: CanvasState; // Текущее состояние холста
  setState:    React.Dispatch<React.SetStateAction<CanvasState>>;    //    Функция    изменения
состояния
  camera: Camera; // Позиция и масштаб камеры
  setCamera: React.Dispatch<React.SetStateAction<Camera>>; // Функция изменения камеры
  leftIsMinimized: boolean; // Состояние левой панели (свернута/развернута)
  setLeftIsMinimized: React.Dispatch<React.SetStateAction<boolean>>;
  roomColor: {
    readonly r: number;
    readonly g: number;
    readonly b: number;
  } | null; // Цвет комнаты
  layerIds: readonly string[] | null; // Массив ID слоев
  pencilDraft: [x: number, y: number, pressure: number][] | null; // Текущий рисуемый путь
  history: ReturnType<typeof useHistory>; // История изменений
  MAX_LAYERS: number; // Максимальное количество слоев
}

// Создание контекста
const CanvasContext = createContext<CanvasContextType | undefined>(undefined);

// Провайдер контекста холста
export function CanvasProvider({ children }: { children: React.ReactNode }) {
  // Состояние холста
  const [canvasState, setState] = useState<CanvasState>({
    mode: CanvasMode.None,
  });
  // Состояние камеры (позиция и масштаб)
  const [camera, setCamera] = useState<Camera>({ x: 0, y: 0, zoom: 1 });
  // Состояние левой панели
  const [leftIsMinimized, setLeftIsMinimized] = useState(false);

  // Получение данных из хранилища Liveblocks
  const roomColor = useStorage((root) => root.roomColor);
  const layerIds = useStorage((root) => root.layerIds);
  const pencilDraft = useSelf((me) => me.presence.pencilDraft);
  const history = useHistory();
  const MAX_LAYERS = 100; // Ограничение на количество слоев

  return (
    <CanvasContext.Provider
```

Продолжение листинга Г.2

```
      value={{
        canvasState,
        setState,
        camera,
        setCamera,
        leftIsMinimized,
        setLeftIsMinimized,
        roomColor,
        layerIds,
        pencilDraft,
        history,
        MAX_LAYERS,
      }}
    >
      {children}
    </CanvasContext.Provider>
  );
}

// Хук для использования контекста холста в компонентах
export function useCanvas() {
  const context = useContext(CanvasContext);
  if (!context) {
    throw new Error("useCanvas должен использоваться внутри CanvasProvider");
  }
  return context;
}
```

Листинг Г.3 – Исходный код хука управления масштабированием use-zoom.ts

```
"use client";

/**
 * Хук для управления масштабированием и панорамированием холста
 * Позволяет изменять масштаб и положение холста с помощью колесика мыши
 */

import { useCallback, useEffect } from "react";
import type { Camera } from "~/types";
import { pointerEventToCanvasPoint } from "~/utils";

const MIN_ZOOM = 0.02; // Минимальный масштаб 2%
const MAX_ZOOM = 10; // Максимальный масштаб 1000%

/**
 * Хук для управления масштабированием и перемещением по холсту
 *
 * @param camera - Текущее состояние камеры (позиция и масштаб)
 * @param setCamera - Функция для обновления состояния камеры
 * @param svgRef - Ссылка на SVG-элемент холста
 * @returns Объект с функциями для управления масштабом
 */
```

Продолжение листинга Г.3

```
export const useZoom = (
  camera: Camera,
  setCamera: (updater: (prev: Camera) => Camera) => void,
  svgRef: React.RefObject<SVGSVGElement>,
) => {
  /**
   * Обработчик события прокрутки колеса мыши
   * Изменяет масштаб при зажатом Ctrl или перемещает холст
   */
  const handleWheel = useCallback(
    (e: WheelEvent) => {
      // Предотвращаем стандартное поведение браузера
      e.preventDefault();
      e.stopPropagation();

      if (e.ctrlKey) {
        // Масштабирование при зажатом Ctrl
        const point = pointerEventToCanvasPoint(
          e as unknown as React.WheelEvent,
          camera,
        );
        const zoomFactor = 1.2;
        const delta = e.deltaY < 0 ? zoomFactor : 1 / zoomFactor;

        setCamera((prev: Camera) => {
          const newZoom = Math.min(
            Math.max(prev.zoom * delta, MIN_ZOOM),
            MAX_ZOOM,
          );

          // Рассчитываем смещение для зума относительно курсора
          const dx = point.x * (1 - delta);
          const dy = point.y * (1 - delta);

          return {
            x: prev.x + dx * prev.zoom,
            y: prev.y + dy * prev.zoom,
            zoom: newZoom,
          };
        });
      } else {
        // Панорамирование холста при обычной прокрутке
        const deltaX = e.shiftKey ? e.deltaY : e.deltaX;
        const deltaY = e.shiftKey ? 0 : e.deltaY;

        setCamera((prev: Camera) => ({
          x: prev.x - deltaX,
          y: prev.y - deltaY,
          zoom: prev.zoom,
        }));
      }
    },
    [camera, setCamera],
  );
```

Продолжение листинга Г.3

```
// Добавляем обработчик события прокрутки к SVG-элементу
useEffect(() => {
  const svg = svgRef.current;
  if (!svg) return;

  svg.addEventListener("wheel", handleWheel, { passive: false });
  return () => {
    svg.removeEventListener("wheel", handleWheel);
  };
}, [handleWheel, svgRef]);

/**
 * Увеличение масштаба холста
 * Масштабирование происходит относительно центра видимой области
 */
const zoomIn = useCallback(() => {
  const svg = document.querySelector("svg");
  if (svg) {
    const rect = svg.getBoundingClientRect();
    const centerPoint = {
      x: (rect.width / 2 - camera.x) / camera.zoom,
      y: (rect.height / 2 - camera.y) / camera.zoom,
    };
    const zoomFactor = 1.2;

    setCamera((prev: Camera) => {
      const newZoom = Math.min(prev.zoom * zoomFactor, MAX_ZOOM);
      const dx = centerPoint.x * (1 - zoomFactor);
      const dy = centerPoint.y * (1 - zoomFactor);

      return {
        x: prev.x + dx * prev.zoom,
        y: prev.y + dy * prev.zoom,
        zoom: newZoom,
      };
    });
  }
}, [camera, setCamera]);

/**
 * Уменьшение масштаба холста
 * Масштабирование происходит относительно центра видимой области
 */
const zoomOut = useCallback(() => {
  const svg = document.querySelector("svg");
  if (svg) {
    const rect = svg.getBoundingClientRect();
    const centerPoint = {
      x: (rect.width / 2 - camera.x) / camera.zoom,
      y: (rect.height / 2 - camera.y) / camera.zoom,
    };
    const zoomFactor = 1 / 1.2;

    setCamera((prev: Camera) => {
```

Окончание листинга Г.3

```
        const newZoom = Math.max(prev.zoom * zoomFactor, MIN_ZOOM);
        const dx = centerPoint.x * (1 - zoomFactor);
        const dy = centerPoint.y * (1 - zoomFactor);

        return {
          x: prev.x + dx * prev.zoom,
          y: prev.y + dy * prev.zoom,
          zoom: newZoom,
        };
      });
    }
  }, [camera, setCamera]);

  return { zoomIn, zoomOut };
};
```

Листинг Г.4 – Исходный код скрипта для предотвращения стандартного масштабирования ClientScripts.tsx

```
"use client";

/**
 * Компонент для отключения масштабирования страницы в браузере
 * Предотвращает разные жесты масштабирования
*/

import { useEffect } from "react";

/**
 * Компонент ClientScripts блокирует нежелательные жесты масштабирования
 * Не имеет визуального представления, работает только на уровне событий
 */
export default function ClientScripts() {
  useEffect(() => {
    /**
     * Предотвращает масштабирование с помощью колесика мыши при зажатом Ctrl/Cmd
     * @param {WheelEvent} e - Событие прокрутки колесиком мыши
     */
    const preventZoom = (e: WheelEvent) => {
      if (e.ctrlKey || e.metaKey) {
        e.preventDefault();
      }
    };

    /**
     * Предотвращает масштабирование жестом щипка на сенсорных устройствах
     * @param {TouchEvent} e - Событие касания экрана
     */
    const preventPinch = (e: TouchEvent) => {
      if (e.touches.length > 1) {
        e.preventDefault();
```

Окончание листинга Г.4

```
    }
  };

  /**
   * Предотвращает масштабирование жестами на MacOS-устройствах
   * @param {Event} e - Событие жеста масштабирования
   */
  const preventGesture = (e: Event) => {
    e.preventDefault();
  };

  /**
   * Предотвращает масштабирование двойным тапом на мобильных устройствах
   * Определяет быстрые последовательные касания и блокирует их
   * @param {TouchEvent} e - Событие окончания касания экрана
   */
  let lastTouchEnd = 0;
  const preventDoubleTap = (e: TouchEvent) => {
    const now = new Date().getTime();
    if (now - lastTouchEnd <= 300) {
      e.preventDefault();
    }
    lastTouchEnd = now;
  };

  // Добавление обработчиков событий для всех типов жестов масштабирования
  document.addEventListener("wheel", preventZoom, { passive: false });
  document.addEventListener("touchstart", preventPinch, { passive: false });
  document.addEventListener("gesturestart", preventGesture);
  document.addEventListener("touchend", preventDoubleTap, false);

  // Удаление обработчиков при размонтировании компонента для предотвращения утечек памяти
  return () => {
    document.removeEventListener("wheel", preventZoom);
    document.removeEventListener("touchstart", preventPinch);
    document.removeEventListener("gesturestart", preventGesture);
    document.removeEventListener("touchend", preventDoubleTap);
  };
}, []);

// Компонент не рендерит никакой UI
return null;
}
```

Листинг Г.5 – Исходный код контекстного меню SelectionTools.tsx

```
"use client";

/**
 * Компонент для отображения контекстного меню выделенных элементов на холсте.
 * Позволяет управлять порядком слоев выделенных элементов (передний/задний план).
 */
```

Продолжение листинга Г.5

```
import useSelectionBounds from "~/hooks/use-selection-bounds";
import type { Camera } from "~/types";
import { CanvasMode } from "~/types";
import { useMutation, useSelf } from "@liveblocks/react";
import { memo } from "react";
import * as Dropdown from "~/components/ui/dropdown";
import * as Kbd from "~/components/ui/kbd";

/**
 * Интерфейс пропсов компонента SelectionTools
 */
interface SelectionToolsProps {
  camera: Camera;
  canvasMode: CanvasMode;
}

/**
 * Компонент контекстного меню для управления выделенными элементами
 */
function SelectionTools({ camera, canvasMode }: SelectionToolsProps) {
  // Получаем ID выделенных элементов из хранилища Liveblocks
  const selection = useSelf((me) => me.presence.selection);

  /**
   * Мутация для перемещения выделенных элементов на передний план
   */
  const bringToFront = useMutation(
    ({ storage }) => {
      const liveLayerIds = storage.get("layerIds");
      const indices: number[] = [];
      const arr = liveLayerIds.toArray();

      // Находим индексы выделенных элементов
      for (let i = 0; i < arr.length; i++) {
        const element = arr[i];
        if (element !== undefined && selection?.includes(element)) {
          indices.push(i);
        }
      }

      // Перемещаем элементы на передний план (в конец массива)
      for (let i = indices.length - 1; i >= 0; i--) {
        const element = indices[i];
        if (element !== undefined) {
          liveLayerIds.move(element, arr.length - 1 - (indices.length - 1 - i));
        }
      }
    },
    [selection],
  );

  /**
   * Мутация для перемещения выделенных элементов на задний план
   */
```

Продолжение листинга Г.5

```
const sendToBack = useMutation(
  ({ storage }) => {
    const liveLayerIds = storage.get("layerIds");
    const indices: number[] = [];
    const arr = liveLayerIds.toArray();

    // Находим индексы выделенных элементов
    for (let i = 0; i < arr.length; i++) {
      const element = arr[i];
      if (element !== undefined && selection?.includes(element)) {
        indices.push(i);
      }
    }

    // Перемещаем элементы на задний план (в начало массива)
    for (let i = 0; i < indices.length; i++) {
      const element = indices[i];
      if (element !== undefined) {
        liveLayerIds.move(element, i);
      }
    }
  },
  [selection],
);

/**
 * Мутация для перемещения выделенных элементов на один слой вперед
 */
const bringForward = useMutation(
  ({ storage }) => {
    const liveLayerIds = storage.get("layerIds");
    const indices: number[] = [];
    const arr = liveLayerIds.toArray();

    // Находим индексы выделенных элементов
    for (let i = 0; i < arr.length; i++) {
      const element = arr[i];
      if (element !== undefined && selection?.includes(element)) {
        indices.push(i);
      }
    }

    // Сортируем индексы в порядке убывания для перемещения снизу вверх
    indices.sort((a, b) => b - a);

    // Перемещаем каждый элемент на одну позицию вперед
    for (const index of indices) {
      if (index < arr.length - 1) {
        liveLayerIds.move(index, index + 1);
      }
    }
  },
  [selection],
);
```

Продолжение листинга Г.5

```
/**
 * Мутация для перемещения выделенных элементов на один слой назад
 */
const sendBackward = useMutation(
  ({ storage }) => {
    const liveLayerIds = storage.get("layerIds");
    const indices: number[] = [];
    const arr = liveLayerIds.toArray();

    // Находим индексы выделенных элементов
    for (let i = 0; i < arr.length; i++) {
      const element = arr[i];
      if (element !== undefined && selection?.includes(element)) {
        indices.push(i);
      }
    }

    // Сортируем индексы в порядке возрастания для перемещения сверху вниз
    indices.sort((a, b) => a - b);

    // Перемещаем каждый элемент на одну позицию назад
    for (const index of indices) {
      if (index > 0) {
        liveLayerIds.move(index, index - 1);
      }
    }
  },
  [selection],
);

// Получаем границы выделения
const selectionBounds = useSelectionBounds();
if (!selectionBounds) {
  return null;
}

// Рассчитываем позицию для меню относительно выделения с учетом масштаба
const x =
  (selectionBounds.width / 2 + selectionBounds.x) * camera.zoom + camera.x;
const y = selectionBounds.y * camera.zoom + camera.y;

// Отображаем меню только в режиме правого клика
if (canvasMode !== CanvasMode.RightClick) return null;

return (
  <Dropdown.Root open modal={false}>
    <Dropdown.Content
      align="start"
      sideOffset={0}
      className="absolute w-60"
      style={{
        transform: `translate(calc(${x}px - 50%), calc(${y - 8}px - 100%))`,
      }}
    >
```

Окончание листинга Г.5

```
          <Dropdown.Group className="flex flex-col gap-0">
            <Dropdown.Item
              onSelect={bringToFront}
              className="flex items-center justify-between"
            >
              На передний план
              <Kbd.Root>]</Kbd.Root>
            </Dropdown.Item>
            <Dropdown.Item
              onSelect={bringForward}
              className="flex items-center justify-between"
            >
              Переместить вперед
              <Kbd.Root>Ctrl ]</Kbd.Root>
            </Dropdown.Item>

            <Dropdown.Item
              onSelect={sendToBack}
              className="flex items-center justify-between"
            >
              На задний план
              <Kbd.Root>[</Kbd.Root>
            </Dropdown.Item>

            <Dropdown.Item
              onSelect={sendBackward}
              className="flex items-center justify-between"
            >
              Переместить назад
              <Kbd.Root>Ctrl [</Kbd.Root>
            </Dropdown.Item>
          </Dropdown.Group>
        </Dropdown.Content>
      </Dropdown.Root>
    );
}

export default memo(SelectionTools);
```

ПРИЛОЖЕНИЕ Д
ИСХОДНЫЙ КОД УПРАВЛЕНИЯ СЛОЯМИ

Листинг Д.1 – Исходный код компонента слоя LayerComponent.tsx

```
"use client";

/**
 * Универсальный компонент для рендеринга различных типов слоев на холсте.
 * Работает как диспетчер, определяющий какой специализированный компонент
 * нужно отрисовать в зависимости от типа слоя (прямоугольник, эллипс,
 * треугольник, текст, путь, изображение, фрейм).
 *
 * Мемоизирован для оптимизации производительности при частых перерисовках.
 */

import { useStorage } from "@liveblocks/react";
import React, { memo } from "react";
import type { CanvasMode } from "~/types";
import { LayerType } from "~/types";
import { colorToCss } from "~/utils";

// Импортируем компоненты для разных типов слоев
import Rectangle from "./shapes/Rectangle";
import Ellipse from "./shapes/Ellipse";
import Text from "./shapes/Text";
import Path from "./shapes/Path";
import Frame from "./shapes/Frame";
import Image from "./shapes/Image";

/**
 * Интерфейс пропсов для компонента LayerComponent
 *
 * @param id - Уникальный идентификатор слоя
 * @param onLayerPointerDown - Обработчик события нажатия указателя на слой
 * @param canvasMode - Текущий режим холста
 * @param setIsEditingText - Функция для установки состояния редактирования текста
 * @param inFrame - Флаг, указывающий, находится ли слой внутри фрейма
 */
interface LayerComponentProps {
  id: string;
  onLayerPointerDown: (e: React.PointerEvent, layerId: string) => void;
  canvasMode: CanvasMode;
  setIsEditingText: (isEditing: boolean) => void;
  _inFrame?: boolean;
}

/**
 * Компонент для рендеринга различных типов слоев на холсте.
 * Определяет тип слоя и рендерит соответствующий компонент.
 */
const LayerComponent = memo(
  ({
    id,
    onLayerPointerDown,
```

Продолжение листинга Д.1

```
  canvasMode,
  setIsEditingText,
  _inFrame = false,
}: LayerComponentProps) => {
  // Получение данных слоя из хранилища Liveblocks по его id
  const layer = useStorage((root) => root.layers.get(id));

  // Если слой отсутствует, ничего не рендерим
  if (!layer) {
    return null;
  }

  // Рендеринг слоя в зависимости от его типа
  switch (layer.type) {
    // Прямоугольник
    case LayerType.Rectangle:
      return (
        <Rectangle
          onPointerDown={onLayerPointerDown}
          id={id}
          layer={layer}
          canvasMode={canvasMode}
        />
      );

    // Эллипс
    case LayerType.Ellipse:
      return (
        <Ellipse
          onPointerDown={onLayerPointerDown}
          id={id}
          layer={layer}
          canvasMode={canvasMode}
        />
      );

    // Путь (рукописная линия)
    case LayerType.Path:
      return (
        <Path
          onPointerDown={(e) => onLayerPointerDown(e, id)}
          points={layer.points}
          x={layer.x}
          y={layer.y}
          fill={layer.fill ? colorToCss(layer.fill) : "#000"}
          stroke={layer.stroke ? colorToCss(layer.stroke) : undefined}
          opacity={layer.opacity}
          blendMode={layer.blendMode}
          canvasMode={canvasMode}
          strokeWidth={layer.strokeWidth}
        />
      );

    // Текст
```

Продолжение листинга Д.1

```
case LayerType.Text:
  return (
    <Text
      onPointerDown={onLayerPointerDown}
      id={id}
      layer={layer}
      setIsEditingText={setIsEditingText}
      canvasMode={canvasMode}
    />
  );

// Изображение
case LayerType.Image:
  return (
    <Image
      onPointerDown={onLayerPointerDown}
      id={id}
      layer={layer}
      canvasMode={canvasMode}
      aria-label="image"
    />
  );

// Фрейм (контейнер, содержащий другие элементы)
case LayerType.Frame:
  // Получаем массив ID дочерних элементов фрейма
  const childIds = layer.childIds ?? [];

  // Создаем уникальный ID для области обрезки
  const clipPathId = `clip-path-${id}`;

  return (
    <g>
      {/* Определяем область обрезки для дочерних элементов */}
      <defs>
        <clipPath id={clipPathId}>
          <rect
            x={layer.x}
            y={layer.y}
            width={layer.width}
            height={layer.height}
            rx={layer.cornerRadius ?? 0}
            ry={layer.cornerRadius ?? 0}
          />
        </clipPath>
      </defs>

      {/* Рендерим сам фрейм */}
      <Frame
        onPointerDown={onLayerPointerDown}
        id={id}
        layer={layer}
        canvasMode={canvasMode}
      />
```

Окончание листинга Д.1

```
            {/* Рендерим все дочерние элементы внутри области обрезки */}
            <g clipPath={`url(#${clipPathId})`}>
              {childIds.map((childId) => (
                <LayerComponent
                  key={childId}
                  id={childId}
                  onLayerPointerDown={onLayerPointerDown}
                  canvasMode={canvasMode}
                  setIsEditingText={setIsEditingText}
                  _inFrame={true}
                />
              ))}
            </g>
          </g>
        );

      // Неизвестный тип слоя - выводим предупреждение в консоль
      default:
        console.warn("Неизвестный тип слоя");
        return null;
    }
  },
);

// Устанавливаем displayName для более удобной отладки
LayerComponent.displayName = "LayerComponent";

export default LayerComponent;
```

Листинг Д.2 – Исходный код создания слоев LayerFactory.tsx

```
"use client";

/**
 * Функциональность для создания различных типов слоев на холсте
 *  Предоставляет единый интерфейс для создания всех поддерживаемых типов слоев с
необходимыми свойствами
 */

import { LiveObject } from "@liveblocks/client";
import {
  LayerType,
  type EllipseLayer,
  type RectangleLayer,
  type TextLayer,
  type TriangleLayer,
  type FrameLayer,
  type ImageLayer,
} from "~/types";

// Константы для текстовых слоев
const DEFAULT_FONT_SIZE = 16;
const DEFAULT_LINE_HEIGHT_COEFFICIENT = 1.2;
```

Продолжение листинга Д.2

```
/**
 * Создает новый слой заданного типа с указанными параметрами
 * @param layerType - Тип создаваемого слоя
 * @param x - X-координата слоя
 * @param y - Y-координата слоя
 * @param width - Ширина слоя
 * @param height - Высота слоя
 * @param textContent - Текстовое содержимое (для текстовых слоев)
 * @param isFixedSize - Флаг фиксированного размера (для текстовых слоев)
 * @param imageUrl - URL изображения (для слоев с изображениями)
 * @param aspectRatio - Соотношение сторон (для слоев с изображениями)
 * @returns LiveObject слоя или null, если тип не поддерживается
 */
export function createLayer(
  layerType: LayerType,
  x: number,
  y: number,
  width: number,
  height: number,
  textContent?: string,
  isFixedSize = false,
  imageUrl?: string,
  aspectRatio?: number,
) {
  // Базовые свойства, общие для всех типов слоев
  const baseProps = {
    x,
    y,
    width,
    height,
    opacity: 100,
    blendMode: "normal",
  };

  if (layerType === LayerType.Rectangle) {
    // Создание прямоугольника
    return new LiveObject<RectangleLayer>({
      type: LayerType.Rectangle,
      ...baseProps,
      fill: { r: 217, g: 217, b: 217 }, // Светло-серый цвет по умолчанию
      stroke: null, // Без обводки по умолчанию
    });
  } else if (layerType === LayerType.Ellipse) {
    // Создание эллипса
    return new LiveObject<EllipseLayer>({
      type: LayerType.Ellipse,
      ...baseProps,
      fill: { r: 217, g: 217, b: 217 },
      stroke: null,
    });
  } else if (layerType === LayerType.Triangle) {
    // Создание треугольника
    return new LiveObject<TriangleLayer>({
      type: LayerType.Triangle,
```

Окончание листинга Д.2

```
      ...baseProps,
      fill: { r: 217, g: 217, b: 217 },
      stroke: null,
    });
  } else if (layerType === LayerType.Text) {
    // Создание текстового блока
    return new LiveObject<TextLayer>({
      type: LayerType.Text,
      ...baseProps,
      text: textContent ?? "", // Текст по умолчанию - пустая строка
      fontSize: DEFAULT_FONT_SIZE,
      fontWeight: 400,
      fontFamily: "Inter",
      lineHeight: DEFAULT_LINE_HEIGHT_COEFFICIENT,
      letterSpacing: 0,
      stroke: null,
      fill: { r: 0, g: 0, b: 0 }, // Черный цвет текста по умолчанию
      isFixedSize,
    });
  } else if (layerType === LayerType.Frame) {
    // Создание рамки (контейнера)
    return new LiveObject<FrameLayer>({
      type: LayerType.Frame,
      ...baseProps,
      fill: { r: 255, g: 255, b: 255 }, // Белый фон по умолчанию
      stroke: null,
      strokeWidth: 1,
      cornerRadius: 0,
      childIds: [], // Пустой массив дочерних элементов
    });
  } else if (layerType === LayerType.Image && imageUrl && aspectRatio) {
    // Создание слоя с изображением
    return new LiveObject<ImageLayer>({
      type: LayerType.Image,
      ...baseProps,
      url: imageUrl,
      aspectRatio: aspectRatio,
    });
  }

  // Для неподдерживаемых типов возвращаем null
  return null;
}
```

Листинг Д.3 – Исходный код компонента прямоугольника Rectangle.tsx

```
"use client";

/**
 * Компонент для отображения прямоугольника на холсте
 * Поддерживает настройку цвета заливки, обводки, прозрачности и скругления углов
 */
```

Продолжение листинга Д.3

```
import React from "react";
import { CanvasMode, type RectangleLayer } from "~/types";
import { colorToCss } from "~/utils";
import { useCanvas } from "../helper/CanvasContext";

type RectangleProps = {
  id: string;
  layer: RectangleLayer;
  onPointerDown: (e: React.PointerEvent, layerId: string) => void;
  canvasMode: CanvasMode;
};

export default function Rectangle({
  id,
  layer,
  onPointerDown,
  canvasMode,
}: RectangleProps) {
  // Получаем camera для масштабирования обводки
  const { camera } = useCanvas();

  // Деструктуризация свойств из объекта слоя
  const {
    x,
    y,
    width,
    height,
    fill,
    stroke,
    opacity,
    cornerRadius,
    blendMode,
    strokeWidth,
  } = layer;

  // Получаем CSS-представление цветов с учетом их непрозрачности
  const fillColor = fill ? colorToCss(fill) : "none";
  const strokeColor = stroke ? colorToCss(stroke) : "none";

  // Толщина обводки с учетом масштаба
  const scaledStrokeWidth = stroke ? (strokeWidth ?? 1) / camera.zoom : 0;
  // Толщина обводки для выделения/наведения
  const outlineStrokeWidth = 3 / camera.zoom;

  // Определяем классы для контура при наведении и выделении
  const outlineClass = [
    CanvasMode.None,
    CanvasMode.RightClick,
    CanvasMode.SelectionNet,
    CanvasMode.Translating,
    CanvasMode.Pressing,
  ].includes(canvasMode)
    ? "pointer-events-none opacity-0 group-hover:opacity-100"
    : "pointer-events-none opacity-0";
```

Окончание листинга Д.3

```
  return (
    <g className="group">
      {/* Контур прямоугольника при выделении или наведении */}
      <rect
        style={{ transform: `translate(${x}px, ${y}px)` }}
        width={width}
        height={height}
        fill="none"
        stroke="#4183ff"
        strokeWidth={outlineStrokeWidth}
        className={outlineClass}
        rx={cornerRadius ?? 0}
        ry={cornerRadius ?? 0}
      />

      {/* Фигура прямоугольника */}
      <rect
        onPointerDown={(e) => onPointerDown(e, id)}
        style={{
          transform: `translate(${x}px, ${y}px)`,
          mixBlendMode:
            (blendMode as React.CSSProperties["mixBlendMode"]) ?? "normal",
          opacity: `${opacity ?? 100}%`,
        }}
        width={width}
        height={height}
        fill={fillColor}
        stroke={strokeColor}
        strokeWidth={scaledStrokeWidth}
        rx={cornerRadius ?? 0}
        ry={cornerRadius ?? 0}
      />
    </g>
  );
}
```

Листинг Д.4 – Исходный код создания слоев ShapeDrawingFunctions.tsx

```
"use client";

/**
 * Функциональность для создания и добавления новых слоев на холст
 * Обеспечивает создание различных типов слоев как с помощью клика, так и перетаскивания
 */

import { useMutation } from "@liveblocks/react";
import { type LiveObject } from "@liveblocks/client";
import { nanoid } from "nanoid";
import { CanvasMode, LayerType, type Point, type Layer, type ImageDataEvent } from
"~/types";
import { useCanvas } from "~/components/canvas/helper/CanvasContext";
import { createLayer } from "~/components/canvas/helper/LayerFactory";
import { useEffect, useRef } from "react";
```

Продолжение листинга Д.4

```
export function useCreateLayerFunctions() {
  const { canvasState, setState, MAX_LAYERS } = useCanvas();

  // Хранилище данных изображения
  const imageDataRef = useRef<ImageDataEvent | null>(null);

  // Обработчик события для получения данных изображения
  useEffect(() => {
    const handleImageDataReady = (event: CustomEvent<ImageDataEvent>) => {
      imageDataRef.current = event.detail;
    };

    window.addEventListener(
      "imageDataReady",
      handleImageDataReady as EventListener,
    );

    return () => {
      window.removeEventListener(
        "imageDataReady",
        handleImageDataReady as EventListener,
      );
    };
  }, []);

  // Вставка слоя (фигуры или текста) одним кликом
  const insertLayerByClick = useMutation(
    ({ storage, setMyPresence }, point: Point, layerType: LayerType) => {
      const liveLayers = storage.get("layers");
      // Проверка на максимальное количество слоев
      if (liveLayers.size >= MAX_LAYERS) return;

      const layerId = nanoid();
      let layer: LiveObject<Layer> | null;

      if (layerType === LayerType.Text) {
        // Создание пустого текстового блока с минимальными размерами
        layer = createLayer(layerType, point.x, point.y, 10, 20, "", false);

        // Устанавливаем слой в режим редактирования сразу после создания
        if (layer) {
          storage.get("layerIds").push(layerId);
          liveLayers.set(layerId, layer);
          setMyPresence({ selection: [layerId] }, { addToHistory: true });
          setState({ mode: CanvasMode.None });
        }
      } else if (layerType === LayerType.Frame) {
        // Рамки создаются с фиксированным размером
        const size = 200;
        layer = createLayer(
          layerType,
          point.x - size / 2,
          point.y - size / 2,
          size,
```

Продолжение листинга Д.4

```
    size,
  );

  if (layer) {
    storage.get("layerIds").push(layerId);
    liveLayers.set(layerId, layer);
    setMyPresence({ selection: [layerId] }, { addToHistory: true });
    setState({ mode: CanvasMode.None });
  }
} else if (layerType === LayerType.Image) {
  // Получаем данные изображения из useRef
  const imageData = imageDataRef.current;

  if (imageData) {
    const { width, height, url, aspectRatio } = imageData;

    // Создаем слой изображения и центрируем его относительно точки клика
    layer = createLayer(
      layerType,
      point.x - width / 2,
      point.y - height / 2,
      width,
      height,
      undefined,
      false,
      url,
      aspectRatio,
    );

    if (layer) {
      storage.get("layerIds").push(layerId);
      liveLayers.set(layerId, layer);
      setMyPresence({ selection: [layerId] }, { addToHistory: true });
      setState({ mode: CanvasMode.None });

      // Очищаем данные изображения после использования
      imageDataRef.current = null;
    }
  } else {
    console.error("Не найдены данные изображения для создания слоя");
  }
} else {
  // Фигуры создаются с фиксированным размером
  const size = 100;
  layer = createLayer(
    layerType,
    point.x - size / 2,
    point.y - size / 2,
    size,
    size,
  );

  if (layer) {
    storage.get("layerIds").push(layerId);
```

Продолжение листинга Д.4

```
        liveLayers.set(layerId, layer);
        setMyPresence({ selection: [layerId] }, { addToHistory: true });
        setState({ mode: CanvasMode.None });
      }
    }
  },
  [],
);

// Завершение рисования фигуры или текста (тянущийся клик)
const insertLayerByDragging = useMutation(
  (
    { storage, setMyPresence },
    position: { x: number; y: number; width: number; height: number },
  ) => {
    if (
      canvasState.mode !== CanvasMode.CreatingShape ||
      !canvasState.origin ||
      !canvasState.current
    ) {
      return;
    }

    const liveLayers = storage.get("layers");
    if (liveLayers.size >= MAX_LAYERS) return;

    const layerId = nanoid();
    let layer: LiveObject<Layer> | null;

    if (canvasState.layerType === LayerType.Text) {
      // Текстовый блок с фиксированной шириной и переносом строк
      layer = createLayer(
        canvasState.layerType,
        position.x,
        position.y,
        10,
        20,
        "",
        false,
      );
    } else if (canvasState.layerType === LayerType.Image) {
      // Для изображений мы используем только вставку одиночным кликом
      // При попытке вставить изображение перетаскиванием - отменяем операцию
      setState({ mode: CanvasMode.None });
      imageDataRef.current = null; // Очищаем данные изображения
      return;
    } else {
      // Обработка рамок и других фигур
      layer = createLayer(
        canvasState.layerType,
        position.x,
        position.y,
        position.width,
        position.height,
```

Окончание листинга Д.4

```
        );
      }

      if (layer) {
        storage.get("layerIds").push(layerId);
        liveLayers.set(layerId, layer);
        setMyPresence({ selection: [layerId] }, { addToHistory: true });
      }

      setState({ mode: CanvasMode.None });
    },
    [canvasState, setState],
  );

  return {
    insertLayerByClick,
    insertLayerByDragging,
  };
}
```

Листинг Д.5 – Исходный код функций манипуляции слоями LayerManipulation.tsx

```
"use client";

/**
 * Функциональность для манипуляции слоями на холсте
 * Предоставляет инструменты для изменения размера, перемещения и взаимодействия с рамками
 */

import { useCallback, useEffect } from "react";
import {
  CanvasMode,
  LayerType,
  type Point,
  type Side,
  type XYWH,
  type FrameLayer,
} from "~/types";
import { useMutation } from "@liveblocks/react";
import { resizeBounds } from "~/utils";
import { useCanvas } from "~/components/canvas/helper/CanvasContext";

export function useLayerManipulation() {
  const { canvasState, setState, history } = useCanvas();

  // Проверка, находится ли точка внутри слоя
  const isPointInLayer = (point: Point, layer: { x: number; y: number; width: number;
height: number }) => {
    return (
      point.x >= layer.x &&
```

Продолжение листинга Д.5

```
    point.x <= layer.x + layer.width &&
    point.y >= layer.y &&
    point.y <= layer.y + layer.height
  );
};

// Обработка нажатия и отпускания клавиши Shift во время изменения размера
useEffect(() => {
  const handleKeyDown = (e: KeyboardEvent) => {
    if (canvasState.mode === CanvasMode.Resizing && e.key === "Shift") {
      setState((prev) => ({
        ...prev,
        isShiftPressed: true,
      }));
    }
  };
  const handleKeyUp = (e: KeyboardEvent) => {
    if (canvasState.mode === CanvasMode.Resizing && e.key === "Shift") {
      setState((prev) => ({
        ...prev,
        isShiftPressed: false,
      }));
    }
  };

  // Добавляем обработчики событий для клавиш
  window.addEventListener("keydown", handleKeyDown);
  window.addEventListener("keyup", handleKeyUp);

  // Убираем обработчики при размонтировании компонента
  return () => {
    window.removeEventListener("keydown", handleKeyDown);
    window.removeEventListener("keyup", handleKeyUp);
  };
}, [canvasState.mode, setState]);

// Обработчик нажатий на маркеры изменения размера
const onResizeHandlePointerDown = useCallback(
  (corner: Side, initialBounds: XYWH) => {
    history.pause(); // Приостанавливаем запись истории

    // Устанавливаем режим изменения размера
    setState((prev) => ({
      ...prev,
      mode: CanvasMode.Resizing,
      initialBounds, // Запоминаем начальные границы элемента
      corner, // Запоминаем угол, с которого началось изменение размера
      isShiftPressed: false, // Изначально Shift не нажат
    }));
  },
  [setState, history],
);

// Функция изменения размера выбранного слоя
```

Продолжение листинга Д.5

```
const resizeSelectedLayer = useMutation(
  ({ storage, self }, point: Point) => {
    // Проверяем, что режим - изменение размера
    if (canvasState.mode !== CanvasMode.Resizing) {
      return;
    }

    // Вычисляем новые границы слоя с учетом начальных границ, угла и положения курсора
    const bounds = resizeBounds(
      canvasState.initialBounds,
      canvasState.corner,
      point,
      canvasState.isShiftPressed,
    );

    const liveLayers = storage.get("layers"); // Получаем слои из хранилища

    // Обновляем границы выбранного слоя, если он существует и выделен
    if (self.presence.selection.length > 0) {
      const layer = liveLayers.get(self.presence.selection[0]!);
      if (layer) {
        layer.update(bounds); // Обновляем границы слоя
      }
    }
  },
  [canvasState],
);

// Функция перемещения выделенных слоев
const translateSelectedLayers = useMutation(
  ({ storage, self }, point: Point) => {
    // Проверка, что режим - перемещение
    if (canvasState.mode !== CanvasMode.Translating) {
      return;
    }

    // Вычисляем смещение от предыдущей до текущей позиции
    const offset = {
      x: point.x - canvasState.current.x,
      y: point.y - canvasState.current.y,
    };

    const liveLayers = storage.get("layers");
    const layerIds = storage.get("layerIds");

    // Сохраняем информацию о слоях, которые находятся внутри фреймов
    // для последующей проверки, нужно ли их выносить из фрейма
    const layersToMoveOutOfFrames = new Map();

    // Проверяем для каждого выбранного слоя, находится ли он внутри какого-либо фрейма
    for (const id of self.presence.selection) {
      const layer = liveLayers.get(id);
      if (!layer) continue;
```

Продолжение листинга Д.5

```
  // Проверяем все фреймы, чтобы найти родительский для данного слоя
  let parentFrameId = null;
  for (const potentialParentId of layerIds) {
    const potentialParent = liveLayers.get(potentialParentId);
    if (
      potentialParent &&
      potentialParent.get("type") === LayerType.Frame
    ) {
      const frameData = potentialParent.toObject() as FrameLayer;
      const childIds = frameData.childIds ?? [];

      // Если ID слоя есть в списке дочерних фрейма, то это родительский фрейм
      if (childIds.includes(id)) {
        parentFrameId = potentialParentId;
        break;
      }
    }
  }

  // Если слой находится в фрейме, сохраняем эту информацию для последующей обработки
  if (parentFrameId) {layersToMoveOutOfFrames.set(id, parentFrameId);
  }
}

// Перемещаем выбранные слои и их дочерние элементы (если это фреймы)
for (const id of self.presence.selection) {
  const layer = liveLayers.get(id);
  if (layer) {
    const currentX = layer.get("x");
    const currentY = layer.get("y");

    // Обновляем позицию слоя
    layer.update({
      x: currentX + offset.x,
      y: currentY + offset.y,
    });

    // Если слой является фреймом, необходимо переместить все его дочерние элементы
    if (layer.get("type") === LayerType.Frame) {
      const frameData = layer.toObject() as FrameLayer;
      const childIds = frameData.childIds ?? [];

      // Перемещаем каждый дочерний элемент на то же смещение
      for (const childId of childIds) {
        const childLayer = liveLayers.get(childId);
        if (childLayer) {
          const childX = childLayer.get("x");
          const childY = childLayer.get("y");

          childLayer.update({
            x: childX + offset.x,
            y: childY + offset.y,
          });
        }
```

Продолжение листинга Д.5

```
      }
    }
  }
}

// Проверяем, находится ли перетаскиваемый слой над фреймом
let isOverFrame = false;
let frameLayerId: string | null = null;

// Перебираем все слои, чтобы найти фреймы
for (const id of layerIds) {
  const layer = liveLayers.get(id);
  // Если это фрейм и он не выбран в данный момент
  if (
    layer &&
    layer.get("type") === LayerType.Frame &&
    !self.presence.selection.includes(id)
  ) {
    // Проверяем для каждого выбранного слоя, находится ли его центр над фреймом
    for (const selectedId of self.presence.selection) {
      const selectedLayer = liveLayers.get(selectedId);
      if (selectedLayer) {
        // Вычисляем центр слоя после перемещения
        const centerX =
          selectedLayer.get("x") +
          offset.x +
          selectedLayer.get("width") / 2;
        const centerY =
          selectedLayer.get("y") +
          offset.y +
          selectedLayer.get("height") / 2;

        // Проверяем, находится ли центр слоя внутри фрейма
        if (
          isPointInLayer(
            { x: centerX, y: centerY },
            {
              x: layer.get("x"),
              y: layer.get("y"),
              width: layer.get("width"),
              height: layer.get("height"),
            },
          )
        ) {
          isOverFrame = true;
          frameLayerId = id;
          break;
        }
      }
    }
    if (isOverFrame) break;
  }
}
```

Продолжение листинга Д.5

```
// Обработка перемещения слоев из фрейма наружу
for (const [
  layerId,
  parentFrameId,
] of layersToMoveOutOfFrames.entries()) {
  const layer = liveLayers.get(layerId as string);
  const frameLayer = liveLayers.get(parentFrameId as string);
  if (!layer || !frameLayer) continue;

  // Проверяем, находится ли центр слоя за пределами родительского фрейма
  const centerX = layer.get("x") + layer.get("width") / 2;
  const centerY = layer.get("y") + layer.get("height") / 2;

  // Проверяем, вышел ли слой за пределы родительского фрейма
  const isOutsideParentFrame = !isPointInLayer(
    { x: centerX, y: centerY },
    {
      x: frameLayer.get("x"),
      y: frameLayer.get("y"),
      width: frameLayer.get("width"),
      height: frameLayer.get("height"),
    },
  );

  // Если слой находится за пределами фрейма и не находится над другим фреймом
  // или если слой находится над другим фреймом, но не над родительским
  if (
    isOutsideParentFrame &&
    (!isOverFrame || frameLayerId !== parentFrameId)
  ) {
    // Получаем данные родительского фрейма
    const frameData = frameLayer.toObject() as FrameLayer;
    const childIds = [...(frameData.childIds ?? [])];

    // Удаляем ID слоя из дочерних элементов родительского фрейма
    const index = childIds.indexOf(layerId as string);
    if (index !== -1) {
      childIds.splice(index, 1);

      // Обновляем список дочерних элементов фрейма
      frameLayer.update({
        childIds: childIds,
      });

      // Добавляем слой в список слоев, если он не будет добавлен в другой фрейм
      if (!isOverFrame || frameLayerId !== parentFrameId) {
        // Проверяем, есть ли уже этот слой в списке верхнеуровневых слоев
        const layerIdsArray = layerIds.toArray();
        if (!layerIdsArray.includes(layerId as string)) {
          // Добавляем слой в конец списка слоев верхнего уровня
          layerIds.push(layerId as string);
        }
      }
    }
```

Окончание листинга Д.5

```
        }
      }

      // Если слой перемещается над фреймом, добавляем его в дочерние элементы
      if (isOverFrame && frameLayerId) {
        for (const id of self.presence.selection) {
          const selectedLayer = liveLayers.get(id);
          if (!selectedLayer) continue;

          // Проверяем, что это не фрейм (фреймы нельзя вкладывать в другие фреймы)
          const selectedLayerType = selectedLayer.get("type");

          if (selectedLayerType !== LayerType.Frame) {
            const frameLayer = liveLayers.get(frameLayerId);

            if (!frameLayer) continue;

            // Проверяем, что это действительно фрейм
            const frameLayerType = frameLayer.get("type");

            if (frameLayerType === LayerType.Frame) {
              // Получаем объект фрейма, чтобы работать с массивом childIds
              const frameData = frameLayer.toObject() as FrameLayer;
              const childIds = frameData.childIds ?? [];

              // Добавляем ID слоя в массив дочерних элементов, если его там нет
              if (!childIds.includes(id)) {
                // Обновляем фрейм с новым массивом дочерних элементов
                frameLayer.update({
                  childIds: [...childIds, id] as string[],
                });

                // Удаляем слой из списка слоев верхнего уровня, так как теперь он дочерний
                const index = layerIds.indexOf(id);
                if (index !== -1) {
                  layerIds.delete(index);
                }
              }
            }
          }
        }
      }
      // Обновляем текущую позицию в состоянии холста
      setState({ mode: CanvasMode.Translating, current: point });
    },
    [canvasState],
  );

  return {
    onResizeHandlePointerDown,
    resizeSelectedLayer,
    translateSelectedLayers,
  };
}
```

ПРИЛОЖЕНИЕ Е
ИСХОДНЫЙ КОД МОДУЛЯ ИНТЕРФЕЙСА

Листинг Е.1 – Исходный код холста интерфейса UICanvas.tsx

```
"use client";

/**
 * Компонент для отображения интерфейса редактора холста. Отвечает за управление
 * отображением левой и правой боковых панелей, обработку состояния свёрнутых/развёрнутых
 * панелей, передачу данных о выбранных слоях между компонентами, а также синхронизацию
*/

import React from "react";
import { useSelf, useStorage } from "@liveblocks/react";
import type { UserInfo } from "../../types/user";
import type { Layer } from "~/types";
import { RightSidebar, MinimizedRightSidebar } from "./RightSidebar";
import { LeftSidebar, MinimizedLeftSidebar } from "./LeftSidebar";
import TopPanel from "./TopPanel";

export default function Sidebar({
  leftIsMinimized,
  setLeftIsMinimized,
  roomName,
  roomId,
  othersWithAccessToRoom,
  owner,
}: {
  leftIsMinimized: boolean;
  setLeftIsMinimized: (value: boolean) => void;
  roomName: string;
  roomId: string;
  othersWithAccessToRoom: UserInfo[];
  owner: UserInfo;
}) {
  const me = useSelf();
  const isOwner = me?.info.name === owner.email;

  // Получаем выбранный слой, если он есть
  const selectedLayer = useSelf((me) => {
    const selection = me.presence.selection;
    return selection.length === 1 ? selection[0] : null;
  });

  // Получаем объект слоя из хранилища, если выбран слой
  const layer = useStorage((root) => {
    if (!selectedLayer) {
      return null;
    }
    return root.layers.get(selectedLayer) ?? null;
  }) as Layer | null;

  return (
    <>
```

Окончание листинга Е.1

```
      {/* Верхний панель */}
      {!leftIsMinimized && (
        <TopPanel
          roomName={roomName}
          roomId={roomId}
          othersWithAccessToRoom={othersWithAccessToRoom}
          owner={owner}
        />
      )}

      {/* Левая панель */}
      {!leftIsMinimized ? (
        <LeftSidebar
          setLeftIsMinimized={setLeftIsMinimized}
        />
      ) : (
        <MinimizedLeftSidebar
          setLeftIsMinimized={setLeftIsMinimized}
          roomName={roomName}
        />
      )}

      {/* Правая панель */}
      {!leftIsMinimized || layer ? (
        <RightSidebar
          leftIsMinimized={leftIsMinimized}
          layer={layer}
          _roomId={roomId}
          _othersWithAccessToRoom={othersWithAccessToRoom}
          _owner={owner}
          _isOwner={isOwner}
        />
      ) : (
        <MinimizedRightSidebar
          roomId={roomId}
          othersWithAccessToRoom={othersWithAccessToRoom}
          owner={owner}
          isOwner={isOwner}
        />
      )}
    </>
  );
}
```

Листинг Е.2 – Исходный код верхней панели TopPanel.tsx

```
/**
 * Компонент верхней панели. Отвечает за отображение логотипа, названия проекта,
 * списка активных пользователей в комнате и меню для совместного доступа.
 * Позволяет навигацию к дашборду и управление правами доступа к проекту.
 */

import React from "react";
```

Продолжение листинга Е.2

```
import Link from "next/link";
import Image from "next/image";
import { useSelf, useOthers } from "@liveblocks/react";
import { RiArrowDownSLine } from "@remixicon/react";
import { connectionIdToColor } from "~/utils";
import UserAvatar from "./UserAvatar";
import ShareMenu from "./ShareMenu";
import type { UserInfo } from "~/types/user";
import {Root as AvatarGroup, Overflow as AvatarGroupOverflow} from "~/components/ui/avatar-
group";

interface TopPanelProps {
  roomName: string;
  roomId: string;
  othersWithAccessToRoom: UserInfo[];
  owner: UserInfo;
}

export const TopPanel: React.FC<TopPanelProps> = ({
  roomName,
  roomId,
  othersWithAccessToRoom,
  owner,
}) => {
  const me = useSelf();
  const others = useOthers();
  const isOwner = me?.info.name === owner.email;

  return (
    <div  className="fixed  relative  left-0  top-0  flex  h-[48px]  w-full  select-none  items-
center justify-between border-b border-stroke-soft-200 bg-bg-white-0 px-4 py-2">
      {/* Левая часть панели с логотипом и навигацией */}
      <div className="flex flex-row items-center gap-2">
        <Image
          src="/icon/ridex-logo.svg"
          alt="RideX"
          width={32}
          height={32}
          className="h-[32px] w-[32px]"
        />
        <div className="flex items-center gap-2 text-paragraph-sm">
          <Link
            href="/dashboard"
            className="text-text-sub-600 hover:text-text-strong-950"
          >
            Черновик
          </Link>
          <span className="text-text-sub-600">/</span>

          {/* Кнопка с названием текущего проекта */}
          <button className="flex items-center gap-1">
            <span className="text-text-strong-950">{roomName}</span>
            <RiArrowDownSLine  className="h-4  w-4  text-text-strong-950  transition-transform
duration-200 ease-out hover:translate-y-[2px]" />
```

Окончание листинга Е.2

```
            </button>
          </div>
        </div>

        {/* Правая часть панели с аватарами пользователей и меню доступа */}
        <div className="flex flex-row items-center gap-3">
          <AvatarGroup size="32">
            {me && (
              <UserAvatar
                color={connectionIdToColor(me.connectionId)}
                name={me.info.name}
                image={me.info.image}
              />
            )}
            {/* Отображение аватаров других пользователей с ограничением */}
            {others.length > 2 ? (
              <>
                {others.slice(0, 2).map((other) => (
                  <UserAvatar
                    key={other.connectionId}
                    color={connectionIdToColor(other.connectionId)}
                    name={other.info.name}
                    image={other.info.image}
                  />
                ))}
                {/* Счетчик дополнительных пользователей */}
                <AvatarGroupOverflow>
                  +{others.length - 2}
                </AvatarGroupOverflow>
              </>
            ) : (
              others.map((other) => (
                <UserAvatar
                  key={other.connectionId}
                  color={connectionIdToColor(other.connectionId)}
                  name={other.info.name}
                  image={other.info.image}
                />
              ))
            )}
          </AvatarGroup>
          {/* Меню для управления доступом к проекту */}
          <ShareMenu
            roomId={roomId}
            othersWithAccessToRoom={othersWithAccessToRoom}
            owner={owner}
            isOwner={isOwner}
          />
        </div>
      </div>
    );
};
export default TopPanel;
```

Листинг Е.3 – Исходный код левой боковой панели LeftSidebar.tsx

```
/**
 * Левая боковая панель приложения отвечает за отображение иерархии слоев проекта,
 * переключение между режимами просмотра слоев и шаблонов.
*/

import React, { useState } from "react";
import * as Divider from "~/components/ui/divider";
import ModeButton from "~/components/ui/mode-button";
import { Minimized_icon } from "~/icon";
import { useStorage, useSelf } from "@liveblocks/react";
import AllLayersTree from "~/components/canvas/LayerTree";
import { RiArrowDownSLine } from "@remixicon/react";
import TemplatesTab from "./TemplatesTab";
import Image from "next/image";

interface LeftSidebarProps {setLeftIsMinimized: (value: boolean) => void;}

export const LeftSidebar: React.FC<LeftSidebarProps> = ({
  setLeftIsMinimized,
}) => {
  // Состояние для переключения между режимами: "слои" и "шаблоны"
  const [layerMode, setLayerMode] = useState("layers_mode");
  // Получаем список слоев из Liveblocks-хранилища
  const layers = useStorage((root) => root.layers);
  // Получаем текущее выделение пользователя
  const selection = useSelf((me) => me.presence.selection);

  return (
    // Основной контейнер левой панели с фиксированным позиционированием
    <div  className="fixed  left-0  top-[48px]  flex  h-screen  w-[260px]  select-none  flex-col
border-r border-stroke-soft-200 bg-bg-white-0">
      <div className="p-4 pb-2">
        <div className="flex items-center justify-between">
          <div className="flex flex-row gap-1">
            {/* Кнопка переключения в режим слоев */}
            <ModeButton
              onSelect={() => setLayerMode("layers_mode")}
              active={layerMode === "layers_mode"}
              text="Слои"
            />

            {/* Кнопка переключения в режим шаблонов */}
            <ModeButton
              onSelect={() => setLayerMode("templates")}
              active={layerMode === "templates"}
              text="Шаблоны"
            />
          </div>

          {/* Кнопка для сворачивания панели */}
          <div className="cursor-pointer rounded-md p-1 hover:bg-bg-weak-50">
            <Minimized_icon onClick={() => setLeftIsMinimized(true)} />
          </div>
        </div>
```

Продолжение листинга Е.3

```
      </div>

      <div className="w-full max-w-96 px-4">
        <Divider.Root />
      </div>

      {/* Отображение дерева слоев или вкладки шаблонов в зависимости от режима */}
      {layerMode === "layers_mode" && (
        <div className="flex flex-col gap-1 p-4 pt-2 h-full overflow-hidden hover:overflow-
y-auto pr-2 [&::-webkit-scrollbar]:w-1 [&::-webkit-scrollbar-track]:bg-transparent [&::-
webkit-scrollbar-thumb]:rounded-full [&::-webkit-scrollbar-thumb]:bg-stroke-soft-200
hover:[&::-webkit-scrollbar-thumb]:bg-stroke-soft-300">
          {/* Дерево слоев отображается только если есть данные о слоях и выделении */}
          {layers && selection && (
            <AllLayersTree
              layers={layers}
              selection={selection || []}
            />
          )}
        </div>
      )}

      {layerMode === "templates" && (
        <div className="h-full overflow-hidden">
          <TemplatesTab />
        </div>
      )}
    </div>
  );
};

// Компонент свернутой левой панели: отображает логотип и название комнаты, а также кнопку
для разворачивания панели
export const MinimizedLeftSidebar: React.FC<{
  setLeftIsMinimized: (value: boolean) => void;
  roomName: string;
}> = ({ setLeftIsMinimized, roomName }) => {
  return (
    <div className="fixed left-3 top-3 flex h-[48px] w-[280px] select-none items-center
justify-between rounded-xl border bg-white p-4 shadow-regular-sm">
      <div className="flex items-center gap-2">
        {/* Логотип приложения */}
        <Image
          src="/icon/ridex-logo.svg"
          alt="RideX"
          width={32}
          height={32}
          className="h-[32px] w-[32px]"
        />
        {/* Название комнаты с иконкой раскрытия */}
        <button className="flex items-center gap-1 text-paragraph-sm">
          <span className="text-text-strong-950">{roomName}</span>
          <RiArrowDownSLine className="h-4 w-4 text-text-strong-950 transition-transform
duration-200 ease-out hover:translate-y-[2px]" />
```

Окончание листинга Е.3

```
        </button>
      </div>

      {/* Кнопка для разворачивания панели */}
      <div className="cursor-pointer rounded-md p-1 hover:bg-bg-weak-50">
        <Minimized_icon onClick={() => setLeftIsMinimized(false)} />
      </div>
    </div>
  );
};

export default LeftSidebar;
```

Листинг Е.4 – Исходный код дерева слоев LayerTree.tsx

```
"use client";

/**
 * Компонент для отображения иерархической структуры слоев на холсте.
 * Поддерживает вложенные слои, различные типы элементов (фреймы, текст, изображения),
 * а также интерактивное управление (выбор, разворачивание/сворачивание).
 * Обеспечивает синхронизацию с другими пользователями через Liveblocks.
 */

import React, { useState } from "react";
import { useStorage, useRoom } from "@liveblocks/react";
import type { LiveMap, LiveObject } from "@liveblocks/client";
import { Rectangle_16, Ellipse_16, Text_16, Frame_16, Image_16 } from "~/icon";
import LayerButton from "~/components/ui/layer-button";
import { LayerType } from "~/types";
import type { FrameLayer, Layer } from "~/types";
import { generateLayerName } from "~/utils";

// Тип для хранилища слоев, который может быть LiveMap или обычным Map
export type LayersStorage =
  | LiveMap<string, LiveObject<Layer>>
  | Map<string, Layer>
  | ReadonlyMap<string, Layer>
  | ReadonlyMap<string, LiveObject<Layer>>;

// Пропсы для компонента RenderLayersList
interface RenderLayersListProps {
  layerIds: readonly string[];      // Массив ID слоев
  layers: LayersStorage;            // Хранилище слоев
  selection: readonly string[];     // Массив выбранных слоев
  level?: number;                   // Уровень вложенности
  processedIds?: Set<string>;       // Множество обработанных ID
  roomId: string;                   // ID комнаты
}

// Рекурсивный компонент для отображения слоев с поддержкой вложенности
export const RenderLayersList: React.FC<RenderLayersListProps> = ({
```

Продолжение листинга Е.4

```
  layerIds,
  layers,
  selection,
  level = 0,
  processedIds = new Set<string>(),
  roomId,
}) => {
  // Состояние для отслеживания развернутых слоев
  const [expandedLayers, setExpandedLayers] = useState<Set<string>>(() => {
    // Инициализация: все фреймы развернуты по умолчанию
    const initialExpanded = new Set<string>();

    // Вспомогательная функция для поиска всех фреймов и их разворачивания
    const findFramesRecursive = (
      ids: readonly string[],
      processed = new Set<string>(),
    ) => {
      ids.forEach((id) => {
        if (processed.has(id)) return;
        processed.add(id);

        const layer = layers?.get(id);
        if (!layer) return;

        // Проверяем тип слоя
        const layerType = "get" in layer ? layer.get("type") : layer.type;

        if (layerType === LayerType.Frame) {
          // Добавляем фрейм в множество развернутых
          initialExpanded.add(id);

          // Получаем дочерние ID и обрабатываем их (для вложенных фреймов)
          const childIds =
            "get" in layer
              ? (layer.toObject() as FrameLayer).childIds ?? []
              : (layer as FrameLayer).childIds ?? [];

          findFramesRecursive(childIds, processed);
        }
      });
    };

    // Начинаем с верхнеуровневых слоев
    findFramesRecursive(layerIds);

    return initialExpanded;
  });

  // Функция для переключения состояния развернутости слоя
  const toggleLayerExpanded = (layerId: string) => {
    setExpandedLayers((prevExpanded) => {
      const newExpanded = new Set(prevExpanded);
      if (newExpanded.has(layerId)) {
        newExpanded.delete(layerId);
```

Продолжение листинга Е.4

```
    } else {
      newExpanded.add(layerId);
    }
    return newExpanded;
  });
};

// Отображаем слои в обратном порядке (новые сверху) только для верхнего уровня
const displayLayerIds = level === 0 ? [...layerIds].reverse() : layerIds;

return (
  <div className="flex flex-col gap-1" style={{ paddingLeft: level * 16 }}>
    {displayLayerIds.map((id) => {
      // Пропускаем уже обработанные ID для избежания дублирования
      if (processedIds.has(id)) {
        const layer = layers?.get(id);
        if (!layer) return null;

        const isLiveObject = "get" in layer;
        const layerType = isLiveObject ? layer.get("type") : layer.type;

        // Если это не фрейм, пропускаем
        if (layerType !== LayerType.Frame) {
          return null;
        }
      }

      // Отмечаем как обработанный для избежания дубликатов
      const localProcessedIds = new Set(processedIds);
      localProcessedIds.add(id);

      const layer = layers?.get(id);
      if (!layer) return null;

      // Обработка свойств для прямых объектов и LiveObjects
      const isLiveObject = "get" in layer;
      const layerType = isLiveObject ? layer.get("type") : layer.type;

      // Получаем дочерние элементы, если это фрейм
      let childIds: readonly string[] = [];
      if (layerType === LayerType.Frame) {
        if (isLiveObject) {
          const frameData = layer.toObject() as FrameLayer;
          childIds = frameData.childIds ?? [];
        } else {
          childIds = (layer as FrameLayer).childIds ?? [];
        }
      }

      const hasChildren = childIds.length > 0;
      const isExpanded = expandedLayers.has(id);
      const isSelected = selection?.includes(id);

      let icon;
```

Продолжение листинга Е.4

```
let layerName = "";

// Генерируем имя слоя на основе его типа и ID
if (layerType !== undefined) {
  layerName = generateLayerName(id, layerType, roomId);
}

// Определяем иконку в зависимости от типа слоя
if (layerType === LayerType.Rectangle) {
  icon = <Rectangle_16 className="color-icon-strong-950" />;
} else if (layerType === LayerType.Ellipse) {
  icon = <Ellipse_16 className="color-icon-strong-950" />;
} else if (layerType === LayerType.Path) {
  icon = (
    <svg
      width="16"
      height="16"
      viewBox="0 0 16 16"
      className="text-text-strong-950"
    >
      <path
        d={((): string => {
          // Проверяем, является ли слой PathLayer с точками
          if (isLiveObject) {
            const layerObj = layer.toObject();
            if (
              "points" in layerObj &&
              Array.isArray(layerObj.points)
            ) {
              const points = layerObj.points;
              const width = layer.get("width");
              const height = layer.get("height");
              return points
                .map((point: number[], index: number) => {
                  const [x = 0, y = 0] = (point.length >= 2 ? point : [0, 0]) as
[number, number];

                  const scaledX = (x / (width ?? 1)) * 10 + 3;
                  const scaledY = (y / (height ?? 1)) * 10 + 3;
                  return `${index === 0 ? "M" : "L"} ${scaledX} ${scaledY}`;
                })
                .join(" ");
            }
          } else if ("points" in layer && Array.isArray(layer.points)) {
            const points = layer.points;
            const width = layer.width;
            const height = layer.height;
            return points
              .map((point: number[], index: number) => {
                const [x = 0, y = 0] = (point.length >= 2 ? point : [0, 0]) as
[number, number];

                const scaledX = (x / (width ?? 1)) * 10 + 3;
                const scaledY = (y / (height ?? 1)) * 10 + 3;
                return `${index === 0 ? "M" : "L"} ${scaledX} ${scaledY}`;
              })
```

Продолжение листинга Е.4

```
              .join(" ");
          }
          return "M0 0"; // Пустой путь по умолчанию
        })()}
        stroke="currentColor"
        fill="none"
        strokeWidth="1"
      />
    </svg>
  );
} else if (layerType === LayerType.Text) {
  icon = <Text_16 className="color-icon-strong-950" />;

  // Безопасно получаем текстовое содержимое с проверкой типов
  const textContent = ((): string => {
    if (isLiveObject) {
      const layerObj = layer.toObject();
      return "text" in layerObj ? layerObj.text : "";
    } else if ("text" in layer) {
      return layer.text;
    }
    return "";
  })();

  // Используем текст как название, если он не пустой
  if (textContent.length > 0) {
    const singleLineText = textContent.split("\n").join(" ");
    if (singleLineText.length <= 20) {
      layerName = singleLineText;
    } else {
      const maxChars = 20;
      layerName = singleLineText.length > maxChars
        ? singleLineText.slice(0, maxChars) + "..."
        : singleLineText;
    }
  }
} else if (layerType === LayerType.Frame) {
  icon = <Frame_16 className="text-white bg-primary-base rounded-[4px]" />;
} else if (layerType === LayerType.Image) {
  icon = <Image_16 className="color-icon-strong-950" />;

  // Для слоя изображения используем короткое название с расширением
  layerName = layerName.replace('Image', 'image.jpg');
}

return (
  <React.Fragment key={id}>
    <LayerButton
      layerId={id}
      text={layerName}
      isSelected={isSelected ?? false}
      icon={icon}
      hasChildren={hasChildren}
      isExpanded={isExpanded}
```

Окончание листинга Е.4

```
            onToggleExpand={() => toggleLayerExpanded(id)}
          />

          {/* Рендерим дочерние элементы только если слой имеет детей и развернут */}
          {hasChildren && isExpanded && (
            <RenderLayersList
              layerIds={childIds}
              layers={layers}
              selection={selection}
              level={level + 1}
              processedIds={localProcessedIds}
              roomId={roomId}
            />
          )}
        </React.Fragment>
      );
    })}
  </div>
);
};

// Пропсы для компонента AllLayersTree
interface AllLayersTreeProps {
  layers: LayersStorage;            // Хранилище слоев
  selection: readonly string[];     // Массив выбранных слоев
}

// Компонент AllLayersTree для отображения всех слоев, включая вложенные
export const AllLayersTree: React.FC<AllLayersTreeProps> = ({
  layers,
  selection,
}) => {
  // Получаем все ID слоев из хранилища
  const layerIds = useStorage((root) => root.layerIds);
  const room = useRoom();
  const roomId = room.id;

  if (!layerIds) return null;

  return (
    <div className="flex flex-col gap-1">
      <RenderLayersList
        layerIds={layerIds}
        layers={layers}
        selection={selection}
        roomId={roomId}
      />
    </div>
  );
};

export default AllLayersTree;
```

Листинг Е.5 – Исходный код правой боковой панели RightSidebar.tsx

```
/**
 * Правая боковая панель приложения отвечает за редактирование свойств выбранного слоя,
 * управление режимами, отображение пользователей и доступ к функциям совместной работы.
 */

import React, { useState, useEffect } from "react";
import { useMutation, useSelf, useOthers, useStorage } from "@liveblocks/react";
import * as Divider from "~/components/ui/divider";
import ModeButton from "~/components/ui/mode-button";
import * as ScaleButton from "~/components/ui/scale-button";
import * as Button from "~/components/ui/button";
import { scaleItems } from "~/components/ui/scale-button";
import { connectionIdToColor, hexToRgb } from "~/utils";
import { useCanvas } from "~/components/canvas/helper/CanvasContext";
import ColorRow from "./props/ColorRow";
import OpacityRow from "./props/OpacityRow";
import BasicSettings from "./props/BasicSettings";
import StrokeRow from "./props/StrokeRow";
import BgColor from "./props/BgColor";
import TextRow from "./props/TextRow";
import UserAvatar from "./UserAvatar";
import { Tutorial } from "./Tutorial";
import type { Layer, Color } from "~/types";
import type { UserInfo } from "~/types/user";
import { LayerType } from "~/types";
import {
  Root as AvatarGroup,
  Overflow as AvatarGroupOverflow,
} from "~/components/ui/avatar-group";
import ShareMenu from "./ShareMenu";

// Адаптер для BasicSettings
const createBasicSettingsAdapter = (
  updateLayer: (updates: {
    x?: number;
    y?: number;
    width?: number;
    height?: number;
    opacity?: number;
    cornerRadius?: number;
    fill?: Color | null;
    stroke?: Color | null;
    strokeWidth?: number;
    fontSize?: number;
    fontWeight?: number;
    fontFamily?: string;
    lineHeight?: number;
    letterSpacing?: number;
    tiltAngle?: number;
    blendMode?: string;
  }) => void
) => {
  return (updates: {
    x?: number;
```

Продолжение листинга Е.5

```
    y?: number;
    width?: number;
    height?: number;
    opacity?: number;
    cornerRadius?: number;
    fill?: string | null;
    stroke?: string;
    fontSize?: number;
    fontWeight?: number;
    fontFamily?: string;
    tiltAngle?: number;
    blendMode?: string;
  }) => {
    const { fill, stroke, ...rest } = updates;
    updateLayer({
      ...rest,
      ...(fill !== undefined && { fill: fill ? hexToRgb(fill) : null }),
      ...(stroke !== undefined && { stroke: stroke ? hexToRgb(stroke) : null }),
    });
  };
};

// Адаптер для ColorRow
const createColorRowAdapter = (
  updateLayer: (updates: {
    fill?: Color | null;
  }) => void
) => {
  return (updates: { fill?: string | null }) => {
    const { fill } = updates;
    updateLayer({
      fill: fill ? hexToRgb(fill) : null,
    });
  };
};

// Адаптер для StrokeRow
const createStrokeRowAdapter = (
  updateLayer: (updates: {
    stroke?: Color | null;
    strokeWidth?: number;
  }) => void
) => {
  return (updates: { stroke?: string | null; strokeWidth?: number }) => {
    const { stroke, strokeWidth } = updates;
    updateLayer({
      ...(stroke !== undefined && { stroke: stroke ? hexToRgb(stroke) : null }),
      ...(strokeWidth !== undefined && { strokeWidth }),
    });
  };
};

interface RightSidebarProps {
  leftIsMinimized: boolean;
```

Продолжение листинга Е.5

```
  layer: Layer | null;
  _roomId: string;
  _othersWithAccessToRoom: UserInfo[];
  _owner: UserInfo;
  _isOwner: boolean;
}

export const RightSidebar: React.FC<RightSidebarProps> = ({
  leftIsMinimized,
  layer,
  _roomId,
  _othersWithAccessToRoom,
  _owner,
  _isOwner,
}) => {
  // Получаем состояние камеры холста и функцию для его изменения
  const { camera, setCamera } = useCanvas();
  // Состояние для переключения между режимами панели: дизайн/учебник
  const [rightSidebarMode, setRightSidebarMode] = useState("design");
  // Состояние выбранного масштаба
  const [selectedScale, setSelectedScale] = useState("1");
  // Цвет комнаты из Liveblocks-хранилища
  const roomColor = useStorage((root) => root.roomColor);
  // Данные о текущем пользователе и других участниках
  const me = useSelf();
  const others = useOthers();

  // Получаем ID выбранного слоя для обновлений
  const selectedLayerId = useSelf((me) => {
    const selection = me.presence.selection;
    return selection.length === 1 ? selection[0] : null;
  });

  // Мутация для обновления свойств слоя через Liveblocks
  const updateLayer = useMutation(
    (
      { storage },
      updates: {
        x?: number;
        y?: number;
        width?: number;
        height?: number;
        opacity?: number;
        cornerRadius?: number;
        fill?: Color | null;
        stroke?: Color | null;
        strokeWidth?: number;
        fontSize?: number;
        fontWeight?: number;
        fontFamily?: string;
        lineHeight?: number;
        letterSpacing?: number;
        tiltAngle?: number;
        blendMode?: string;
```

Продолжение листинга Е.5

```
    },
  ) => {
    if (!layer || !selectedLayerId) return;

    const liveLayers = storage.get("layers");
    const layerObj = liveLayers.get(selectedLayerId);

    if (layerObj) {
      layerObj.update(updates);
    }
  },
  [selectedLayerId],
);

// Создаем адаптеры для передачи обновлений в дочерние компоненты
const basicSettingsAdapter = React.useMemo(
  () => createBasicSettingsAdapter(updateLayer),
  [updateLayer]
);
const colorRowAdapter = React.useMemo(
  () => createColorRowAdapter(updateLayer),
  [updateLayer]
);
const strokeRowAdapter = React.useMemo(
  () => createStrokeRowAdapter(updateLayer),
  [updateLayer]
);

// Функция для изменения масштаба холста и центрирования относительно текущего положения
const handleScaleChange = (value: string) => {
  setSelectedScale(value);
  const zoomValue = parseFloat(value);
  const canvas = document.querySelector("svg");
  if (canvas) {
    const centerPoint = {
      x: canvas.clientWidth / 2,
      y: canvas.clientHeight / 2,
    };
    setCamera((prevCamera) => {
      const scale = zoomValue / prevCamera.zoom;
      const newX = centerPoint.x - (centerPoint.x - prevCamera.x) * scale;
      const newY = centerPoint.y - (centerPoint.y - prevCamera.y) * scale;
      return { ...prevCamera, zoom: zoomValue, x: newX, y: newY };
    });
  }
};

// Следим за изменением масштаба камеры и обновляем выбранное значение
useEffect(() => {
  const zoomValue = camera.zoom.toString();
  if (scaleItems.some((item) => item.value === zoomValue)) {
    setSelectedScale(zoomValue);
  } else {
    // Находим ближайшее значение масштаба
```

Продолжение листинга Е.5

```
    const closestScale = scaleItems.reduce((prev, curr) => {
      return Math.abs(parseFloat(curr.value) - camera.zoom) <
        Math.abs(parseFloat(prev.value) - camera.zoom)
        ? curr
        : prev;
    });
    setSelectedScale(closestScale.value);
  }
}, [camera.zoom]);

// Обработчик изменения цвета (заливка или обводка)
const handleColorChange = (color: string | Color, type: "fill" | "stroke") => {
  if (!layer || !color) return;

  try {
    // Если цвет уже является объектом Color, используем его напрямую
    if (
      typeof color === "object" &&
      "r" in color &&
      "g" in color &&
      "b" in color
    ) {
      updateLayer({
        [type]: color,
      });
      return;
    }

    // Иначе преобразуем hex в RGB объект
    const colorObj = hexToRgb(color);
    if (!colorObj) return;

    // Обновляем слой с новым цветом
    updateLayer({
      [type]: colorObj,
    });
  } catch (error) {
    console.error("Error updating color:", error);
  }
};

// Обработчики для цвета обводки и заливки
const handleStrokeChange = (color: string | Color) => {
  handleColorChange(color, "stroke");
};

const handleFillChange = (color: string | Color) => {
  handleColorChange(color, "fill");
};

// Мутация для обновления цвета комнаты
const updateRoomColor = useMutation(({ storage }, color: Color) => {
  storage.set("roomColor", color);
}, [])
```

Продолжение листинга Е.5

```
  // Формируем класс для контейнера в зависимости от состояния панели и наличия выбранного
слоя
  const className = `fixed ${
    leftIsMinimized && layer ? "bottom-3 right-3 top-3 flex rounded-xl" : ""
  } ${!leftIsMinimized && !layer ? "h-screen" : ""} ${
    !leftIsMinimized && layer ? "bottom-0 top-[48px] h-screen" : ""
  } right-0 ${
    leftIsMinimized ? "top-3" : "top-[48px]"
  } flex w-[280px] select-none flex-col border-l border-stroke-soft-200 bg-bg-white-0`;

  return (
    <div className={className}>
      {/* Отображаем аватары пользователей и кнопку "Поделиться", если левая панель свёрнута
и есть выбранный слой */}
      {leftIsMinimized && layer && (
        <div className="flex items-center justify-between px-4 pb-0 pt-2">
          <AvatarGroup size="32">
            {me && (
              <UserAvatar
                color={connectionIdToColor(me.connectionId)}
                name={me.info.name}
                image={me.info.image}
              />
            )}
            {others.length > 2 ? (
              <>
                {others.slice(0, 2).map((other) => (
                  <UserAvatar
                    key={other.connectionId}
                    color={connectionIdToColor(other.connectionId)}
                    name={other.info.name}
                    image={other.info.image}
                  />
                ))}
                <AvatarGroupOverflow>
                  +{others.length - 2}
                </AvatarGroupOverflow>
              </>
            ) : (
              others.map((other) => (
                <UserAvatar
                  key={other.connectionId}
                  color={connectionIdToColor(other.connectionId)}
                  name={other.info.name}
                  image={other.info.image}
                />
              ))
            )}
          </AvatarGroup>
          <Button.Root variant="primary" mode="lighter" size="xsmall">
            Поделиться
          </Button.Root>
        </div>
```

Продолжение листинга Е.5

```
)}

{/* Переключатели режимов и масштаб */}
<div className="flex flex-col gap-2 p-4 px-3">
  <div className="flex items-center justify-between">
    <div className="flex flex-row gap-1">
      <ModeButton
        onSelect={() => setRightSidebarMode("design")}
        active={rightSidebarMode === "design"}
        text="Дизайн"
      />

      <ModeButton
        onSelect={() => setRightSidebarMode("tutorial")}
        active={rightSidebarMode === "tutorial"}
        text="Учебник"
      />
    </div>

    <ScaleButton.Root
      value={selectedScale}
      onValueChange={handleScaleChange}
    >
      <ScaleButton.Trigger></ScaleButton.Trigger>
      <ScaleButton.Content align="center">
        {scaleItems.map((item) => (
          <ScaleButton.Item key={item.value} value={item.value}>
            {item.label}
          </ScaleButton.Item>
        ))}
      </ScaleButton.Content>
    </ScaleButton.Root>
  </div>

  <Divider.Root />
</div>

{/* Содержимое панели в зависимости от выбранного режима */}
{layer && rightSidebarMode === "design" ? (
  <>
    {/* Базовые настройки: позиция, размер, скругление углов */}
    <div className="flex flex-col gap-2 p-4 pb-2 pt-0">
      <BasicSettings layer={layer} onUpdateLayer={basicSettingsAdapter} />
    </div>

    <div className="w-full max-w-96 p-3"><Divider.Root /></div>

    {/* Прозрачность и режим наложения */}
    <div className="flex flex-col gap-2 p-4 py-0">
      <OpacityRow layer={layer} onUpdateLayer={updateLayer} />
    </div>
    <div className="w-full max-w-96 p-3"><Divider.Root /></div>

    {/* Настройки текста, если слой является текстовым */}
```

Продолжение листинга Е.5

```
          {layer.type === LayerType.Text && (
            <>
              <div className="flex flex-col gap-2 p-4 py-0">
                <TextRow layer={layer} onUpdateLayer={updateLayer} />
              </div>

              <div className="w-full max-w-96 p-3">
                <Divider.Root />
              </div>
            </>
          )}

          {/* Цвет заливки */}
          <div className="flex flex-col gap-2 p-4 py-0">
            <ColorRow
              layer={layer}
              onUpdateLayer={colorRowAdapter}
              onColorChange={handleFillChange}
            />
          </div>

          <div className="w-full max-w-96 p-3"><Divider.Root /></div>

          {/* Настройки обводки */}
          <div className="flex flex-col gap-2 p-4 py-0">
            <StrokeRow
              layer={layer}
              onUpdateLayer={strokeRowAdapter}
              onColorChange={handleStrokeChange}
            />
          </div>

          <div className="w-full max-w-96 p-3"><Divider.Root /></div>

        </>
      ) : rightSidebarMode === "tutorial" ? (
        <div className="px-4">
          <Tutorial />
        </div>
      ) : (
        <div className="flex flex-col gap-2 p-4 py-0">
          <BgColor roomColor={roomColor} setRoomColor={updateRoomColor} />
        </div>
      )}
    </div>
  );
};

// Компонент для отображения свернутой правой панели: показывает аватары и меню
export const MinimizedRightSidebar: React.FC<{
  roomId: string;
  othersWithAccessToRoom: UserInfo[];
  owner: UserInfo;
  isOwner: boolean;
```

Окончание листинга Е.5

```
}> = ({ roomId, othersWithAccessToRoom, owner, isOwner }) => {
  const me = useSelf();
  const others = useOthers();

  return (
    <div className="py2 fixed right-3 top-3 flex h-[48px] w-[280px] select-none items-center
justify-between rounded-xl border bg-white px-4 shadow-regular-sm">
      <div className="flex w-full flex-row items-center justify-between">
        <AvatarGroup size="32">
          {me && (
            <UserAvatar
              color={connectionIdToColor(me.connectionId)}
              name={me.info.name}
              image={me.info.image}
            />
          )}
          {others.length > 2 ? (
            <>
              {others.slice(0, 2).map((other) => (
                <UserAvatar
                  key={other.connectionId}
                  color={connectionIdToColor(other.connectionId)}
                  name={other.info.name}
                  image={other.info.image}
                />
              ))}
              <AvatarGroupOverflow>
                +{others.length - 2}
              </AvatarGroupOverflow>
            </>
          ) : (
            others.map((other) => (
              <UserAvatar
                key={other.connectionId}
                color={connectionIdToColor(other.connectionId)}
                name={other.info.name}
                image={other.info.image}
              />
            ))
          )}
        </AvatarGroup>

        <ShareMenu
          roomId={roomId}
          othersWithAccessToRoom={othersWithAccessToRoom}
          owner={owner}
          isOwner={isOwner}
        />
      </div>
    </div>
  );
};

export default RightSidebar;
```

ПРИЛОЖЕНИЕ Ж
ИСХОДНЫЙ КОД МОДУЛЯ ШАБЛОНОВ

Листинг Ж.1 – Исходный код вкладки шаблонов TemplatesTab.tsx

```
/**
 * Вкладка с шаблонами отвечает за отображение доступных категорий шаблонов, навигацию между категориями и просмотр шаблонов внутри выбранной категории.
 * Компонент обеспечивает удобный выбор шаблонов для дальнейшего использования в проекте.
 */

import React, { useState, useEffect } from "react";
import { TemplateCategory } from "~/types";
import TemplatesList from "./TemplatesList";
import { RiArrowLeftLine } from "@remixicon/react";
import * as CompactButton from "~/components/ui/compact-button";
import Image from "next/image";

interface Template {
  category: TemplateCategory;
  // Add other template properties as needed
}

// Маппинг категорий на иконки/миниатюры
const categoryIcons: Record<TemplateCategory, string> = {
  [TemplateCategory.Presentation]: "/templates/categories/thumbnail_presentation_16_9.svg",
  [TemplateCategory.SocialMedia]: "/templates/categories/thumbnail_vk_post.svg",
  [TemplateCategory.BusinessCard]: "/templates/categories/thumbnail_business_card.svg",
  [TemplateCategory.Poster]: "/templates/categories/thumbnail_poster.svg",
  [TemplateCategory.Logo]: "/templates/categories/thumbnail_logo.svg",
  [TemplateCategory.Document]: "/templates/categories/thumbnail_doc.svg",
};

// Функция для склонения слова "шаблон"
const getTemplateWord = (count: number): string => {
  const lastDigit = count % 10;
  const lastTwoDigits = count % 100;

  if (lastTwoDigits >= 11 && lastTwoDigits <= 19) {
    return "шаблонов";
  }

  if (lastDigit === 1) {
    return "шаблон";
  }

  if (lastDigit >= 2 && lastDigit <= 4) {
    return "шаблона";
  }

  return "шаблонов";
};

const TemplatesTab: React.FC = () => {
  // Состояния для хранения данных о категориях и выбранной категории
```

Продолжение листинга Ж.1

```
  const [categories, setCategories] = useState<TemplateCategory[]>([]);
  const [selectedCategory, setSelectedCategory] =
    useState<TemplateCategory | null>(null);
  const [isLoading, setIsLoading] = useState(true);
  const [templateCounts, setTemplateCounts] = useState<Record<TemplateCategory, number>>({}
as Record<TemplateCategory, number>);

  // Загружаем список доступных категорий и количество шаблонов
  useEffect(() => {
    const loadCategories = async () => {
      try {
        setIsLoading(true);
        setCategories(Object.values(TemplateCategory));

        const response = await fetch("/api/templates");
        const templates = await response.json() as Template[];

        // Считаем количество шаблонов в каждой категории
        const counts = Object.values(TemplateCategory).reduce((acc, category) => {
          acc[category] = templates.filter((t) => t.category === category).length;
          return acc;
        }, {} as Record<TemplateCategory, number>);

        setTemplateCounts(counts);
      } catch (err) {
        console.error("Ошибка при загрузке категорий:", err);
      } finally {
        setIsLoading(false);
      }
    };

    void loadCategories();
  }, []);

  // Обработчик выбора категории
  const handleCategorySelect = (category: TemplateCategory) => {
    setSelectedCategory(category);
  };

  // Обработчик возврата к списку категорий
  const handleBack = () => {
    setSelectedCategory(null);
  };

  // Если выбрана категория - показываем шаблоны этой категории
  if (selectedCategory) {
    return (
      <div className="flex h-full flex-col">
          {/* Заголовок с кнопкой возврата */}
          <div className="flex flex-row items-center gap-2 px-4 py-3 shrink-0">
            <CompactButton.Root
              size="medium"
              variant="ghost"
              onClick={handleBack}
```

Продолжение листинга Ж.1

```
            >
              <CompactButton.Icon as={RiArrowLeftLine} />
            </CompactButton.Root>
            <span className="text-paragraph-sm text-text-sub-600">
            {selectedCategory}
            </span>
          </div>

          {/* Список шаблонов выбранной категории */}
          <div  className="px-4  py-2  hover:[&::-webkit-scrollbar-thumb]:bg-stroke-soft-300
flex-1 overflow-hidden hover:overflow-y-auto [&::-webkit-scrollbar-thumb]:rounded-full [&::-
webkit-scrollbar-thumb]:bg-stroke-soft-200 [&::-webkit-scrollbar-track]:bg-transparent [&::-
webkit-scrollbar]:w-1">
            <TemplatesList selectedCategory={selectedCategory} />
          </div>

      </div>
    );
  }

  // Если категория не выбрана - показываем список категорий
  return (
    <div className="flex h-full flex-col">
      {/* Заголовок раздела */}
      <div className="shrink-0 px-4 py-2">
        <span className="text-label-sm text-text-strong-950">Категории</span>
      </div>

      {/* Индикатор загрузки или список категорий */}
      {isLoading ? (
        <div className="px-4 py-2 text-paragraph-sm text-text-sub-600">
          Загрузка категорий...
        </div>
      ) : (
        <div     className="hover:[&::-webkit-scrollbar-thumb]:bg-stroke-soft-300     flex-1
overflow-y-auto  px-4  py-1  [&::-webkit-scrollbar-thumb]:rounded-full  [&::-webkit-scrollbar-
thumb]:bg-stroke-soft-200      [&::-webkit-scrollbar-track]:bg-transparent      [&::-webkit-
scrollbar]:w-1">
          <div className="space-y-5 pb-16">
            {/* Отображаем карточки категорий */}
            {categories.map((category) => (
              <div key={category} className="flex flex-col gap-2">
                {/* Карточка категории с миниатюрой */}
                <div
                  className="hover:border-primary-500 flex cursor-pointer flex-col overflow-
hidden rounded-lg border border-stroke-soft-200 transition-colors"
                  onClick={() => handleCategorySelect(category)}
                >
                  <div className="bg-bg-weak_alt-100 relative h-32 w-full">
                    <Image
                      src={categoryIcons[category]}
                      alt={category}
                      fill
                      className="object-cover"
```

Окончание листинга Ж.1

```
                    onError={(e) => {
                      const target = e.target as HTMLImageElement;
                      target.style.display = "none";
                    }}
                  />
                </div>
              </div>

              {/* Название категории и количество шаблонов */}
              <div className="flex flex-col">
                <div className="text-paragraph-sm text-text-strong-950">
                  {category}
                </div>
                <div className="text-paragraph-xs text-text-sub-600">
                  {templateCounts[category]                    ||                    0}
{getTemplateWord(templateCounts[category] || 0)}
                </div>
              </div>
            </div>
          ))}
        </div>
      </div>
    )}
  </div>
);
};

export default TemplatesTab;
```

Листинг Ж.2 – Исходный код списка шаблонов TemplatesList.tsx

```
/**
 * Список шаблонов отвечает за отображение и загрузку шаблонов выбранной категории, а также
за применение выбранного шаблона на холст проекта.
 * Компонент обеспечивает взаимодействие с сервером и интеграцию шаблонов в рабочее
пространство пользователя.
 */

import React, { useState, useEffect } from "react";
import type { Template, TemplateCategory } from "~/types";
import { useMutation } from "@liveblocks/react";
import { LiveObject } from "@liveblocks/client";
import Image from "next/image";

interface TemplatesListProps {
  selectedCategory: TemplateCategory;
}

const TemplatesList: React.FC<TemplatesListProps> = ({ selectedCategory }) => {
  // Состояния для хранения данных о шаблонах, загрузке и ошибках
  const [templates, setTemplates] = useState<Template[]>([]);
  const [isLoading, setIsLoading] = useState(true);
```

Продолжение листинга Ж.2

```
const [error, setError] = useState<string | null>(null);

// Мутация для применения шаблона на холст
const applyTemplate = useMutation(({ storage }, template: Template) => {
  // Получаем существующие слои
  const layers = storage.get("layers");
  const layerIds = storage.get("layerIds");

  // Добавляем слои из шаблона
  Object.entries(template.layers).forEach(([id, layer]) => {
    layers.set(id, new LiveObject(layer));
  });

  // Добавляем rootLayerIds шаблона в общий список слоев
  template.rootLayerIds.forEach(id => {
    layerIds.push(id);
  });
}, []);

// Загрузка шаблонов выбранной категории с сервера
useEffect(() => {
  const loadTemplates = async () => {
    try {
      setIsLoading(true);
      // Здесь должен быть запрос к API для получения шаблонов
      const response = await fetch("/api/templates");
      const data = (await response.json()) as Template[];

      // Фильтруем шаблоны по выбранной категории
      const filteredData = data.filter(
        (template) => template.category === selectedCategory
      );
      setTemplates(filteredData);
    } catch (err) {
      setError("Не удалось загрузить шаблоны");
      console.error(err);
    } finally {
      setIsLoading(false);
    }
  };

  void loadTemplates();
}, [selectedCategory]);

// Отображаем индикатор загрузки
if (isLoading) {
  return (
    <div className="flex items-center justify-center h-full text-paragraph-sm">
      <div className="animate-pulse text-text-sub-600">Загрузка шаблонов...</div>
    </div>
  );
}

// Отображаем сообщение об ошибке
```

Окончание листинга Ж.2

```
  if (error) {
    return (
      <div className="text-text-sub-600 text-paragraph-sm">
        {error}
      </div>
    );
  }

  // Отображаем сообщение, если шаблоны не найдены
  if (templates.length === 0) {
    return (
      <div className="text-text-sub-600 text-paragraph-sm">
        Шаблоны в категории &quot;{selectedCategory}&quot; не найдены
      </div>
    );
  }

  // Отображаем список шаблонов
  return (
    <div className="space-y-6 pb-16">
      {templates.map((template) => (
        <div
          key={template.id}
          className="hover:border-primary-500 flex cursor-pointer flex-col overflow-hidden
rounded-lg border border-stroke-soft-200 transition-colors"
          onClick={() => applyTemplate(template)}
        >
          <div className="relative w-full h-32 bg-bg-weak_alt-100">
            <Image
              src={template.thumbnail}
              alt={template.name}
              fill
              className="object-cover"
            />
          </div>
        </div>
      ))}
    </div>
  );
};

export default TemplatesList;
```

ПРИЛОЖЕНИЕ З
ИСХОДНЫЙ КОД МОДУЛЯ ОБУЧЕНИЯ

Листинг З.1 – Исходный код панели обучения Tutorial.tsx

```
"use client";

/**
 * Компонент обучения отвечает за отображение списка обучающих курсов, отслеживание и
сохранение прогресса пользователя, а также за показ содержимого выбранного курса в окне.
 * Обеспечивает удобное взаимодействие с учебными материалами и хранение прогресса.
 */

import React, { useState, useEffect } from 'react';
import { tutorialCourses } from './tutorial/courses';
import type { TutorialCourse } from '~/types';
import { CourseCard } from './tutorial/CourseCard';
import { TutorialModal } from './tutorial/TutorialModal';

// Ключ для хранения данных о прогрессе обучения в localStorage
const TUTORIAL_PROGRESS_KEY = 'tutorial_progress';

// Тип для хранения прогресса обучения по курсам
type TutorialProgress = Record<string, {
  completedLessons: string[];
}>;

export const Tutorial: React.FC = () => {
  // Состояния для хранения курсов и выбранного курса
  const [courses, setCourses] = useState<TutorialCourse[]>([]);
  const [selectedCourse, setSelectedCourse] = useState<TutorialCourse | null>(null);
  const [isCourseOpen, setIsCourseOpen] = useState(false);

  // Загружаем курсы при первом рендере, восстанавливая прогресс из localStorage
  useEffect(() => {
    loadCoursesWithProgress();
  }, []);

  // Функция загрузки курсов с прогрессом пользователя
  const loadCoursesWithProgress = () => {
    const savedProgressString = localStorage.getItem(TUTORIAL_PROGRESS_KEY);

    if (savedProgressString) {
      try {
        // Парсим сохраненный прогресс
        const savedProgress = JSON.parse(savedProgressString) as TutorialProgress;

        // Применяем сохраненный прогресс к курсам
        const coursesWithProgress = tutorialCourses.map(course => {
          const courseProgress = savedProgress[course.id];

          if (courseProgress) {
            // Обновляем статусы уроков на основе сохраненных данных
            const updatedTopics = course.topics.map(topic => {
              const updatedLessons = topic.lessons.map(lesson => {
```

Продолжение листинга 3.1

```
            const isCompleted = courseProgress.completedLessons.includes(lesson.id);
            return { ...lesson, completed: isCompleted };
          });
          return { ...topic, lessons: updatedLessons };
        });

        // Рассчитываем общее количество выполненных уроков
        let completedCount = 0;
        updatedTopics.forEach(topic => {
          topic.lessons.forEach(lesson => {
            if (lesson.completed) completedCount++;
          });
        });

        return {
          ...course,
          topics: updatedTopics,
          completedLessons: completedCount
        };
      }

      return course;
    });

    setCourses(coursesWithProgress);
  } catch (e) {
    console.error('Ошибка восстановления прогресса:', e);
    setCourses(tutorialCourses);
  }
} else {
  // Если сохраненного прогресса нет, используем исходные курсы
  setCourses(tutorialCourses);
}
};

// Обработчик клика по карточке курса
const handleCourseClick = (course: TutorialCourse) => {
  setSelectedCourse(course);
  setIsCourseOpen(true);
};

// Обновляем прогресс по уроку и сохраняем в localStorage
const handleUpdateProgress = (courseId: string, lessonId: string, completed: boolean) => {
  // Обновляем состояние курсов с новым прогрессом
  setCourses(prevCourses => {
    const updatedCourses = prevCourses.map(course => {
      if (course.id === courseId) {
        // Найдем урок и обновим его статус
        const updatedTopics = course.topics.map(topic => {
          const updatedLessons = topic.lessons.map(lesson => {
            if (lesson.id === lessonId) {
              return { ...lesson, completed };
            }
            return lesson;
```

Продолжение листинга З.1

```
        });
        return { ...topic, lessons: updatedLessons };
      });

      // Посчитаем общее количество выполненных уроков
      let completedCount = 0;
      updatedTopics.forEach(topic => {
        topic.lessons.forEach(lesson => {
          if (lesson.completed) completedCount++;
        });
      });

      return {
        ...course,
        topics: updatedTopics,
        completedLessons: completedCount
      };
    }
    return course;
  });

  // Сохраняем обновленный прогресс в localStorage
  saveProgressToLocalStorage(updatedCourses);

  return updatedCourses;
});

// Также обновляем выбранный курс, если это он
if (selectedCourse && selectedCourse.id === courseId) {
  const updatedTopics = selectedCourse.topics.map(topic => {
    const updatedLessons = topic.lessons.map(lesson => {
      if (lesson.id === lessonId) {
        return { ...lesson, completed };
      }
      return lesson;
    });
    return { ...topic, lessons: updatedLessons };
  });

  let completedCount = 0;
  updatedTopics.forEach(topic => {
    topic.lessons.forEach(lesson => {
      if (lesson.completed) completedCount++;
    });
  });

  setSelectedCourse({
    ...selectedCourse,
    topics: updatedTopics,
    completedLessons: completedCount
  });
}
};
```

Окончание листинга З.1

```
  // Функция для сохранения прогресса в localStorage
  const saveProgressToLocalStorage = (updatedCourses: TutorialCourse[]) => {
    try {
      // Создаем объект с прогрессом по каждому курсу
      const progressData: Record<string, { completedLessons: string[] }> = {};

      updatedCourses.forEach(course => {
        const completedLessons: string[] = [];

        // Собираем ID всех выполненных уроков
        course.topics.forEach(topic => {
          topic.lessons.forEach(lesson => {
            if (lesson.completed) {
              completedLessons.push(lesson.id);
            }
          });
        });

        progressData[course.id] = { completedLessons };
      });

      // Сохраняем прогресс в localStorage
      localStorage.setItem(TUTORIAL_PROGRESS_KEY, JSON.stringify(progressData));
    } catch (e) {
      console.error('Ошибка сохранения прогресса:', e);
    }
  };

  return (
    <>
      {/* Список карточек курсов */}
      <div className="space-y-4">
        {courses.map(course => (
          <CourseCard
            key={course.id}
            course={course}
            onClick={handleCourseClick}
          />
        ))}
      </div>

      {/* Модальное окно с содержимым курса */}
      {selectedCourse && (
        <TutorialModal
          course={selectedCourse}
          isOpen={isCourseOpen}
          onOpenChange={setIsCourseOpen}
          onUpdateProgress={handleUpdateProgress}
          variant="command"
        />
      )}
    </>
  );
};
```

Листинг З.2 – Исходный код карточки курса CourseCard.tsx

```
/**
 * Карточка учебного курса отвечает за отображение информации о курсе, визуализацию статуса прохождения и обработку клика для открытия содержимого курса.
 * Компонент используется в списке курсов для представления прогресса пользователя.
 */

import React from "react";
import type { TutorialCourse } from "~/types";
import Image from "next/image";

interface CourseCardProps {
  course: TutorialCourse;
  onClick: (course: TutorialCourse) => void;
}

export const CourseCard: React.FC<CourseCardProps> = ({ course, onClick }) => {
  // Рассчитываем процент прохождения курса
  const progressPercent = course.completedLessons
    ? Math.round((course.completedLessons / course.lessonsCount) * 100)
    : 0;

  // Определяем статус курса
  const getCourseStatus = () => {
    if (progressPercent === 100) return "completed";
    if (progressPercent > 0) return "pending";
    if (course.isNew) return "pending";
    return "disabled";
  };

  // Определяем текст статуса
  const getStatusText = () => {
    if (progressPercent === 100) return "Пройден";
    if (progressPercent > 0) return "В процессе";
    if (course.isNew) return "Новый";
    return "Не начат";
  };

  return (
    <div
      className="flex cursor-pointer flex-col rounded-xl border border-stroke-soft-200 bg-bg-white-0 p-4 transition-colors hover:bg-bg-weak-50"
      onClick={() => onClick(course)}
    >

      {/* Верхняя часть карточки с иконкой и информацией */}
      <div className="mb-3 flex items-center gap-3">
        {/* Иконка курса */}
        <div className="flex-shrink-0">
          <Image
            src={course.iconPath}
            alt={course.title}
            width={32}
            height={32}
            className="rounded"
```

Окончание листинга З.2

```
          />
        </div>

        <div className="flex-2">
          {/* Название курса */}
          <h3 className="text-label-sm text-text-strong-950">
            {course.title}
          </h3>

          {/* Информация о продолжительности, уровне и статусе */}
          <div className="flex items-center gap-4 text-paragraph-xs text-text-soft-400">
            <div className="flex items-center gap-1">
              <span>{course.duration}  •  {course.level}</span>
            </div>
            <span className={`px-2 py-0.5 rounded text-label-xs ${
              getCourseStatus() === 'completed' ? 'bg-green-100 text-green-800' :
              getCourseStatus() === 'pending' ? 'bg-blue-100 text-blue-800' :
              'bg-gray-100 text-gray-800'
            }`}>
              {getStatusText()}
            </span>
          </div>
        </div>
      </div>

      {/* Описание курса */}
      <p className="line-clamp-2 text-paragraph-xs text-text-sub-600">
        {course.description}
      </p>
    </div>
  );
};
```

Листинг З.3 – Исходный код окна обучения TutorialModal.tsx

```
/**
 * Модальное окно обучения отвечает за отображение содержимого учебного курса, навигацию
между темами и уроками, а также за отслеживание и отображение прогресса пользователя.
*/

import React, { useState, useEffect, useCallback, useMemo } from "react";
import {
  RiCloseLine,
  RiCheckboxCircleFill,
  RiCheckboxBlankCircleLine,
  RiArrowRightLine,
  RiLockLine,
  RiCheckDoubleLine,
} from "@remixicon/react";
import type { TutorialCourse, TutorialLesson, TutorialTopic } from "~/types";
import * as Modal from "~/components/ui/modal";
import * as CommandMenu from "~/components/ui/command-menu";
import * as Divider from "~/components/ui/divider";
```

Продолжение листинга З.3

```
import * as TabMenuVertical from "~/components/ui/tab-menu-vertical";
import * as Button from "~/components/ui/button";

interface TutorialModalProps {
  isOpen: boolean;
  onClose?: () => void;
  onOpenChange?: (open: boolean) => void;
  course: TutorialCourse;
  initialTopicId?: string;
  initialLessonId?: string;
  onCompleteLesson?: (lessonId: string, completed: boolean) => void;
  onUpdateProgress?: (
    courseId: string,
    lessonId: string,
    completed: boolean,
  ) => void;
  variant?: "modal" | "command";
}

export const TutorialModal: React.FC<TutorialModalProps> = ({
  isOpen,
  onClose,
  onOpenChange,
  course,
  initialTopicId,
  initialLessonId,
  onCompleteLesson,
  onUpdateProgress,
  variant = "modal", // По умолчанию используем стандартное модальное окно
}) => {
  // Создаем пустые заглушки для случаев, когда данных нет
  const emptyLesson = useMemo(() => ({
    id: "empty-lesson",
    title: "Занятия отсутствуют",
    content: "В данной теме пока нет материалов.",
    completed: false,
  }), []);

  const emptyTopic = useMemo(() => ({
    id: "empty-topic",
    title: "Темы отсутствуют",
    lessons: [emptyLesson],
  }), [emptyLesson]);

  // Получаем начальный топик
  const getInitialTopic = useCallback((): TutorialTopic => {
    // Если в курсе нет топиков, возвращаем заглушку
    if (!course.topics || course.topics.length === 0) {
      return emptyTopic;
    }

    // Если указан ID начального топика, ищем его
    if (initialTopicId) {
      for (const topic of course.topics) {
```

Продолжение листинга З.3

```
        if (topic.id === initialTopicId) {
          return topic;
        }
      }
    }

    // Иначе возвращаем первый топик
    return course.topics[0] ?? emptyTopic;
  }, [course.topics, initialTopicId, emptyTopic]);

  // Получаем начальный урок для указанного топика
  const getInitialLesson = useCallback((topic: TutorialTopic): TutorialLesson => {
    // Если в топике нет уроков, возвращаем заглушку
    if (!topic.lessons || topic.lessons.length === 0) {
      return emptyLesson;
    }

    // Если указан ID начального урока, ищем его
    if (initialLessonId) {
      for (const lesson of topic.lessons) {
        if (lesson.id === initialLessonId) {
          return lesson;
        }
      }
    }

    // Иначе возвращаем первый урок
    return topic.lessons[0] ?? emptyLesson;
  }, [initialLessonId, emptyLesson]);

  // Состояние для активного топика и урока
  const [activeTopic, setActiveTopic] =
    useState<TutorialTopic>(getInitialTopic);
  const [activeLesson, setActiveLesson] = useState<TutorialLesson>(() => {
    const topic = getInitialTopic();
    return getInitialLesson(topic);
  });

  // Инициализируем активную тему и урок при открытии курса
  useEffect(() => {
    if (isOpen && course) {
      setActiveTopic(getInitialTopic());
      setActiveLesson(getInitialLesson(getInitialTopic()));
    }
  },     [isOpen,     course,     initialTopicId,     initialLessonId,     getInitialTopic,
getInitialLesson]);

  // Обработчик выбора урока с учетом только что выполненного урока
  const  handleSelectLessonWithCompletedId  =  (topicId:  string,  lesson:  TutorialLesson,
justCompletedLessonId?: string) => {
    // Проверка, можно ли открыть этот урок (предыдущие должны быть выполнены)
    if (!canAccessLesson(topicId, lesson, justCompletedLessonId)) {
      return;
    }
```

Продолжение листинга З.3

```
    // Поиск топика по ID
    const foundTopic = course.topics?.find((t) => t.id === topicId);

    // Если топик найден, обновляем состояние
    if (foundTopic) {
      setActiveTopic(foundTopic);
      setActiveLesson(lesson);
    }
  };

  // Обработчик выбора урока - теперь вызывает новую функцию без передачи
justCompletedLessonId
  const handleSelectLesson = (topicId: string, lesson: TutorialLesson) => {
    handleSelectLessonWithCompletedId(topicId, lesson);
  };

  // Функция для определения, может ли пользователь получить доступ к уроку
  const canAccessLesson = (
    topicId: string,
    lesson: TutorialLesson,
    justCompletedLessonId?: string,
  ): boolean => {
    // Если у нас нет функции отслеживания прогресса, разрешаем доступ ко всем урокам
    if (!onCompleteLesson && !onUpdateProgress) {
      return true;
    }

    const currentTopic = course.topics?.find((t) => t.id === topicId);
    if (!currentTopic) return false;

    // Индекс текущего урока в топике
    const lessonIndex = currentTopic.lessons.findIndex((l) => l.id === lesson.id);

    // Первый урок всегда доступен
    if (lessonIndex === 0) return true;

    // Проверяем, выполнен ли предыдущий урок
    const previousLesson = currentTopic.lessons[lessonIndex - 1];

    // Если предыдущий урок был только что отмечен как выполненный, учитываем это
    if (justCompletedLessonId && previousLesson?.id === justCompletedLessonId) {
      return true;
    }

    return Boolean(previousLesson?.completed);
  };

  // Функция для навигации к следующему уроку
  const goToNextLesson = () => {
    // Сохраняем текущий урок который нужно будет отметить как выполненным
    const lessonToMarkCompleted = activeLesson;
    const shouldMarkCompleted = !lessonToMarkCompleted.completed && (onCompleteLesson ??
onUpdateProgress);
```

Продолжение листинга З.3

```
// Находим индекс текущего урока в активном топике
const currentLessonIndex = activeTopic.lessons.findIndex(
  (l) => l.id === activeLesson.id,
);

// Определяем следующий урок
let nextTopic = activeTopic;
let nextLesson = null;

// Если это последний урок в топике
if (currentLessonIndex === activeTopic.lessons.length - 1) {
  // Находим индекс текущего топика
  const currentTopicIndex = course.topics.findIndex(
    (t) => t.id === activeTopic.id,
  );

  // Если есть следующий топик, переходим к его первому уроку
  if (currentTopicIndex < course.topics.length - 1) {
    const foundNextTopic = course.topics[currentTopicIndex + 1];
    if (foundNextTopic) {
      nextTopic = foundNextTopic;

      if (nextTopic.lessons?.length > 0) {
        nextLesson = nextTopic.lessons[0];
      }
    }
  }
} else {
  // Иначе переходим к следующему уроку в текущем топике
  nextLesson = activeTopic.lessons[currentLessonIndex + 1];
}

// Если есть следующий урок, сохраняем его для последующей установки
if (nextLesson) {
  // Сначала отмечаем текущий урок как выполненный
  if (shouldMarkCompleted) {
    handleCompleteLesson(lessonToMarkCompleted.id, true);

    // Сохраняем информацию о следующем уроке в замыкании
    const nextLessonCopy = nextLesson;
    const nextTopicCopy = nextTopic;

    // Устанавливаем короткую задержку перед переходом к следующему уроку
    // для того, чтобы состояние обновилось в родительском компоненте
    setTimeout(() => {
      // Обновляем состояние - устанавливаем следующий урок как активный
      setActiveTopic(nextTopicCopy);
      setActiveLesson(nextLessonCopy);
    }, 50); // Очень короткая задержка, которая не будет заметна пользователю
  } else {
    // Если урок уже отмечен выполненным, переходим к следующему без задержки
    // Обновляем состояние - устанавливаем следующий урок как активный
    setActiveTopic(nextTopic);
    setActiveLesson(nextLesson);
```

Продолжение листинга З.3

```
    }
  }
};

// Проверка наличия следующего урока
const hasNextLesson = (): boolean => {
  const currentLessonIndex = activeTopic.lessons.findIndex(
    (l) => l.id === activeLesson.id,
  );

  // Если это последний урок в топике, проверяем наличие следующего топика
  if (currentLessonIndex === activeTopic.lessons.length - 1) {
    const currentTopicIndex = course.topics.findIndex(
      (t) => t.id === activeTopic.id,
    );
    return currentTopicIndex < course.topics.length - 1;
  }
  // Иначе в текущем топике есть следующий урок
  return currentLessonIndex < activeTopic.lessons.length - 1;
};

// Обработчик отметки урока как выполненного
const handleCompleteLesson = (lessonId: string, completed: boolean) => {
  if (onCompleteLesson) {
    onCompleteLesson(lessonId, completed);
  }
  if (onUpdateProgress) {
    onUpdateProgress(course.id, lessonId, completed);
  }
};

// Обработчик закрытия модального окна
const handleOpenChange = (open: boolean) => {
  if (onOpenChange) {
    onOpenChange(open);
  } else if (!open && onClose) {
    onClose();
  }
};

// Проверка, является ли текущий урок последним в курсе
const isLastLessonInCourse = (): boolean => {
  const currentLessonIndex = activeTopic.lessons.findIndex(
    (l) => l.id === activeLesson.id);

  const isLastInTopic = currentLessonIndex === activeTopic.lessons.length - 1;
  if (!isLastInTopic) return false;

  const currentTopicIndex = course.topics.findIndex(
    (t) => t.id === activeTopic.id,
  );

  return currentTopicIndex === course.topics.length - 1;
};
```

Продолжение листинга 3.3

```
  // Обработчик завершения курса
  const handleCompleteCourse = () => {
    // Отмечаем текущий урок как выполненный, если еще не отмечен
    if (!activeLesson.completed && (onCompleteLesson || onUpdateProgress)) {
      handleCompleteLesson(activeLesson.id, true);
    }

    // Закрываем модальное окно
    if (onClose) {
      onClose();
    } else if (onOpenChange) {
      onOpenChange(false);
    }
  };

  // Основной компонент содержимого модального окна
  const contentComponent = (
    <div className="flex h-[680px]">
      {/* Сайдбар с темами и уроками */}
      <div  className="hover:[&::-webkit-scrollbar-thumb]:bg-stroke-soft-300  h-full  w-60
flex-col gap-4 overflow-hidden border-r border-stroke-soft-200 p-4 pr-2 hover:overflow-y-
auto   [&::-webkit-scrollbar-thumb]:rounded-full   [&::-webkit-scrollbar-thumb]:bg-stroke-
soft-200 [&::-webkit-scrollbar-track]:bg-transparent [&::-webkit-scrollbar]:w-1">
        <TabMenuVertical.Root className="mb-4">
          <TabMenuVertical.List>
            {course.topics.map((topic) => (
              <React.Fragment key={topic.id}>
                <h4 className="mb-2 px-2 py-1 text-subheading-xs uppercase text-text-soft-
400">
                  {topic.title}
                </h4>
                {topic.lessons.map((lesson) => {
                  const isAccessible = canAccessLesson(topic.id, lesson);
                  return (
                    <button
                      key={lesson.id}
                      onClick={() => handleSelectLesson(topic.id, lesson)}
                      disabled={!isAccessible}
                      className={`flex  w-full  items-center  gap-2  rounded-lg  px-3  py-2
text-left text-paragraph-sm transition-colors ${
                        activeLesson.id === lesson.id
                          ? "bg-bg-weak-50 text-text-strong-950"
                          : isAccessible
                            ? "text-text-sub-600 hover:bg-bg-weak-50"
                            : "text-text-disabled-300 opacity-60"
                      }`}
                    >
                      <div
                        className={`flex  size-5  flex-shrink-0  items-center  justify-center
rounded-full ${
                          lesson.completed
                            ? "text-success-base"
                            : !isAccessible
                              ? "text-text-disabled-300"
```

Продолжение листинга З.3

```
                        : "text-text-soft-400"
                    }`}
                  >
                    {lesson.completed ? (
                      <RiCheckboxCircleFill className="size-5" />
                    ) : !isAccessible ? (
                      <RiLockLine className="size-5" />
                    ) : (
                      <RiCheckboxBlankCircleLine className="size-5" />
                    )}
                  </div>
                  <span className="line-clamp-1 flex-1">
                    {lesson.title}
                  </span>
                </button>
              );
            })}
          </React.Fragment>
        ))}
      </TabMenuVertical.List>
    </TabMenuVertical.Root>

    <Divider.Root className="" />

    {/* Индикатор прогресса */}
    <div className="mt-2 pt-3">
      <div className="mb-3 flex items-center justify-between">
        <span className="text-paragraph-xs text-text-sub-600">
          Прогресс
        </span>
        <span className="text-paragraph-xs text-text-sub-600">
          {Math.round(
            ((course.completedLessons ?? 0) / course.lessonsCount) * 100,
          )}
          %
        </span>
      </div>
      <div className="h-1.5 w-full overflow-hidden rounded-full bg-bg-soft-200">
        <div
          className="h-full rounded-full bg-primary-base"
          style={{
            width: `${Math.round(((course.completedLessons ?? 0) / course.lessonsCount)
* 100)}%`,
          }}
        ></div>
      </div>
    </div>
  </div>

  {/* Содержимое урока */}
  <div className="flex h-full flex-1 flex-col">
    <div className="w-full max-w-3xl p-6 h-full flex flex-col">
      <header className=" flex items-center justify-between">
```

Продолжение листинга З.3

```
            {variant === "modal" && (
              <Modal.Close className="flex size-10 items-center justify-center rounded-full
hover:bg-bg-weak-50">
                <RiCloseLine className="size-6 text-text-soft-400" />
              </Modal.Close>
            )}
          </header>

          {/* Контент урока */}
          <div  className="flex-1  overflow-y-auto  pr-2  [&::-webkit-scrollbar]:w-1  [&::-
webkit-scrollbar-track]:bg-transparent    [&::-webkit-scrollbar-thumb]:rounded-full    [&::-
webkit-scrollbar-thumb]:bg-stroke-soft-200 mb-4">
            <div className="prose prose-lg max-w-none">
              <div
                dangerouslySetInnerHTML={{
                  __html: formatContent(activeLesson.content),
                }}
              />
            </div>
          </div>

          {/* Кнопки навигации */}
          {isLastLessonInCourse() ? (
            <div className="mt-auto flex justify-end">
              <Button.Root
                variant="primary"
                mode="filled"
                onClick={handleCompleteCourse}
                className="flex items-center gap-2"
              >
                Завершить
                <Button.Icon as={RiCheckDoubleLine} />
              </Button.Root>
            </div>
          ) : hasNextLesson() && (
            <div className="mt-auto flex justify-end">
              <Button.Root
                variant="primary"
                mode="lighter"
                onClick={goToNextLesson}
                className="flex items-center gap-2"
              >
                Далее
                <Button.Icon as={RiArrowRightLine} />
              </Button.Root>
            </div>
          )}
        </div>
      </div>
    </div>
  );

  // Выбираем тип модального окна в зависимости от переданного варианта
  if (variant === "command") {
```

Окончание листинга З.3

```
    return (
      <CommandMenu.Dialog
        open={isOpen}
        onOpenChange={handleOpenChange}
        className="h-[680px] max-w-[980px] p-0"
      >
        <CommandMenu.DialogTitle className="sr-only">
          {course.title}
        </CommandMenu.DialogTitle>
        {contentComponent}
      </CommandMenu.Dialog>
    );
  }
  return (
    <Modal.Root open={isOpen} onOpenChange={handleOpenChange}>
      <Modal.Content className="h-[680px] w-[900px] max-w-[90vw] overflow-hidden p-0">
        <Modal.Title className="sr-only">{course.title}</Modal.Title>
        {contentComponent}
      </Modal.Content>
    </Modal.Root>
  );
};

// Вспомогательная функция для форматирования markdown-контента
function formatContent(content: string): string {
  // Замена markdown на HTML
  let formatted = content
    .replace(/^# (.*)$/gm, "<h1 style='font-size: 1.25rem; font-weight: 600; letterSpacing:
'0em'; margin-bottom: 0.75rem; margin-top: 1.25rem; color:#171717'>$1</h1>")
    .replace(/^##    (.*)$/gm,    "<h2    style='font-size:    1.125rem;    font-weight:    600;
letterSpacing: '0em'; margin-bottom: 0.75rem; margin-top: 1.25rem; color:#171717'>$1</h2>")
    .replace(/^### (.*)$/gm, "<h3 style='font-size: 1rem; font-weight: 600; letterSpacing:
'0em'; margin-bottom: 0.75rem; margin-top: 1.25rem; color:#171717'>$1</h3>")
    .replace(/\*\*(.*?)\*\*/g, "<strong class='font-bold'>$1</strong>")
    .replace(/\*(.*?)\*/g, "<em class='italic'>$1</em>")
    .replace(/!\[(.*?)\]\((.*?)\)/g, '<img alt="$1" src="$2" class="my-4 rounded-lg max-w-
full" />')
    .replace(/\[(.*?)\]\((.*?)\)/g,       '<a       href="$2"       class="text-primary-base
hover:underline">$1</a>')
    .replace(/^- (.*?)$/gm, "<li class='mb-1'>$1</li>")
    .replace(/^(\d+)\. (.*?)$/gm, "<li class='mb-0'>$2</li>");

  // Замена списков
  formatted = formatted
    .replace(
      /<li>.*?<\/li>(\s*<li>.*?<\/li>)+/g,
      (match) => `<ul class='list-disc pl-6 my-4 space-y-1'>${match}</ul>`,
    )
    .replace(/\n\n/g, "<p class='mb-4'></p>");

  return formatted;
}
```

ПРИЛОЖЕНИЕ И
ИСХОДНЫЙ КОД МОДУЛЯ КОНСТРАСТНОСТИ

Листинг И.1 – Исходный код утилит доступности accessibility-utils.ts

```
/**
 * Утилиты для проверки доступности
 * Содержит функции для расчета и оценки контраста между цветами
 * по стандарту APCA (Advanced Perceptual Contrast Algorithm) для WCAG 3.0
 */

/**
 * Реализация алгоритма APCA (Advanced Perceptual Contrast Algorithm) для проверки контраста
по WCAG 3.0
 *
 * @param textColorHex - Цвет текста в формате hex
 * @param backgroundColorHex - Цвет фона в формате hex
 * @returns Значение контраста в диапазоне от -108 до 106 (абсолютное значение > 60
считается хорошим)
 */
export function calculateAPCAContrast(
  textColorHex: string,
  backgroundColorHex: string,
): number {
  // Функция для преобразования hex в RGB
  const hexToRGBValues = (hex: string): [number, number, number] => {
    // Удаляем # если он есть
    const cleanHex = hex.replace("#", "");

    // Получаем RGB значения
    const r = parseInt(cleanHex.substring(0, 2), 16);
    const g = parseInt(cleanHex.substring(2, 4), 16);
    const b = parseInt(cleanHex.substring(4, 6), 16);

    return [r, g, b];
  };

  // Функция для преобразования sRGB в линейный RGB
  const sRGBtoLinear = (val: number): number => {
    const v = val / 255;
    if (v <= 0.04045) {
      return v / 12.92;
    }
    return ((v + 0.055) / 1.055) ** 2.4;
  };

  // Получаем RGB значения для текста и фона
  const [textR, textG, textB] = hexToRGBValues(textColorHex);
  const [bgR, bgG, bgB] = hexToRGBValues(backgroundColorHex);

  // Преобразуем в линейный RGB
  const linearTextR = sRGBtoLinear(textR);
  const linearTextG = sRGBtoLinear(textG);
  const linearTextB = sRGBtoLinear(textB);
```

Продолжение листинга И.1

```
  const linearBgR = sRGBtoLinear(bgR);
  const linearBgG = sRGBtoLinear(bgG);
  const linearBgB = sRGBtoLinear(bgB);

  // Вычисляем яркость (Y)
  const textY =
    0.2126 * linearTextR + 0.7152 * linearTextG + 0.0722 * linearTextB;
  const bgY = 0.2126 * linearBgR + 0.7152 * linearBgG + 0.0722 * linearBgB;

  // Определяем полярность (текст темнее или светлее фона)
  const polarity = bgY > textY ? 1 : -1;

  // Коэффициенты APCA
  const SAPC_Y_LIGHT = 1.14;
  const SAPC_Y_DARK = 0.56;
  const SAPC_OFFSET_LIGHT = 0.027;
  const SAPC_OFFSET_DARK = 0.027;

  // Вычисляем контраст по формуле APCA
  let contrast: number;

  if (polarity === 1) {
    // Текст темнее фона
    contrast = bgY ** SAPC_Y_LIGHT - textY ** SAPC_Y_DARK;
    contrast =
      contrast < SAPC_OFFSET_LIGHT ? 0 : (contrast - SAPC_OFFSET_LIGHT) * 100;
  } else {
    // Текст светлее фона
    contrast = bgY ** SAPC_Y_DARK - textY ** SAPC_Y_LIGHT;
    contrast =
      contrast < SAPC_OFFSET_DARK ? 0 : (contrast - SAPC_OFFSET_DARK) * -100;
  }

  return Math.round(contrast * 100) / 100;
}

/**
 * Оценивает контраст по APCA и возвращает текстовое описание уровня доступности
 * @param contrast - Значение контраста, полученное из calculateAPCAContrast
 * @returns Объект с оценкой контрастности, описанием и цветом для отображения
 */
export function evaluateAPCAContrast(contrast: number): {
  level: string;
  description: string;
  color: string;
} {
  const absContrast = Math.abs(contrast);

  if (absContrast >= 90) {
    return {
      level: "AAA+",
      description: "Отлично! Подходит для мелкого текста",
      color: "#1FC16B", // зеленый
    };
```

Окончание листинга И.1

```
  } else if (absContrast >= 75) {
    return {
      level: "AAA",
      description: "Очень хорошо. Подходит для основного текста",
      color: "#1FC16B", // зеленый
    };
  } else if (absContrast >= 60) {
    return {
      level: "AA",
      description: "Хорошо. Подходит для больших текстов",
      color: "#84cc16", // желто-зеленый
    };
  } else if (absContrast >= 45) {
    return {
      level: "A",
      description: "Удовлетворительно. Для крупных заголовков",
      color: "#F6B51E", // желтый
    };
  } else if (absContrast >= 30) {
    return {
      level: "Низкий",
      description: "Только для декоративных элементов",
      color: "#f97316", // оранжевый
    };
  } else {
    return {
      level: "Плохой",
      description: "Недостаточный контраст",
      color: "#FB3748", // красный
    };
  }
}
```

Листинг И.2 – Исходный код отображения контраста ContrastDisplay.tsx

```
"use client";

/**
 * Компонент отображает контраста между цветом объекта и фоном под ним по стандарту APCA.
 * Используется для оценки доступности и визуального качества дизайна на холсте.
 */

import React, { useState, useEffect } from "react";
import {
  colorToCss,
  calculateAPCAContrast,
  evaluateAPCAContrast,
} from "~/utils";
import type { Layer as BaseLayer, Color } from "~/types";
import { LayerType } from "~/types";
import { useCanvas } from "~/components/canvas/helper/CanvasContext";
import { useStorage, useSelf } from "@liveblocks/react";
import * as Tooltip from "~/components/ui/tooltip";
```

Продолжение листинга И.2

```
import { RiCloseLine } from "@remixicon/react";

interface LiveObject<T> {
  get<K extends keyof T>(key: K): T[K];
}

type LayerWithFill = Extract<BaseLayer, { fill: Color | null }>;
type Layer = BaseLayer;

interface BackgroundInfo {
  name: string;
  color: string;
}

function isLiveObject<T>(obj: LiveObject<T> | T): obj is LiveObject<T> {
  return typeof obj === 'object' && obj !== null && 'get' in obj && typeof obj.get ===
'function';
}

function getLayerProperty<T extends Layer, K extends keyof T>(
  layer: LiveObject<T> | T,
  property: K
): T[K] | undefined {
  if (!layer) return undefined;

  if (isLiveObject(layer)) {
    return layer.get(property);
  }
  return layer[property];
}

// Основной компонент для отображения информации о контрасте
interface ContrastDisplayProps {
  colorHex: string;
  layer: LiveObject<Layer> | Layer;
}

export const ContrastDisplay: React.FC<ContrastDisplayProps> = ({
  colorHex,
  layer,
}) => {
  const { roomColor } = useCanvas();
  const [backgroundInfo, setBackgroundInfo] = useState<BackgroundInfo | null>(null);

  // Get the layer object from storage if a layer is selected
  const layers = useStorage((root) => root.layers);
  const layerIds = useStorage((root) => root.layerIds);
  const selectedId = useSelf((me) => me.presence.selection[0] ?? null);

  // Проверяем тип слоя - для изображений контраст не показываем
  const layerType = getLayerProperty(layer, "type");

  // Определяем какой слой находится под выделенным слоем
  useEffect(() => {
```

Продолжение листинга И.2

```
if (layerType === LayerType.Image) return;
if (!selectedId || !layers || !layerIds || !layer) return;

// По умолчанию это фон холста
const canvasBackground = roomColor ? colorToCss(roomColor) : "#EFEFEF";
let backgroundLayer: BackgroundInfo = { name: "холстом", color: canvasBackground };

// Получаем координаты выбранного слоя
const selectedLayer = selectedId ? layers.get(selectedId) : undefined;
if (!selectedLayer) {
  setBackgroundInfo(backgroundLayer);
  return;
}

// Получаем координаты выбранного слоя
const selectedX = Number(getLayerProperty(selectedLayer, "x") ?? 0);
const selectedY = Number(getLayerProperty(selectedLayer, "y") ?? 0);
const selectedWidth = Number(getLayerProperty(selectedLayer, "width") ?? 0);
const selectedHeight = Number(getLayerProperty(selectedLayer, "height") ?? 0);

// Проверяем все слои и ищем те, что находятся под выбранным
const centerX = selectedX + selectedWidth / 2;
const centerY = selectedY + selectedHeight / 2;

// Получаем z-порядок слоев
const layerIdsArray = Array.from(layerIds);

// Переменные для хранения найденных слоев
let foundParentFrame: LiveObject<Layer> | Layer | null = null;
let underlyingLayer: LiveObject<Layer> | Layer | null = null;

// Перебираем в обратном порядке, чтобы начать с верхних слоев
for (let i = layerIdsArray.length - 1; i >= 0; i--) {
  const id = layerIdsArray[i];
  if (id === selectedId) continue; // Пропускаем сам выбранный слой

  const curLayer = id ? layers.get(id) : undefined;
  if (!curLayer) continue;

  // Получаем координаты текущего слоя
  const curX = Number(getLayerProperty(curLayer, "x") ?? 0);
  const curY = Number(getLayerProperty(curLayer, "y") ?? 0);
  const curWidth = Number(getLayerProperty(curLayer, "width") ?? 0);
  const curHeight = Number(getLayerProperty(curLayer, "height") ?? 0);
  const curType = getLayerProperty(curLayer, "type");

  // Проверяем, находится ли центр выбранного слоя внутри текущего
  const isInside =
    centerX >= curX &&
    centerX <= curX + curWidth &&
    centerY >= curY &&
    centerY <= curY + curHeight;

  if (isInside) {
```

Продолжение листинга И.2

```
        // Если это фрейм, отмечаем его как родительский
        if (curType === LayerType.Frame) {
          foundParentFrame = curLayer;
        }

        // Сохраняем первый найденный слой как находящийся под выбранным
        underlyingLayer ??= curLayer;
      }
    }

    // Определяем приоритет отображения фона:
    // 1. Слой под объектом
    // 2. Родительская рамка
    // 3. Фон холста
    if (underlyingLayer) {
      const layerWithFill = layer;
      const layerFill = getLayerProperty(layerWithFill as LayerWithFill, "fill" as keyof
LayerWithFill);

      if (layerFill) {
        let layerName = "объектом";

        const layerType = getLayerProperty(layerWithFill, "type");
        if (layerType === LayerType.Rectangle) {
          layerName = "прямоугольником";
        } else if (layerType === LayerType.Ellipse) {
          layerName = "эллипсом";
        } else if (layerType === LayerType.Triangle) {
          layerName = "треугольником";
        } else if (layerType === LayerType.Text) {
          layerName = "текстом";
        }

        backgroundLayer = {
          name: layerName,
          color: colorToCss(layerFill as Color),
        };
      }
    } else if (foundParentFrame) {
      const frameWithFill = foundParentFrame as LiveObject<LayerWithFill> | LayerWithFill;
      const frameFill = getLayerProperty(frameWithFill, "fill");

      if (frameFill) {
        backgroundLayer = {
          name: "рамкой",
          color: colorToCss(frameFill),
        };
      }
    }

    setBackgroundInfo(backgroundLayer);
  }, [selectedId, layers, layerIds, layer, roomColor, layerType]);

  // Если нет данных о фоне, не показываем ничего
```

Окончание листинга И.2

```
  if (!backgroundInfo) return null;

  // Расчет контраста между объектом и фоном
  const contrastValue = calculateAPCAContrast(colorHex, backgroundInfo.color);
  const evalContrast = evaluateAPCAContrast(contrastValue);

  return (
    <>
      <div className="mt-1">
        <div className="flex items-center justify-between">
          <div className="flex items-center gap-2">
            <span className="text-paragraph-sm text-text-strong-950">Контраст APCA:</span>
          </div>
          <Tooltip.Provider>
            <Tooltip.Root>
              <Tooltip.Trigger asChild>
                <div className="flex items-center gap-1 cursor-help">
                  <span className="text-paragraph-sm">
                    {Math.abs(contrastValue).toFixed(0)}
                  </span>
                  {(evalContrast.level === 'Низкий' || evalContrast.level === 'Плохой') ? (
                    <RiCloseLine   style={{   color:   evalContrast.color   }}   size={16}
className="text-paragraph-sm text-text-strong-950" />
                  ) : (
                    <span
                      style={{ color: evalContrast.color }}
                      className="text-paragraph-sm"
                    >
                      {evalContrast.level}
                    </span>
                  )}
                </div>
              </Tooltip.Trigger>
              <Tooltip.Content size="small" variant='light' side='left'>
                <div className="max-w-[220px]">
                  <p className="text-label-sm">
                    WCAG  APCA:  {evalContrast.level  !==  'Низкий'  &&  evalContrast.level  !==
'Плохой' ? evalContrast.level : 'Недостаточный'} (
                    {Math.abs(contrastValue).toFixed(0)})
                  </p>
                  <p>{evalContrast.description}</p>
                </div>
              </Tooltip.Content>
            </Tooltip.Root>
          </Tooltip.Provider>
        </div>
      </div>
    </>
  );
};

export default ContrastDisplay;
```