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.

Программа для автоматизации деятельности фирмы по закупке оборудования

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ
ЮУРГУ-090301.2025.406 ПЗ ВКР

Руководитель работы,
к.т.н., доцент каф. ЭВМ
_____ В.А. Парасич
«___»_____2025 г.

Автор работы,
студент группы КЭ-406
_____ Д.Р. Аскаров
«___»_____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

студенту группы КЭ-406
Аскарову Данилу Руслановичу
обучающемуся по направлению
09.03.01 «Информатика и вычислительная техника»

1. **Тема работы: «**Программа для автоматизации деятельности фирмы по закупке оборудования» 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. Основные параметры системы:

- тип: веб-приложение для автоматизации закупки оборудования;
- поддерживаемые платформы: сервер с ОС Linux или Windows, веб-браузеры (Chrome, Firefox, Edge);
- протокол передачи данных: HTTP/HTTPS, REST API для возможной интеграции;
- источники данных: сайты поставщиков (Citilink, DNS, OnlineTrade, Регард, Торг-ПК).

3.2. Бизнес-требования:

- операторам – автоматический поиск оборудования и формирование конфигураций на основе заявок;
- фирме – сокращение временных и трудозатрат на обработку предложений поставщиков и оформление заказов.

3.3. Атрибуты качества:

- время сбора данных с одного сайта – не более 5 секунд при нормальных условиях соединения;
- отображение результатов поиска и конфигураций – в пределах 20–30 секунд;
- возможность масштабирования и расширения функциональности без переработки ядра системы.

3.4. Основной функционал приложения:

- автоматизированный парсинг информации о товарах с торговых площадок;
- сравнение цен, наличия и условий поставки по нескольким поставщикам;
- формирование и сохранение заявок на закупку, а также шаблонов конфигураций оборудования;
- административный интерфейс для управления товарами, ценами и заказами.

4. **Перечень подлежащих разработке вопросов:**

4.1. Аналитический обзор научно-технической, нормативной и методической литературы.

4.2. Разработка серверного программного модуля.

4.3. Разработка программного комплекса для закупки компьютерного оборудования.

4.4. Проведение тестирования разработанных модулей.

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

Руководитель работы _____/ *В.А. Парасич/*

Студент _____/ *Д.Р. Аскаров/*

КАЛЕНДАРНЫЙ ПЛАН

Этап	Срок сдачи	Подпись руководителя
Введение и обзор литературы	03.03.2025	
Разработка программного комплекса для закупки компьютерного оборудования	22.03.2025	
Проведение тестирования	12.04.2025	
Компоновка текста работы и сдача на нормоконтроль	22.05.2025	
Подготовка презентации и доклада	30.05.2025	

Руководитель работы _____/ *В.А. Парасич /*

Студент _____/ *Д.Р. Аскаров /*

Аннотация

Д.Р. Аскаров. Программа для автоматизации деятельности фирмы по закупке оборудования. – Челябинск: ФГАОУ ВО «ЮУрГУ (НИУ)», ВШ ЭКН; 2025, 113 с., 20 ил., библиогр. список – 46 наим.

Выпускная квалификационная работа посвящена разработке программного комплекса для автоматизации деятельности фирмы по закупке компьютерного оборудования.

В первой главе проведен анализ предметной области, рассмотрены существующие аналоги разрабатываемых приложений.

Во второй главе обоснованы функциональные и нефункциональные требования к программному комплексу, а также представлены критерии выбора основного программного обеспечения. Осуществлен выбор фреймворков, с учетом масштабируемости и возможности добавления новых функциональных компонентов.

Третья глава посвящена разработке модулей и базы данных программного комплекса для автоматизации закупки оборудования.

В четвертой главе проведено тестирование программного комплекса для автоматизации закупки компьютерного оборудования.

В пятой главе подведены итоги проделанной работы, сформулированы выводы о достигнутых результатах, а также обозначены направления дальнейшего развития программного комплекса. Рассмотрены перспективы масштабирования системы, интеграции с внешними сервисами расширения набора поддерживаемых поставщиков и добавления интеллектуальных модулей анализа.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Эффективное управление материально-техническим снабжением – одна из основ стабильной и бесперебойной работы любого современного предприятия, независимо от его отраслевой принадлежности. Особенно это становится актуальным в условиях стремительного технологического прогресса и активного внедрения цифровых решений.

На сегодняшний день ни одно предприятие – будь то производственная компания, образовательное учреждение, государственная организация или частная фирма – не может обойтись без использования компьютерной техники и сопутствующего оборудования.

Компьютеры, комплектующие и периферийные устройства стали неотъемлемой частью информационной инфраструктуры организации, влияя на все аспекты её деятельности: от документооборота и бухгалтерии до инженерных расчетов и аналитики.

Процесс закупки компьютерного оборудования – это сложная и многоэтапная процедура, включающая анализ текущих потребностей, формирование технического задания, выбор поставщиков, сбор и анализ коммерческих предложений, заключение договоров, контроль выполнения обязательств и последующий ввод оборудования в эксплуатацию.

Каждый из этих этапов требует не только технической грамотности, но и высокой степени координации между различными подразделениями предприятия: ИТ-отделом, бухгалтерией, отделом снабжения, юридической службой и другими участниками процесса.

До недавнего времени подобные закупки зачастую осуществлялись вручную, с применением электронных таблиц, стандартных офисных программ и бумажного документооборота.

Такой подход ограничивает скорость принятия решений, увеличивает вероятность ошибок, снижает прозрачность и усложняет контроль за выполнением контрактов. В условиях возрастания объемов информации и

роста требований к качеству и срокам поставок становится очевидной необходимость перехода к автоматизированным системам управления закупочной деятельностью.

Автоматизация закупок, особенно в сфере ИТ-оборудования, предоставляет целый ряд преимуществ. Она позволяет:

- стандартизировать процессы закупки и сделать их воспроизводимыми;
- минимизировать влияние человеческого фактора на принятие решений;
- повысить точность выбора оборудования с учетом технических требований;
- оперативно реагировать на изменение рыночных условий и появление новых технологий;
- осуществлять централизованный контроль над закупочной деятельностью;
- интегрировать процессы закупки с другими информационными системами предприятия, такими как системы учета, инвентаризации, технической поддержки и др.

Разработка программного комплекса для автоматизации закупки компьютерной техники требует всестороннего анализа текущего состояния дел в данной области, изучения существующих решений, выявления их сильных и слабых сторон, а также определения требований к будущей системе. Особое внимание должно уделяться возможностям интеллектуального анализа заявок, автоматическому подбору оборудования с учетом совместимости компонентов, учёту истории закупок и анализа работы поставщиков.

Целью данной дипломной работы является разработка специализированного программного комплекса для автоматизации закупки компьютеров, их комплектующих и периферийного оборудования, включающего инструменты для:

- формирования и управления заявками на закупку;

- поиска поставщиков и анализа коммерческих предложений;
- сравнения условий поставки и технических характеристик оборудования;
- ведения сопроводительной документации и отчетности;
- контроля выполнения контрактов и поставок.

Актуальность темы обусловлена стремительным обновлением ассортимента ИТ-оборудования, необходимостью регулярной модернизации технической базы организаций, повышенными требованиями к прозрачности закупочных процедур, а также необходимостью снижения издержек на приобретение техники без потери качества.

Задачи дипломной работы:

- изучить теоретические основы автоматизации процессов закупки компьютерного оборудования;
- провести анализ существующих программных решений;
- выявить недостатки и пробелы современных систем;
- сформировать требования к разрабатываемому программному комплексу;
- разработать архитектуру системы и реализовать ключевые функциональные модули;
- провести тестирование и оценку эффективности предложенного решения;
- разработать предложения по дальнейшему расширению функционала и масштабированию системы.

Объект исследования: процессы закупки компьютерной техники и периферии на предприятиях.

Предмет исследования: методы, алгоритмы и программные средства автоматизации закупочной деятельности в сфере ИТ-оборудования.

Результатом работы станет программное решение, способное существенно повысить эффективность и прозрачность закупок ИТ-

оборудования, сократить временные и финансовые издержки, связанные с анализом коммерческих предложений и выбором поставщиков, минимизировать влияние человеческого фактора на принятие решений, а также обеспечить более высокий уровень технологической обеспеченности организаций. Разрабатываемый программный комплекс будет включать модули автоматического сбора данных с торговых площадок, интеллектуального анализа и сравнения товаров, формирования конфигураций компьютерной техники с учётом совместимости компонентов и хранения истории заказов, что позволит предприятиям выстраивать более структурированные и управляемые закупочные процессы. Кроме того, интеграция с корпоративными системами учёта и возможность масштабирования платформы под разные категории оборудования обеспечат её практическую ценность в условиях цифровой трансформации.

1. АНАЛИТИЧЕСКИЙ ОБЗОР НАУЧНО-ТЕХНИЧЕСКОЙ, НОРМАТИВНОЙ И МЕТОДИЧЕСКОЙ ЛИТЕРАТУРЫ

1.1. ОСНОВНЫЕ ПОНЯТИЯ И ОПРЕДЕЛЕНИЯ АВТОМАТИЗАЦИИ ЗАКУПКИ КОМПЬЮТЕРНОГО ОБОРУДОВАНИЯ

Закупка оборудования – это процесс приобретения технических средств, необходимых для обеспечения функционирования предприятия. В контексте данной работы под оборудованием понимаются компьютеры, их комплектующие (процессоры, материнские платы, оперативная память, накопители, блоки питания, корпуса и т.д.) и периферийные устройства (мониторы, принтеры, сканеры, клавиатуры, мыши, устройства ИБП и другое вспомогательное оборудование).

Современные предприятия сталкиваются с необходимостью регулярного обновления и расширения ИТ-инфраструктуры [1], обусловленного: развитием технологий и ростом вычислительных потребностей;

- устареванием ранее приобретённой техники, выходом новых поколений оборудования;
- необходимостью соответствия нормативным и отраслевым стандартам (например, по информационной безопасности);
- обеспечением бесперебойной работы сервисов и поддержки удалённых или гибридных форматов работы сотрудников.

В условиях цифровой трансформации и повышения степени автоматизации бизнес-процессов особое значение приобретает качественное техническое оснащение. От выбора оборудования напрямую зависит производительность труда, эффективность выполнения служебных задач и конкурентоспособность компании в целом.

Процесс закупки компьютерного оборудования включает следующие ключевые этапы: анализ потребностей – формируется исходя из планов

развития ИТ-инфраструктуры, заявок от подразделений и текущих технических дефицитов;

- формирование технического задания (ТЗ) – включает описание требуемых параметров, ограничений по стоимости, условий поставки, совместимости с имеющимися решениями и эксплуатационными характеристиками;
- поиск поставщиков и получение коммерческих предложений – выполняется вручную или с помощью специализированного программного обеспечения, предусматривает сбор информации с торговых площадок, маркетплейсов и прямых запросов;
- сравнение предложений – осуществляется по набору критериев: цена, срок доставки, наличие на складе, условия гарантии, отзывы, опыт сотрудничества;
- принятие решения и заключение договора – включает юридическую и финансовую экспертизу документов, согласование с ответственными лицами, оформление сопроводительной документации;
- контроль исполнения поставки – отслеживание сроков, приемка оборудования с проверкой соответствия заявленным характеристикам, ввод в эксплуатацию.

Автоматизация закупки оборудования – это внедрение информационных технологий в процессы управления закупочной деятельностью, направленное на повышение эффективности [42], прозрачности и управляемости данных процессов. Автоматизация может охватывать как отдельные этапы, так и весь цикл закупки [3].

Ключевые понятия, связанные с автоматизацией закупок:

1. Электронная заявка – цифровая форма, содержащая перечень требуемого оборудования [29], обоснование потребности и согласование с ответственными подразделениями.
2. Электронный каталог – база данных, содержащая технические

спецификации и предложения от разных поставщиков. Каталог может включать фильтры по параметрам, брендам, цене, доступности и совместимости.

3. Механизмы согласования – встроенные процессы одобрения заявок, ТЗ и контрактов с учетом организационной структуры предприятия [2].
4. Алгоритмы подбора конфигураций – средства автоматического формирования совместимых сборок компьютеров, рабочих станций и серверов с учётом предполагаемой нагрузки, стандартов предприятия и совместимости компонентов [36].
5. Ранжирование поставщиков – оценка на основе многокритериального анализа [41], включающего как количественные (стоимость, сроки), так и качественные (надежность, лояльность, удобство взаимодействия) параметры.
6. Интеграция с ERP/CRM/ITSM системами – обеспечивает сквозную обработку информации, автоматическое формирование документов, учет остатков, автоматическое закрытие заявок по поставкам и списаниям.

Дополнительно в рамках автоматизации может реализовываться: ведение истории закупок с аналитикой (наиболее эффективные поставщики, динамика цен);

- формирование шаблонов типовых конфигураций техники под разные нужды (офис, разработка, графика, обучение) [40];
- проверка соответствия ТЗ корпоративным ИТ-стандартам и политикам безопасности;
- поддержка интеграции с государственными закупочными системами (например, ЕИС) [35];
- формирование автоматических уведомлений и контрольных точек (напоминания, дедлайны, статусы поставок).

Преимущества автоматизации закупки компьютерной техники:

- значительное сокращение времени на подготовку и согласование документов;
- повышение точности благодаря использованию шаблонов и встроенной логики согласования;
- снижение рисков закупки некачественного или несовместимого оборудования;
- возможность централизованного управления закупками в крупных и распределённых организациях;
- автоматическая генерация сопроводительной документации (контракты, акты, спецификации);
- повышение прозрачности, управляемости и ответственности за результат;
- формирование аналитической базы для планирования будущих закупок и оценки эффективности работы поставщиков.

Исходя из указанной выше информации, автоматизация закупки оборудования – это необходимое условие эффективной цифровой трансформации предприятия [45]. Она позволяет не только сократить издержки и ускорить процессы, но и выстраивать стратегические подходы к обновлению ИТ-инфраструктуры, способствуя повышению общей устойчивости и технологической зрелости организации.

1.2. ОБЗОР СУЩЕСТВУЮЩИХ РЕШЕНИЙ ПРОГРАММ ДЛЯ АВТОМАТИЗАЦИИ ЗАКУПКИ ОБОРУДОВАНИЯ

На современном рынке представлено множество программных решений, направленных на автоматизацию закупочной деятельности. Они различаются по функциональности, области применения, способам внедрения, масштабируемости и интеграции с другими корпоративными системами. В контексте закупки компьютерной техники особое значение имеют такие функции, как автоматический подбор конфигураций, интеграция с

поставщиками, мониторинг наличия оборудования, контроль цен и поддержка документооборота [4].

Среди наиболее распространённых классов программ для автоматизации закупок можно выделить:

ERP-системы (Enterprise Resource Planning) [33] – это мощные корпоративные платформы, охватывающие все ключевые бизнес-процессы. Они обеспечивают централизованное управление закупками, начиная от планирования и формирования заявок, заканчивая оплатой и логистикой поставок [5].

Например, SAP ERP предоставляет модуль MM (Materials Management), который охватывает весь жизненный цикл закупки: от потребности до поставки.

На рисунке 1 показана экранная форма SAP ERP.

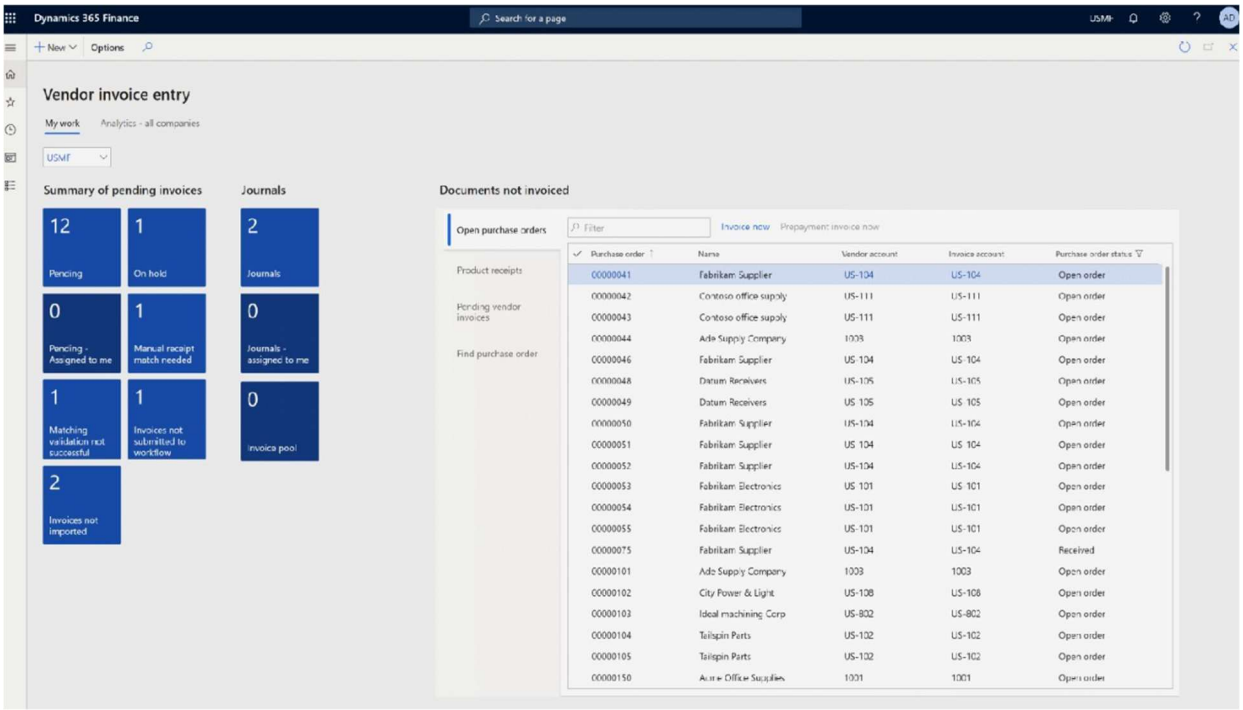

Рисунок 1 – Экранная форма SAP ERP

1С:ERP особенно популярен в российских компаниях и позволяет адаптировать процесс под нормативные требования, автоматизировать документооборот и вести учёт по специфике закупки ИТ-оборудования.

Эти системы поддерживают многоуровневое согласование заявок [7],

контроль лимитов бюджета, взаимодействие с поставщиками и логистическими службами.

На рисунке 2 показана экранная форма 1С:ERP.

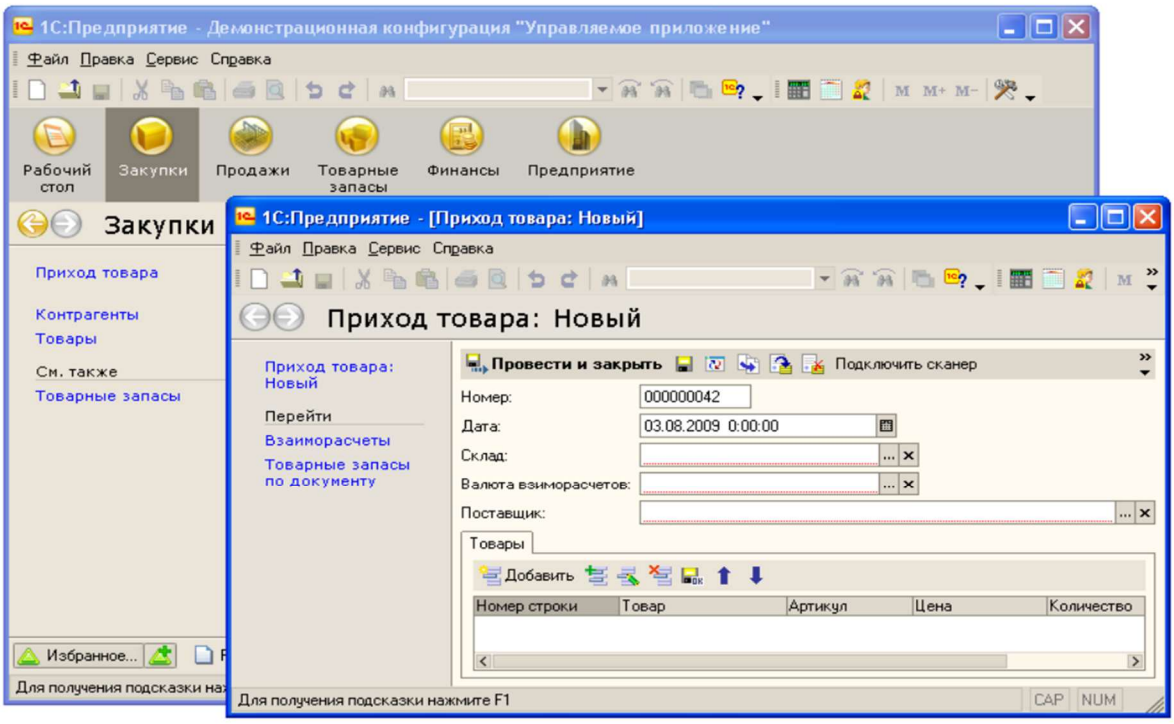

Рисунок 2 – Экранная форма1С:ERP

SRM-системы (Supplier Relationship Management) – специализированные инструменты для построения и управления отношениями с поставщиками [6].

Такие системы, как SAP Ariba, позволяют эффективно организовать процессы отбора, аккредитации и рейтинга поставщиков, что особенно важно при закупке компьютерной техники, где качество и срок поставки критичны.

SRM-системы предоставляют функциональность для автоматизации электронных тендеров, обработки предложений, расчёта total cost of ownership (TCO) и формирования контрактов [11].

Кроме того, они позволяют анализировать поведение поставщиков на протяжении длительного времени, выявлять наиболее надёжных и прогнозировать потенциальные риски в поставках [41].

На рисунке 3 показана экранная форма SAP Ariba.

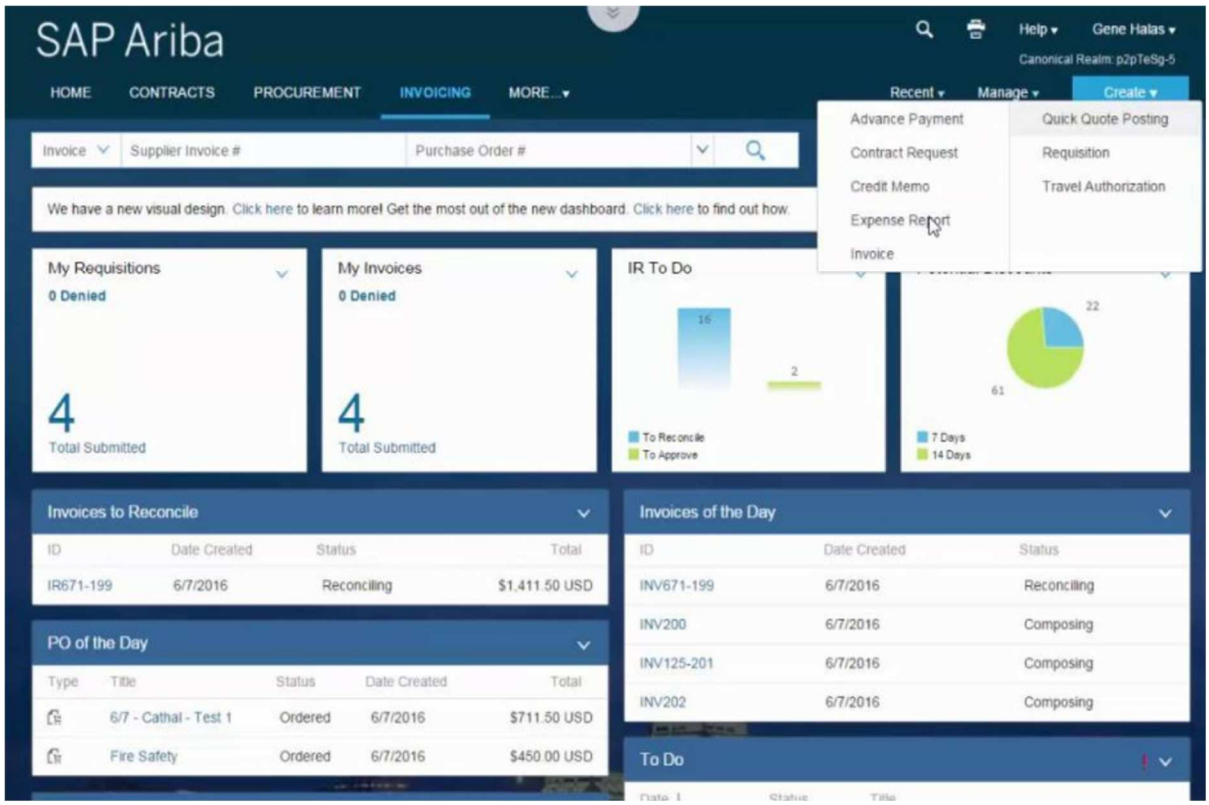

Рисунок 3 – Экранная форма SAP Ariba

Системы электронных торговых площадок (ЭТП) – важный элемент современной закупочной инфраструктуры. ЭТП предоставляют заказчикам возможность организовать закупки в виде аукционов, конкурсов, запросов котировок и предложений [9].

Системы, такие как ЕИС, B2B-Center, Сбербанк-АСТ, позволяют автоматизировать взаимодействие между организациями и поставщиками. B2B-Center [44], например, широко применяется для закупки ИТ-оборудования среди корпоративных клиентов.

Эти системы содержат обширные каталоги товаров, поддерживают API-интеграцию с внутренними системами заказчиков и позволяют отслеживать все стадии закупочного процесса в режиме реального времени.

На рисунке 4 показана экранная форма B2B-Center.

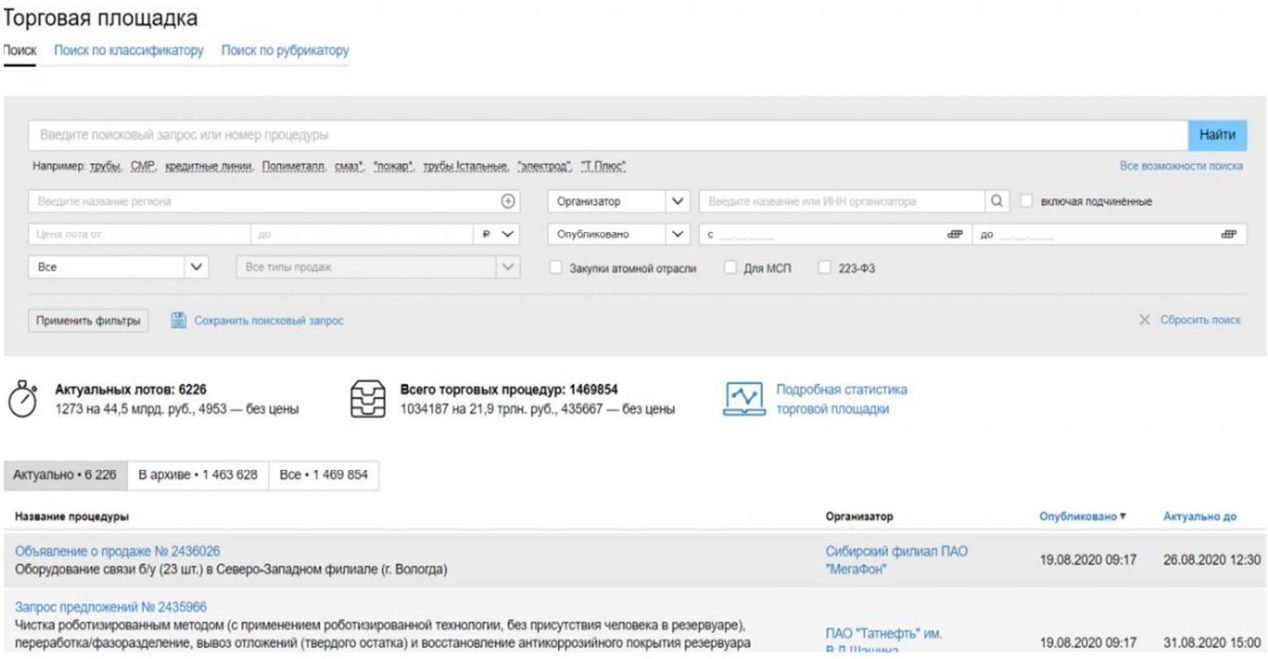

Рисунок 4 – Экранная форма B2B-Center

Системы управления ИТ-активами (IT Asset Management) – эти решения ориентированы на учёт, инвентаризацию и мониторинг состояния всех ИТ-ресурсов компании [8].

Примеры таких систем: Lansweeper, ManageEngine AssetExplorer и ServiceNow ITAM.

Они позволяют отслеживать жизненный цикл каждого устройства, своевременно планировать его замену и, таким образом, инициировать закупочные процессы.

Встроенные модули аналитики дают возможность спрогнозировать будущие потребности на основе текущего износа, загрузки и частоты отказов оборудования.

Кроме того, данные из этих систем часто используются для оценки эффективности ИТ-инфраструктуры и обоснования инвестиционных решений в новое оборудование.

На рисунке 5 показана экранная форма AssetExplorer.

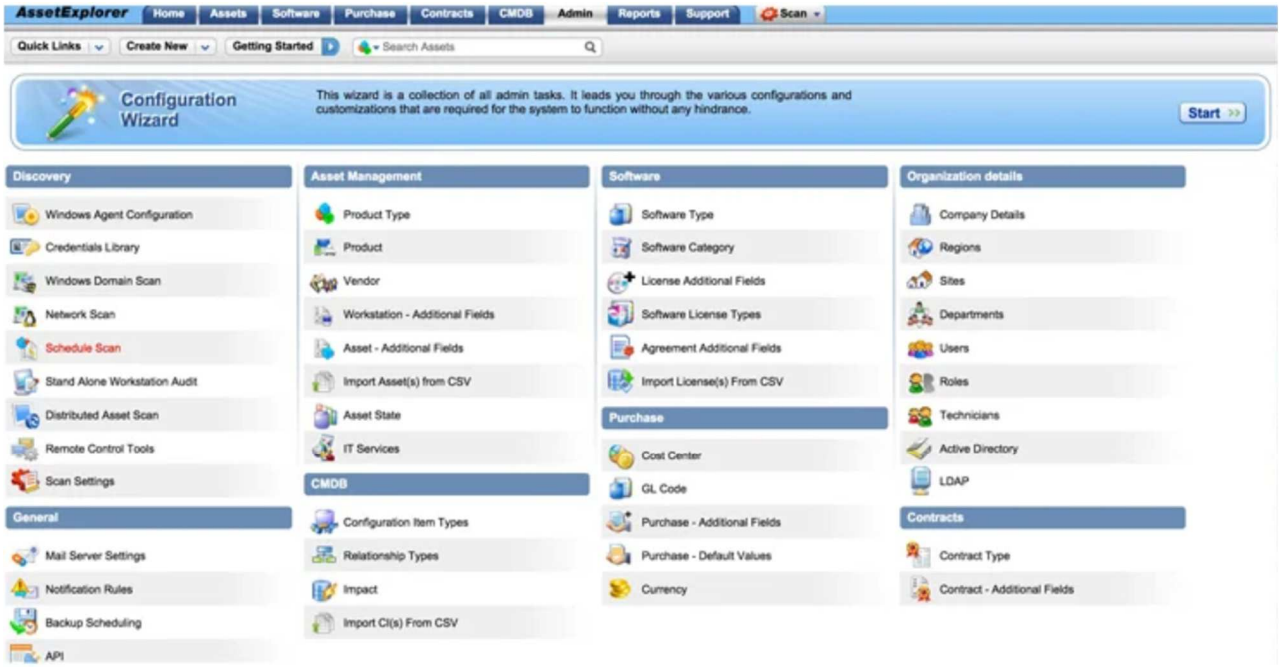

Рисунок 5 – Экранная форма AssetExplorer

Системы конфигурирования ИТ-оборудования – специализированные программные модули, предназначенные для автоматического подбора и сборки компьютеров и серверов [10].

Они особенно актуальны при необходимости массовых закупок или индивидуальной сборки систем под конкретные задачи.

Платформы, такие как конфигураторы от Merlion, Регард, Компьюлинк, позволяют учитывать совместимость компонентов, требуемую производительность, поддержку ПО, энергопотребление и другие параметры.

Эти решения существенно сокращают время на подготовку технического задания, а также исключают ошибки при подборе несовместимых компонентов.

В дополнение к этому, они предоставляют актуальные данные по ценам, наличию и срокам поставки, формируют спецификации в формате Excel или PDF и могут быть интегрированы с корпоративными ERP-системами [13].

На рисунке 6 показана экранная форма Merlion(Axapta).

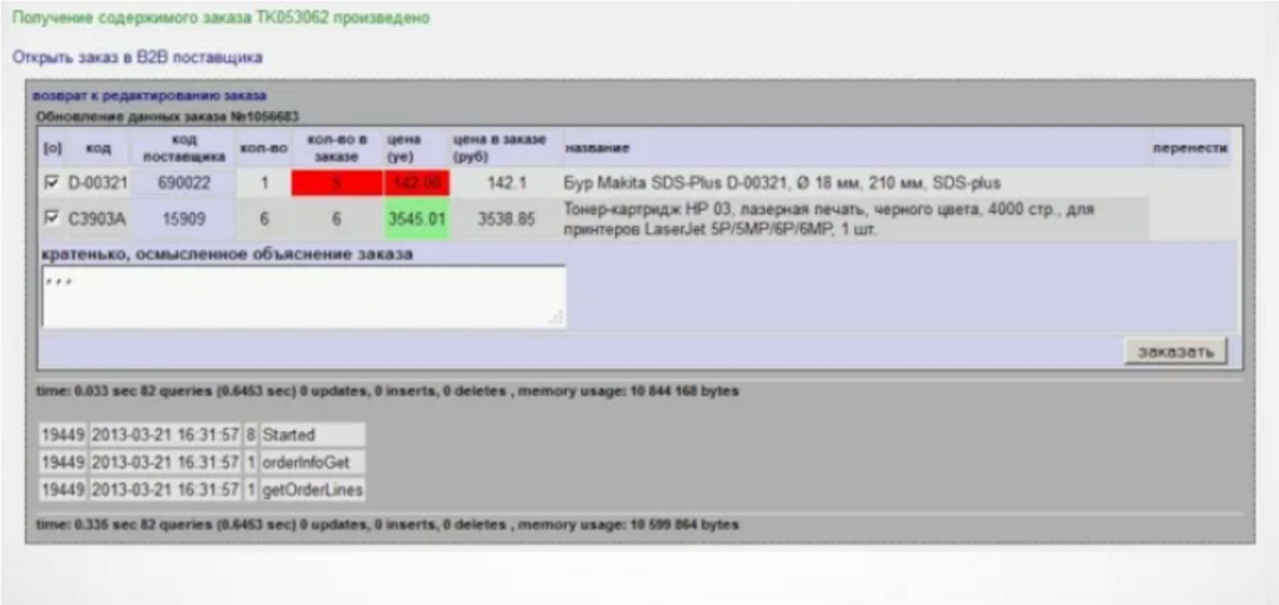

Рисунок 6 – Экранная форма Merlion(Axapta)

Облачные SaaS-решения для малого и среднего бизнеса – это простые в использовании онлайн-сервисы, не требующие сложной настройки и установки.

Примеры: Seldon.Basis, Zakupki360, Контур.Закупки.

Они предоставляют минимальный, но достаточный набор функций: формирование и согласование заявок, взаимодействие с поставщиками, шаблоны контрактов и актов, базу документов, отчётность [15].

Для МСП такие решения позволяют быстро автоматизировать закупки без необходимости крупных вложений в ИТ-инфраструктуру.

Некоторые из них предлагают интеграцию с государственными системами (например, с ЕИС) [46], что важно при работе с госсектором. Они часто используются для закупки оргтехники, периферии, расходных материалов и сервисного оборудования [12].

На рисунке 7 показана экранная форма Seldon.Basis.

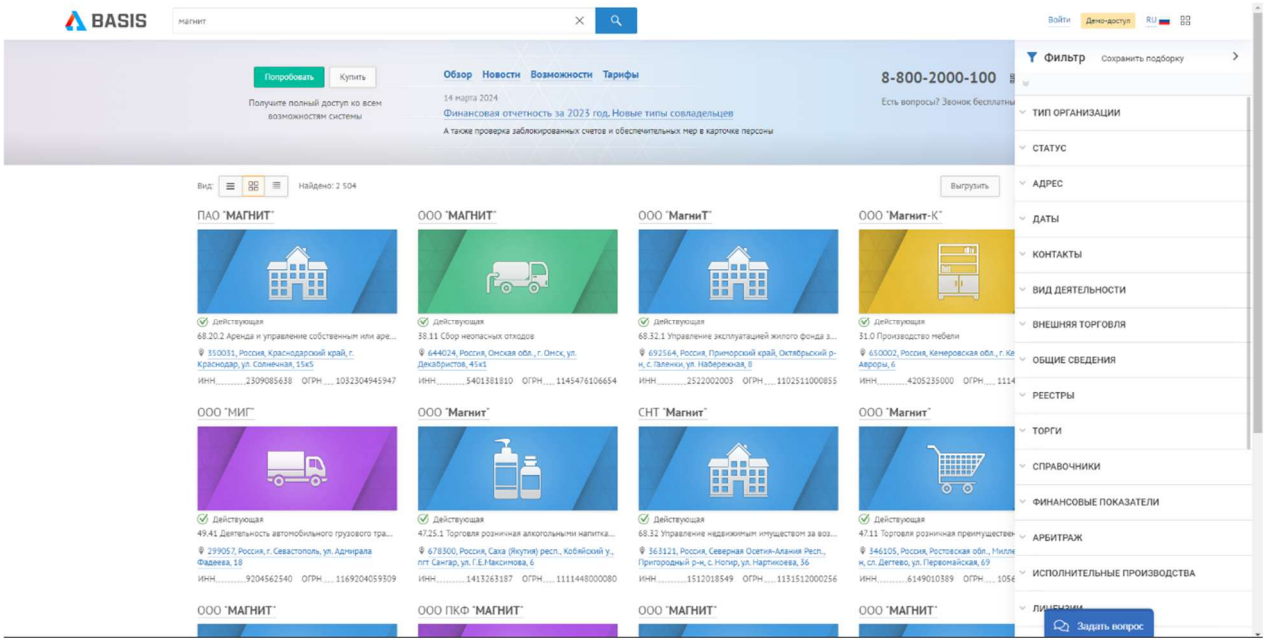

Рисунок 7 – Экранная форма Seldon.Basis

Каждая из представленных категорий программного обеспечения играет важную роль в автоматизации закупки компьютерного оборудования и комплектующих.

Выбор конкретного инструмента зависит от масштаба организации, уровня зрелости её бизнес-процессов, бюджета, а также специфики закупаемой техники и стратегических задач предприятия.

Внедрение таких решений позволяет не только повысить прозрачность и контролируемость закупок, но и существенно сократить временные и финансовые издержки, связанные с ручной обработкой информации и человеческим фактором.

1.3. ПРИНЦИПЫ И МЕТОДЫ АВТОМАТИЗАЦИИ ЗАКУПКИ ОБОРУДОВАНИЯ

Автоматизация закупки оборудования, включая компьютеры, комплектующие и периферийные устройства, представляет собой не только инструмент повышения эффективности бизнес-процессов, но и важный шаг к улучшению прозрачности, снижению издержек и обеспечению высокого уровня контроля за закупками. Этот процесс охватывает различные аспекты управления, от планирования потребностей и выбора поставщиков до

управления поставками и документооборотом. Правильная автоматизация позволяет минимизировать ошибки, ускорить процессы и повысить их прозрачность, а также улучшить взаимодействие между различными участниками процесса [15].

Автоматизация закупки оборудования должна быть основана на нескольких принципах, которые обеспечат её эффективность и успешное внедрение в организацию, например:

1. Централизация данных – целью централизованного подхода является создание единого информационного пространства, где собрана вся информация, касающаяся закупок. Это включает не только данные о наличии и стоимости оборудования, но и информацию о поставщиках, сроках поставки, условиях контрактов и пр. Централизованная база данных гарантирует, что все участники процесса имеют доступ к актуальной информации, что минимизирует вероятность ошибок и упрощает контроль за закупками [16].
2. Скорость и эффективность процесса – одной из основных задач автоматизации является значительное сокращение времени на выполнение стандартных операций. Например, ручной процесс составления запросов на закупку, согласования и утверждения заявок может занимать несколько дней, тогда как в автоматизированной системе это происходит за считанные минуты. Ускорение этих этапов влечет за собой сокращение времени, необходимого для реализации закупки, и, как следствие, повышение общей производительности.
3. Интеграция с внешними системами и поставщиками – важно, чтобы автоматизированные системы закупок могли интегрироваться с внешними платформами и поставщиками. Это значительно ускоряет процесс обработки заказов, обеспечивает более точные и актуальные данные о наличии товаров, ценах, условиях поставки и прочем. Интеграция с такими платформами, как Единая информационная система (ЕИС), помогает сделать закупки прозрачными и

соответствующими нормативным требованиям, а интеграция с API поставщиков позволяет автоматизировать процесс получения информации о стоимости и наличии товаров [17].

4. Прозрачность и контроль – прозрачностью процессов закупки является не только требование к организациям, но и основа для эффективного управления закупками. Важно, чтобы все этапы закупки, включая выбор поставщика, обработку заказов и выполнение платежей, были видимы и отслеживаемы для всех заинтересованных сторон. Это позволяет значительно повысить уровень контроля, минимизировать риски ошибок и мошенничества, а также улучшить отчетность.
5. Аналитика и отчетность – аналитика играет ключевую роль в автоматизации закупок. Современные системы закупок предоставляют подробную аналитику по каждому этапу процесса, позволяя анализировать эффективность работы с поставщиками, стоимость закупок, сроки поставок, а также выявлять тенденции и прогнозировать будущие потребности. Это позволяет не только оперативно принимать решения по закупкам, но и оптимизировать затраты и улучшать отношения с поставщиками.

Существует несколько методов автоматизации закупки оборудования, каждый из которых применяется в зависимости от размера компании, объема закупок, специфики закупаемого оборудования и целей бизнеса.

Внедрение автоматизированных решений зависит от специфики и потребностей организации.

Методы автоматизации закупки оборудования:

1. Метод электронных закупок – электронные закупки с помощью специализированных платформ позволяют автоматизировать процесс выбора и оценки поставщиков. Платформы, такие как B2B-Center, SAP Ariba или Сбербанк-АСТ, позволяют легко организовать тендеры, запросы котировок, конкурсы и аукционы. Электронные закупки помогают снизить затраты на организацию закупок и ускорить процесс

выбора поставщика. Система автоматического формирования тендеров, обработки заявок и проведения аукционов делает закупку более конкурентоспособной, а заказчику позволяет найти наилучшее предложение [20].

2. Метод интеграции с ERP-системами – интеграция автоматизированных решений для закупки оборудования с ERP-системами (например, SAP ERP, 1C:ERP) позволяет автоматизировать весь процесс – от планирования закупок до расчетов и отслеживания поставок. Эти системы могут отслеживать запасы [32], вести учет по каждой единице оборудования, автоматизировать запросы и заказ материалов. Система помогает следить за состоянием бюджета, управлять финансами и получать отчеты о выполнении планов закупок.
3. Метод конфигураторного подхода – для закупок технически сложного оборудования, например, серверов, рабочих станций или сетевых устройств, часто используются конфигураторы. Эти системы автоматически подбирают оборудование по заданным требованиям, таким как производительность, совместимость компонентов, энергопотребление и другие параметры [25]. Примеры таких платформ включают конфигураторы от Merlion, Компьюлинк и Регард, которые позволяют собрать оптимальную конфигурацию оборудования под конкретные задачи заказчика [18].
4. Метод прогнозирования потребностей – прогнозирование потребностей в оборудовании основывается на исторических данных и аналитике состояния текущих ИТ-ресурсов. Системы управления ИТ-активами, такие как Lansweeper или ServiceNow, помогают отслеживать статус каждого устройства в компании, анализировать его использование и производительность, а также прогнозировать, когда оно потребует замены [23]. Это позволяет заранее планировать закупки и обеспечивать бесперебойную работу компании.
5. Метод цифровизации документооборота – автоматизация

документооборота помогает ускорить процессы согласования, подписания и хранения контрактов [22], актов приема-передачи и других документов, связанных с закупками. Цифровые документы сокращают время на их обработку, исключают ошибки при вводе данных, а также облегчают хранение и поиск документов. Важно, чтобы системы обеспечивали возможность цифровой подписи и интеграции с корпоративными базами данных для хранения и дальнейшего использования документов [14].

Важной частью является выбор правильного метода. Для эффективной автоматизации необходимо учитывать несколько факторов: размер компании, специфику закупаемого оборудования [24], внутренние процессы, бюджет на внедрение и последующую эксплуатацию системы.

Например, для малых и средних предприятий может быть достаточно внедрения облачных решений с базовыми функциями для организации закупок и взаимодействия с поставщиками (например, Zakupki360 или Контур.Закупки).

Крупные компании, работающие с большими объемами закупок, могут выбрать более сложные решения [21], такие как ERP-системы или специализированные SRM-платформы.

Также стоит учитывать, что выбор метода автоматизации должен опираться на стратегические цели компании.

Например, если основной целью является сокращение времени на заключение договоров с поставщиками, то более эффективно будет использовать электронные торговые площадки и SRM-системы [37], обеспечивающие быструю обработку тендеров и предложений.

Если целью является оптимизация внутренних процессов, то лучше подойдут ERP-системы с интеграцией всех этапов закупки и учета.

Перспективы и вызовы в области автоматизации закупки оборудования. Автоматизация закупок будет продолжать развиваться, внедряя новые технологии, такие как искусственный интеллект, машинное обучение и

блокчейн. Эти технологии будут использованы для оптимизации закупок, повышения точности прогнозирования и минимизации ошибок. Искусственный интеллект сможет анализировать данные о рынке, предсказывать потребности в оборудовании и рекомендовать оптимальные поставки.

Однако, несмотря на все преимущества, внедрение автоматизации связано с рядом вызовов.

В частности, это проблемы интеграции новых решений с существующими информационными системами, необходимость обучения сотрудников работе с новыми инструментами и возможные высокие первоначальные затраты на внедрение.

Важно понимать, что автоматизация закупок не является конечной целью. Это лишь часть общего процесса трансформации бизнеса, который должен быть направлен на повышение гибкости и способности быстро адаптироваться к изменениям внешней среды [19].

Внедрение автоматизированных решений должно быть частью более широкой стратегии цифровой трансформации компании, которая затрагивает все аспекты работы.

Автоматизация закупки оборудования становится не только возможностью для бизнеса, но и необходимостью для достижения высокой конкурентоспособности, снижения издержек и улучшения качества обслуживания клиентов [21].

ВЫВОДЫ ПО ГЛАВЕ 1

В ходе анализа основных понятий и технологий, связанных с автоматизацией закупок, были рассмотрены ключевые элементы, которые влияют на успешность внедрения подобных решений в рамках различных бизнес-структур.

Одним из главных выводов является необходимость чёткого понимания того, что представляет собой автоматизация закупок. Это не просто внедрение

технологий, а комплексное изменение подхода к управлению закупочными процессами, включая улучшение процессов согласования, выбора поставщиков, обработки заказов и контроля исполнения договоров.

Принципы, такие как централизованный учет данных, прозрачность всех этапов закупки и интеграция с внешними системами, являются основой для построения эффективной системы автоматизации.

Программные решения, представленные на рынке, предлагают разнообразные подходы к автоматизации. Речь идет как о крупных комплексных системах, таких как ERP и SRM, так и более специализированных решениях для управления ИТ-активами или конфигурирования оборудования. Системы электронных торговых площадок также играют важную роль в создании конкурентных условий для закупки, предоставляя заказчикам возможность провести процедуры выбора и аккредитации поставщиков на основе прозрачных данных

Особое внимание в главе было уделено методам и принципам автоматизации. Методы, такие как электронные закупки, прогнозирование потребностей и интеграция с внешними и внутренними системами, позволяют значительно повысить скорость и точность принятия решений, снизить затраты и улучшить управление запасами. Внедрение таких методов требует не только технической составляющей, но и стратегического подхода, учитывающего специфику работы каждой организации

Важность правильного выбора метода автоматизации, который будет зависеть от масштаба бизнеса, объема закупок и специфики оборудования, подчеркивает, что нет универсального решения для всех организаций. Важно учитывать потребности и стратегические цели предприятия для выбора подходящей системы.

Автоматизация закупки оборудования представляет собой не только инструмент для оптимизации текущих процессов, но и важный элемент в цифровой трансформации организации. Внедрение правильных решений способствует повышению эффективности бизнеса, улучшению

взаимодействия с поставщиками, сокращению времени и затрат на закупки, а также повышению прозрачности и контроля на всех этапах процесса.

Выводы, сделанные в данной главе, демонстрируют, что правильный выбор технологий и методов автоматизации закупки является ключом к успешному развитию бизнеса в условиях современной цифровой экономики.

2. ВЫБОР СРЕДСТВ РЕАЛИЗАЦИИ ПРОГРАММНОГО КОМПЛЕКСА ДЛЯ АВТОМАТИЗАЦИИ ЗАКУПКИ КОМПЬЮТЕРНОГО ОБОРУДОВАНИЯ

2.1. ФУНКЦИОНАЛЬНЫЕ ТРЕБОВАНИЯ

Формирование требований является одним из важнейших этапов разработки любого программного комплекса.

Грамотно составленные требования позволят четко определить цели и функционал разрабатываемых программ: возможность ручного ввода и редактирования информации о закупке оператором;

- автоматический сбор данных о товарах с различных торговых площадок (Citilink, DNS, Регард, OnlineTrade, ТоргПЦ);
- сравнительный анализ поставщиков по критериям: цена, наличие, сроки поставки;
- представление результатов в виде ссылок на товары;
- возможность создания конфигурации целого персонального компьютера;
- возможность создания шаблонов готовой сборки компьютера и отдельно взятых комплектующих и заявок готовых к заказу.

2.2. НЕФУНКЦИОНАЛЬНЫЕ ТРЕБОВАНИЯ

Разрабатываемый программный комплекс должен соответствовать ряду нефункциональных требований, обеспечивающих его удобство использования, надежность и пригодность для последующего сопровождения.

К ним относятся: удобный веб-интерфейс для взаимодействия с системой;

- поддержка расширяемости (возможность добавления новых источников данных);
- минимальные требования к оборудованию и возможность локального

развертывания;

- защита от сбоев в процессе сбора данных;
- простота в обслуживании и сопровождении проекта;
- использование свободного программного обеспечения и открытых стандартов.

2.3. АНАЛИЗ ОСНОВНЫХ ТЕХНОЛОГИЧЕСКИХ РЕШЕНИЙ

На этапе проектирования программного комплекса необходимо было выбрать оптимальные средства реализации, учитывая поставленные функциональные и нефункциональные требования.

Ниже приведён анализ ключевых технологических компонентов: языка программирования, веб-фреймворка, системы управления базами данных, инструментов сбора данных, а также архитектурных решений.

Язык программирования:

1. Python.
2. Java.
3. C#.
4. JavaScript (Node.js).

В качестве языка программирования был выбран Python.

Преимущества: простой и читаемый синтаксис, снижающий вероятность ошибок;

- большое количество готовых библиотек и модулей;
- активное сообщество и хорошая документация;
- универсальность – подходит как для веб-разработки, так и для автоматизации и анализа данных.

Недостатки: относительно низкая скорость выполнения по сравнению с компилируемыми языками;

- ограничения многопоточности в стандартной реализации (из-за GIL).

Java и C#, которые обеспечивают высокую производительность и

строгую типизацию, что важно для масштабируемых корпоративных решений, однако требуют более сложной конфигурации и большего объема кода.

Node.js удобен при разработке высоконагруженных веб-приложений с большим количеством параллельных соединений, но менее удобен для реализации сложной логики обработки данных.

Веб-фреймворк:

1. Django.
2. Flask.
3. FastAPI.
4. Spring Boot (Java).
5. ASP.NET Core (C#).

В качестве веб-фреймворка был выбран Django.

Преимущества: содержит ORM, маршрутизацию, авторизацию, работу с шаблонами и административную панель;

- поддержка шаблона MVC (Model-View-Controller);
- хорошо структурированная архитектура;
- широко используем в корпоративной разработке.

Недостатки:

- относительно высокая сложность для небольших и микросервисных решений;
- меньшая гибкость по сравнению с легковесными фреймворками.

Flask и FastAPI – лёгкие и гибкие фреймворки, хорошо подходящие для микросервисов и REST API. Однако они требуют ручной настройки многих компонентов.

Spring Boot и ASP.NET Core обладают высокой производительностью и поддерживают сложные архитектурные решения, но связаны с более тяжёлой инфраструктурой и менее быстры в начальной разработке.

Система управления базами данных (СУБД):

1. PostgreSQL.
2. MySQL / MariaDB.
3. SQLite.
4. MongoDB (NoSQL).

В качестве СУБД был выбран PostgreSQL.

Преимущества: поддержка полноценных транзакций и ACID-свойств;

- расширяемость (пользовательские функции, типы данных, индексы);
- надёжность и устойчивость при работе с большими объемами данных;
- соответствие стандартам SQL.

Недостатки: требует больше ресурсов, чем SQLite;

- более сложная начальная настройка.

MySQL и MariaDB также являются популярными СУБД, однако PostgreSQL превосходит их в поддержке стандартов и расширяемости.

SQLite подходит для небольших и автономных решений, но не поддерживает полноценную многопользовательскую работу.

MongoDB – документно-ориентированная СУБД, хорошо подходит для гибких, слабо структурированных данных, но менее надёжна при сложных реляционных связях.

Средства парсинга и сбора данных:

1. BeautifulSoup + Requests.
2. Scrapy.
3. Selenium.
4. Puppeteer.

В качестве средства парсинга и сбора данных были выбраны BeautifulSoup и Requests.

Преимущества: простота в использовании и настройке;

- поддержка большинства HTML-структур;
- широкое сообщество и множество примеров;
- быстрая реализация базового парсера.

Недостатки: не подходит для работы с динамическим контентом, загружаемым через JavaScript;

- ограниченные средства автоматизации и масштабирования по сравнению со специализированными фреймворками.

Scrapy – мощный фреймворк для масштабируемого и асинхронного парсинга, но требует дополнительного времени на освоение.

Selenium и Puppeteer позволяют работать с полнофункциональными браузерами и динамическим контентом, но являются более ресурсоёмкими решениями.

Архитектурный подход: монолитная архитектура;

- микросервисная архитектура;
- клиент-сервер с REST API.

В качестве архитектурного подхода была выбрана монолитная архитектура.

Преимущества: простота реализации и развёртывания;

- целостность логики приложения;
- упрощённое тестирование и отладка;
- подходит для начальной версии продукта (MVP).

Недостатки: трудности масштабирования отдельных компонентов;

- повышенные риски при внесении изменений – затрагивается вся система;
- ограниченная гибкость при расширении функциональности.

Микросервисная архитектура обеспечивает масштабируемость, гибкость и отказоустойчивость, но требует сложной инфраструктуры и управления межсервисным взаимодействием.

REST API на основе клиент-серверной архитектуры позволяет разносить интерфейс и бизнес-логику, что удобно при интеграции с мобильными и веб-клиентами, но требует отдельной реализации и поддержки двух независимых частей.

ВЫВОДЫ ПО ГЛАВЕ 2

В данной главе были определены как функциональные, так и нефункциональные требования к разрабатываемому программному комплексу, что позволило чётко сформулировать цели, задачи и критерии успешности проекта.

Под функциональными требованиями рассматривались ключевые возможности системы – от автоматического поиска оборудования до формирования заказов и отображения результатов пользователю.

Нефункциональные требования охватывали такие важные аспекты, как надёжность, производительность, масштабируемость, безопасность и удобство интерфейса.

На основе сформулированных требований был проведён анализ современных технологических решений, после чего осуществлён обоснованный выбор инструментов реализации.

В качестве языка программирования выбран Python – один из наиболее распространённых и удобных для быстрой разработки веб-приложений.

Для построения серверной логики и управления моделью данных использован мощный фреймворк Django, предоставляющий встроенные механизмы маршрутизации, аутентификации и администрирования.

В качестве системы управления базами данных выбрана PostgreSQL, обладающая высокой надёжностью, поддержкой транзакций и расширенными возможностями работы с реляционными данными.

Для автоматизированного получения данных о товарах были выбраны библиотеки, поддерживающие парсинг HTML, такие как BeautifulSoup и requests.

Принятые технологические решения позволили заложить прочную архитектурную основу для разработки устойчивой, расширяемой и простой в сопровождении системы. Это обеспечит надёжную автоматизацию ключевых процессов закупки компьютерного оборудования, сокращение ручного труда,

минимизацию ошибок при поиске и сравнении поставщиков, а также повышение общей эффективности закупочной деятельности.

Глава 2 не только определила концептуальные рамки и требования к проекту, но и зафиксировала архитектурные и технологические ориентиры, которые определяют направление всей дальнейшей разработки программного комплекса.

3. ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО КОМПЛЕКСА ДЛЯ АВТОМАТИЗАЦИИ ЗАКУПКИ ОБОРУДОВАНИЯ

Проектирование программного комплекса является важнейшим этапом создания эффективной системы автоматизации закупки компьютерного оборудования. Именно на этом этапе формируется основа будущего программного решения – определяется внутренняя структура приложения, выделяются ключевые модули, разрабатывается архитектура базы данных и продумывается логика взаимодействия между всеми компонентами системы. Грамотно выполненное проектирование позволяет обеспечить согласованность всех элементов, упростить дальнейшую разработку и обеспечить масштабируемость и устойчивость системы при реальной эксплуатации.

В ходе проектирования учитывались как текущие, так и перспективные требования к системе. Были проанализированы типовые сценарии работы операторов, определены возможные пути расширения функциональности, заложены механизмы обработки ошибок и обеспечения устойчивости к отказам внешних сервисов. Особое внимание уделялось вопросам удобства использования, надёжности хранения данных, а также интеграции с внешними источниками информации – сайтами поставщиков оборудования.

Конечная цель проектирования – создание надёжного, масштабируемого и удобного в использовании программного комплекса, полностью соответствующего как функциональным, так и нефункциональным требованиям. Именно проектные решения, заложенные на этом этапе, обеспечивают основу для эффективной реализации, тестирования и дальнейшего сопровождения системы.

На рисунке 8 представлена блок-схема алгоритма действия программы.

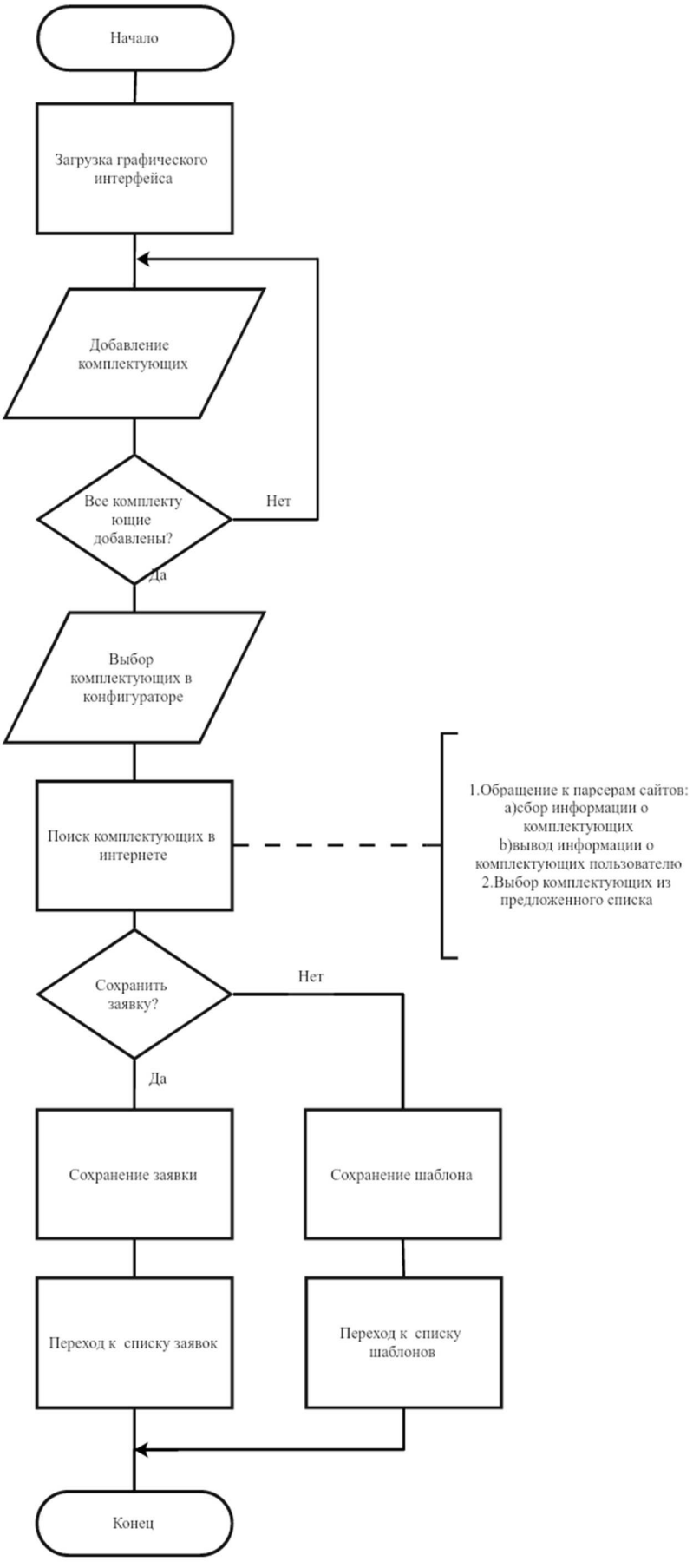

Рисунок 8 – Блок-схема алгоритма действия программы

3.1. МОДУЛИ ПРОГРАММНОГО КОМПЛЕКСА ДЛЯ АВТОМАТИЗАЦИИ ЗАКУПКИ КОМПЬЮТЕРНОГО ОБОРУДОВАНИЯ

Программный комплекс разделён на несколько функциональных модулей, каждый из которых выполняет чётко определённые задачи и взаимодействует с другими компонентами системы.

Парсеры сайтов – автоматизируют процесс сбора информации о товарах и ценах с сайтов поставщиков.

Парсеры в программном комплексе выполняют автоматизированный сбор данных о товарах с веб-сайтов поставщиков. Каждый парсер реализован в виде отдельного модуля и ориентирован на конкретный сайт с учётом его HTML-структуры, системы навигации и возможных ограничений против автоматических запросов.

Модули парсеров располагаются в директории app/diploma/ и включают следующие файлы: parsingCitilink.py;

- parsingDNS.py;
- parsingONLINETRADE.py;
- parsingREGARD2.py;
- parsingTorgPC.py.

Каждый из этих файлов соответствует одному поставщику и реализует однотипную логику за исключением парсинга сайта Citilink.ru, у него есть отличия от остальных парсингов: загрузку HTML-страниц, парсинг карточек товаров, очистку данных и сохранение в базу.

В листингах кода А.1, А.2, А.3, А.4, А.5 приложения А показан листинг кода парсеров.

Основные функции парсеров: формирование и отправка HTTP-запросов – запросы к сайтам осуществляются с заголовками, имитирующими обычный браузер, для избежания блокировки:

- парсинг HTML-страниц с помощью BeautifulSoup – загруженная страница разбирается на элементы для дальнейшего извлечения нужных

блоков:

- извлечение данных о товарах – из карточки каждого товара извлекается имя, цена, URL;
- очистка и нормализация данных – удаляются пробелы, спецсимволы и знаки валют, а цены преобразуются в числовой формат;
- сохранение информации в базу данных – результаты сохраняются через модели Django, определённые в models.py, функция update_or_create() позволяет избежать дублирования данных и при этом автоматически обновляет старые записи;

Особенности реализации:

- заголовки и защита от блокировок – во всех парсерах используются заголовки User-Agent, задержки и ограничения по времени ответа (timeout). Это позволяет обходить блокировку от сайтов и имитировать действия обычных пользователей;
- обработка ошибок – для устойчивости к сбоям реализована система перехвата исключений;
- модульная структура – каждый сайт обрабатывается своим модулем. Такой подход упрощает поддержку и расширение системы: при изменении структуры сайта достаточно обновить один конкретный модуль;
- единый формат функции запуска – во всех парсерах реализована функция parse() или run(), что позволяет централизованно вызывать их при необходимости.

Запуск парсеров parsingDNS.py, parsingONLINETRADE.py, parsingREGARD2.py и parsingTorgPC.py реализованы по единому шаблону.

В каждом из файлов определена функция parse(), выполняющая: итерацию по страницам каталога – на сайтах поставщика каталог товаров разбит на страницы. Парсер обходит несколько страниц (обычно 3–5) путём генерации URL с параметром page. Это позволяет собрать больше товаров, чем

только с первой страницы;

- сбор HTML-данных – для каждой страницы выполняется HTTP-запрос с заголовками, имитирующими браузер. Полученный HTML-код сохраняется в переменную и передаётся для парсинга;
- обработку карточек товаров – HTML-страница разбирается с помощью BeautifulSoup. Из неё извлекаются блоки-карточки, содержащие информацию о каждом товаре. Далее из карточек достаются нужные поля: название, цена, ссылка, наличие;
- сохранение результатов в базу данных – после извлечения информации она сохраняется в базу данных с помощью Django ORM. Используется метод update_or_create(), который обновляет существующую запись или создаёт новую, если такой ещё нет. Это позволяет избежать дублирования.

В отличие от других парсеров, parsingCitilink.py использует входной параметр category, что позволяет запускать его для разных товарных групп.

Такой подход позволяет повторно использовать одну и ту же функцию для различных категорий товаров (например, видеокарты, процессоры, SSD) и упрощает масштабирование – не нужно создавать отдельный файл под каждую категорию.

URL для каждой страницы генерируется динамически с учётом категории и номера страницы. Цикл перебирает страницы каталога:

Например, при вызове parse('videokarty') будут собраны страницы: «https://www.citilink.ru/catalog/videokarty/?p=1», «https://www.citilink.ru/catalog/videokarty/?p=2» и т. д.

HTML парсится библиотекой BeautifulSoup.

Товары расположены в product_data__gtm-js, откуда берутся имя, цена и URL, обработка price_tag через if позволяет избежать ошибки, если товар временно недоступен или блок отсутствует

Модели Django – определяют структуру данных и обеспечивают работу с базой данных через объектно-ориентированный подход.

Модели в Django – это фундамент взаимодействия с базой данных. Они позволяют описать структуру таблиц с помощью Python-классов, определяя поля, связи и бизнес-логику объектов.

Основные модели:

1. Product: идентификатор, название, категория, артикул.
2. Supplier: наименование, URL сайта, приоритет.
3. ProductPrice: внешний ключ на товар и поставщика, цена, наличие, ссылка, дата обновления.

Ключевые особенности:

1. Использование ForeignKey обеспечивает логичные связи между записями и поддержку каскадного удаления.
2. Индексы на часто используемых полях (product_id, date) ускоряют фильтрацию.
3. Методы моделей позволяют быстро извлекать свежие предложения, сортировать их по цене и наличию.

В таблице 1 представлена структура основных моделей базы данных.

Таблица 1 – Структура основных моделей базы данных.

Таблица	**Поле**	**Тип данных**	**Назначение**
Product	id	AutoField (PK)	Первичный ключ
	name	CharField(255)	Название товара
	category	CharField(255)	Категория товара
	article	CharField(100)	Артикул
Supplier	id	AutoField (PK)	Первичный ключ
	name	CharField(100)	Название поставщика
	url	URLField	Сайт поставщика
	priority	IntegerField	Приоритет

Продолжение таблицы 1

Таблица	Поле	Тип данных	Назначение
ProductPrice	id	AutoField (PK)	Первичный ключ
	product	ForeignKey → Product	Связь с товаром
	supplier	ForeignKey → Supplier	Связь с поставщиком
	price	DecimalField(10, 2)	Цена
	availability	BooleanField	Наличие
	url	URLField	Ссылка на товар
	updated_at	DateTimeField	Дата обновления
ComponentType	id	AutoField (PK)	Первичный ключ
	name	CharField(100)	Название типа
	manufacturer	CharField(100)	Производитель
Component	id	AutoField (PK)	Первичный ключ
	name	CharField(200)	Название компонента
	type	ForeignKey → ComponentType	Тип компонента
ConfigurationTemplate	id	AutoField (PK)	Первичный ключ
	name	CharField(100)	Название шаблона
	created_at	DateTimeField	Дата создания
ConfigurationItem	id	AutoField (PK)	Первичный ключ
	template	ForeignKey → ConfigurationTemplate	Шаблон сборки
	component	ForeignKey → Component	Компонент в шаблоне
Order	id	AutoField (PK)	Первичный ключ
	created_at	DateTimeField	Дата создания
	total_price	DecimalField(12, 2)	Общая стоимость
	order_number	CharField(20)	Номер заказа

Продолжение таблицы 1

Таблица	Поле	Тип данных	Назначение
OrderItem	id	AutoField (PK)	Первичный ключ
	order	ForeignKey → Order	Связь с заказом
	category	CharField(100)	Категория
	name	CharField(255)	Название позиции
	link	URLField	Ссылка на товар
	price	DecimalField(10, 2)	Цена

Использование ORM упрощает работу с данными, обеспечивает безопасность запросов и снижает зависимость от конкретной СУБД.

В листинге Б.1 приложения Б показан код для нижеописанных моделей.

Модели Product хранит базовую информацию о товаре: наименование, категорию и артикул.

Пояснение: name – наименование товара (например, «Ноутбук ASUS»);

- category – категория (например, «noutbuki»);
- article – артикул (уникальный идентификатор товара у поставщика).

Метод __str__ нужен для удобного отображения объекта в Django-админке и shell.

Модель Supplier содержит наименование, URL-адрес и приоритет отображения.

Пояснение: priority позволяет сортировать предложения (например, приоритетнее показывать цены от определённых поставщиков);

- используется при отборе лучших предложений.

Самая важная модель – ProductPrice, связывающая товар и поставщика с конкретной ценой, ссылкой, наличием и датой обновления.

Пояснение: ForeignKey устанавливает связь с Product и Supplier;

- DecimalField используется для точных денежных значений;
- availability указывает наличие на сайте;
- updated_at автоматически обновляется при каждом сохранении.

Особенность реализации связи между моделями, ForeignKey(...,

on_delete=models.CASCADE) означает, что при удалении товара или поставщика все связанные предложения будут удалены.

Связь между моделями также предотвращает «висячие» записи в базе.

Индексы и производительность, Django автоматически создаёт индекс на ForeignKey-полях (product, supplier), что ускоряет фильтрацию.

В реальных проектах для ускорения поиска часто добавляют индекс вручную.

Методы можно добавлять для получения самых дешёвых предложений или актуальных цен.

В листингах В.1, В.2, В.3 приложения В показан код миграции БД.

Файлы миграций создаются автоматически после определения моделей и содержат инструкции для создания таблиц.

Например, 0001_initial.py описывает структуру таблиц при первом запуске команды python manage.py makemigrations.

Вывод: модели Django в проекте описывают все ключевые сущности: товары, поставщиков и цены.

Особенности сущностей: гарантируют целостность данных через внешние ключи;

- упрощают работу с БД за счёт ORM;
- позволяют легко интегрировать данные, полученные парсерами;
- удобны для вывода в административном интерфейсе или через API.

За счёт чётко структурированных моделей система становится масштабируемой и удобной в сопровождении, что особенно важно для автоматизированной платформы мониторинга цен.

Пользовательский интерфейс – предоставляет операторам доступ к данным, возможность просматривать, фильтровать и редактировать информацию.

В проекте используется встроенный административный интерфейс Django, который обеспечивает полный контроль над данными – товарами, поставщиками и актуальными ценами. Это позволяет отказаться от создания

отдельного пользовательского интерфейса, не жертвуя удобством и функциональностью (примеры кода берутся из листингов 1, 2, 3, 12, 16, 17)

В листинге Г.1 приложения Г показан код файла настройки ASGI-приложения для Django.

В листинге Д.1 приложения Д показан код для добавления регистрации моделей для админ-панели, и конфигурации проекта Django.

В листинге Е.1 приложения Е показан код для маршрута веб-интерфейса.

В листинге Ж.1 приложения Ж показан код для регистрации.

В листинге И.1 приложения И показан код основной логики программы.

В листинге К.1 приложения К показан код для запуска скрипта Django.

Просмотр списка товаров, цен и поставщиков – модели Product, Supplier и ProductPrice зарегистрированы в административной панели и визуализируются как таблицы. Это позволяет видеть список всех объектов с их основными атрибутами.

Поиск и фильтрация по различным критериям (дата, цена, наличие) – поддержка поиска и фильтрации реализуется через параметры search_fields и list_filter, пример настройки для модели ProductPrice.

Оператор может быстро найти нужную цену по названию товара или отфильтровать по наличию и дате обновления.

Редактирование и удаление записей – каждая запись в таблице (например, товар или цена) открывается как форма, где можно изменить поля, пример формы редактирования – модель Supplier.

Добавление новых товаров и поставщиков вручную – кнопка «Добавить» в панели позволяет вручную создать новую запись. Для модели Product это будет форма с полями, поля определяются в models.py.

Экспорт таблиц (например, в CSV или Excel) – Django не поддерживает экспорт «из коробки», но можно использовать встроенные действия (actions) для экспорта, например, CSV.

Особенности интерфейса:

1. Настройка через admin.py – административные классы позволяют настроить.
2. Инлайн-редактирование – цены можно редактировать внутри карточки товара (если подключён TabularInline).
3. Безопасность и авторизация- интерфейс защищён логином по адресу /admin, только зарегистрированные пользователи с правами администратора имеют доступ к управлению.

Вывод: панель администратора в проекте – это мощный инструмент, обеспечивающий;

- оперативное управление данными без программирования;
- безопасный и централизованный доступ;
- гибкую настройку отображения и взаимодействия с объектами.

Благодаря использованию встроенного функционала Django, пользовательский интерфейс получился одновременно простым, расширяемым и надёжным.

Логика отображения страниц и обработки действий – связывает интерфейс с данными, обрабатывает действия пользователя, управляет маршрутизацией и отображением контента.

Несмотря на использование стандартной административной панели Django, проект также включает в себя реализацию пользовательских представлений (views), шаблонов (templates) и маршрутов (urls), что позволяет гибко расширять функциональность, реализовать интерфейс конфигуратора и организовать обработку действий оператора.

Функции: обработка запросов от пользователей через формы;

- генерация HTML-страниц с данными из базы;
- отображение конфигуратора и составленных заявок;
- поддержка фильтрации, поиска и создания заказов;
- сопровождение действий, таких как добавление компонентов в сборку или оформление заявки.

В листинге Л.1 приложения Л показан код базового HTML-шаблона.

В листинге М.1 приложения М показан код шаблона конфигуратора ПК.

В листинге Н.1 приложения Н показан код отображения списка заявок с выпадающими блоками.

В листинге П.1 приложения П показан код для отображения результатов поиска товаров, собранных с разных сайтов.

В листинге Р.1 приложения Р показан код для отображения сохраненных конфигураций.

Представления (views.py) определены функции и классы, которые получают данные из моделей, формируют бизнес-логику и передают контекст в шаблоны, например, функция отображения списка заказов, эта функция получает все заказы из базы, сортирует по дате передает в шаблон orders_list.html.

Шаблоны (templates/) оформлены на HTML с поддержкой Bootstrap и тегов шаблонизатора Django ({% for %}, {% if %}, {{variable}} и т.д.), например, список компонентов в шаблоне конфигуратора configurator.html, компоненты передаются из views.py и отображаются в виде таблицы.

В маршрутизация (urls.py) задаются URL-адреса и соответствующие им функции или классы представлений, например, URL /configurator/ вызывает функцию configurator, а /orders/ – отображение заказов.

Пример пользовательского действия: оператор вводит название товара в поисковую форму → форма отправляется на сервер → views.py фильтрует данные из базы → отфильтрованные компоненты отображаются в search.html.

Дополнительные возможности: сохранение конфигураций – пользователь может собрать конфигурацию и сохранить её в базу. Это реализовано через POST-запросы, форму и модели ConfigurationTemplate и Order;

- редактирование конфигурации компоненты можно удалять и заменять. Логика реализована через сессию и представления;
- оформление заявки – сохранённая конфигурация передаётся в

представление для генерации заявки, после чего сохраняется в базу как объект модели Order.

Возможность расширения – также предусмотрена возможность подключения REST API – для интеграции с внешними сервисами или создания собственного frontend-клиента. Это может быть реализовано с использованием Django REST Framework.

Вывод: комбинация представлений, шаблонов и маршрутов обеспечивает удобный пользовательский интерфейс, реализующий логику;

- отображения конфигуратора;
- создания и сохранения заказов;
- поиска и фильтрации компонентов.

Подход основан на принципах MVC (Model-View-Controller), что обеспечивает надёжность, удобство сопровождения и масштабируемость интерфейса.

3.2. БАЗА ДАННЫХ

Для хранения и обработки информации в проекте используется объектно-реляционная система управления базами данных PostgreSQL. Эта СУБД выбрана за счёт её надёжности, высокой производительности, гибкости в работе с данными и полной совместимости с Django ORM.

База данных содержит все ключевые сущности проекта: товары, поставщиков, предложения, компоненты, шаблоны конфигураций, заказы и их позиции.

Архитектура построена по принципу нормализации и включает взаимосвязанные таблицы с внешними ключами.

Связи реализованы с использованием ForeignKey и отражают реальные бизнес-отношения: один Product может иметь много ProductPrice;

- один Supplier может быть связан с множеством цен;
- один Order содержит список OrderItem;

Эти связи реализованы в models.py через параметры related_name, on_delete=models.CASCADE, что обеспечивает каскадное удаление и логическую целостность.

Особенности реализации: целостность данных – все связи построены на основе ForeignKey, что гарантирует правильную иерархию и каскадное удаление зависимых записей;

- индексы – автоматически создаются на всех внешних ключах. Дополнительно можно задавать вручную для ускорения запросов.
- методы моделей – для повышения удобства работы с БД добавлены пользовательские методы.

Преимущества используемой СУБД PostgreSQL: полная поддержка транзакций;

- совместимость с Django ORM;
- масштабируемость;
- поддержка расширений (например, pg_trgm для поиска по нечетким совпадениям).

Вывод: база данных проекта играет ключевую роль в обеспечении стабильной и эффективной работы программного комплекса. Её структура построена на основе принципов нормализации, что позволяет минимизировать избыточность данных, упростить сопровождение и обеспечить логическую целостность.

Наиболее важные качества реализованной базы данных: строгая нормализация: каждая таблица отвечает за отдельную сущность – товары, поставщиков, компоненты, заказы и т.д. Это обеспечивает чёткое разграничение данных и исключает дублирование информации;

- гибкость и масштабируемость: архитектура базы данных позволяет легко добавлять новые модели и связи без риска нарушить существующую логику. Это особенно важно для будущего расширения функциональности (например, добавление REST API, внешних систем

учёта, новых категорий товаров);

- целостность и устойчивость к сбоям: использование связей ForeignKey с каскадным удалением и автоматическое создание индексов гарантирует защиту от потери данных и поддерживает целостность при удалении или обновлении записей;
- эффективная работа с данными: благодаря использованию Django ORM, взаимодействие с базой данных происходит на высоком уровне абстракции. Это снижает количество ручного SQL-кода, упрощает отладку, защищает от SQL-инъекций и ускоряет разработку;
- интеграция с административным интерфейсом: все модели интегрированы в Django Admin, что позволяет работать с базой визуально, без использования сторонних СУБД-клиентов или командной строк;
- поддержка временных данных и истории изменений: за счёт поля updated_at в модели ProductPrice обеспечивается отслеживание даты последнего обновления предложения;
- поддержка пользовательских сборок и заказов: таблицы Order, OrderItem, ConfigurationTemplate и ConfigurationItem позволяют организовать хранение не только справочной информации, но и результат пользовательской активности: собранных конфигураций, заказов и их позиций.

Использование СУБД PostgreSQL позволило обеспечить высокую надёжность, скорость и расширяемость проекта. Благодаря тесной интеграции с Django и поддержке транзакций PostgreSQL стал оптимальным выбором для реализации серверной части системы.

Таким образом, база данных является фундаментом архитектуры всего программного комплекса, объединяя в себе как техническую устойчивость, так и бизнес-логику хранения информации о товарах, поставщиках, конфигурациях и заказах.

ВЫВОДЫ ПО ГЛАВЕ 3

В результате проектирования программного комплекса была сформирована чётко структурированная модульная архитектура, охватывающая все основные функциональные аспекты системы: автоматизированный сбор данных с торговых площадок, управление и хранение информации в базе данных, логика взаимодействия с пользователем через веб-интерфейс, а также инструменты администрирования и оформления заказов.

Модули были спроектированы с соблюдением принципов разделения ответственности и слабой связности, что позволило добиться высокой степени независимости между компонентами.

Это, в свою очередь, упростило реализацию, упростило локализацию ошибок в процессе тестирования и повысило удобство дальнейшего сопровождения и модификации системы.

Использование веб-фреймворка Django обеспечило надёжную основу для построения серверной логики и интерфейсной части. Благодаря встроенной модели ORM, работа с базой данных стала безопасной, удобной и масштабируемой. Административная панель Django также позволила быстро организовать базовую управляемость данными без необходимости разработки отдельных модулей.

Выбор PostgreSQL в качестве системы управления базами данных оказался оправданным, так как данная СУБД сочетает высокую производительность, расширенные средства работы с реляционными данными, поддержку транзакций и стабильность при многопользовательском доступе.

Кроме того, PostgreSQL обладает хорошей масштабируемостью и гибкостью, что делает её пригодной для использования в крупных и развивающихся системах.

Особое внимание при проектировании было уделено вопросам

расширяемости и адаптации под новые условия.

Архитектура комплекса спроектирована таким образом, чтобы при необходимости было возможно добавление новых парсеров, изменение бизнес-логики или подключение внешних API без необходимости переработки всей системы. Это делает программный продукт жизнеспособным не только в текущей конфигурации, но и в перспективе – при его возможной интеграции в более сложные ИТ-экосистемы предприятий.

Таким образом, проектирование программного комплекса создало прочную и гибкую архитектурную основу, соответствующую заявленным требованиям и задачам, и заложило все необходимые условия для надёжной и эффективной реализации системы автоматизации закупки компьютерного оборудования.

4. ТЕСТИРОВАНИЕ ПРОГРАММНОГО КОМПЛЕКСА ДЛЯ АВТОМАТИЗАЦИИ ЗАКУПКИ КОМПЬЮТЕРНОГО ОБОРУДОВАНИЯ

4.1. ПРОВЕРКА ПОЛНОЙ РАБОТОСПОСОБНОСТИ ПРОГРАММНОГО КОМПЛЕКСА ДЛЯ АВТОМАТИЗАЦИИ ЗАКУПКИ ОБОРУДОВАНИЯ

Программа содержит набор скриптов, каждый из которых отвечает за получение информации о товарах с конкретного интернет-магазина. В систему интегрированы следующие парсеры: parsingCitilink.py — парсинг сайта Ситилинк;

- parsingDNS.py — парсинг сайта DNS;
- parsingREGARD2.py — парсинг сайта Регард;
- parsingONLINETRADE.py — парсинг сайта Онлайнтрейд;
- parsingTorgPC.py — парсинг сайта Торг-ПК.

  Проверенные параметры: наименование товара;
- актуальная цена;
- наличие на складе;
- ссылка на товар.

Пример для проведения тестирования: поступил заказ на покупку пяти процессоров и пяти видеокарт мощных и не самых дорогих. Оператор выбрал и согласовал с заказчиком, что это будут процессоры «Intel Core i3-10100» и видеокарты «ASUS GeForce RTX 4090 TUF Gaming OG OC Edition».

Веб-интерфейс разработан на базе Django с использованием HTML-шаблонов. Он обеспечивает выбор компонентов, просмотр характеристик, расчёт стоимости и сохранение конфигурации. Проверялась как логика отображения, так и корректность действий пользователя.

Основные элементы интерфейса: страница конфигуратора (configurator.html);

- форма добавления компонентов (add_components.html);
- базовый шаблон (base.html).

Проверенные функции:

- добавление компонентов;
- удаление компонентов;
- расчёт итоговой стоимости;
- сохранение конфигурации;
- загрузка ранее сохранённой конфигурации;
- обработка ошибок (например, добавление дублирующего компонента).

В проекте используются модели Django для представления конфигураций, заказов и компонентов. Были проверены связи между моделями, корректность миграций, а также сохранение и извлечение данных из базы данных.

Модели, прошедшие проверку:

1. Component – базовая модель оборудования.
2. ConfigurationTemplate – шаблон конфигурации.
3. Order и OrderItem – модели для оформления заказов.
4. Category – категории оборудования (видеокарты, процессоры и т.д.).

Проверки: создание объектов через Django Admin;

- связь «один ко многим» между конфигурациями и компонентами;
- проверка каскадного удаления (удаление конфигурации – удаление привязанных компонентов);
- обработка полей null и blank;
- проверка уникальности и индексации при большом объёме данных.

Система протестирована на предмет устойчивости к ошибкам, включая: потерю соединения с интернетом;

- недоступность внешних сервисов;
- ввод некорректных или пустых значений пользователем;
- одновременные действия нескольких пользователей (в тестовой среде).

Интерфейс при выполнении заказа на закупку пяти процессоров «Intel Core i3-10100» и пяти видеокарт «ASUS GeForce RTX 4090 TUF Gaming OG OC Edition».

На рисунке 9 показано начало работы сервера.

Рисунок 9 – Начало работы сервера

На рисунке 10 показана панель добавления комплектующих.

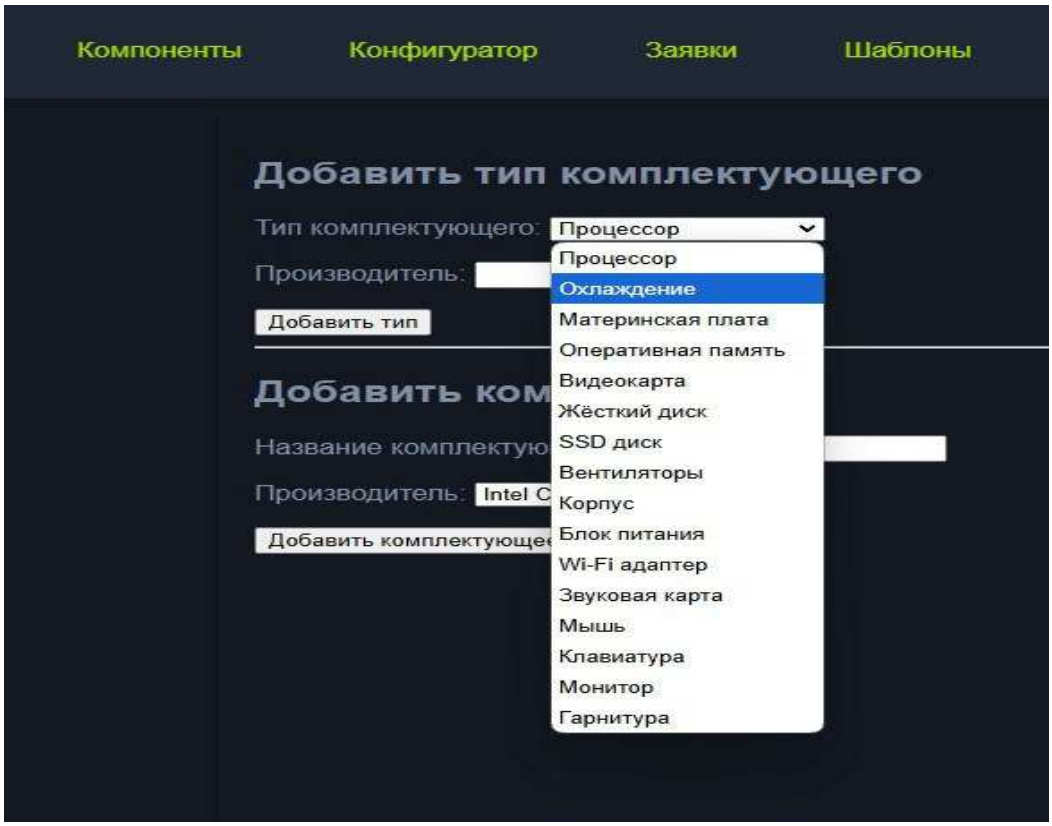

Рисунок 10 – панель добавления комплектующих

На рисунке 11 показан ручной ввод комплектующих.

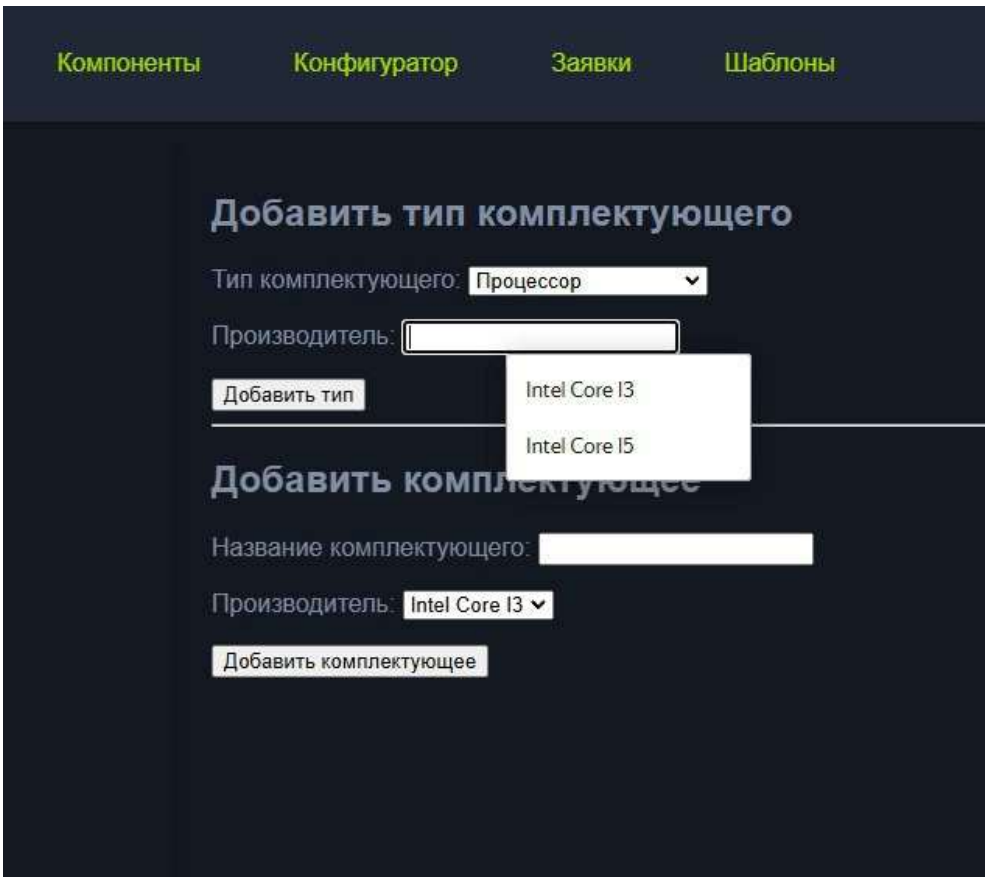

Рисунок 11 – Ручной ввод комплектующих

На рисунке 12 показан конфигуратор с выбором комплектующих.

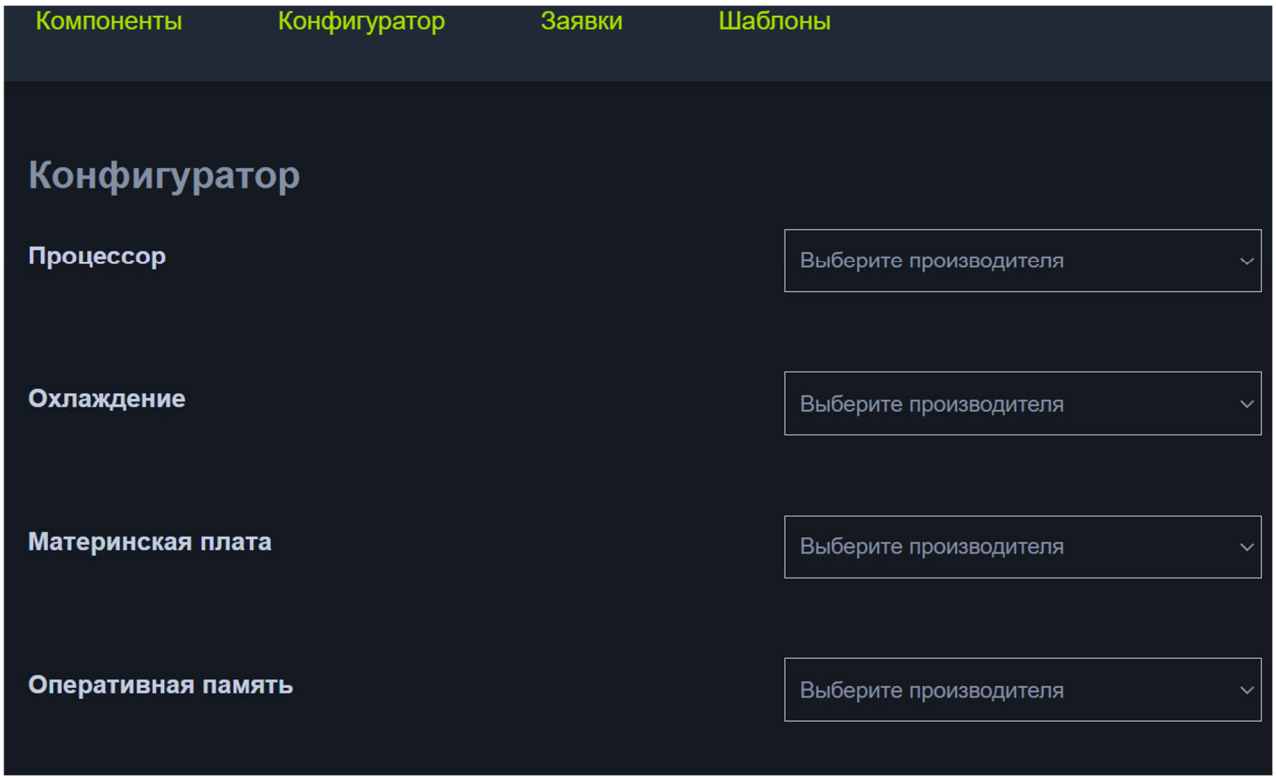

Рисунок 12 – Конфигуратор с выбором комплектующих

На рисунке 13 показан выбор комплектующих, которые были оговорены

с заказчиком.

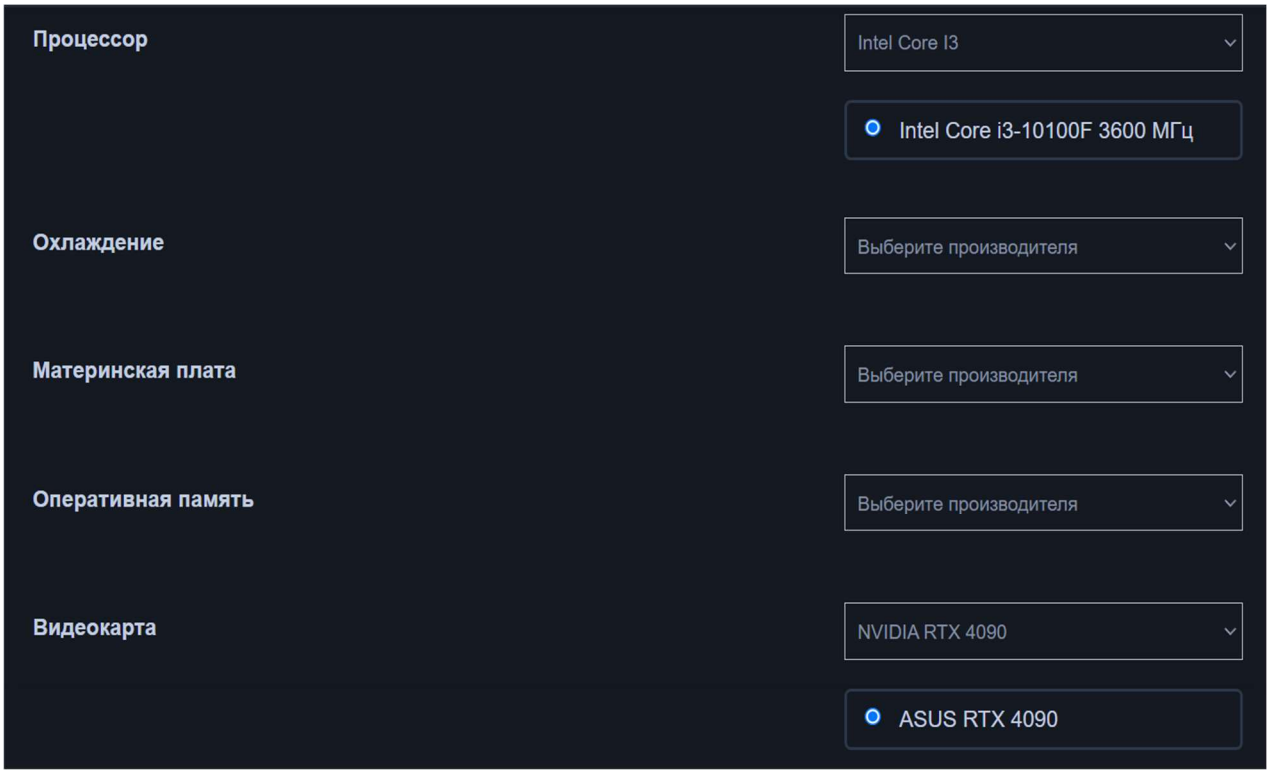

Рисунок 13 – Выбор комплектующих, которые были оговорены с заказчиком

На рисунке 14 показан поиск в интернете комплектующих по заказу.

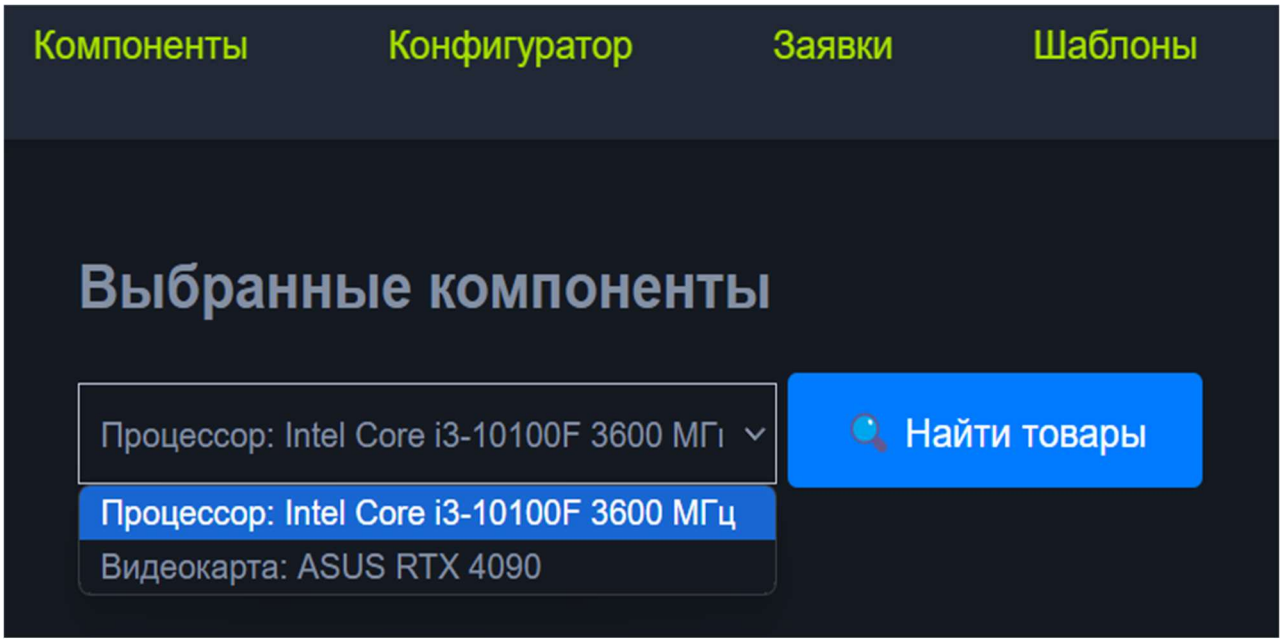

Рисунок 14 – Поиск в интернете комплектующих по заказу

На рисунке 15 показаны найденные по запросу процессоры с возможностью добавить их в заявку.

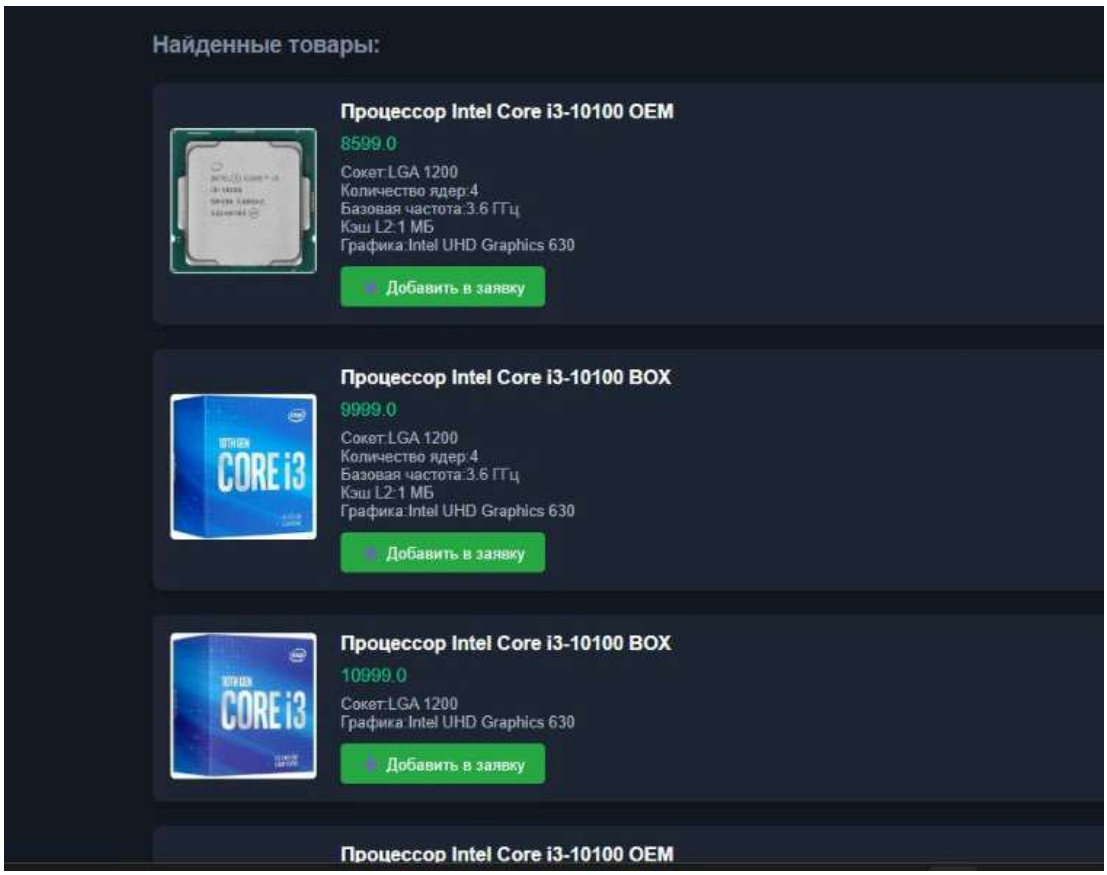

Рисунок 15 – Найденные по запросу процессоры с возможностью добавить их в заявку

На рисунке 16 показаны найденные по запросу видеокарты с возможностью добавить их в заявку.

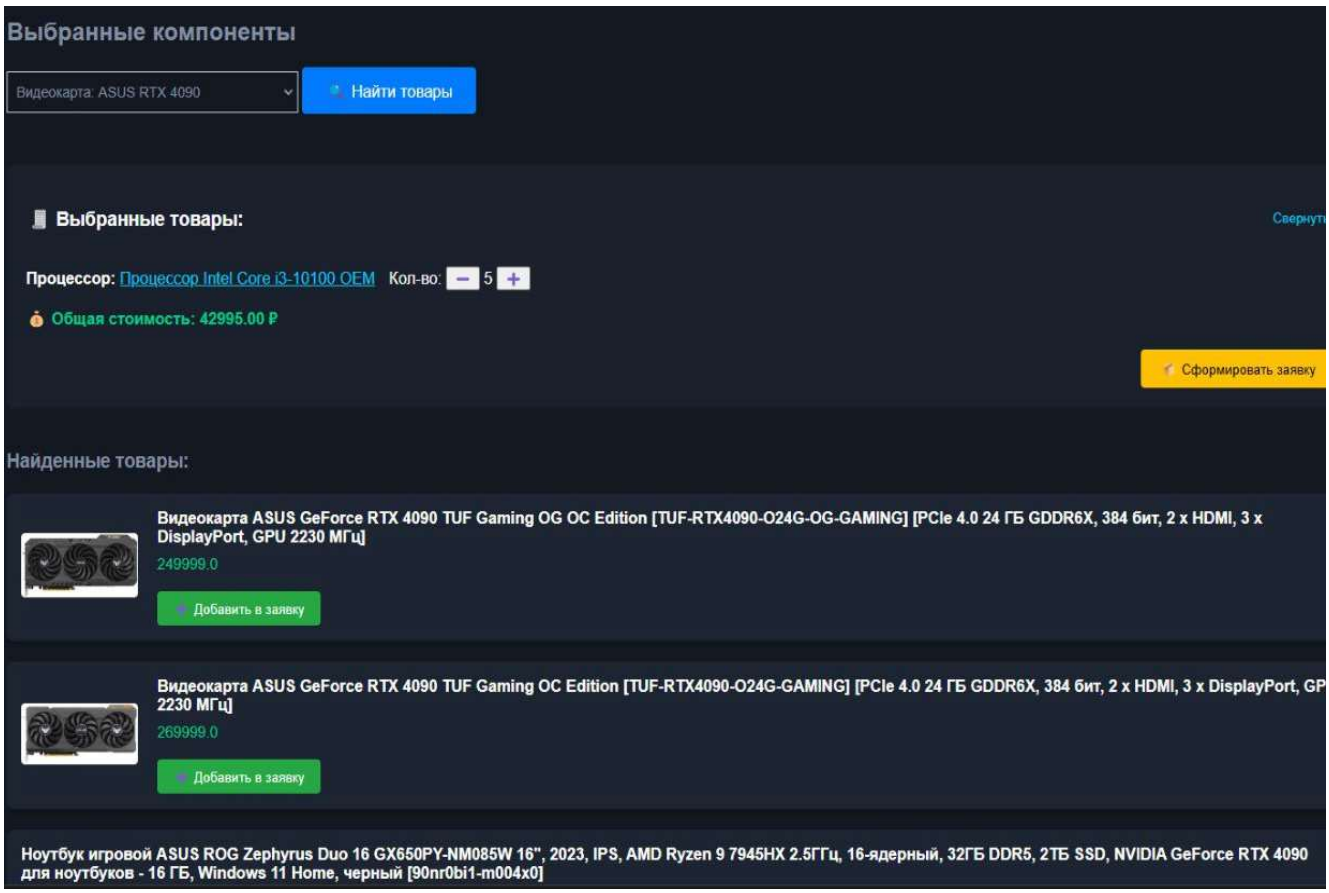

Рисунок 16 – Найденные по запросу видеокарты с возможностью добавить их в заявку

На рисунке 17 показана заявка на заказ комплектующих.

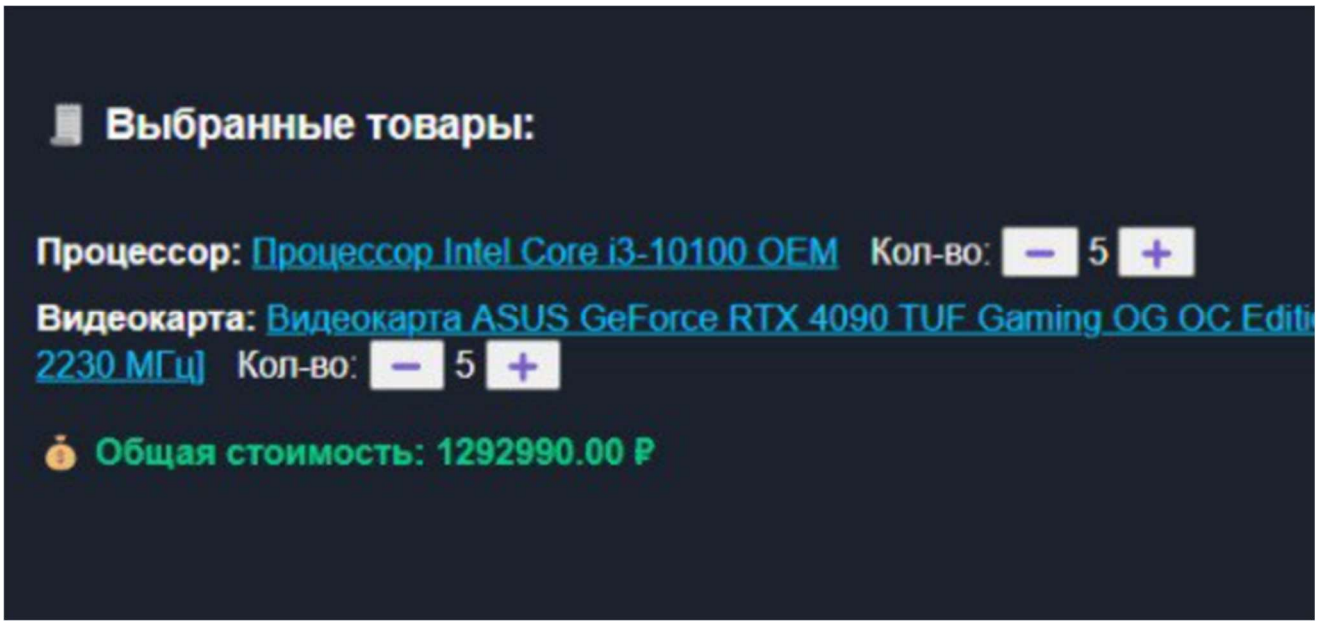

Рисунок 17 – Заявка на заказ комплектующих

На рисунке 18 показан список сделанных заявок.

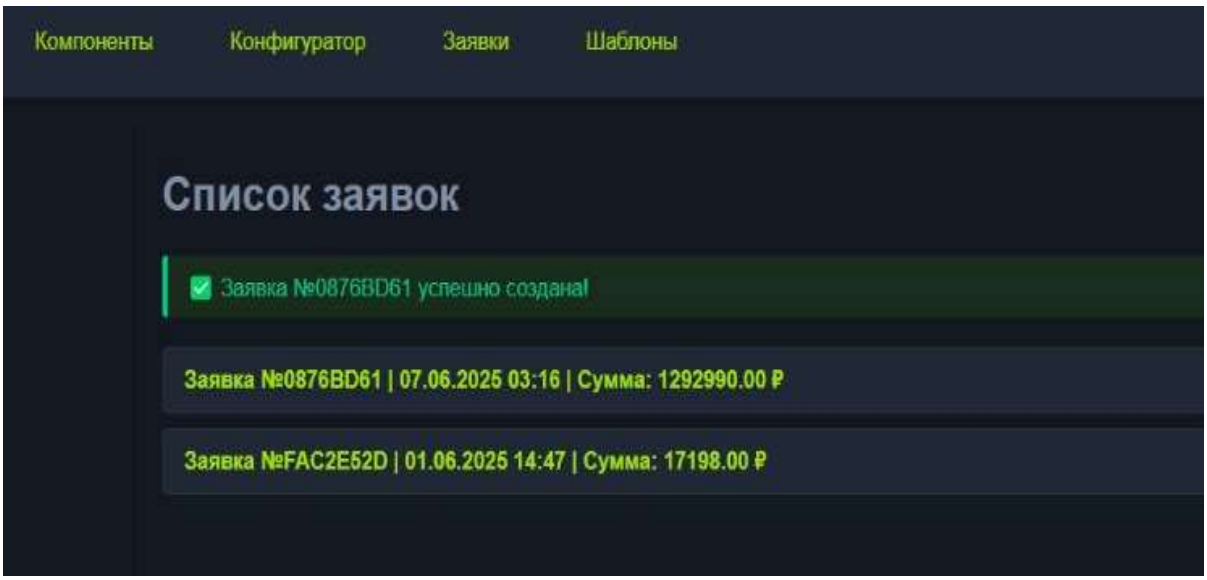

Рисунок 18 – Список сделанных заявок

На рисунке 19 показана собранная заявка.

Рисунок 19 – Собранная заявка

На рисунке 20 показан сохраненный шаблон.

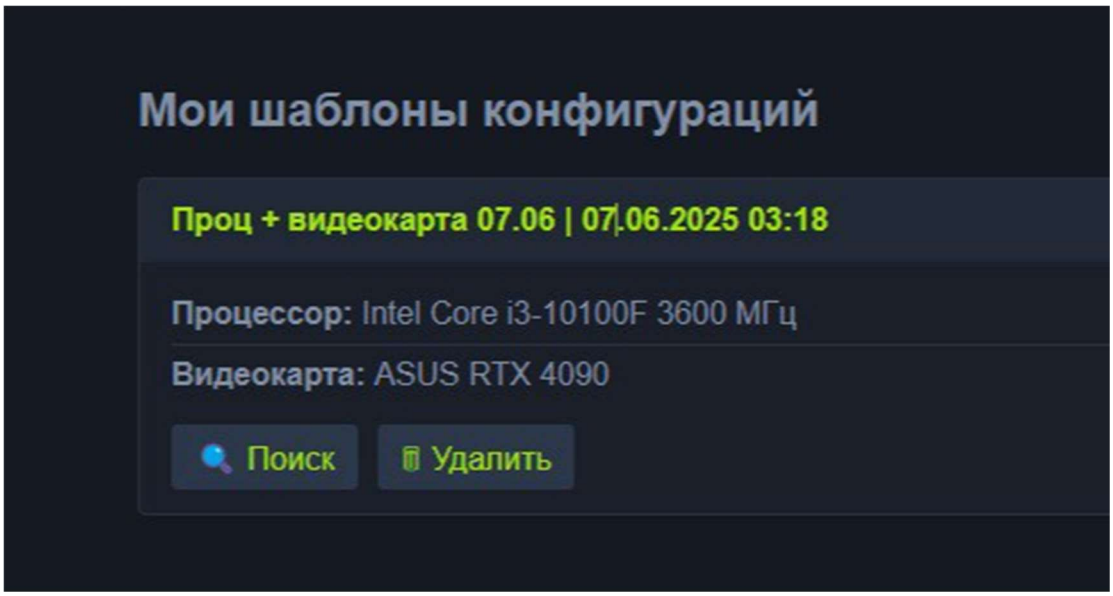

Рисунок 20 – Сохраненный шаблон

В таблице 2 показан сценарий тестирования №1 и его результат.

Таблица 2 – Сценарий тестирования №1

Действие	**Ожидаемый результат**	**Итоговый результат**
Парсинг по запросу «GeForce RTX 4060»	Возвращается список товаров с корректными названиями и ценами	Успешно
Запрос с несуществующим товаром	Возвращается пустой список или сообщение об отсутствии результатов	Успешно
Изменение HTML-структуры сайта	Парсер должен обрабатывать исключения и не вызывать сбой всей программы	Успешно
Отсутствие интернет-соединения	Обработка исключения с уведомлением пользователя	Успешно

В таблице 3 показан сценарий тестирования №2 и его результат.

Таблица 3 – Сценарий тестирования №2

Действие	**Ожидаемый результат**	**Итоговый результат**
Открытие конфигуратора	Загружается список компонентов	Успешно

Продолжение таблицы 3

Действие	Ожидаемый результат	Итоговый результат
Добавление двух одинаковых видеокарт	Выдаётся предупреждение о дублировании	Успешно
Удаление товара из списка	Товар исчезает, итоговая сумма пересчитывается	Успешно
Нажатие «Сохранить конфигурацию»	Конфигурация сохраняется в базе данных	Успешно
Просмотр конфигурации позже	Загружаются все ранее выбранные компоненты	Успешно

Проведённое тестирование показало, что программный продукт стабилен и готов к использованию.

Несмотря на отсутствие модульных и интеграционных тестов в виде автоматизированных сценариев (например, с использованием unittest или pytest), ручное тестирование охватило все ключевые компоненты системы.

В будущем возможно дополнение проекта автоматическими тестами для повышения надёжности при масштабировании или обновлении.

Таким образом, разработка программы прошла все необходимые этапы – от проектирования до реализации и тестирования.

В результате получено работоспособное решение, соответствующее современным требованиям к программному обеспечению: оно надёжно, удобно, расширяемо и готово к интеграции в реальные процессы закупки оборудования.

ВЫВОДЫ ПО ГЛАВЕ 4

Этап тестирования подтвердил функциональную полноту, корректность работы и стабильность разработанного программного комплекса.

В процессе ручного тестирования были проверены все основные компоненты системы, включая: модули парсинга, которые корректно обрабатывают структуру HTML-страниц различных поставщиков и

сохраняют полученные данные в базу без потерь и искажений;

- взаимодействие с базой данных, где тестирование показало корректную реализацию связей между сущностями, целостность данных при множественных операциях чтения и записи, а также быстрое выполнение типовых SQL-запросов;
- пользовательский интерфейс, где были протестированы функции фильтрации, сортировки, редактирования и удаления записей;
- обработка пользовательских сценариев, включая типовые и нестандартные действия операторов, а также поведение системы при отсутствии подключения к интернету или при получении некорректных данных от одного из сайтов;
- устойчивость к сбоям, включая ситуации, когда внешние сайты изменяют структуру страниц, временно недоступны или выдают ошибочные ответы.

Интерфейс показал стабильную работу даже при большом объёме данных и отсутствии доступа к одному из источников.

Парсеры продемонстрировали способность обрабатывать исключения без краха всей системы, сохраняя лог ошибок и продолжая работу с другими источниками.

Отдельно стоит отметить, что несмотря на отсутствие формализованных автоматических юнит- и интеграционных тестов, ручное тестирование было системным и полно охватило все критически важные участки.

Были выявлены и устранены мелкие недочёты в валидации входных данных и логике обработки повторяющихся записей, что положительно сказалось на стабильности системы в целом.

По результатам тестирования можно сделать вывод, что программный комплекс готов к практическому использованию в условиях реального бизнеса.

В будущем планируется внедрение автоматизированной системы тестирования (на базе PyTest или встроенных средств Django), что позволит

проводить регрессионные проверки при добавлении нового функционала и повысит надёжность системы на стадии сопровождения и масштабирования.

Проведённое тестирование подтвердило, что архитектурные решения, заложенные при проектировании системы, обеспечивают устойчивую работу комплекса при взаимодействии с внешними сайтами, включая защиту от частичной недоступности и корректную обработку ошибок в процессе парсинга. Это особенно важно в условиях нестабильной работы интернет-ресурсов и изменяющейся HTML-разметки, что требует от системы высокой адаптивности. Кроме того, результаты проверки показали, что применённые алгоритмы фильтрации и нормализации данных эффективно справляются с извлечением ключевой информации, такой как наименование, цена, наличие и ссылка, что подтверждает пригодность системы к реальному промышленному использованию.

5. ЗАКЛЮЧЕНИЕ

Данная дипломная работа была направлена на разработку программного комплекса для автоматизации закупки компьютерного оборудования. В процессе выполнения работы были решены следующие ключевые задачи: определены функциональные и нефункциональные требования к системе;

- произведён анализ и обоснование выбора средств реализации;
- выполнено проектирование архитектуры программного комплекса;
- реализованы все основные модули системы, включая парсеры, модели, интерфейс и базу данных;
- проведено тестирование и анализ работоспособности системы.

На основе анализа существующих решений и потребностей, была сформулирована цель автоматизации рутинных операций при сборе, обработке и анализе информации о поставщиках оборудования. Это позволило повысить эффективность принятия решений при закупке и снизить человеческий фактор.

Для реализации программного комплекса были выбраны современные и проверенные технологии: язык программирования Python, фреймворк Django, система управления базами данных PostgreSQL. Также применены сторонние библиотеки для парсинга сайтов и обработки данных.

В результате была создана веб-система, включающая следующие модули: модуль автоматического сбора информации с сайтов поставщиков (парсеры);

- модуль хранения и управления структурированными данными (модели Django и база PostgreSQL);
- административный пользовательский интерфейс на базе Django admin;
- логика обработки пользовательских действий, реализующая фильтрацию, поиск и редактирование данных.

Программный комплекс успешно прошёл ручное тестирование, которое подтвердило его функциональную полноту, устойчивость к сбоям и

готовность к практическому применению. Несмотря на отсутствие автоматических тестов, были проверены все критические участки системы, выявлены и устранены недочёты, что повысило надёжность работы приложения.

Разработанная система обладает рядом преимуществ: автоматизация рутинных операций сбора данных;

- масштабируемая архитектура с возможностью подключения новых источников и расширения функционала;
- удобный интерфейс для операторов без необходимости установки дополнительных программ;
- надёжность хранения и обработки информации за счёт использования PostgreSQL и ORM Django.

В рамках данной работы также были заложены основы для дальнейшего развития программного комплекса, включая: добавление автоматических тестов и системы мониторинга;

- реализацию REST API для внешнего взаимодействия;
- улучшение интерфейса с использованием современных JavaScript-фреймворков;
- интеграцию с корпоративными системами учёта и документооборота.

Разработанный программный продукт полностью соответствует поставленным задачам и может быть использован как основа для внедрения в реальную практику автоматизации закупок в организациях различного масштаба.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Андреев, А. А. Управление логистическими процессами / А.А. Андреев Москва: Изд-во Юрайт, 2021. – 352 с.
2. Балабанов, И. Т. Основы менеджмента / И.Т. Балабанов. – Санкт-Петербург: Изд-во Питер, 2020. – 416 с.
3. Безруков, А. В. Разработка программных комплексов / А.В. Безруков. – Москва: Изд-во Форум, 2019. – 287 с.
4. Богданов И. А. Информационные системы в экономике / И.А. Богданов. – Москва: Изд-во Инфра-М, 2022. – 310 с.
5. Воронков А. А. Системы автоматизации бизнес-процессов / А.А. Воронков Санкт-Петербург: Изд-во Питер, 2023. – 376 с.
6. Гусев В. Е. Программная инженерия / Е.В. Гусев Санкт-Петербург: Изд-во БХВ-Петербург, 2020. – 460 с.
7. Дьяконов В. П. SQL. Полное руководство / В.П. Дьяконов. – Санкт-Петербург: Изд-во Питер, 2021. – 560 с.
8. Елисеев А. Е. Автоматизация управления закупками / А.Е. Елисеев. – Москва: Изд-во КноРус, 2020. – 298 с.
9. Ермаков С. Н. ERP-системы и управление предприятием / С.Н. Ермаков. – Москва: Изд-во Финансы и статистика, 2019. – 348 с.
10. Жарков С. П. Управление поставками / С.П. Жарков. – Санкт-Петербург: Изд-во Питер, 2022. – 240 с.
11. Злобин А. А. Информационные технологии в логистике / А.А. Злобин. – Москва: Изд-во Логистика, 2020. – 372 с.
12. Ильин И. В. Моделирование бизнес-процессов / И.В. Ильин. – Санкт-Петербург: Изд-во Питер, 2021. – 432 с.
13. Кабанов П. А. Разработка и внедрение корпоративных информационных систем / П.А. Кабанов. – Москва: Изд-во Инфра-М, 2023. – 410 с.

14. Каляев И. В. Закупки и снабжение: теория и практика / И.В. Каляев. – Москва: Изд-во Юрайт, 2022. – 364 с.

15. Карпов В. В. Программирование на C# / В.В. Карпов Санкт-Петербург: Изд-во Питер, 2021. – 504 с.

16. Официальная документация Wildberries Open API. – https://openapi.wildberries.ru. - С. 22-26.

17. Документация Ozon API. – https://docs.ozon.dev. - С. 10-13.

18. B2B-Center — электронная торговая площадка. – https://www.b2b-center.ru. - С. 55-59.

19. Техническая документация Yandex.Market API. – https://yandex.ru/dev/market/partner-api. - С. 2-5.

20. Документация по Django 4.x. – https://docs.djangoproject.com/en/4.2/. - С. 10-15.

21. Официальный сайт Ozon API. – https://docs.ozon.dev. - С. 16-19.

22. Документация Wildberries Open API. – https://openapi.wildberries.ru. - С. 10-14.

23. Платформа B2B-Center. – https://www.b2b-center.ru. - С. 1-3.

24. Яндекс.Маркет Partner API. – https://yandex.ru/dev/market/partner-api. - С. 1-8.

25. Документация по REST API GitHub. – https://docs.github.com/ru/rest. - С. 9-14.

26. Microsoft Learn: Архитектура веб-приложений. – https://learn.microsoft.com/ru-ru/aspnet/core/fundamentals/?view=aspnetcore-6.0 - С. 1-5.

27. Документация Python (v3.11). – https://docs.python.org/3.11/. - С. 38-43.

28. Портал государственных закупок Республики Казахстан. – https://goszakup.gov.kz. - С. 16-20.

29. Документация FastAPI. – https://fastapi.tiangolo.com/ru/. - С. 12-17.

30. Документация PostgreSQL. – https://www.postgresql.org/docs/. - С.

6-10.

31. Цифровая платформа Сбербанк-АСТ. – https://www.sberbank-ast.ru. - С. 7-9.

32. SAP Help Portal. – https://help.sap.com. - С. 1-3.

33. Документация BeautifulSoup. – https://www.crummy.com/software/BeautifulSoup/bs4/doc/. - С. 9-13.

34. Официальная документация по Django ORM. – https://docs.djangoproject.com/en/4.2/topics/db/models/. - С. 15-24.

35. Gartner: Magic Quadrant for Procure-to-Pay Suites. – https://www.gartner.com/en/documents/4001232. - С. 15-21.

36. Документация по API REGARD. – https://www.regard.ru/info/api. - С. 14-88.

37. Документация по API Onlinetrade. – https://www.onlinetrade.ru/site/api. - С. 60-63.

38. Платформа Контур.Закупки. – https://zakupki.kontur.ru. - С. 1-5.

39. Документация Django Templates. [Электронный ресурс] // URL: https://docs.djangoproject.com/en/4.2/ref/templates/language/ - С. 5-8.

40. Платформа Zakupki360. – https://zakupki360.ru. - С. 13-37.

41. Документация Pytest. – https://docs.pytest.org/en/latest/. - С. 50-52

42. Сервис TorgPC. – https://torgpc.ru. - С. 13-40.

43. Документация SQLite. [Электронный ресурс] // URL: https://www.sqlite.org/docs.html - С. 55-60.

44. Документация по requests (Python). – https://docs.python-requests.org/en/latest/. - С. 10-15.

45. Документация по Git. – https://git-scm.com/doc. - С. 32-35.

46. Lansweeper ITAM Documentation. – https://www.lansweeper.com/knowledgebase/. - С. 40-45.

ПРИЛОЖЕНИЕ А
ЛИСТИНГ КОДА ПАРСЕРОВ

Листинг А.1. – parsingCitilink.py

```
import time
import undetected_chromedriver as uc
from bs4 import BeautifulSoup
from selenium.common import NoSuchElementException, TimeoutException,
StaleElementReferenceException
from selenium.webdriver import Keys
from selenium.webdriver.common.by import By
from selenium.webdriver.support import expected_conditions as EC
from selenium.webdriver.support.ui import WebDriverWait

def get_products_page(item_name: str) -> list[str]:
    """
    возвращаем список, каждый элемент которого это html страница
    :param item_name: str
    :return:  list[str]
    """
    content = []
    options = uc.ChromeOptions()
    headers = {
        "User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 Chrome/123.0.0.0 Safari/537.36"
    }
    driver = uc.Chrome(options=options, use_subprocess=True,
version_main=136)
    driver.get("https://www.citilink.ru/")
    time.sleep(2)
    wait = WebDriverWait(driver, 4)
    element = driver.find_element(By.CSS_SELECTOR, 'input[type="search"]')
    element.clear()
    element.send_keys(item_name)
    element.send_keys(Keys.ENTER)
    time.sleep(1)
    max_page = 5

    while True:
        content.append(driver.page_source)

        try:
            if max_page == 0:
                break
            max_page -= 1

            # Ждём появления кнопки на новой странице
            wait.until(
                EC.presence_of_element_located((By.CSS_SELECTOR, '[data-meta-
name="PaginationElement__go-forward"]')))

            # Заново ищем элемент после перехода на новую страницу
            element = driver.find_element(By.CSS_SELECTOR, '[data-meta-
name="PaginationElement__go-forward"]')
            driver.execute_script("arguments[0].scrollIntoView({block:
'center'});", element)
            driver.execute_script("arguments[0].click();", element)

            # Подождать обновление DOM
```

Окончание листинга А.1

```
            time.sleep(2)
        except (NoSuchElementException, TimeoutException):
            print("Кнопка не найдена — достигли последней страницы.")
            break

        except StaleElementReferenceException:
            print("DOM обновился — повторное получение элемента.")
            continue

    driver.quit()

    return content


def parsing_content(content: list[str]) -> list[dict]:
    products = []

    for html in content:
        soup = BeautifulSoup(html, 'html.parser')
        product_blocks = soup.find_all('div', attrs={'data-meta-name':
'ProductVerticalSnippet'})

        for product in product_blocks:
            if 'Нет в наличии' in product.get_text():
                continue

            name_tag = product.find('a')
            name = name_tag['title'] if name_tag and
name_tag.has_attr('title') else ''

            img_tag = product.find('img')
            img_link = img_tag['src'] if img_tag and img_tag.has_attr('src')
else ''

            a_tag = product.find('a')
            product_link = 'https://www.citilink.ru/' + a_tag['href'] if
a_tag and a_tag.has_attr('href') else ''

            price_span = product.find('span', attrs={'data-meta-price':
True})
            price = f"{price_span['data-meta-price']} ₽" if price_span else
'Нет цены'

            products.append({
                'product_link': product_link,
                'img_link': img_link,
                'name': name,
                'price': price
            })

    return products
```

Листинг А.2 – «parsingDNS.py»

```
import time
import undetected_chromedriver as uc
from bs4 import BeautifulSoup
from selenium.webdriver import Keys
from selenium.webdriver.common.by import By
```

Продолжение листинга А.2

```
def get_products_page(item_name: str) -> list[str]:
    возвращаем список, каждый элемент которого это html страница
    :param item_name: str
    :return:  list[str]

    content = []

    options = uc.ChromeOptions()

    # Создаём драйвер
    driver = uc.Chrome(options=options, use_subprocess=True,
version_main=136)
    driver.get("https://www.dns-shop.ru")
    driver.save_screenshot("dns_debug.png")
    time.sleep(2)

    #Находим строку поиска и вбиваем нужный товар
    element = driver.find_element(By.CLASS_NAME, 'presearch__input')
    element.clear()
    element.send_keys(item_name)
    element.send_keys(Keys.ENTER)

    max_page = 5

    #Перебираем все страницы
    while True:
        if max_page == 0:
            break
        max_page -= 1

        content.append(driver.page_source)

        next_buttons = driver.find_elements(By.CSS_SELECTOR, ".pagination-
widget__page-link_next")
        if next_buttons:
            next_buttons[0].click()
        else:
            break

        time.sleep(1)

    driver.quit()

    return content

def parsing_content(content: list[str]) -> list[dict]:
    # content = get_products_page(item_name)
    parsing_data = []

    for page in content:
        soup = BeautifulSoup(page, "html.parser")
        target_div = soup.find('div', class_='catalog-products view-simple')

        if not target_div:
            continue

        product_cards = target_div.find_all('div', class_='catalog-product')

        for card in product_cards:
            # ⛔ Пропуск если товара нет в наличии
```

Окончание листинга А.2

```
            not_available_block = card.find('div', class_='order-avail-wrap
order-avail-wrap_main order-avail-wrap_not-avail')
            if not_available_block and "Товара нет в наличии" in
not_available_block.get_text(strip=True):
                continue
            data = {}
            # Картинка
            source = card.find('source')
            if source:
                data['img_link'] = source.get('data-srcset')

            # Ссылка и наименование
            link_tag = card.find('a', class_='catalog-product__name ui-link
ui-link_black')
            if link_tag:
                link = link_tag.get('href')
                data['product_link'] = f'https://www.dns-shop.ru{link}'

                span = link_tag.find('span')
                if span:
                    for content in span.contents:
                        if isinstance(content, str):
                            name = content.strip()
                            break
                    else:
                        name = None
                    data['name'] = name

            # Информация о товаре
            specs_block = card.find('div', class_='catalog-product__price-
specs')
            if specs_block:
                specs = [div.get_text(strip=True) for div in
specs_block.find_all('div', recursive=False)]
                data['info'] = specs

            # 💰 Цена
            price_div = card.find('div', class_='product-buy__price')
            if price_div:
                data['price'] = price_div.get_text(strip=True)

            parsing_data.append(data)

    return parsing_data
```

Листинг А.3 – «parsingONLINETRADE.py»

```
import time
import undetected_chromedriver as uc
from bs4 import BeautifulSoup
from selenium.webdriver import Keys
from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC

def get_products_page(item_name: str, max_pages: int = 3) -> str:
    """Получаем HTML страницы товаров с onlinetrade.ru"""
    content = ''
    options = uc.ChromeOptions()
    options.add_argument("--disable-blink-features=AutomationControlled")
```

Продолжение листинга А.3

```
    driver = uc.Chrome(options=options, version_main=136)
    driver.get("https://www.onlinetrade.ru/")

    try:
        # Ожидаем появления поисковой строки
        search_input = WebDriverWait(driver, 10).until(
            EC.presence_of_element_located((By.CSS_SELECTOR,
"input.header__search__inputText"))
        )
        search_input.clear()
        search_input.send_keys(item_name)
        search_input.send_keys(Keys.ENTER)
        time.sleep(1)

        # Прокрутка и загрузка товаров
        last_height = driver.execute_script("return
document.body.scrollHeight")

        while max_pages > 0:
            max_pages -= 1

            # Прокрутка вниз
            driver.execute_script("window.scrollTo(0,
document.body.scrollHeight);")
            time.sleep(1)

            # Проверяем кнопку "Показать ещё"
            try:
                load_more = WebDriverWait(driver, 5).until(
                    EC.element_to_be_clickable((By.CSS_SELECTOR,
"a.js__ajaxListingMore")))
                driver.execute_script("arguments[0].click();", load_more)
                time.sleep(1)
            except:
                break

            new_height = driver.execute_script("return
document.body.scrollHeight")
            if new_height == last_height:
                break
            last_height = new_height

    finally:
        content = driver.page_source
        driver.quit()
    return content

def parse_products(content: str) -> list[dict]:
    """Парсим данные о товарах с onlinetrade.ru"""
    products = []
    soup = BeautifulSoup(content, "html.parser")
    cards = soup.find_all("div", class_="indexGoods__item")
    for card in cards:
        product = {}

        # Название и ссылка
        title = card.find("a", class_="indexGoods__item__name")
        if title:
            product["name"] = title.get_text(strip=True)
```

Окончание листинга А.3

```
            product["product_link"] =
f"https://www.onlinetrade.ru{title['href']}"

        # Картинка
        img = card.find("img")
        if img and 'src' in img.attrs:
            product["img_link"] = img['src']

        # Цена
        price = card.find("span", class_="price")
        if price:
            product["price"] = price.get_text(strip=True).replace(" ", "")

        if product:  # Добавляем только если есть данные
            products.append(product)

    return products
```

Листинг А.4 – «parsingREGARD2.py»

```
import time
import random
import undetected_chromedriver as uc
from bs4 import BeautifulSoup
from selenium.webdriver import Keys
from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
from selenium.common.exceptions import TimeoutException, WebDriverException
import re

def get_products_page(item_name: str, max_pages: int = 3) -> str:
    """Получаем HTML страницы товаров с regard.ru с обходом защиты"""
    content = ''

    # Настройки для более естественного поведения
    options = uc.ChromeOptions()
    options.add_argument("--disable-blink-features=AutomationControlled")
    options.add_argument("--headless")
    options.add_argument("--disable-gpu")
    options.add_argument("--window-size=1920,1080")
    options.add_argument("--no-sandbox")
    options.add_argument("--disable-dev-shm-usage")
    options.add_argument(
        "user-agent=Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/120.0.0.0 Safari/537.36")

    try:
        # Автоматический подбор версии ChromeDriver
        driver = uc.Chrome(
            options=options,
            version_main=136  # Автоматически подберет подходящую версию
        )

        # Имитируем человеческое поведение
        driver.execute_cdp_cmd("Network.setUserAgentOverride", {
            "userAgent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/120.0.0.0 Safari/537.36"
        })
```

Продолжение листинга А.4

```
        # Добавляем случайные задержки
        time.sleep(random.uniform(0.7, 1.5))

        driver.get("https://www.regard.ru/")
        time.sleep(random.uniform(0.5, 2))

        try:
            # Закрываем куки-баннер, если есть
            try:
                cookie_btn = WebDriverWait(driver, 5).until(
                    EC.element_to_be_clickable((By.CSS_SELECTOR,
"button.cookie-notice__agree"))
                )
                cookie_btn.click()
                time.sleep(random.uniform(1, 2))
            except:
                pass

            # Поиск товара
            search_input = WebDriverWait(driver, 15).until(
                EC.presence_of_element_located((By.CSS_SELECTOR,
"input[class*='Search_searchWrap']"))
            )

            # Имитируем человеческий ввод
            for char in item_name:
                search_input.send_keys(char)
                time.sleep(random.uniform(0.1, 0.3))

            time.sleep(random.uniform(0.5, 1.5))
            search_input.send_keys(Keys.ENTER)
            time.sleep(random.uniform(0.7, 2))

            # Прокрутка страницы
            last_height = driver.execute_script("return
document.body.scrollHeight")
            scroll_attempts = 0

            while max_pages > 0 and scroll_attempts < 5:
                max_pages -= 1
                scroll_attempts += 1

                # Плавная прокрутка
                for i in range(1, 5):
                    driver.execute_script(f"window.scrollTo(0,
document.body.scrollHeight/{5 - i}*{i});")
                    time.sleep(random.uniform(0.5, 1.5))

                # Проверяем кнопку "Показать ещё"
                try:
                    load_more = WebDriverWait(driver, 10).until(
                        EC.element_to_be_clickable((By.CSS_SELECTOR,
"button[class*='Pagination_loadMore']"))
                    )
                    driver.execute_script("arguments[0].click();", load_more)
                    time.sleep(random.uniform(2, 4))
                except:
                    break
```

Продолжение листинга А.4

```
                new_height = driver.execute_script("return
document.body.scrollHeight")
                if new_height == last_height:
                    break
                last_height = new_height

            content = driver.page_source

        except Exception as e:
            print(f"Ошибка при работе с сайтом: {str(e)}")
            # Сохраняем скриншот для диагностики
            driver.save_screenshot("error_screenshot.png")

    except Exception as e:
        print(f"Ошибка при запуске браузера: {str(e)}")
        # Если undetected_chromedriver не может подобрать версию, попробуем
установить явно
        try:
            driver = uc.Chrome(
                options=options,
                version_main=120  # Явно указываем версию Chrome 120
            )
            return get_products_page(item_name, max_pages)  # Рекурсивный
вызов
        except:
            pass
    finally:
        try:
            driver.quit()
        except:
            pass

    return content


def parse_products(content: str) -> list[dict]:
    """Парсим данные о товарах с учетом возможных изменений в структуре
сайта"""
    products = []

    if not content:
        return products

    soup = BeautifulSoup(content, "html.parser")

    # Несколько вариантов селекторов на случай изменения структуры
    card_selectors = [
        "div.Card_wrap__hES44",  # Основной селектор
        "div[class*='Card_listing']",  # Альтернативный вариант
        "div.product-card"  # Еще один возможный вариант
    ]

    cards = None
    for selector in card_selectors:
        cards = soup.select(selector)
        if cards:
            break

    if not cards:
```

Окончание листинга А.4

```
        print("Не удалось найти карточки товаров")
        return products

    for card in cards:
        try:
            product = {}

            # Название и ссылка
            title = card.select_one("a[class*='CardText_link'], a.product-
title")
            if title:
                product["name"] = title.get_text(strip=True)
                href = title.get("href", "")
                if href and not href.startswith("http"):
                    href = f"https://www.regard.ru{href}"
                product["product_link"] = href

            # Цена
            price = card.find("span", class_="Price_price__m2aSe")
            if price:
                price_text = price.get_text(strip=True)
                # Удаляем все нецифровые символы и преобразуем в число
                clean_price = re.sub(r'[^\d]', '', price_text)
                try:
                    product["price"] = int(clean_price) if clean_price else
None
                except ValueError:
                    product["price"] = None

            # Картинка
            img = card.select_one("img[class*='CardImageSlider'],
img.product-image")
            if img:
                src = img.get("src", img.get("data-src", ""))
                if src and not src.startswith("http"):
                    src = f"https://www.regard.ru{src}"
                product["img_link"] = src

            if product:
                products.append(product)

        except Exception as e:
            print(f"Ошибка при парсинге карточки товара: {e}")
            continue

    return products
```

Листинг А.5 – «parsingTorgPC.py»

```
import time
import undetected_chromedriver as uc
from bs4 import BeautifulSoup
from selenium.webdriver import Keys
from selenium.webdriver.common.by import By
from selenium.webdriver.support import expected_conditions as EC
from selenium.webdriver.support.ui import WebDriverWait

def get_products_page(item_name: str) -> list[str]:
```

Продолжение листинга А.5

```
    """
    возвращаем список, каждый элемент которого это html страница
    :param item_name: str
    :return:  list[str]
    """
    content = []

    options = uc.ChromeOptions()

    # Создаём драйвер
    driver = uc.Chrome(options=options, use_subprocess=True,
version_main=136)
    driver.get("https://torg-pc.ru/")
    wait = WebDriverWait(driver, 10)

    #Находим строку поиска и вбиваем нужный товар
    element = wait.until(EC.element_to_be_clickable((By.ID, "title-search-
input_fixed")))
    element.clear()
    element.send_keys(item_name)
    element.send_keys(Keys.ENTER)

    #Перебираем все страницы
    max_page = 5
    while True:
        time.sleep(2)

        max_page -= 1
        if not max_page:
            break
        content.append(driver.page_source)

        next_buttons = driver.find_elements(By.CLASS_NAME, "flex-nav-next")
        if next_buttons:
            next_buttons[0].click()
        else:
            break

    driver.quit()
    return content

def parsing_content(content: list[str]) -> list[dict]:
    # content = get_products_page(item_name)
    parsing_data = []

    for page in content:
        soup = BeautifulSoup(page, "html.parser")
        target_div = soup.find('div', class_='display_list')

        if not target_div:
            continue

        product_cards = target_div.find_all('div', class_='list_item_wrapp')

        for item in product_cards:
            # Ссылка на товар
            product_link_tag = item.find('a', class_='thumb')
            product_link = 'https://torg-pc.ru' + product_link_tag['href'] if
product_link_tag else ''
```

Окончание листинга А.5

```
            # Ссылка на картинку
            img_tag = product_link_tag.find('img') if product_link_tag else
None
            if img_tag:
                if img_tag.has_attr('data-src'):
                    image_url = 'https://torg-pc.ru' + img_tag['data-src']
                else:
                    image_url = 'https://torg-pc.ru' + img_tag['src']
            else:
                image_url = ''

            # Название товара
            title_tag = item.find('div', class_='item-title')
            title = title_tag.get_text(strip=True) if title_tag else ''

            # Цена
            price_tag = item.find('div', class_='price')
            price = price_tag.get_text(strip=True) if price_tag else ''

            parsing_data.append(
                {
                    "img_link": image_url,
                    "product_link": product_link,
                    "name": title,
                    "price": price
                },
            )
    return parsing_data
```

ПРИЛОЖЕНИЕ Б
ЛИСТИНГ КОДА МОДЕЛЕЙ

Листинг Б.1 – Код моделей

```
   from django.db import models
class ComponentType(models.Model):
    name = models.CharField(max_length=100, verbose_name="Тип
комплектующего")
    manufacturer = models.CharField(max_length=100,
verbose_name="Производитель")

class Component(models.Model):
    name = models.CharField(max_length=200, verbose_name="Название
комплектующего")
    type = models.ForeignKey(ComponentType, on_delete=models.CASCADE,
related_name="components", verbose_name="Тип")

class Order(models.Model):
    created_at = models.DateTimeField(auto_now_add=True)
    total_price = models.DecimalField(max_digits=12, decimal_places=2)
    order_number = models.CharField(max_length=20, unique=True)

    def __str__(self):
        return f"Заявка #{self.order_number}"


class OrderItem(models.Model):
    order = models.ForeignKey(Order, on_delete=models.CASCADE,
related_name="items")
    category = models.CharField(max_length=100)
    name = models.CharField(max_length=255)
    link = models.URLField()
    price = models.DecimalField(max_digits=10, decimal_places=2)
    quantity = models.PositiveIntegerField(default=1)

    def total(self):
        return self.price * self.quantity

class ConfigurationTemplate(models.Model):
    name = models.CharField(max_length=100)
    created_at = models.DateTimeField(auto_now_add=True)

class ConfigurationItem(models.Model):
    template = models.ForeignKey(ConfigurationTemplate,
on_delete=models.CASCADE, related_name='items')
    category = models.CharField(max_length=100)
    component = models.ForeignKey(Component, on_delete=models.CASCADE)
```

ПРИЛОЖЕНИЕ В
ЛИСТИНГ КОДА МИГРАЦИИ ДАННЫХ

Листинг В.1 – Первая миграция данных

```
   from django.db import migrations, models
import django.db.models.deletion


class Migration(migrations.Migration):

    initial = True

    dependencies = [
    ]

    operations = [
        migrations.CreateModel(
            name='ComponentType',
            fields=[
                ('id', models.BigAutoField(auto_created=True,
primary_key=True, serialize=False, verbose_name='ID')),
                ('name', models.CharField(max_length=100, verbose_name='Тип
комплектующего')),
                ('manufacturer', models.CharField(max_length=100,
verbose_name='Производитель')),
            ],
        ),
        migrations.CreateModel(
            name='Component',
            fields=[
                ('id', models.BigAutoField(auto_created=True,
primary_key=True, serialize=False, verbose_name='ID')),
                ('name', models.CharField(max_length=200,
verbose_name='Название комплектующего')),
                ('type',
models.ForeignKey(on_delete=django.db.models.deletion.CASCADE,
related_name='components', to='diploma.componenttype', verbose_name='Тип')),
            ],
        ),
    ]
```

Листинг В.2 – 0002_configurationtemplate_order_orderitem_and_more.py

```
from django.db import migrations, models
import django.db.models.deletion
class Migration(migrations.Migration):

    initial = True
    dependencies = [
    ]
    operations = [
        migrations.CreateModel(
            name='ComponentType',
            fields=[
                ('id', models.BigAutoField(auto_created=True,
primary_key=True, serialize=False, verbose_name='ID')),
                ('name', models.CharField(max_length=100, verbose_name='Тип
комплектующего')),
                ('manufacturer', models.CharField(max_length=100,
```

Окончание листинга В.2

```
verbose_name='Производитель')),
            ],
        ),
        migrations.CreateModel(
            name='Component',
            fields=[
                ('id', models.BigAutoField(auto_created=True,
primary_key=True, serialize=False, verbose_name='ID')),
                ('name', models.CharField(max_length=200,
verbose_name='Название комплектующего')),
                ('type',
models.ForeignKey(on_delete=django.db.models.deletion.CASCADE,
related_name='components', to='diploma.componenttype', verbose_name='Тип')),
            ],
        ),
    ]
```

Листинг В.3 – add_components.html

```
   {% extends 'base.html' %}
{% block content%}
    <h2>Добавить тип комплектующего</h2>
    <form method="post">
      {% csrf_token %}
      {{ type_form.as_p }}
      <button type="submit" name="add_type">Добавить тип</button>
    </form>

    <hr>

    <h2>Добавить комплектующее</h2>
    <form method="post">
      {% csrf_token %}
      {{ component_form.as_p }}
      <button type="submit" name="add_component">Добавить
комплектующее</button>
    </form>
{% endblock %}
```

ПРИЛОЖЕНИЕ Г
ЛИСТИНГ КОДА ФАЙЛА НАСТРОЙКИ ASGI-ПРИЛОЖЕНИЯ ДЛЯ DJANGO

Листинг Г.1 – asgi.py

```
   from django.core.asgi import get_asgi_application

os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'app.settings')

application = get_asgi_application()
```

ПРИЛОЖЕНИЕ Д
ЛИСТИНГ КОДА ДОБАВЛЕНИЯ РЕГИСТРАЦИИ МОДЕЛЕЙ ДЛЯ АДМИН-ПАНЕЛИ DJANGO

Листинг Д.1 – код для добавления регистрации моделей для админ-панели Django

```
from django.contrib import admin
from pathlib import Path

BASE_DIR = Path(__file__).resolve().parent.parent
SECRET_KEY = 'django-insecure-
k8=s$r^i*&#fjy&0o2&)j=@)r_qfi5c8r+ej!4#i1oh$*yl@&9'
DEBUG = True
ALLOWED_HOSTS = []
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'diploma.apps.DiplomaConfig',
]
MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
]

ROOT_URLCONF = 'app.urls'

TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]

WSGI_APPLICATION = 'app.wsgi.application'

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql',
        'NAME': 'diplomPC',
```

Окончание листинга Д.1

```
        'USER': 'postgres',
        'PASSWORD': "286670nekl",
        'HOST': 'localhost',
        'PORT': '5432',
    }
}
AUTH_PASSWORD_VALIDATORS = [
    {
        'NAME':
'django.contrib.auth.password_validation.UserAttributeSimilarityValidator',
    },
    {
        'NAME':
'django.contrib.auth.password_validation.MinimumLengthValidator',
    },
    {
        'NAME':
'django.contrib.auth.password_validation.CommonPasswordValidator',
    },
    {
        'NAME':
'django.contrib.auth.password_validation.NumericPasswordValidator',
    },
]
LANGUAGE_CODE = 'en-us'
TIME_ZONE = 'UTC'
USE_I18N = True
USE_TZ = True
STATIC_URL = 'static/'
DEFAULT_AUTO_FIELD = 'django.db.models.BigAutoField'
```

ПРИЛОЖЕНИЕ Е

ЛИСТИНГ КОДА ДЛЯ МАРШРУТА ВЕБ-ИНТЕРФЕЙСА

Листинг Е.1 – Код для маршрута веб-интерфейса

```
from django.contrib import admin
from django.urls import path

from diploma import views

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', views.main_page, name='home_page'),
    path('add_components', views.add_component, name="add_components"),
    path('ajax/load-components/', views.load_components,
name='load_components'),
    path('configurator/', views.configurator_view, name='configurator'),
    path('search/', views.search_view, name='search'),
    path('submit-order/', views.submit_order, name='submit_order'),
    path('orders/', views.orders_view, name='orders_list'),
    path('create-template/', views.create_template, name='create_template'),
    path('templates/', views.template_list_view, name='template_list'),
    path('templates/delete/<int:pk>/', views.delete_template_view,
name='template_delete'),
    path('templates/use/<int:pk>/', views.use_template_view,
name='template_use'),
]
```

ПРИЛОЖЕНИЕ Ж
ЛИСТИНГ КОДА РЕГИСТРАЦИИ

Листинг Ж.1 – Код для регистрации

```
    from django.apps import AppConfig

class DiplomaConfig(AppConfig):
    default_auto_field = 'django.db.models.BigAutoField'
    name = 'diploma'
```

ПРИЛОЖЕНИЕ И

ЛИСТИНГ КОДА ОСНОВНОЙ ЛОГИКИ ПРОГРАММЫ

Листинг И.1 – Код основной логики программы

```
from django import forms
from .models import ComponentType, Component, ConfigurationTemplate,
ConfigurationItem
from . import parsingDNS, parsingCitilink, parsingONLINETRADE, parsingTorgPC,
parsingREGARD2
import uuid
from decimal import Decimal
import json
from .models import Order, OrderItem
import re
from django.shortcuts import render, redirect, get_object_or_404

def clean_price(raw_price):
    """Убираем все символы кроме цифр и точки и возвращаем float"""
    if not raw_price:
        return 0.0
    if not isinstance(raw_price, int):
        clean = re.sub(r"[^\d.]", "", raw_price.replace(',', '.'))
    else:
        return raw_price
    try:
        return float(clean)
    except ValueError:
        return 0.0

menu = [
    {'url_name': 'home_page', 'text': 'Главная'},
    {'url_name': 'add_components', 'text': 'Компоненты'},
    {'url_name': 'configurator', 'text': 'Конфигуратор'},
    {'url_name': 'orders_list', 'text': 'Заявки'},
    {'url_name': 'template_list', 'text': 'Шаблоны'},
]

# Форма для ввода нового типа комплектующих
components_type_choices = [
    ("Процессор", "Процессор"),
    ("Охлаждение", "Охлаждение"),
    ("Материнская плата", "Материнская плата"),
    ("Оперативная память", "Оперативная память"),
    ("Видеокарта", "Видеокарта"),
    ("Жёсткий диск", "Жёсткий диск"),
    ("SSD диск", "SSD диск"),
    ("Вентиляторы", "Вентиляторы"),
    ("Корпус", "Корпус"),
    ("Блок питания", "Блок питания"),
    ("Wi-Fi адаптер", "Wi-Fi адаптер"),
    ("Звуковая карта", "Звуковая карта"),
    ("Мышь", "Мышь"),
    ("Клавиатура", "Клавиатура"),
    ("Монитор", "Монитор"),
    ("Гарнитура", "Гарнитура"),
]

class ComponentTypeForm(forms.Form):
    name = forms.ChoiceField(choices=components_type_choices, label="Тип
комплектующего")
```

Продолжение листинга И.1

```
    manufacturer = forms.CharField(max_length=100, label="Производитель")

# Форма для ввода нового комлпектующего
manufacturer_choices = ComponentType.objects.values_list('manufacturer',
flat=True).distinct()
manufacturer_choices = [(m, m) for m in manufacturer_choices]

class ComponentForm(forms.Form):
    name = forms.CharField(
        max_length=200,
        label="Название комплектующего"
    )
    manufacturer = forms.ChoiceField(
        label="Производитель"
    )

    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)
        manufacturer_choices =
ComponentType.objects.values_list('manufacturer', flat=True).distinct()
        self.fields['manufacturer'].choices = [(m, m) for m in
manufacturer_choices]
def load_components(request):
    category = request.GET.get('category')
    manufacturer = request.GET.get('manufacturer')
    try:
        component_type = ComponentType.objects.filter(name=category,
manufacturer=manufacturer).first()
        components = Component.objects.filter(type=component_type) if
component_type else []
        data = {
            'components': [{'id': c.id, 'name': c.name} for c in components]
        }
        return JsonResponse(data)
    except Exception as e:
        return JsonResponse({'error': str(e)}, status=500)
def main_page(request):
    return render(request, 'home_page.html', {'menu': menu})
```

Продолжение листинга И.1

```
def add_component(request):
    # создаём формы в динамическом режиме (внутри условий)
    if request.method == 'POST':
        if 'add_type' in request.POST:
            type_form = ComponentTypeForm(request.POST)
            component_form = ComponentForm()  # обновим тут
            if type_form.is_valid():
                ComponentType.objects.create(
                    name=type_form.cleaned_data['name'],
                    manufacturer=type_form.cleaned_data['manufacturer']
                )
                return redirect('add_components')
        elif 'add_component' in request.POST:
            component_form = ComponentForm(request.POST)
            type_form = ComponentTypeForm()  # обновим тут
            if component_form.is_valid():
                manufacturer = component_form.cleaned_data['manufacturer']
                name = component_form.cleaned_data['name']
                comp_type =
ComponentType.objects.filter(manufacturer=manufacturer).first()
                if comp_type:
                    Component.objects.create(name=name, type=comp_type)
                return redirect('add_components')
    else:
        # GET-запрос — создаём формы здесь
        type_form = ComponentTypeForm()
        component_form = ComponentForm()

    return render(request, 'add_components.html', {
        'menu': menu,
        'type_form': type_form,
        'component_form': component_form,
    })
def configurator_view(request):
    categories = [
        "Процессор", "Охлаждение", "Материнская плата", "Оперативная память",
"Видеокарта",
        "Жёсткий диск", "SSD диск 1", "SSD диск 2", "Вентиляторы", "Корпус",
        "Блок питания", "Wi-Fi адаптер", "Звуковая карта", "Операционная
система", "Мышь",
```

Продолжение листинга И.1

```
        "Клавиатура", "Монитор", "Гарнитура"
    ]

    category_data = []
    for name in categories:
        types = ComponentType.objects.filter(name=name)
        manufacturers = types.values_list('manufacturer',
flat=True).distinct()
        category_data.append({
            'label': name,
            'manufacturers': manufacturers
        })

    if request.method == 'POST':
        selected_components = {}
        # Перебираем все категории и собираем выбранные компоненты
        for category in categories:
            selected_radio = request.POST.get(f'component_{category}')
            if selected_radio:
                selected_components[category] = selected_radio

        # Формируем строку запроса для перенаправления на страницу поиска
        query_string = '&'.join([f"{key}={value}" for key, value in
selected_components.items()])
        return redirect(f'/search/?clear=true&{query_string}')

    return render(request, 'configurator.html', {'categories': category_data,
'menu': menu})

from django.views.decorators.csrf import csrf_exempt
from django.http import JsonResponse

@csrf_exempt
def create_template(request):
    if request.method == 'POST':
        name = request.POST.get("template_name")
        if not name:
            return JsonResponse({"success": False, "error": "Название не
указано"})
```

Продолжение листинга И.1

```
        template = ConfigurationTemplate.objects.create(name=name)

        for key, value in request.POST.items():
            if key.startswith("component_"):
                category = key.replace("component_", "")
                try:
                    component = Component.objects.get(id=value)
                    ConfigurationItem.objects.create(template=template,
category=category, component=component)
                except Component.DoesNotExist:
                    continue

        return JsonResponse({"success": True})

    return JsonResponse({"success": False, "error": "Метод не разрешён"},
status=405)
def search_view(request):
    components = {}
    selected_components = request.GET.dict()
    clear_local_storage = request.GET.get("clear") == "true"  # <-- сюда!

    selected_category = None
    for category, component_id in selected_components.items():
        if category != "clear":  # <-- защита от "clear" как категории
            try:
                component = Component.objects.get(id=component_id)
                components[category] = component.name
            except Component.DoesNotExist:
                pass

    products = []
    if request.method == 'POST':
        selected_category = request.POST.get('category')
        component_name = components.get(selected_category)

        if component_name:
            content = parsingDNS.get_products_page(component_name)
            raw_products = parsingDNS.parsing_content(content)

            content = parsingCitilink.get_products_page(component_name)
```

Продолжение листинга И.1

```
            raw_products.extend(parsingCitilink.parsing_content(content))
            content = parsingREGARD2.get_products_page(component_name)
            raw_products.extend(parsingREGARD2.parse_products(content))

            content = parsingONLINETRADE.get_products_page(component_name)
            raw_products.extend(parsingONLINETRADE.parse_products(content))

            content = parsingTorgPC.get_products_page(component_name)
            raw_products.extend(parsingTorgPC.parsing_content(content))

            # Обрабатываем цену для каждого товара
            for p in raw_products:
                p["price"] = clean_price(p.get("price"))
                products.append(p)

            for product in products:
                print('{')
                for key, value in product.items():
                    print(f'    {key}: {value}')
                print('}')

        # Не очищаем localStorage после POST!
        clear_local_storage = False

    return render(request, 'search.html', {
        'components': components,
        'products': products,
        'selected_category': selected_category,
        'clear_local_storage': clear_local_storage,
        'menu': menu,
    })

@csrf_exempt
def submit_order(request):
    if request.method == 'POST':
        try:
            data = json.loads(request.body)

            total = Decimal('0.00')
            items = []
```

Продолжение листинга И.1

```
            for cat, info in data.items():
                quantity = int(info.get("quantity", 1))
                price = Decimal(str(info.get("price", "0")))
                total += price * quantity
                items.append({
                    "category": cat,
                    "name": info["name"],
                    "link": info["link"],
                    "price": price,
                    "quantity": quantity
                })

            order = Order.objects.create(
                total_price=total,
                order_number=str(uuid.uuid4())[:8].upper()
            )
            for item in items:
                OrderItem.objects.create(order=order, **item)

            return JsonResponse({"success": True, "order_number":
order.order_number})
        except Exception as e:
            return JsonResponse({"success": False, "error": str(e)},
status=400)
    return JsonResponse({"error": "Method not allowed"}, status=405)
def orders_view(request):
    orders = Order.objects.prefetch_related('items').order_by('-created_at')
    created = request.GET.get("created")  # <-- добавлено
    return render(request, 'orders_list.html', {
        'orders': orders,
        'created': created,  # <-- добавлено
        'menu': menu
    })
def template_list_view(request):
    templates =
ConfigurationTemplate.objects.prefetch_related('items__component__type').orde
r_by('-created_at')
    return render(request, 'template_list.html', {'templates': templates,
'menu': menu})
```

Окончание листинга И.1

```
def delete_template_view(request, pk):
    template = get_object_or_404(ConfigurationTemplate, pk=pk)
    template.delete()
    return redirect('template_list')  # Убедись, что имя пути совпадает

def use_template_view(request, pk):
    template = get_object_or_404(ConfigurationTemplate, pk=pk)
    query_params = {
        item.component.type.name: item.component.id for item in
template.items.all()
    }
    query_string = "&".join(f"{key}={value}" for key, value in
query_params.items())
    return redirect(f"/search/?clear=true&{query_string}")
```

ПРИЛОЖЕНИЕ К

ЛИСТИНГ КОДА ЗАПУСКА СКРИПТА DJANGO

Листинг К.1 – Код для запуска скрипта Django

```
   #!/usr/bin/env python
"""Django's command-line utility for administrative tasks."""
import os
import sys
def main():
    """Run administrative tasks."""
    os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'app.settings')
    try:
        from django.core.management import execute_from_command_line
    except ImportError as exc:
        raise ImportError(
            "Couldn't import Django. Are you sure it's installed and "
            "available on your PYTHONPATH environment variable? Did you "
            "forget to activate a virtual environment?"
        ) from exc
    execute_from_command_line(sys.argv)


if __name__ == '__main__':
    main()
```

ПРИЛОЖЕНИЕ Л

ЛИСТИНГ КОДА БАЗОВОГО HTML-ШАБЛОНА

Листинг Л.1 – Код для базового HTML-шаблона

```
   <!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>{{ page_title }}</title>
        <style>
            * {
                box-sizing: border-box;
            }

            body {
                font-family: Arial, sans-serif;
                margin: 0;
                padding: 0;
                background-color: #151a22;
                display: flex;
                color: #8490a4;
                flex-direction: column;
                justify-content: flex-start;
                align-items: center;
                height: 100vh;
            }

            header {
                width: 100%; /* Устанавливаем хедер на всю ширину экрана */
                background-color: #212936;
                padding: 15px 0; /* Дополнительный отступ по вертикали */
                box-shadow: 0 4px 6px rgba(0, 0, 0, 0.1);
                position: fixed; /* Закрепляем хедер */
                top: 0;
                left: 0;
                z-index: 1000; /* Убедимся, что хэдер будет поверх контента
*/
                display: flex; /* Используем flexbox для выравнивания
элементов */
                justify-content: flex-start; /* Элементы выравнены по левому
краю */
                align-items: center; /* Выравниваем по вертикали */
            }

            nav {
                display: flex;
                justify-content: flex-start;
                gap: 20px;
                padding: 0 20px;
            }

            .content {
                width: 75%;
                background-color: #151a22;
                box-shadow: 0 4px 10px rgba(0, 0, 0, 0.1);
                padding: 20px;
                margin-top: 80px; /* Делаем отступ на высоту хедера */
                min-height: calc(100vh — 80px); /* Учитываем отступ, чтобы
контент занимал все пространство */
            }
```

Окончание листинга Л.1

```
            a.button {
                text-decoration: none; /* Убираем подчеркивание */
                color: #a7e200; /* Цвет текста как для ссылок */
                background: none; /* Убираем фон */
                padding: 0 20px; /* Добавляем горизонтальные отступы */
                font-size: 16px; /* Размер шрифта */
                border: none; /* Убираем границу */
                cursor: pointer; /* Курсор в виде руки */
                display: inline-block; /* Элемент будет вести себя как блок,
но с инлайновыми свойствами */
                height: 100%; /* Высота кнопки совпадает с высотой хедера */
                line-height: 45px; /* Устанавливаем вертикальное выравнивание
текста (высоту строки) */
                transition: color 0.3s ease; /* Плавный переход цвета при
наведении */
            }
            a.button:hover {
                color: white; /* Цвет текста при наведении */
            }
            a.button:active {
                color: white; /* Цвет текста при активном нажатии */
            }
            a.button:focus {
                outline: none; /* Убираем стандартный фокус */
                box-shadow: 0 0 5px rgba(0, 123, 255, 0.8); /* Добавляем
фокусный эффект */
            }
            main {
                padding: 20px;
                width: 100%;
            }
        </style>
</head>
<body>
    <header>
        <nav>
            {% for button in menu %}
                <a href="{% url button.url_name %}" class="button">{{
button.text }}</a>
            {% endfor %}
        </nav>
    </header>

    <div class="content">
        {% block content %} {% endblock %}
    </div>
</body>
</html>
```

ПРИЛОЖЕНИЕ М

ЛИСТИНГ КОДА ШАБЛОНА КОНФИГУРАТОРА ПК

Листинг М.1 – Код для шаблона конфигуратора ПК

```
   {% extends 'base.html' %}

{% block content %}
  <h2>Конфигуратор</h2>
  <form method="post" id="configForm">
    {% csrf_token %}
    {% for category in categories %}
      <div style="margin-bottom: 30px; display: flex; justify-content: space-
between; align-items: flex-start;">
        <label style="color: #c3cee1; font-size: 16px; margin-top:
8px;"><strong>{{ category.label }}</strong></label>

        <div style="text-align: right; width: 300px;">
          <select
            onchange="loadComponents({{ forloop.counter }}, '{{
category.label }}', this.value)"
            style="width: 300px; height: 40px; font-size: 14px; background-
color: #151a22; color: #8490a4; border: 1px solid #c3cee1; padding: 5px;
margin-bottom: 10px;">
            <option value="">Выберите производителя</option>
            {% for m in category.manufacturers %}
              <option value="{{ m }}">{{ m }}</option>
            {% endfor %}
          </select>

          <div id="components-{{ forloop.counter }}" style="margin-top:
10px;"></div>
        </div>
      </div>
    {% endfor %}
    <button type="submit" id="collectButton">Собрать</button>
    <button type="button" onclick="openTemplateModal()" style="margin-left:
20px;">🖫 Создать шаблон</button>
  </form>

  <!-- Модальное окно -->
  <div id="templateModal" style="display:none; position:fixed; top:0; left:0;
width:100%; height:100%; background:rgba(0,0,0,0.6); justify-content:center;
align-items:center;">
    <div style="background:#1d232e; padding:20px; border-radius:8px;
width:300px; text-align:center;">
      <h3 style="color:#c3cee1;">Название шаблона</h3>
      <input type="text" id="templateName" placeholder="Мой шаблон"
style="width:100%; padding:8px; margin:10px 0;">
      <div style="margin-top: 10px;">
        <button onclick="saveTemplate()">Сохранить</button>
        <button onclick="closeTemplateModal()">Отмена</button>
      </div>
    </div>
  </div>

  <script>
    function loadComponents(id, category, manufacturer) {
      const container = document.getElementById('components-' + id);
      if (!manufacturer) {
        container.innerHTML = '';
```

Продолжение листинга М.1

```
      return;
      }

      container.innerHTML = 'Загрузка...';

      fetch(`/ajax/load-
components/?category=${encodeURIComponent(category)}&manufacturer=${encodeURI
Component(manufacturer)}`)
        .then(response => response.json())
        .then(data => {
          container.innerHTML = '';
          data.components.forEach(component => {
            const wrapper = document.createElement('div');
            wrapper.style.border = '2px solid #2c3849';
            wrapper.style.padding = '8px';
            wrapper.style.borderRadius = '5px';
            wrapper.style.marginBottom = '10px';
            wrapper.style.width = '300px';

            const label = document.createElement('label');
            label.setAttribute('for', `comp-${component.id}`);
            label.style.display = 'flex';
            label.style.alignItems = 'flex-start';
            label.style.gap = '10px';
            label.style.color = '#c3cee1';
            label.style.wordBreak = 'break-word';
            label.style.lineHeight = '1.4';
            label.style.whiteSpace = 'normal';
            label.style.width = '100%';

            const radio = document.createElement('input');
            radio.type = 'radio';
            radio.name = 'component_' + category;
            radio.value = component.id;
            radio.id = `comp-${component.id}`;
            radio.style.marginTop = '3px';
            radio.style.flexShrink = '0';

            const text = document.createElement('span');
            text.innerText = component.name;
            text.style.flexGrow = '1';
            text.style.textAlign = 'left';

            label.appendChild(radio);
            label.appendChild(text);
            wrapper.appendChild(label);
            container.appendChild(wrapper);
          });

          if (data.components.length === 0) {
            container.innerHTML = '<em style="color: #8490a4;">Нет доступных
компонентов</em>';
          }
        })
        .catch(err => {
          container.innerHTML = '<em style="color: red;">Ошибка при загрузке
компонентов</em>';
          console.error('Ошибка при загрузке компонентов:', err);
        });
```

Окончание листинга М.1

```
    }
    function openTemplateModal() {
      document.getElementById("templateModal").style.display = "flex";
    }

    function closeTemplateModal() {
      document.getElementById("templateModal").style.display = "none";
    }

    function saveTemplate() {
      const name = document.getElementById("templateName").value;
      if (!name.trim()) {
        alert("Введите название шаблона");
        return;
      }

      const form = document.getElementById("configForm");
      const formData = new FormData(form);
      formData.append("template_name", name);

      fetch("/create-template/", {
        method: "POST",
        headers: {
          "X-CSRFToken": getCookie("csrftoken")
        },
        body: formData
      })
      .then(response => response.json())
      .then(data => {
        if (data.success) {
          alert("Шаблон сохранён");
          closeTemplateModal();
        } else {
          alert("Ошибка: " + data.error);
        }
      })
      .catch(err => {
        alert("Ошибка при отправке");
        console.error(err);
      });
    }

    function getCookie(name) {
      let cookieValue = null;
      if (document.cookie && document.cookie !== "") {
        const cookies = document.cookie.split(";");
        for (let cookie of cookies) {
          cookie = cookie.trim();
          if (cookie.startsWith(name + "=")) {
            cookieValue = decodeURIComponent(cookie.slice(name.length + 1));
            break;
          }
        }
      }
      return cookieValue;
    }
  </script>
{% endblock %}
```

ПРИЛОЖЕНИЕ Н

ЛИСТИНГ КОДА ОТОБРАЖЕНИЯ СПИСКА ЗАЯВОК С ВЫПАДАЮЩИМИ БЛОКАМИ

Листинг Н.1 – Код для отображения списка заявок с выпадающими блоками

```
   {% extends 'base.html' %}

{% block content %}
    <h1>Список заявок</h1>
    {% if created %}
    <p style="color: #00cc88; background-color: #1a2d1f; padding: 10px 15px;
border-left: 4px solid #00cc88; border-radius: 4px; margin-bottom: 20px;">
        ✓ Заявка №{{ created }} успешно создана!
    </p>
    {% endif %}

    <style>
        .order-block {
            margin-bottom: 10px;
            border: 1px solid #2c3443;
            background-color: #1d232e;
            border-radius: 4px;
            overflow: hidden;
        }

        .order-header {
            cursor: pointer;
            padding: 12px 16px;
            background-color: #212936;
            color: #a7e200;
            font-weight: bold;
        }

        .order-body {
            display: none;
            padding: 12px 16px;
            background-color: #1a1f29;
            color: #8490a4;
        }

        .order-body ul {
            list-style: none;
            padding: 0;
            margin: 0;
        }

        .order-body li {
            padding: 6px 0;
            border-bottom: 1px solid #2c3443;
        }

        .order-body li:last-child {
            border-bottom: none;
        }

        .order-link {
```

Окончание листинга Н.1

```
            color: #a7e200;
            text-decoration: underline;
        }
        .order-link:hover {
            color: #ffffff;
        }
    </style>

    {% for order in orders %}
        <div class="order-block">
            <div class="order-header" onclick="toggleOrder('order-{{ order.id
}}')">
                Заявка №{{ order.order_number }} | {{
order.created_at|date:"d.m.Y H:i" }} | Сумма: {{ order.total_price }} ₽
            </div>
            <div class="order-body" id="order-{{ order.id }}">
                <ul>
                    {% for item in order.items.all %}
                        <li>
                            <strong>{{ item.category }}:</strong> {{
item.name }}<br>
                            <a href="{{ item.link }}" target="_blank"
class="order-link">Ссылка</a><br>
                            Кол-во: {{ item.quantity }} | Цена: {{ item.price
}} ₽ | Итого: {{ item.total }} ₽
                        </li>
                    {% endfor %}
                </ul>
            </div>
        </div>
    {% empty %}
        <p>Заявок пока нет.</p>
    {% endfor %}

    <script>
        function toggleOrder(id) {
            const el = document.getElementById(id);
            el.style.display = el.style.display === 'block' ? 'none' :
'block';
        }
    </script>
{% endblock %}
```

ПРИЛОЖЕНИЕ П
ЛИСТИНГ КОДА ОТОБРАЖЕНИЯ РЕЗУЛЬТАТОВ ПОИСКА ТОВАРОВ, СОБРАННЫХ С РАЗНЫХ САЙТОВ

Листинг П.1 – Код для отображения результатов поиска товаров, собранных с разных сайтов

```
   {% extends 'base.html' %}

{% block content %}
<h2>Выбранные компоненты</h2>

<form method="post" onsubmit="clearSelectionIfNewComponent()" style="margin-
top: 20px;">
  {% csrf_token %}
  <input type="hidden" id="selected-category" name="selected_category"
value="{{ selected_category }}">

  <select name="category"
          style="width: 300px; height: 40px; font-size: 14px; background-
color: #151a22; color: #8490a4; border: 1px solid #c3cee1; padding: 5px;
margin-bottom: 20px;">
    {% for category, component_name in components.items %}
      <option value="{{ category }}"
              {% if category == selected_category %}selected{% endif %}>
        {{ category }}: {{ component_name }}
      </option>
    {% empty %}
      <option disabled selected>Нет выбранных компонентов</option>
    {% endfor %}
  </select>

  <button type="submit"
          style="padding: 12px 24px; font-size: 16px; background-color:
#007bff; color: white; border: none; border-radius: 4px; cursor: pointer;">
    🔍 Найти товары
  </button>
</form>

<!-- Выбранные товары -->
<div id="selected-items"
     style="margin-top: 30px; background-color: #1b222d; padding: 20px;
border-radius: 8px; display: none;">
  <div style="display: flex; justify-content: space-between; align-items:
center;">
    <h3 style="color: #ffffff;">📄 Выбранные товары:</h3>
    <button id="toggle-list" style="background: none; border: none; color:
#00ccff; cursor: pointer;">Свернуть</button>
  </div>
  <ul id="selected-list" style="list-style: none; padding: 0; color:
#ffffff;"></ul>

  <p id="total-price" style="color: #00cc88; font-weight: bold; margin-top:
15px;"></p>
  <div style="text-align: right; margin-top: 20px;">
    <button onclick="submitOrder()" style="padding: 10px 20px; background-
color: #ffc107; color: #000; border: none; border-radius: 4px; cursor:
pointer;">
```

Продолжение листинга П.1

```
      🎁 Сформировать заявку
    </button>
  </div>
</div>

{% if products %}
  <h3 style="margin-top: 40px;">Найденные товары:</h3>
  <div style="display: flex; flex-direction: column; gap: 20px; margin-top:
20px;">
    {% for product in products %}
      <div class="product-card"
           data-name="{{ product.name }}"
           data-link="{{ product.product_link }}"
           data-category="{{ selected_category }}"
           data-price="{{ product.price|default:"0"|floatformat }}"
           style="display: flex; align-items: center; background-color:
#1e2633; border-radius: 8px; box-shadow: 0 4px 6px rgba(0, 0, 0, 0.2);
overflow: hidden; padding: 15px; color: #ffffff; transition: background-color
0.2s ease;"
           onmouseover="this.style.backgroundColor='#2a3445'"
           onmouseout="this.style.backgroundColor='#1e2633'">
        <a href="{{ product.product_link }}" target="_blank" style="text-
decoration: none;">
          <img src="{{ product.img_link }}" alt="{{ product.name }}"
               style="width: 120px; height: auto; object-fit: contain;
margin-right: 20px; border-radius: 4px;">
        </a>
        <div style="flex: 1;">
          <p style="font-size: 16px; font-weight: 600; margin: 0 0 8px;">{{
product.name }}</p>

          {% if product.price %}
            <p style="font-size: 15px; font-weight: 500; color: #00cc88;
margin: 0 0 8px;">
              {{ product.price }}
            </p>
          {% endif %}

          {% if product.info %}
            <div style="font-size: 13px; color: #b0b8c2;">
              {% for line in product.info %}
                <div>{{ line }}</div>
              {% endfor %}
            </div>
          {% endif %}

          <button class="add-to-request-btn"
                  style="margin-top: 10px; padding: 8px 16px; background-
color: #28a745; color: white; border: none; border-radius: 4px; cursor:
pointer;">
            ➕ Добавить в заявку
          </button>
        </div>
      </div>
    {% endfor %}
  </div>
{% endif %}

<script>
```

Продолжение листинга П.1

```
const CLEAR_LOCAL_STORAGE = {{ clear_local_storage|yesno:"true,false" }};

function renderSelectedItems() {
  const container = document.getElementById("selected-items");
  const list = document.getElementById("selected-list");
  const totalBlock = document.getElementById("total-price");
  list.innerHTML = "";

  const data = JSON.parse(localStorage.getItem("requestItems") || "{}");
  const categories = Object.keys(data);
  let total = 0;

  if (categories.length === 0) {
    container.style.display = "none";
    return;
  }

  container.style.display = "block";

  categories.forEach(category => {
    const item = data[category];
    const quantity = item.quantity || 1;
    const price = parseFloat(item.price || 0);
    total += price * quantity;

    const li = document.createElement("li");
    li.style.marginBottom = "10px";
    li.innerHTML = `
      <strong>${category}:</strong>
      <a href="${item.link}" target="_blank" style="color:
#00ccff;">${item.name}</a>
      <span style="margin-left: 10px;">
        Кол-во:
        <button onclick="changeQuantity('${category}', -1)">━</button>
        <span id="qty-${category}">${quantity}</span>
        <button onclick="changeQuantity('${category}', 1)">✚</button>
      </span>
    `;
    list.appendChild(li);
  });

  totalBlock.innerText = `💰 Общая стоимость: ${total.toFixed(2)} ₽`;
}

function changeQuantity(category, delta) {
  const data = JSON.parse(localStorage.getItem("requestItems") || "{}");
  if (!data[category]) return;
  data[category].quantity = Math.max(1, (data[category].quantity || 1) +
delta);
  localStorage.setItem("requestItems", JSON.stringify(data));
  renderSelectedItems();
}

function clearSelectionIfNewComponent() {
  const currentCategory = document.getElementById("selected-category").value;
  localStorage.setItem("lastSelectedCategory", currentCategory);
}
```

Продолжение листинга П.1

```
document.addEventListener("DOMContentLoaded", function () {
  if (CLEAR_LOCAL_STORAGE) {
    localStorage.removeItem("requestItems");
    localStorage.removeItem("lastSelectedCategory");
  }

  renderSelectedItems();

  const buttons = document.querySelectorAll(".add-to-request-btn");
  buttons.forEach(button => {
    const card = button.closest(".product-card");
    const name = card.dataset.name;
    const link = card.dataset.link;
    const category = card.dataset.category;
    const price = parseFloat(card.dataset.price || "0");

    const stored = JSON.parse(localStorage.getItem("requestItems") || "{}");
    if (stored[category] && stored[category].name === name) {
      button.innerText = "✓ Добавлено";
      button.disabled = true;
      button.style.backgroundColor = "#6c757d";
    }

    button.addEventListener("click", function () {
      buttons.forEach(btn => {
        const btnCard = btn.closest(".product-card");
        if (btnCard.dataset.category === category) {
          btn.innerText = "+ Добавить в заявку";
          btn.disabled = false;
          btn.style.backgroundColor = "#28a745";
        }
      });

      const requestItems = JSON.parse(localStorage.getItem("requestItems") ||
"{}");
      requestItems[category] = {
        name,
        link,
        price,
        quantity: 1
      };
      localStorage.setItem("requestItems", JSON.stringify(requestItems));

      this.innerText = "✓ Добавлено";
      this.disabled = true;
      this.style.backgroundColor = "#6c757d";

      renderSelectedItems();
    });
  });

  const toggleBtn = document.getElementById("toggle-list");
  if (toggleBtn) {
    toggleBtn.addEventListener("click", function () {
      const list = document.getElementById("selected-list");
      if (list.style.display === "none") {
        list.style.display = "block";
        this.innerText = "Свернуть";
      } else {
```

Окончание листинга П.1

```
        list.style.display = "none";
        this.innerText = "Развернуть";
      }
    });
  }
});

function submitOrder() {
  const data = JSON.parse(localStorage.getItem("requestItems") || "{}");

  fetch("/submit-order/", {
    method: "POST",
    headers: {
      "Content-Type": "application/json",
      "X-CSRFToken": getCookie("csrftoken")
    },
    body: JSON.stringify(data)
  })
  .then(response => response.json())
  .then(res => {
    if (res.success) {
      localStorage.removeItem("requestItems");
      window.location.href = "/orders/?created=" + res.order_number;
    } else {
      alert("Ошибка при отправке заявки: " + res.error);
    }
  });
}


function getCookie(name) {
  let cookieValue = null;
  if (document.cookie && document.cookie !== '') {
    const cookies = document.cookie.split(';');
    for (let i = 0; i < cookies.length; i++) {
      const cookie = cookies[i].trim();
      if (cookie.substring(0, name.length + 1) === (name + '=')) {
        cookieValue = decodeURIComponent(cookie.substring(name.length + 1));
        break;
      }
    }
  }
  return cookieValue;
}
</script>
{% endblock %}
```

ПРИЛОЖЕНИЕ Р
ЛИСТИНГ КОДА ОТОБРАЖЕНИЯ СОХРАНЕННЫХ РЕЗУЛЬТАТОВ

Листинг Р.1 – Код для отображения сохраненных результатов

```
   {% extends 'base.html' %}

{% block content %}
  <h2>Мои шаблоны конфигураций</h2>

  <style>
    .template-block {
        margin-bottom: 10px;
        border: 1px solid #2c3443;
        background-color: #1d232e;
        border-radius: 4px;
        overflow: hidden;
    }

    .template-header {
        cursor: pointer;
        padding: 12px 16px;
        background-color: #212936;
        color: #a7e200;
        font-weight: bold;
    }

    .template-body {
        display: none;
        padding: 12px 16px;
        background-color: #1a1f29;
        color: #8490a4;
    }

    .template-body ul {
        list-style: none;
        padding: 0;
        margin: 0;
    }

    .template-body li {
        padding: 6px 0;
        border-bottom: 1px solid #2c3443;
    }

    .template-body li:last-child {
        border-bottom: none;
    }

    .template-buttons {
        margin-top: 10px;
        display: flex;
        gap: 10px;
    }

    .btn {
        padding: 6px 12px;
        background-color: #2c3849;
        color: #a7e200;
```

Окончание листинга Р.1

```
        border: none;
        cursor: pointer;
        border-radius: 3px;
        text-decoration: none;
    }

    .btn:hover {
        background-color: #3b4a5a;
    }
  </style>

  {% for template in templates %}
    <div class="template-block">
        <div class="template-header" onclick="toggleTemplate('tpl-{{
template.id }}')">
            {{ template.name }} | {{ template.created_at|date:"d.m.Y H:i" }}
        </div>
        <div class="template-body" id="tpl-{{ template.id }}">
            <ul>
                {% for item in template.items.all %}
                    <li>
                        <strong>{{ item.component.type.name }}:</strong> {{
item.component.name }}
                    </li>
                {% endfor %}
            </ul>
            <div class="template-buttons">
                <a href="{% url 'template_use' template.id %}" class="btn">🔍
Поиск</a>
                <a href="{% url 'template_delete' template.id %}" class="btn"
onclick="return confirm('Удалить шаблон?')">🗑 Удалить</a>
            </div>
        </div>
    </div>
  {% empty %}
    <p style="color: #8490a4;">Шаблонов пока нет.</p>
  {% endfor %}

  <script>
    function toggleTemplate(id) {
        const el = document.getElementById(id);
        el.style.display = el.style.display === 'block' ? 'none' : 'block';
    }
  </script>
{% endblock %}
```