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
«___» _____ 2024 г.

Разработка мобильного приложения «Интерактивная карта ЮУрГУ» с использованием дополненной реальности

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ
ЮУРГУ-090301.2024.414 ПЗ ВКР

Руководитель работы,
к.п.н., доцент каф. ЭВМ
_____ Ю.Г. Плаксина
«___» _____2024 г.

Автор работы,
студент группы КЭ-406
_____А.Р. Рухлов
«___» _____2024 г.

Нормоконтролёр,
ст. преп. каф. ЭВМ
_____ С.В. Сяськов
«___» _____2024 г.

Челябинск-2024

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
«___» _____2024 г.

ЗАДАНИЕ

на выпускную квалификационную работу бакалавра

студенту группы КЭ-406
Рухлову Антону Романовичу
обучающемуся по направлению
09.03.01 «Информатика и вычислительная техника»

Тема работы: «Разработка мобильного приложения "Интерактивная карта ЮУрГУ" с использованием дополненной реальности» утверждена приказом по университету от «__» _____ 2024 г. № _____

Срок сдачи студентом законченной работы: 1 июня 2024 г.

Исходные данные к работе: поддержка мобильным устройством Depth API; поддержка мобильным устройством OpenGL ES 3.0 и более поздней версии; оперативная память мобильного устройства не менее 4 ГБ.

Перечень подлежащих разработке вопросов:

1) аналитический обзор научно-технической, нормативной и методической литературы по тематике работы;
2) формулирование требований, выбор среды и средств разработки;
3) разработка программного кода мобильного приложения;
4) тестирование мобильного приложения.

Дата выдачи задания: 1 декабря 2023 г.

Руководитель работы _____/ *Ю.Г. Плаксина/*

Студент _____/ *А.Р. Рухлов/*

КАЛЕНДАРНЫЙ ПЛАН

Этап	Срок сдачи	Подпись руководителя
Аналитический обзор научно-технической, нормативной и методической литературы по тематике работы, анализ существующих аналогов	01.02.2024	
Формулирование требований, выбор среды и средств разработки	03.03.2024	
Разработка программного кода мобильного приложения	07.05.2024	
Тестирование мобильного приложения	15.05.2024	
Компоновка работы и сдача на нормоконтроль	25.05.2024	
Подготовка презентации и доклада	30.05.2024	

Руководитель работы _____/ *Ю.Г. Плаксина/*

Студент _____/ *А.Р. Рухлов* /

АННОТАЦИЯ

Рухлов А. Р. Разработка мобильного приложения "Интерактивная карта ЮУрГУ" с использованием дополненной реальности – Челябинск: ЮУрГУ, ВШЭКН; 2024, 70 с., 22 ил., библиогр. список – 51 наим.

В рамках выпускной квалификационной работы проводится анализ перспектив использования технологии дополненной реальности в контексте внутренней навигации в зданиях. Также рассматриваются различные способы получения местоположения внутри здания, проводится ознакомление с алгоритмами расчета кратчайшего маршрута внутри здания.

Работа представляет обзор существующих решений, включая приложения для внутренней навигации, которые предлагают различные функциональные возможности.

Итогом проделанной работы является разработанное мобильное приложение для отображения маршрутов по зданию университета на языке программирования Kotlin для операционной системы Android.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Современные университеты являются сложными пространствами с множеством зданий, коридоров и помещений, что делает навигацию внутри них часто вызывающей трудности для студентов, сотрудников и посетителей. В условиях такого многообразия структуры и организации пространства возникает необходимость в эффективных средствах для ориентирования и перемещения внутри университета.

Вместе с тем, с развитием технологий дополненной реальности (AR) открываются новые возможности для улучшения процесса навигации внутри зданий. Использование AR в мобильных приложениях позволяет пользователям получать визуальные указания и информацию о местоположении объектов прямо на экране своего смартфона или другого устройства.

Исходя из вышесказанного, актуальной задачей становится разработка мобильного приложения для навигации внутри университета с отображением маршрута в AR. Такое приложение позволит пользователям более удобно и эффективно перемещаться по зданиям, быстро находить нужные аудитории, кабинеты, библиотеки и другие объекты университетского комплекса.

1. АНАЛИТИЧЕСКИЙ ОБЗОР СОВРЕМЕННОЙ НАУЧНО-ТЕХНИЧЕСКОЙ, НОРМАТИВНОЙ, МЕТОДИЧЕСКОЙ ЛИТЕРАТУРЫ, ЗАТРАГИВАЮЩЕЙ НАУЧНО-ТЕХНИЧЕСКУЮ ПРОБЛЕМУ, ИССЛЕДУЕМУЮ В РАБОТЕ

В век технологического развития, когда цифровые инновации проникают во все сферы нашей повседневной жизни, вопросы навигации внутри зданий приобретают особую актуальность. Однако стоит признать, что потребность в ориентации в незнакомой среде является фундаментальной для человечества, преследуя нас на протяжении веков. С течением времени, люди всегда стремились создавать инструменты, облегчающие этот процесс и помогающие освоить новые территории [1].

Одним из таких инструментов является перспективное направление в информационных технологиях – дополненная реальность. Дополненная реальность (augmented reality, AR) – технология наложения цифровых данных на физический мир в реальном времени, с использованием компьютерных устройств [2].

Существует несколько типов технологий дополненной реальности: с использованием маркеров и без. Выбор типа применяемой технологии определяет, как будет отображаться информация. AR с использованием маркеров опирается на распознавание изображений для идентификации заранее запрограммированных объектов. При появлении этих объектов в поле зрения, они становятся опорными точками, помогая устройству определить положение камеры и ее ориентацию. Это достигается за счет перевода камеры в режим отображения оттенков серого и поиска маркера для сравнения с информацией в базе данных. При обнаружении совпадения устройство использует эту информацию для математического определения местоположения AR-изображения. AR без использования маркеров более сложен, так как отсутствует конкретная точка фокусировки устройства. Оно вынуждено распознавать объекты по мере их появления в поле зрения. Путем использования алгоритмов распознавания цветов, узоров и других характеристик, устройство определяет тип объекта, а затем, совмещая данные о времени, акселерометре, GPS и компасе, наложит изображение на физическое пространство вокруг вас с помощью камеры [3].

Технология дополненной реальности может работать на смартфонах, планшетах, умных экранах и очках дополненной реальности, что позволяет ее применять практически во всех областях.

Автор статьи [4] приводит примеры применения дополненной реальности в строительстве. В работе рассмотрены возможности использования AR для обучения сотрудников строительного процесса, тестирование и наглядный просмотр строительных конструкций, просмотр виртуального голографического макета строения с различных ракурсов. Таким образом, автор предполагает, что применение дополненной реальности позволит сократить сроки сдачи проектов, материальные затраты и значительно уменьшить влияние человеческого фактора.

В работе [5] описывается возможность применения AR на уроках геометрии. Авторы отмечают, что это позволит решить проблему восприятия учениками трехмерных объектов при изучении стереометрии. Такое применение дополненной реальности разнообразит процесс обучения, позволяя ученикам визуализировать и взаимодействовать с геометрическими фигурами прямо в классе. Это создаст более глубокое понимание пространственных отношений и форм, а также способствует более увлекательному и запоминающемуся опыту обучения.

В исследовании [6] представлены результаты использования технологии дополненной реальности в медицинской практике для улучшения процесса планирования и оперативного лечения пациентов с травмами и заболеваниями опорно-двигательной системы. Автор отмечает, что применение AR позволяет хирургам создавать голографические модели на основе данных предварительного обследования, позволяя им отработать этапы и технику предстоящей операции. Это способствует снижению рисков во время операции и позволяет собирать научную и практическую информацию для дальнейшего анализа.

Автор данной статьи [7] исследует применение технологии дополненной реальности в сфере навигации среди пользователей карт. Он основывает свою идею на проблемах, с которыми сталкиваются многие люди, исследующие новые места и пытающиеся ориентироваться с помощью карт и навигационных приложений. Автор утверждает, что традиционные варианты использования карт, такие как

запуск приложений, установка маршрутов и ориентация с помощью вращения на месте, могут быть неэффективными и затратными по времени для пользователей.

Идея автора заключается в том, чтобы внедрить технологию дополненной реальности непосредственно в процесс пешеходной навигации. Он предлагает использовать AR для отображения направлений движения непосредственно на изображении, полученном с камеры мобильного устройства. Это позволит пользователям легче и быстрее определять нужное направление, минуя необходимость в сложных действиях по поиску пешеходного маршрута с помощью приложений. Автор считает, что внедрение дополненной реальности в навигацию упростит процесс ориентации и улучшит пользовательский опыт, обеспечивая более интуитивный и простой способ навигации в незнакомых местах.

Изучение дополненной реальности в контексте навигации по местности дает представление о её потенциале в области внутреннего позиционирования, особенно в помещениях. Люди также испытывают трудности в поиске маршрутов внутри зданий, особенно в больших университетах, торговых центрах, аэропортах или госпиталях. Перенос технологии дополненной реальности на внутреннюю навигацию может быть важным шагом в облегчении этого процесса. Разработка инструментов, позволяющих пользователям легко находить нужные места внутри здания, обозначать оптимальные маршруты или получать информацию о точном местоположении интересующих объектов, может улучшить опыт их перемещения в таких средах [8].

В ходе разработки системы внутренней навигации, автор [9] выявил проблему точного определения местоположения пользователей. Он пришел к заключению о невозможности эффективного применения традиционных технологий, таких как GPS (глобальная система позиционирования), внутри зданий из-за технических ограничений, связанных с ослаблением сигналов спутников при их прохождении через стены, потолки и другие структурные элементы здания. Это существенно снижает точность определения местоположения и ухудшает качество данных, что делает точное позиционирование внутри здания невозможным.

Одно из решений этой проблемы приводится в статье [10]. Автор предлагает использовать определенные знаки для выявления местоположения и ориентации устройства. Эти знаки обладают уникальными узорами, которые способны быть четко идентифицированы алгоритмами компьютерного зрения. Если каждый маркер будет уникальным и различимым, то это позволит указывать на конкретную точку в реальном пространстве. Один из подходов, который рассматривает автор, предполагает размещение нескольких QR-кодов в помещении для повторной калибровки устройства. При несовпадении визуальных компонентов с реальными объектами, пользователю будет предлагаться ручная калибровка устройства путем сканирования QR-кода.

Статьи [11, 12] содержат альтернативное решение проблемы позиционирования внутри здания. Подход основан на использовании маяков, реализующие протокол Bluetooth Low Energy 4.0 (BLE 4.0), с заранее известными координатами по всей территории здания. Эти устройства представляют собой компактные передатчики, осуществляющие передачу коротких беспроводных сообщений с уникальной идентификацией. Для определения текущего положения с использованием BLE 4.0 используется метод трилатерации. Для применения метода этого метода необходимо не менее двух известных опорных объектов с известными координатами. Трилатерация базируется на измеренном расстоянии между каждым из опорных объектов, которыми являются Bluetooth-маяки, и устройством, чье местоположение требуется определить. Это обеспечивает точное и однозначное определение относительного местоположения объекта на плоскости. Также преимуществами использования Bluetooth-маяков являются компактные размеры устройства и длительное функционирование от одной батарейки в течение многих лет.

В работе [13] авторы предлагают одно из возможных вариантов определения местоположения пользователя – использование визуальных меток с машинным обучением (Simultaneous Localization and Mapping). SLAM основывается на размещении определенных визуальных меток на полу или других поверхностях в помещении и последующем анализе этих меток с помощью алгоритмов машинного

обучения для определения местоположения и направления движения пользователя. Основными преимуществами данного подхода являются возможность обеспечения точного определения местоположения в реальном времени и относительная независимость от ограничений, связанных с сигналами, характерными для других технологий. Однако он также сталкивается с вызовами, такими как необходимость обучения алгоритмов на больших объемах данных и чувствительность к изменениям в окружающей среде.

Исследования и разработки в области технологий определения местоположения внутри зданий неразрывно связаны с задачей оптимизации маршрутов внутри этих помещений. Определение местоположения играет важную роль в создании систем навигации, а алгоритмы поиска оптимального маршрута используют информацию о местоположении для эффективного направления пользователей по зданию.

В данном контексте широко применяются графы как структуры данных для представления пространства внутри. Графы представляют собой совокупность вершин и рёбер, где вершины могут представлять собой ключевые точки внутри помещения, а рёбра – возможные пути между ними [14]. Алгоритмы Дейкстры, Беллмана-Форда и Флойда-Уоршелла, A* (A-star), «прыжковых точек» (jump point search) применяются для решения проблемы поиска кратчайших путей в графах. Кратчайший путь между двумя вершинами в графе представляет собой минимальную последовательность рёбер, соединяющую эти вершины, с наименьшей суммарной стоимостью или длиной.

Алгоритм Дейкстры [15] представляет собой эффективный метод нахождения кратчайших путей во взвешенных графах, где каждому ребру сопоставлен неотрицательный вес. Он является основой классических алгоритмов планирования маршрутов и широко применяется в областях, таких как транспортные системы и сетевое проектирование.

Принцип работы алгоритма Дейкстры базируется на поддержании массива временных расстояний D[u] для каждого узла графа. Начиная с исходного узла s, алгоритм посещает узлы графа в порядке их расстояния от исходного узла, при

этом поддерживая инвариант D[u] = d(s, u) для посещенных узлов. Здесь d(s, u) представляет собой длину кратчайшего пути от исходного узла до узла u.

Когда узел u посещается, алгоритм расслабляет его исходящие ребра. Это означает, что для каждого смежного узла v проверяется, можно ли уменьшить временное расстояние D[v], обновив его как минимум из текущего D[v] и суммы D[u] и веса ребра w(u, v). Этот процесс продолжается, поочередно расслабляя рёбра на каждом этапе, пока все узлы не будут посещены. Алгоритм завершается, когда целевой узел достигнут.

Важным свойством алгоритма Дейкстры является то, что он работает корректно только в графах с неотрицательными весами рёбер. При этом размер поискового пространства, выраженный количеством посещенных узлов, оценивается как O(n), где n - количество узлов в графе, и в среднем n/2 узлов.

Алгоритм Беллмана-Форда [16] является еще одним методом решения задачи поиска кратчайших путей во взвешенном графе. В отличие от алгоритма Дейкстры, алгоритм Беллмана-Форда способен обрабатывать графы с рёбрами отрицательного веса, но при этом он требует больше вычислительных ресурсов.

Принцип работы алгоритма Беллмана-Форда заключается в итеративном обновлении оценок кратчайших путей от исходной вершины ко всем остальным. Алгоритм проводит $|V| - 1$ итераций, где $|V| -$ количество вершин в графе. На каждой итерации алгоритм рассматривает все рёбра графа и пытается улучшить оценку расстояния до каждой вершины. Если на i-й итерации удалось улучшить расстояние до какой-то вершины, то алгоритм считает, что путь длины i найден.

Отличительной особенностью алгоритма Беллмана-Форда является его способность обнаруживать и обрабатывать циклы отрицательного веса в графе. Если после $|V| - 1$ итераций алгоритма произойдёт ещё одно улучшение, то это свидетельствует о наличии цикла отрицательного веса в графе, и алгоритм сможет соответствующим образом обработать эту ситуацию. Это позволяет расширить его область применения, но при этом увеличивается вычислительная сложность до $O(|V| \cdot |E|)$, где E – множество ребер графа.

Алгоритм Флойда-Уоршелла [17] представляет собой метод для поиска всех кратчайших путей между всеми парами вершин в графе. Этот алгоритм обладает полнотой и универсальностью, что позволяет ему эффективно обрабатывать графы с произвольными весами на рёбрах, включая как положительные, так и отрицательные значения. Принцип работы алгоритма Флойда-Уоршелла основан на итеративном обновлении матрицы кратчайших путей. На каждой итерации алгоритм рассматривает все пары вершин и определяет, можно ли добраться из одной вершины в другую через некоторую третью вершину, что приведет к более короткому пути. Если такой путь существует, то матрица кратчайших путей обновляется соответствующим образом.

Следует отметить, что алгоритм Флойда-Уоршелла имеет более высокую вычислительную сложность по сравнению с алгоритмами, такими как Дейкстры или Беллмана-Форда [18]. В частности, его временная сложность составляет $O(V^3)$, где V - количество вершин в графе. Это делает алгоритм менее эффективным для больших графов, но при этом он остается универсальным инструментом для решения задач нахождения кратчайших путей между всеми парами вершин.

Рассмотрев три предыдущих обзора алгоритмов поиска оптимального пути, обратим внимание на алгоритм A* [18]. Он представляет собой модификацию алгоритма Дейкстры, оптимизированной для единственной конечной точки.

Принцип работы алгоритма A* основан на использовании эвристической функции, которая оценивает стоимость достижения целевой вершины из текущей. Порядок обхода определяется эвристической функцией $f(x) = g(x) + h(x)$, где $g(x) -$ стоимость пути от начальной вершины до текущей вершины x, $h(x) -$ эвристическая оценка стоимости пути от текущей вершины x до цели. Алгоритм выбирает для исследования вершину с наименьшей оцененной стоимостью. Это позволяет A* эффективно искать оптимальный путь, направляя внимание к целевой вершине.

A* обеспечивает полноту, что подразумевает обязательное нахождение решения при его наличии. Кроме того, при определенной эвристике h, A*

демонстрирует оптимальную эффективность, что означает, что другие алгоритмы исследуют не менее узлов, чем A*.

Алгоритм «прыжковых точек» (Jump Point Search, JPS) [20] представляет собой метод поиска пути в графе, который используется для ускорения алгоритмов поиска кратчайших путей, таких как A* или Дейкстры, в двумерных сетках. Этот алгоритм является оптимизированной версией A* для работы с сетками и может значительно ускорить процесс поиска, особенно в случае больших и сложных карт.

Принцип работы JPS базируется на идее предварительного определения точек на сетке, называемых "прыжковыми точками" (jump points), которые можно пропустить при поиске, так как они гарантированно не влияют на кратчайший путь. Алгоритм JPS делает выборочные прыжки по сетке, пропуская часть путей, что позволяет ему существенно уменьшить количество рассматриваемых узлов.

Основные шаги алгоритма «прыжковых точек»:

1) исследование направлений: при обнаружении препятствия алгоритм исследует направления, в которых путь может быть ускорено пройден без изменения длины;

2) прыжки: поиск прыжковых точек осуществляется в направлении, где обнаружено препятствие. Эти точки могут быть пропущены при дальнейшем поиске;

3) рекурсивный вызов: если обнаружена прыжковая точка, алгоритм рекурсивно вызывается для этой точки. Процесс повторяется, пока не будет достигнута конечная точка.

Основное преимущество JPS заключается в его способности значительно ускорять процесс поиска пути в сетке, особенно на больших картах с большим количеством препятствий. Однако, его применение ограничивается двумерными сетками, и в некоторых случаях, например, при наличии сложных трехмерных препятствий, другие алгоритмы поиска маршрута могут быть более предпочтительными.

Рассмотрим различные мобильные приложения, предназначенные для обеспечения внутренней навигации. В числе таких приложений выделяются,

например, «Yandex Maps», «Google Maps», «IndoorAtlas Positioning», «Indoar, PolyNavi», «РГГУ Навигатор», «ArcGIS Indoors», «Навигация Политеха AR», «Indoors Navigation» каждое из которых обладает уникальными особенностями и технологическими подходами. Рассмотрим каждое приложение подробнее.

Приложение «Yandex Maps» включает в себя функциональность отображения планов зданий, предоставляя пользователям подробные схемы зданий, охватывающие этажи, коридоры, помещения, и другие инфраструктурные элементы. Помимо основных структурных элементов, приложение предоставляет информацию о различных объектах интереса внутри здания, таких как магазины, рестораны, и другие удобства [21, 22]. Пользователи могут планировать маршруты внутри здания, следуя указаниям Yandex Maps. Однако, уровень детализации и доступность этой функции может различаться в зависимости от конкретного объекта. Также недостатками приложения являются неточное определение местоположения пользователя в здании, отсутствие возможности отображения маршрута с использованием дополненной реальности.

«Google Maps» также, как и «Yandex Maps», предоставляет планы внутренних пространств для некоторых мест, таких как аэропорты, торговые центры и другие крупные объекты [23]. Эти планы помогают пользователям ориентироваться внутри сооружений. В некоторых случаях Google Maps позволяет планировать маршруты внутри зданий, что обеспечивает пользователям возможность навигации от точки А до точки Б внутри объекта. В отличие от «Yandex Maps», в ряде случаев Google Maps предоставляет возможность использовать дополненную реальность для более наглядной навигации в режиме реального времени, где информация о маршруте отображается на изображении камеры устройства. Недостатком приложения является использование GPS, что не позволяет точно определять местоположение пользователя.

«IndoorAtlas Positioning» – это технологическое решение в области indoor навигации, разработанное компанией IndoorAtlas. Эта технология ориентирована на предоставление точного и эффективного определения местоположения внутри определенного зданий, где традиционные системы GPS могут быть менее

эффективными [24]. Решение ставит своей целью предоставление высокой точности определения местоположения внутри здания при минимальном потреблении энергии устройства. «IndoorAtlas» использует встроенные магнитометры и другие сенсоры в мобильных устройствах для сбора данных о магнитных полях внутри зданий. Эти данные затем обрабатываются с использованием алгоритмов машинного обучения для определения местоположения пользователя. Также Технология поддерживает использование AR для более наглядной и интуитивной навигации внутри зданий.

Несмотря на многочисленные преимущества «IndoorAtlas Positioning» имеет свои недостатки. Например, явным недостатком является внедрение и настройка системы «IndoorAtlas», так как требуется значительные усилия и ресурсы со стороны предприятия или организации, особенно при работе с большими и сложными зданиями. Также недостатком можно считать использование SLAM для определения местоположения пользователя в связи с тем, что структурные изменения внутри здания, такие как перемещение мебели или установка нового оборудования, могут повлиять на качество и точность определения местоположения.

«Indoar» – коммерческий продукт, разрабатываемый компанией ViewAR [25]. Также, как и ранее рассмотренное решение, «Indoar» имеет преимущества в виде точного определения местоположения, отображение маршрута с помощью дополненной реальности. Особенностью решения ViewAR является функциональная возможность размещать интерактивные метки с дополнительной информации о различных объектах или местах в здании, которые могут быть размещены на виртуальной карте. Однако, недостатком данного решения является дорогостоящее оборудование необходимое для создания цифрового двойника здания, с помощью которого определяется местоположение устройства.

PolyNavi представляет собой приложение для навигации внутри университета, ориентированное на помощь пользователям в перемещении по зданиям и на территории университетского кампуса [26]. Несмотря на отсутствие некоторых функций, таких как отслеживание местоположения пользователя в

реальном времени и поддержка AR для отображения маршрута, приложение имеет ряд полезных функций. Примером таких функций является возможность поиска объектов и мест в университетском кампусе, планирование маршрута и его отображение на двумерной карте.

«РГГУ Навигатор» представляет собой мобильное приложение, специально разработанное для упрощения внутренней навигации в пределах Российского государственного гуманитарного университета (РГГУ). В своей функциональности «РГГУ Навигатор» схож с «PolyNavi», предоставляя базовые средства для внутренней навигации и поиск по аудиториям, преподавателям и мероприятиям [27]. Также как и «PolyNavi», данное приложение не осуществляет отслеживание местоположения пользователя в реальном времени внутри зданий.

Особенностью «РГГУ Навигатора» является режим для инвалидов, который учитывает потребности пользователей с ограниченными физическими возможностями и позволяет прокладывать маршрут, учитывая это. Приложение ставит акцент на обеспечении доступности пространства и создании комфортных условий для всех категорий пользователей.

Аналогичным решением, как «PolyNavi» и «РГГУ Навигатор», является мобильное приложение для навигации по Московскому политехническому университету «Навигация Политеха AR» [28]. Приложение предоставляет схожий функционал, как у ранее рассмотренных аналогов, в виде поиска аудитории, построение маршрута и его отображение на трехмерной карте. Однако, отличительными чертой приложения является возможность отображения маршрута с помощью дополненной реальности. Также приложение позволяет определить свое местоположение, используя QR-коды, расположенные возле аудиторий. Существенный недостаток «Навигация Политеха AR» заключается в его прекращении разработки, следовательно, отсутствие возможности использования приложения на современных устройствах.

После обзора различных приложений для внутренней навигации, можно прийти к выводу, что каждое из них обладает уникальными характеристиками и функциональными аспектами. Важным фактором является наличие различий в

способности отслеживания местоположения, поддержке технологии дополненной реальности, а также возможностям построения маршрутов и способом его отображения.

Для более систематизированного и наглядного сравнения каждого из рассмотренных приложений была сформирована таблица 1.

Таблица 1 – Сравнение аналогов

Мобильное приложение	Формат представления карты	Построение маршрута между точками внутри здания	Технология определения местоположения	Возможность отображения маршрута в AR
Yandex Maps	2D	-	GPS	–
Google Maps	2D	+/-	GPS	+
IndoorAtlas Positioning	2D	+	BLE маяки, SLAM	+
Indoar	–	+	SLAM	+
PolyNavi	2D	+	–	–
РГГУ Навигатор	2.5D	+	–	–
Навигация Политеха AR	3D	+	QR-код	+

Вывод по разделу один

В ходе проведенного обзора выявлена актуальность и важность темы внутренней навигации, особенно в контексте распространения технологий дополненной реальности. Современные тенденции строительства больших зданий подчеркивают значимость улучшения опыта внутреннего перемещения, и в этом контексте AR выступает как инновационный инструмент для обогащения восприятия окружающей среды внутри помещений.

Применение AR в различных сферах открывает новые перспективы для оптимизации процессов и повышения вовлеченности пользователя. Внутренняя навигация становится неотъемлемым компонентом, обеспечивая точность и комфорт в перемещении внутри зданий.

Обзор алгоритмов поиска кратчайшего пути подчеркнул их значимую роль в обеспечении эффективности навигационных систем. Разнообразие алгоритмов и их

применимость к различным сценариям использования выделяет важные аспекты для дальнейших исследований.

Обзор приложений-аналогов выявил разнообразие функциональных возможностей и показал необходимость гибкости и адаптивности к потребностям пользователей. Однако, выявленные ограничения, такие как отсутствие отслеживания местоположения в реальном времени и ограниченная поддержка AR, предоставляют перспективы для будущих усовершенствований в разработке подобных приложений.

2. ФОРМУЛИРОВАНИЕ ТРЕБОВАНИЙ, ВЫБОР СРЕДЫ И СРЕДСТВ РАЗРАБОТКИ

2.1. Формулирование требований

На основе обзора аналогичных приложений и анализа их функциональных возможностей, сформулируем функциональные и нефункциональные требования разрабатываемого приложения.

2.1.1. Функциональные требования

Опишем функциональные требования к разрабатываемому приложению:

1) определение местоположения пользователя через распознавание текста на двери аудитории;
2) формирование истории прошлых маршрутов пользователя;
3) организация построения кратчайшего маршрута;
4) отображение маршрута с использованием технологии дополненной реальности;
5) реализация ролей активного пользователя, администратора.

Для роли взаимодействия типа Пользователь определим следующие функциональные требования:

1) пользователь должен иметь возможность определения своего текущего местоположения внутри здания путем распознавания текста на двери аудитории с помощью камеры устройства;
2) пользователь должен иметь возможность просматривать список своих предыдущих перемещений, включая начальную и конечную точки и даты использования;
3) пользователь должен иметь возможность указать начальную и конечную точки внутри университета для построения оптимального маршрута между ними, учитывая доступные пути;
4) пользователь должен иметь возможность просматривать свой маршрут на карте с использованием технологии дополненной реальности.

Для краткой иллюстрации функциональных требований к системе со стороны пользователя представим взаимодействия в нотации UML на рисунке 1.

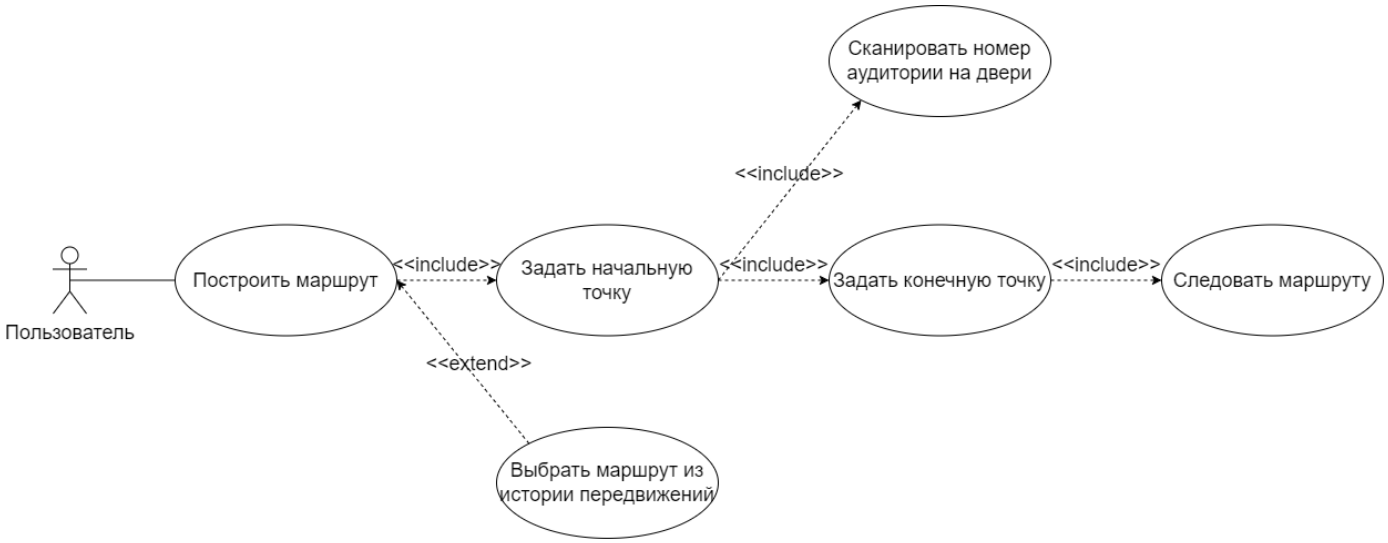

Рисунок 1 – Диаграмма взаимодействия пользователя в рамках приложения

Для роли взаимодействия типа Администратор определим следующие функциональные требования:

1) администратор должен иметь возможность определения своего текущего местоположения внутри здания путем распознавания текста на двери аудитории с помощью камеры устройства;

2) администратор должен иметь возможность добавлять узлы на карту с использованием камеры устройства и технологии дополненной реальности;

3) администратор должен иметь возможность удалять узлы с карты с использованием камеры устройства и технологии дополненной реальности;

4) администратор должен иметь возможность соединять узлы между собой;

5) администратор должен иметь возможность просматривать маршрут на карте с использованием технологии дополненной реальности во время его добавления.

Для краткой иллюстрации функциональных требований к системе со стороны администратора представим взаимодействия в нотации UML на рисунке 2.

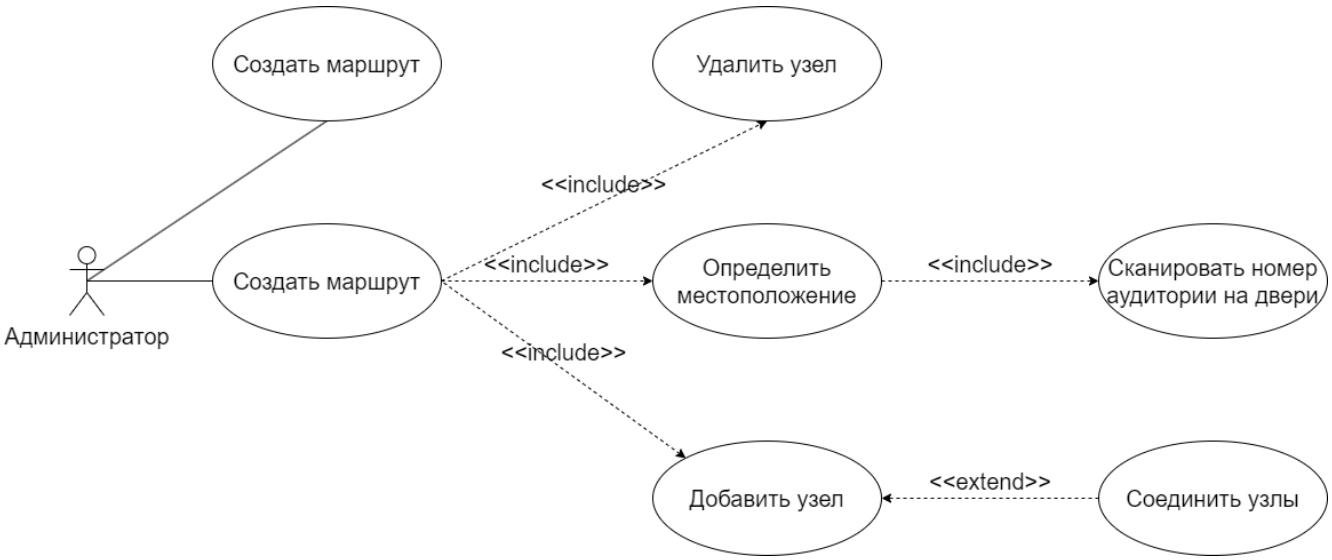

Рисунок 2- Диаграмма взаимодействия администратора в рамках приложения

2.1.2. Нефункциональные требования

Разрабатываемое приложение соответствовать следующим нефункциональным требованиям:

1) время, необходимое для обработки изображения с помощью камеры устройства и распознавания текста, не более 3 с;

2) время, затрачиваемое на построение и отображение маршрута, не более 5 с;

3) адаптивный пользовательский интерфейс.

2.2. Выбор среды и средств разработки

2.2.1. Тип мобильного приложения

Существует несколько типов мобильных приложений, отличающихся по методам разработки, платформам, для которых они предназначены, и способам взаимодействия с пользователем. Основные типы включают в себя нативные приложения, кроссплатформенные приложения и веб-приложения [29].

Нативные приложения разрабатываются специально для конкретной операционной системы, такой как iOS или Android. Они написаны на языках программирования, поддерживаемых соответствующей платформой, и имеют

прямой доступ к функциям и возможностям устройства, что обеспечивает высокую производительность и оптимизацию [30].

Кроссплатформенные приложения, напротив, разрабатываются с использованием единой кодовой базы и могут быть запущены на различных платформах. Они предлагают удобство одновременной разработки для нескольких платформ, но могут иметь ограничения в производительности и доступе к функциям устройства [29, 30].

В отличие от других типов, веб-приложения создаются с использованием веб-технологий, таких как HTML, CSS и JavaScript. Одной из ключевых характеристик такого вида мобильных приложений является возможность доступа к ним через обычные веб-браузеры, что обеспечивает простой доступ к приложениям на различных платформах и устройствах без установки дополнительного программного обеспечения. Однако, следует отметить, что у веб-приложений обычно ограничен доступ к функциям устройства по сравнению с нативными приложениями. Например, доступ к некоторым аппаратным ресурсам, таким как камера, геолокация или датчики, может быть ограничен, что может сказаться на их функциональности в некоторых случаях [31].

В связи с необходимостью использования камеры устройства для реализации функциональности приложения, было принято решение разрабатывать нативное мобильное приложение. Помимо этого, выбор нативного типа приложения обусловлен не только необходимостью использования камеры устройства, но и стремлением к максимальной эффективности, производительности и качеству работы приложения в целом.

2.2.2. Операционная система для разрабатываемого мобильного приложения

Операционные системы Android и iOS являются двумя основными платформами для мобильных устройств в современном мире. Рассмотрим подробно каждую.

Android — это операционная система для мобильных устройств, разработанная компанией Google на основе ядра Linux. Она была впервые

представлена в 2008 году и с тех пор стала одной из наиболее широко распространенных и популярных платформ для смартфонов, планшетов, смарт-телевизоров и других подобных устройств. Одним из фундаментальных принципов Android является его открытость. Это позволяет разработчикам и производителям устройств свободно адаптировать и модифицировать платформу под свои нужды. Этот подход способствует большому разнообразию устройств, работающих на Android, и стимулирует инновации в индустрии мобильных технологий [32].

iOS — это операционная система, разработанная и поддерживаемая компанией Apple, специально для своих мобильных устройств. iOS интегрируется тесно с другими устройствами и сервисами компании-разработчика, что обеспечивает возможность взаимодействовать между собой. В отличие от Android, iOS является закрытой операционной системой, тем самым обеспечивая высокий уровень безопасности и конфиденциальности данных [33].

Рассматривая мобильные операционные системы, нельзя не обратить внимание на их распространение среди пользователей. На рисунке 3 изображен график, отражающий занимаемую долю рынка для каждой операционной системы.

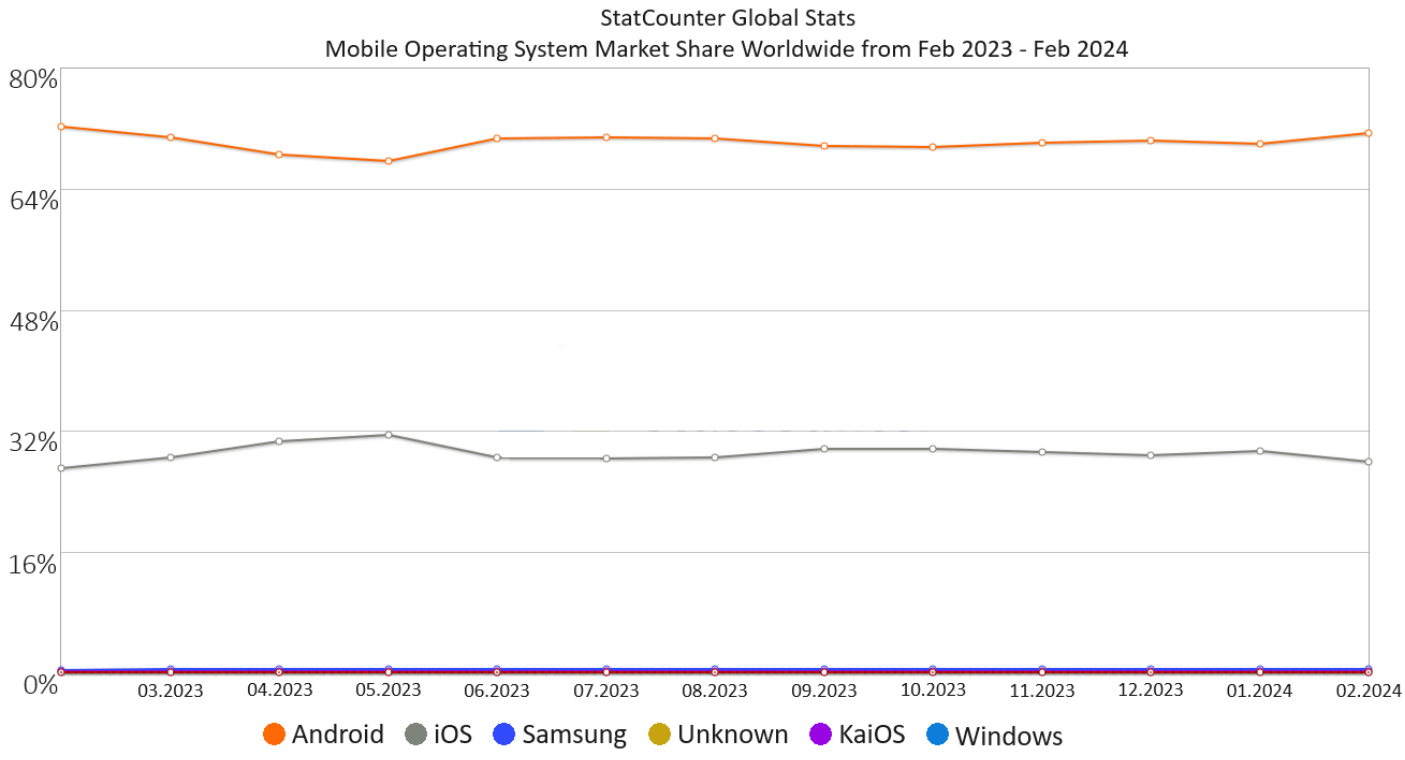

Рисунок 3 – Статистика мобильных операционных систем за год [34]

Таким образом, основываясь на обзоре мобильных операционных систем и данных статистики, принято решение ориентировать разработку мобильного приложения под платформу Android.

2.2.3. Язык программирования

Основываясь на принятом типе и платформе приложения, необходимо подробно рассмотреть доступные варианты языков программирования, учитывая их соответствие требованиям платформы и целям разработки.

Java — один из наиболее распространенных языков программирования для создания мобильных приложений на платформе Android. Он основан на принципах объектно-ориентированного программирования, что делает его удобным и гибким инструментом для разработки сложных приложений. Такой подход позволяет строить программы из взаимосвязанных объектов, что способствует повышению эффективности и повторному использованию кода. Одним из основных преимуществ Java является ее платформонезависимость. Это означает, что приложения, написанные на этом языке, могут быть запущены на различных операционных системах без необходимости внесения изменений в их исходный код. Это особенно важно для мобильной разработки, поскольку приложения могут без проблем работать на различных устройствах под управлением Android, не требуя специальной адаптации для каждой конкретной платформы. Кроме того, Java обладает встроенными механизмами безопасности, такими как проверка типов, обработка исключений и сборка мусора [35]. Это помогает предотвратить множество типичных ошибок программирования и обеспечить стабильную работу приложений.

Несмотря на множество преимуществ, у Java есть определенные недостатки, такие как ограниченная поддержка некоторых современных технологий, медленное развитие по сравнению с другими языками программирования и избыточность кода [36].

Kotlin представляет собой статически типизированный язык программирования, разработанный компанией JetBrains, который получил

широкое признание в мобильной разработке, особенно под Android-платформу [37]. В сравнении с Java, Kotlin предлагает ряд улучшений и дополнительных функций, которые делают его привлекательным выбором для разработчиков мобильных приложений. Одним из основных преимуществ Kotlin является более компактный синтаксис, что позволяет писать код более эффективно и читаемо. Kotlin также предлагает ряд современных функций, таких как расширения функций, корутины и безопасность типов, которые способствуют ускорению разработки и предотвращают множество распространенных ошибок. Помимо этого, Kotlin является его 100% совместимость с Java. Это означает, что существующий Java-код может быть легко интегрирован в проект на Kotlin, и наоборот [38].

Несмотря на все преимущества, которые предлагает Kotlin, по сравнению с Java, он также имеет свои недостатки. Одним из них является отсутствие примитивных типов данных, что может негативно сказываться на производительности и использовании памяти. Кроме того, в Kotlin отсутствуют проверяемые исключения, что может привести к непредсказуемому поведению программы во время выполнения [39].

При выборе между Java и Kotlin для разработки мобильного приложения, предпочтение отдается современному и развивающемуся языку Kotlin. Этот выбор обусловлен значительными преимуществами Kotlin по сравнению с Java, при его незначительных недостатках.

2.2.4. Средство разработки пользовательского интерфейса

Основным средством разработки пользовательского интерфейса мобильных приложений являются XML-макеты. Эти макеты содержат описания размещения и внешнего вида компонентов пользовательского интерфейса, таких как кнопки, текстовые поля, изображения и другие элементы. Использование XML для определения макетов позволяет разработчикам создавать и обслуживать пользовательские интерфейсы, при этом поддерживая разделение логики приложения и его представления. Кроме того, XML-макеты обеспечивают

интерфейс, который может адаптироваться к разным размерам и ориентациям экранов, делая возможным создание приложений, подходящих для различных устройств и разрешений экранов [40].

Существенными недостатками использования XML-макетов являются увеличение объема кода с усложнением интерфейса приложения, что делает дальнейшую поддержку сложной, а также ограниченная гибкость при динамических изменениях интерфейса, требующая создания дополнительного громоздкого кода.

Альтернативным средством разработки пользовательского интерфейса в Android-приложениях является Jetpack Compose. Этот инновационный инструментарий направлен на создание нативных пользовательских интерфейсов с использованием более эффективных средств и API на языке Kotlin с целью упрощения и ускорения процесса разработки. Основными характеристиками Jetpack Compose являются декларативный подход к созданию интерфейса, который позволяет описывать желаемый результат без необходимости указывать способы достижения этого результата, а также возможность легкого повторного использования и комбинирования компонентов для формирования сложных интерфейсов [41].

Основными недостатками Jetpack Compose по сравнению с XML-макетами являются увеличение времени сборки проекта и увеличение размера итогового приложения [42]. Это связано с тем, что Jetpack Compose требует компиляции кода на Kotlin, который может быть более медленным процессом по сравнению с простым парсингом XML-файлов. Также, при использовании Compose могут внедряться дополнительные библиотеки, что может увеличить размер приложения.

Подводя итог, несмотря на некоторые недостатки, Jetpack Compose представляет собой современный и мощный инструмент для разработки пользовательского интерфейса в Android-приложениях. Использование Compose позволит воспользоваться всеми его преимуществами и сделать процесс разработки более эффективным.

2.2.5. Система управления базой данных

Использование базы данных в мобильных приложениях необходимо для обеспечения эффективного хранения и управления данными. База данных предоставляет структурированное хранилище для различных типов информации, включая историю и узлы маршрутов, другие данные, важные для функционирования приложения.

При разработке мобильного приложения, целью которого является навигация внутри университета, использование встроенной системы управления базой данных (СУБД) на устройстве представляется обоснованным с позиции оптимизации процесса доступа к информации и повышения эффективности функционирования приложения. Локальное хранение данных обеспечивает быстрый доступ к информации о местоположении объектов внутри университетского корпуса, что существенно сокращает время ответа приложения на запросы пользователя. Помимо этого, возможность работы даже при отсутствии сетевого подключения предоставляет пользователям непрерывный доступ к необходимой информации, что особенно ценно в условиях переменного качества сетевого покрытия. Дополнительно, хранение данных способствует экономии трафика данных, что в целом повышает уровень функциональности и удобства использования мобильного приложения.

SQLite является встроенной СУБД, широко применяемой в мобильной разработке, включая платформу Android. Она представляет собой компактную и легковесную реляционную базу данных, разработанную для работы с небольшими и средними объемами данных, что делает ее особенно подходящей для мобильных приложений с ограниченными ресурсами. SQLite хранит данные в одном файле базы данных, что упрощает управление и распространение баз данных в приложениях [43].

Одним из основных преимуществ SQLite является его простота использования и интеграции с Android-приложениями. Он предоставляет простой и понятный SQL-подобный язык запросов для работы с данными, что упрощает создание запросов и выполнение операций чтения и записи. Кроме того, SQLite

обеспечивает высокую производительность и эффективное использование ресурсов устройства, что особенно важно для мобильных приложений с ограниченными вычислительными мощностями и энергопотреблением.

Однако, несмотря на свои преимущества, у SQLite также есть некоторые ограничения и недостатки. Во-первых, из-за своей легковесности и простоты, SQLite не обладает всеми функциями и возможностями более мощных серверных баз данных, таких как MySQL или PostgreSQL. Это может быть недостатком для приложений, требующих сложных операций или работающих с большими объемами данных. Кроме того, SQLite не поддерживает одновременный доступ к базе данных несколькими процессами или потоками, что может привести к проблемам с многопоточностью в некоторых сценариях использования.

Кроме того, существует Firebase Realtime Database и Firestore [44], облачные базы данных от Google, которые предоставляют возможность хранения и синхронизации данных в режиме реального времени. Использование Firebase Realtime Database и Firestore обеспечивает удобную интеграцию с облачным хранилищем данных, поддерживая режим реального времени и оффлайн-синхронизацию для мобильных приложений.

Однако, хотя Firebase Realtime Database предоставляет мгновенное обновление данных и простоту в использовании, его ограниченная возможность выполнения сложных запросов и структура данных JSON могут усложнить процесс моделирования данных. С другой стороны, Firestore предлагает более широкий набор функций запросов, структурированные данные в формате коллекций и документов, а также более гибкую модель прав доступа, что делает его предпочтительным выбором для проектов с комплексными потребностями в управлении данными, однако это может повлиять на скорость и затраты на использование Firestore.

Еще одной альтернативой является ObjectBox [45], которая представляет собой быструю и легковесную NoSQL-базу данных для мобильных приложений, обладающую высокой производительностью и простотой использования благодаря своему оптимизированному API и объектно-ориентированному подходу к

хранению данных. Однако, несмотря на свои преимущества, ObjectBox все еще ограничен относительно небольшим сообществом разработчиков и имеет менее развитую экосистему по сравнению с более узнаваемыми СУБД, такими как SQLite, а также обладает ограниченными возможностями запросов и поддержкой платформ, что может сделать его менее подходящим для некоторых проектов.

Учитывая все преимущества и недостатки рассмотренных СУБД, Firebase Firestore оказывается оптимальным выбором для мобильного приложения на платформе Android. Преимущества использования этой СУБД делают её идеальным решением для обеспечения хранения и быстрого доступа к актуальным данным, что соответствует особенностям разрабатываемого приложения.

2.2.6. Среда разработки

Для разработки приложений под платформу Android на языке программирования Kotlin доступны несколько интегрированных сред разработки (IDE), предоставляющих разработчикам различные инструменты и средства для эффективной работы.

Одним из наиболее популярных вариантов является Android Studio [46], официальная IDE для разработки приложений под Android, созданная на базе IntelliJ IDEA. Android Studio обеспечивает широкий спектр инструментов, включая интегрированную среду разработки, визуальные редакторы интерфейсов, инструменты для отладки и профилирования, а также интеграцию с платформой Google Play для публикации приложений. Одним из главных преимуществ Android Studio является его официальная поддержка Android-разработки, а также активное сообщество пользователей и разработчиков, что обеспечивает быстрый доступ к обновлениям и поддержке.

Другой популярной IDE для разработки Android-приложений на Kotlin является IntelliJ IDEA [47], на базе которой создан Android Studio. IntelliJ IDEA предоставляет мощные инструменты для разработки приложений на различных языках программирования, включая Kotlin, Java и другие. Ее главными преимуществами являются богатая функциональность, расширенные возможности

автоматизации и интеграции, а также гибкая система настройки и поддержка различных плагинов. Однако, несмотря на свои мощные возможности, IntelliJ IDEA может быть сложной в освоении и требовать дополнительных затрат на лицензии для полноценного использования.

Также стоит упомянуть Eclipse [48], среду разработки с открытым исходным кодом, предоставляющей инструменты и широкие возможности для разработки программного обеспечения, включая мобильные приложения под Android. Ее преимущества включают в себя гибкость, расширяемость и богатый функционал, однако, высокий порог вхождения и конкуренция с более современными IDE, такими как Android Studio, снизили ее популярность в разработке Android-приложений.

С учетом анализа различных сред разработки для создания мобильных приложений, предпочтение отдается Android Studio. Это обосновано его официальной поддержкой, богатым функционалом и тесной интеграцией с экосистемой Google Play, что делает его наиболее подходящим выбором для разработки приложений под платформу Android.

Вывод по разделу два

В этом разделе были сформированы две основные категории требований: функциональные и нефункциональные. Кроме того, были отдельно подробно рассмотрены функциональные требования для ролей администратора и пользователя. Также были выбраны среда разработки – Android Studio, и средства разработки: операционная система - Android, язык программирования - Kotlin, средство разработки пользовательского интерфейса - Jetpack Compose, система управления базы данных – Firebase Firestore.

3. РАЗРАБОТКА ПРОГРАММНОГО КОДА МОБИЛЬНОГО ПРИЛОЖЕНИЯ

3.1. Разработка архитектуры мобильного приложения

Архитектура мобильного приложения — это структура и организация кода, компонентов и их взаимодействия внутри мобильного приложения. Она определяет, как различные части приложения будут взаимодействовать, обеспечивая масштабируемость, поддержку.

При построении архитектуры мобильного приложения важно придерживаться общепринятых принципов чистой архитектуры и шаблонов для обеспечения чистоты кода и структурирования его компонентов [49]. Согласно принципам чистой архитектуры код мобильного приложения разделяется на слои, каждый из которых имеет свою определенную ответственность. Схема выделяющихся слоев и взаимодействие между ними представлены на рисунке 4.

Рисунок 4 – Разделение кода мобильного приложения на слои [49]

Принципы чистой архитектуры и шаблон Model-View-ViewModel (MVVM) являются двумя подходами к проектированию приложений, которые могут быть интегрированы для создания хорошо структурированных и легко поддерживаемых систем. Чистая архитектура обеспечивает высокоуровневую организацию приложения, разделяя его на независимые слои, в то время как MVVM используется для управления пользовательским интерфейсом и логикой взаимодействия с пользователем. В рамках шаблона MVVM, Model представляет

данные и бизнес-логику приложения, View отвечает за отображение данных пользователю и прием ввода от пользователя, а ViewModel выступает посредником между View и Model, предоставляя данные View и обрабатывая взаимодействия пользователя с View. На рисунке 5 представлена схема шаблона MVVM.

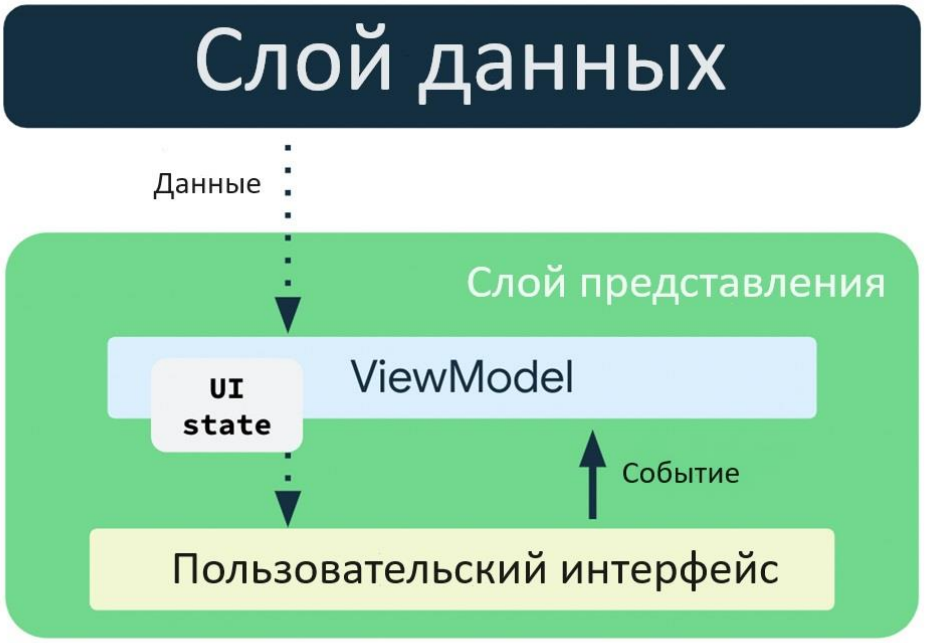

Рисунок 5 – Схема шаблона MVVM [49]

Основываясь на вышеупомянутых принципах, была спроектирована архитектура мобильного приложения. Диаграмма, изображенная на рисунке 6, отражает структурное разделение приложения на три основных слоя: слой данных, слой бизнес-логики и слой представления. Каждый слой имеет четко определенные компоненты и интерфейсы, обеспечивающие взаимодействие между слоями.

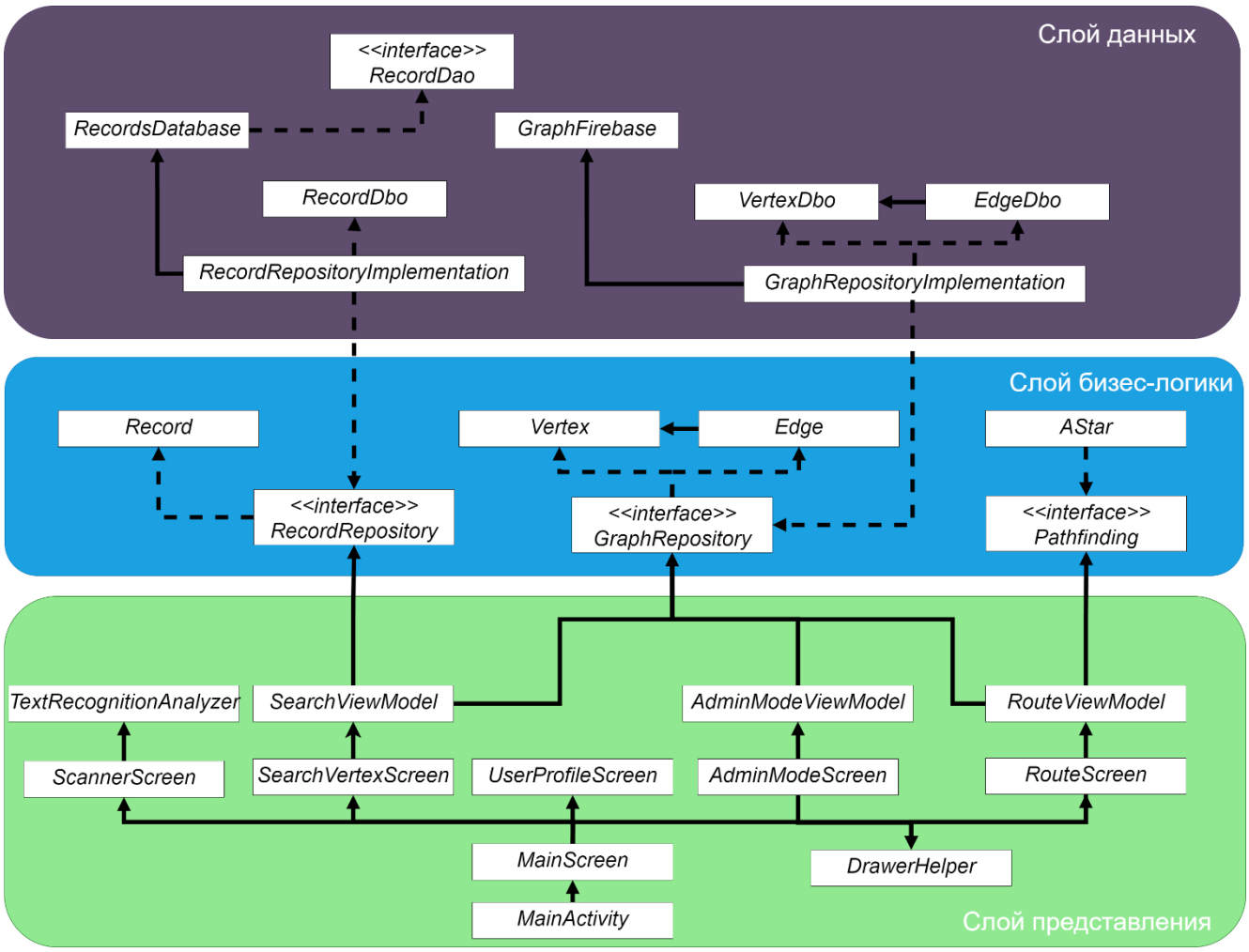

Рисунок 6 – Архитектура мобильного приложения «Интерактивная карта ЮУрГУ»

Слой данных отвечает за управление и хранение данных, включая их получение, сохранение и преобразование. Взаимодействие классов этого слоя отображены на рисунке 7.

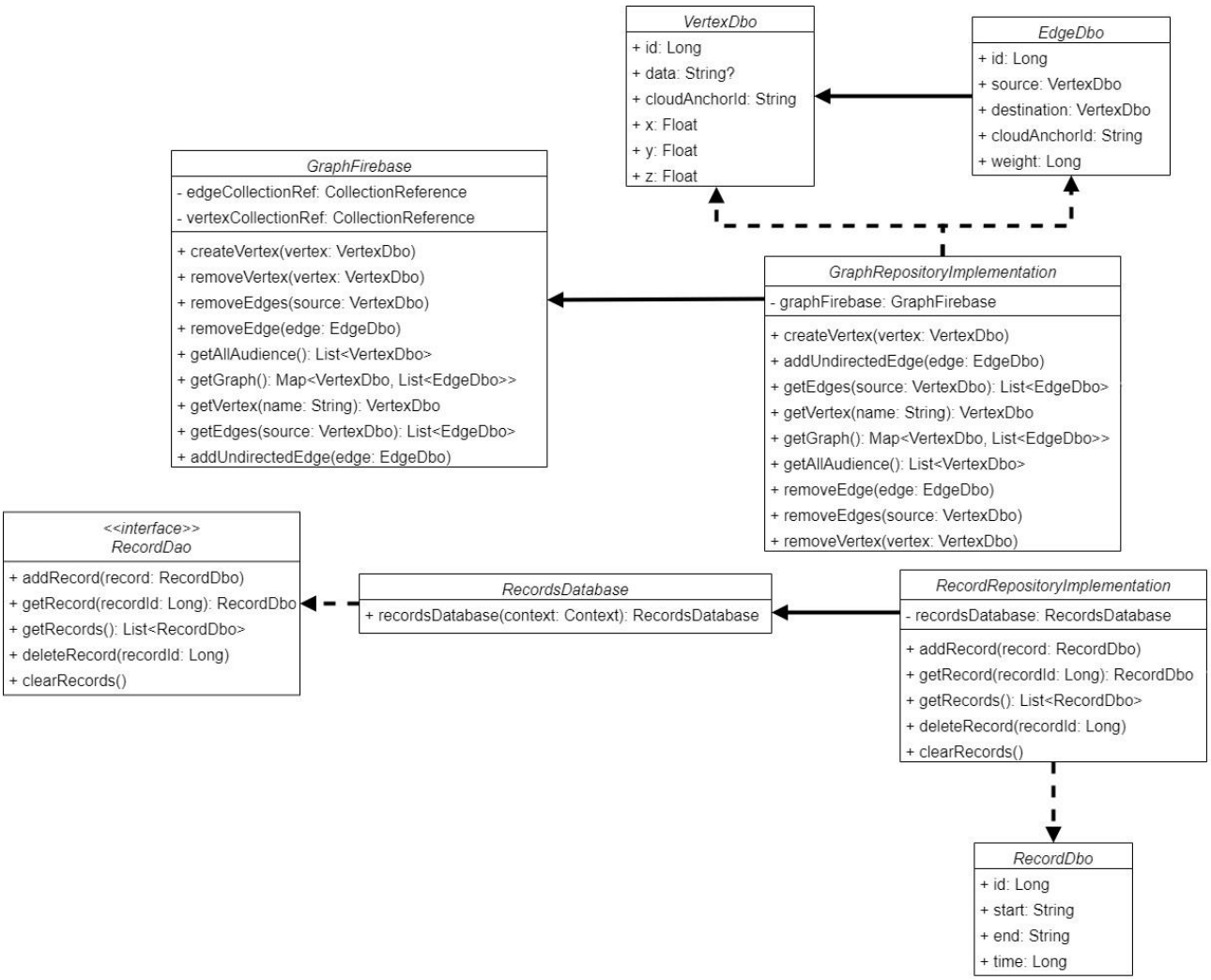

Рисунок 7 – Диаграмма классов слоя данных

На диаграмме классов слоя данных представлены следующие классы:

- TextRecognitionAnalyzer – класс, реализующий распознавание текста;
- RecordsDatabase – класс, представляющий базу данных для хранения записей, также предоставляет методы для взаимодействия с данными о записях;
- RecordDao – интерфейс, предоставляющий методы для работы с данными о записях (добавление, получение, удаление и очистка записей);
- RecordDbo – объект базы данных для записи данных;
- RecordRepositoryImplementation – реализация репозитория для управления записями;

• GraphFirebase – класс, взаимодействующий с Firebase для управления данными графа, предоставляет методы для создания, удаления и получения вершин и ребер графа;

• VertexDbo и EdgeDbo – объекты базы данных для вершин и ребер графа соответственно;

• GraphRepositoryImplementation – реализация репозитория для управления графом.

Слой бизнес-логики содержит основные бизнес-правила и алгоритмы, используемые в приложении. Диаграмма классов слоя бизнес-логики представлена на рисунке 8.

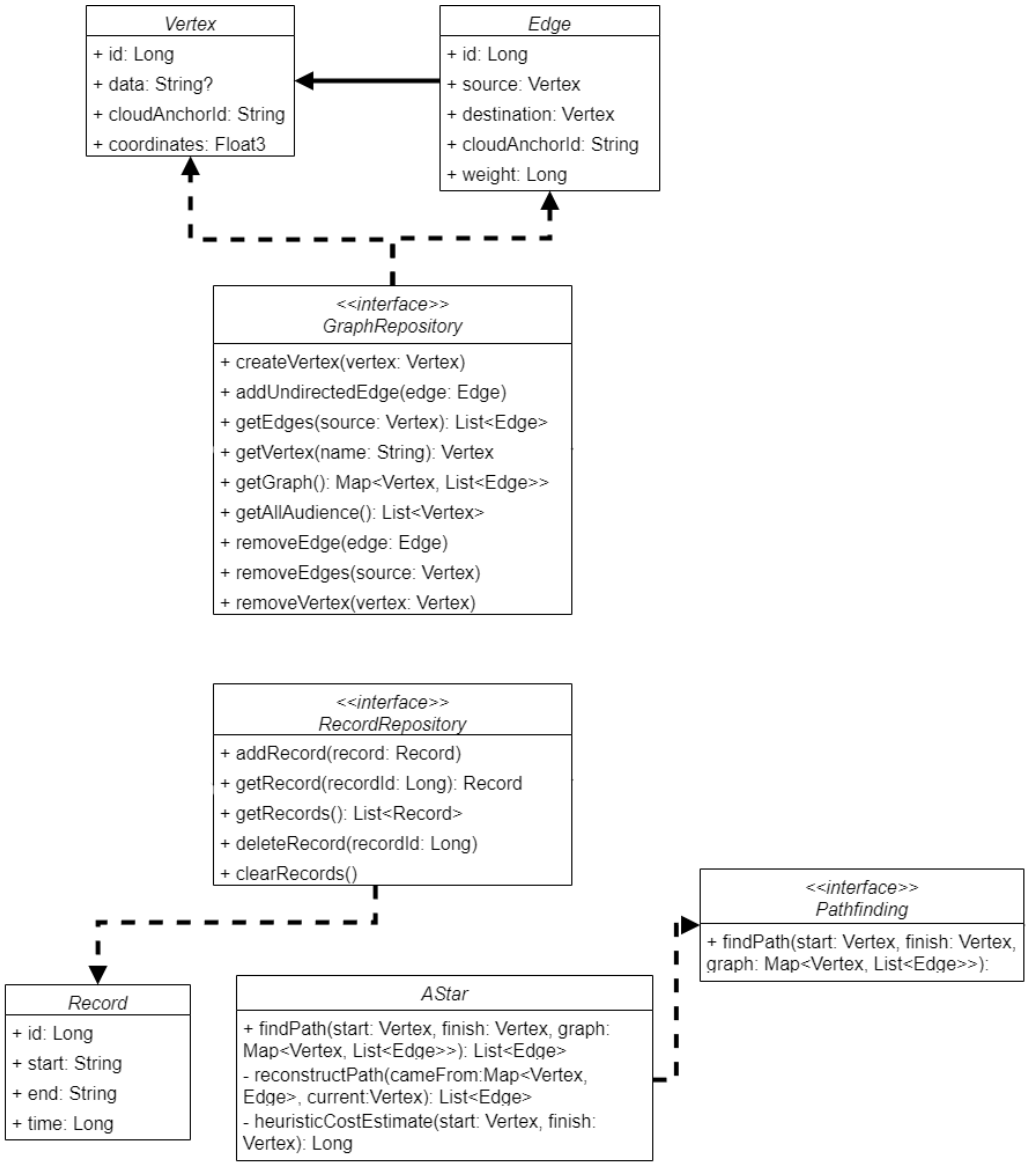

Рисунок 8 – Диаграмма классов слоя бизнес-логики

В слое бизнес-логики представлены:

- Record – класс, отображающий модель записи бизнес-логики,
- Vertex и Edge – классы, отображающие модели узла и ребра бизнес-логики,
- RecordRepository – интерфейс репозитория для управления записями,
- GraphRepository – интерфейс репозитория для управления графом,
- AStar – класс, реализующий алгоритм поиска пути A*,
- Pathfinding: интерфейс для поиска пути.

Слой представления отвечает за отображение данных пользователю и прием пользовательского ввода. Диаграмма классов этого слоя представлена на рисунке 9.

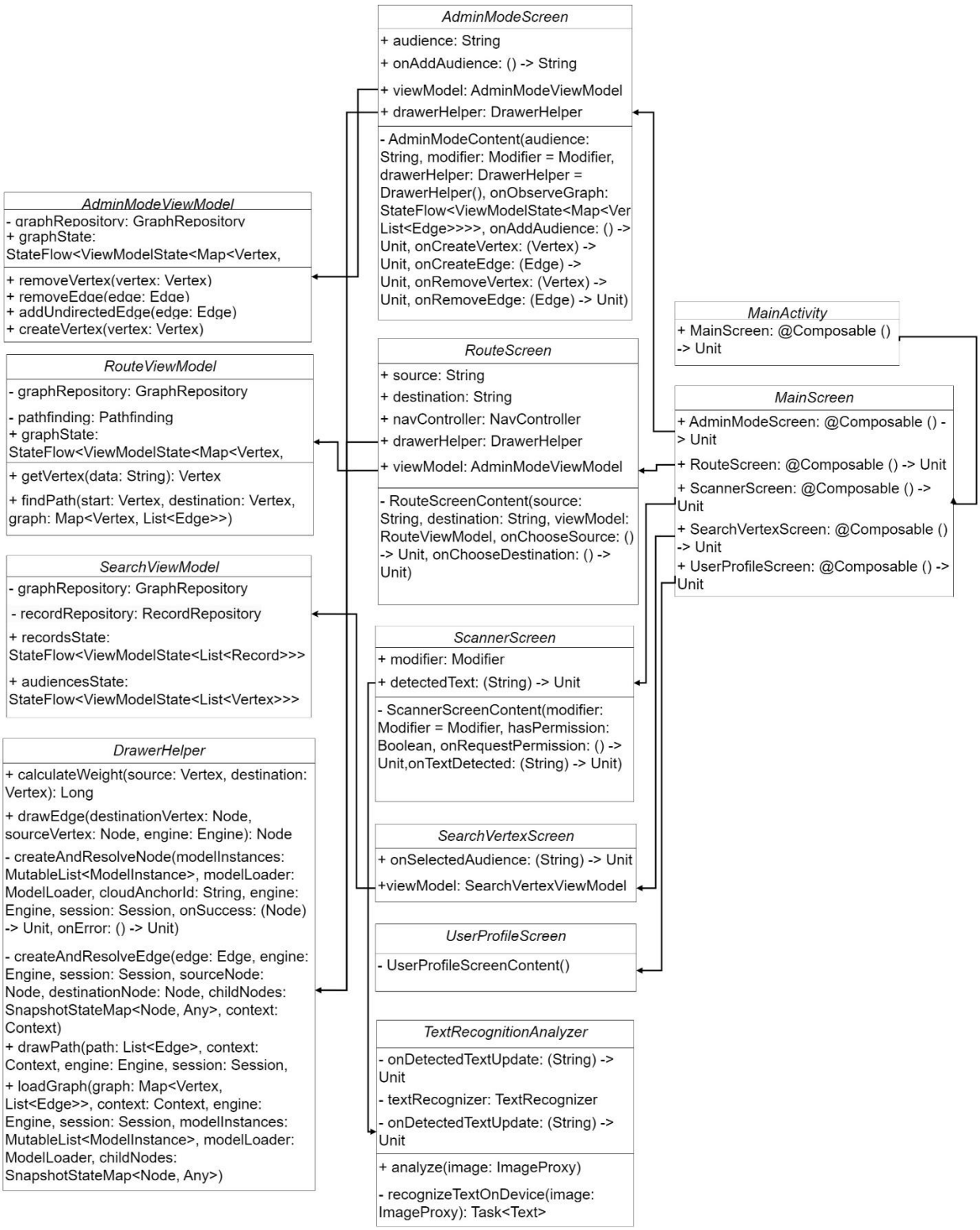

Рисунок 9 – Диаграмма классов слоя представления

Рассмотрим основные компоненты, представленные на диаграмме:

- ScannerScreen – экран для сканирования, отображает рамку, внутри которой распознается номер аудитории;
- SearchVertexScreen – экран для поиска аудиторий и записей;

• UserProfileScreen – экран профиля пользователя;

• MainScreen – главный экран, содержащий ссылки на другие экраны (режим администратора, маршрут, сканирование, поиск, профиль пользователя);

• MainActivity – основная активность, которая запускает MainScreen;

• AdminModeScreen – экран режима администратора, позволяющий управлять графом (добавление/удаление вершин и ребер);

• RouteScreen – экран для отображения маршрута;

• DrawerHelper – вспомогательный класс для работы с графом, включает методы для расчета веса, рисования ребер и загрузки графа;

• SearchViewModel – предоставляет список записей и аудиторий,

• AdminModeViewModel – управляет состоянием графа и операциями добавления, удаления вершин и ребер;

• RouteViewModel – обрабатывает запросы на поиск маршрутов, предоставляет данные о вершинах и путях.

Исходя из архитектуры и требований к мобильному приложению, была разработана концептуальная диаграмма потоков данных, отражающая взаимодействие компонентов приложения и передачу данных в нем. Диаграмма представлена на рисунке 10.

Рисунок 10 – Диаграмма потоков данных

3.2. Разработка алгоритма поиска оптимального маршрута

Алгоритм, изображенный на рисунке 11, реализует метод поиска пути с использованием алгоритма A* (AStar). Этот алгоритм является одним из наиболее эффективных методов поиска кратчайшего пути в графах, сочетающим в себе элементы жадного поиска и алгоритма Дейкстры. В основе алгоритма лежит идея использования эвристической функции для оценки стоимости оставшегося пути от текущей вершины до целевой вершины, что позволяет значительно уменьшить количество обрабатываемых вершин и ускорить поиск.

Инициализация алгоритма начинается с определения двух словарей, fScore и gScore, которые будут хранить оценки полной стоимости пути и стоимости кратчайшего пути до каждой вершины соответственно. Для каждой вершины эти значения инициализируются значением Long.MAX_VALUE (64 бита), за исключением начальной вершины, для которой gScore устанавливается в 0, а fScore — в эвристическую оценку расстояния до целевой вершины. Вершины, которые предстоит, обработать хранятся в очереди с приоритетом openSet, инициализируемой начальной вершиной. Словарь cameFrom используется для хранения информации о предшествующих вершинах, что необходимо для восстановления найденного пути.

Основной цикл алгоритма продолжается до тех пор, пока очередь openSet не станет пустой. На каждом шаге из очереди извлекается вершина с наименьшим значением fScore. Если текущая вершина совпадает с целевой, алгоритм завершает работу и возвращает найденный путь, восстановленный с помощью функции reconstructPath. В противном случае, для каждой соседней вершины вычисляется временная оценка стоимости пути tentativeGScore. Если эта оценка меньше известной стоимости пути к соседней вершине, обновляются данные в cameFrom, gScore и fScore. Если соседняя вершина еще не была добавлена в очередь, она добавляется.

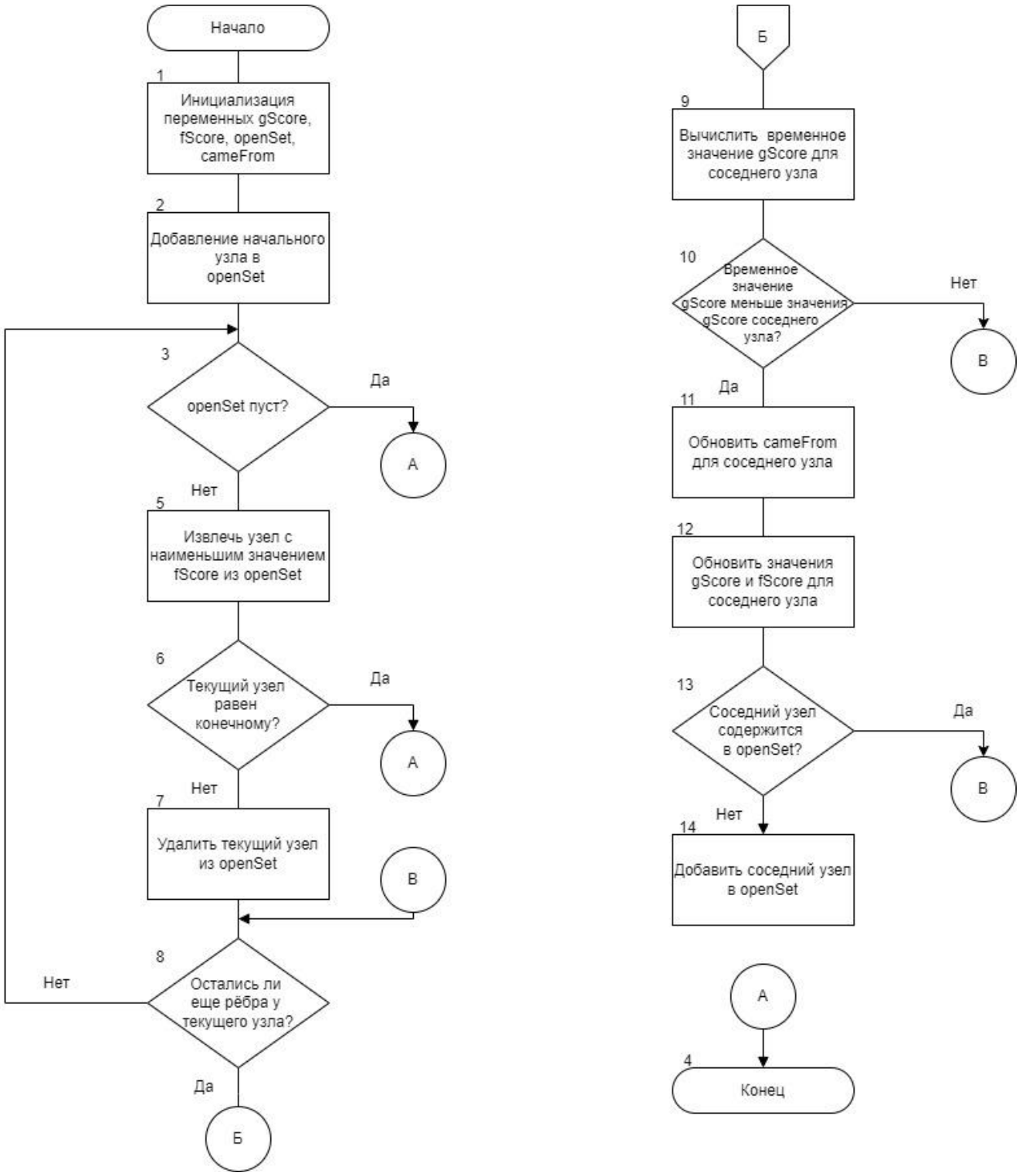

Рисунок 11 – Блок-схема алгоритма A*

Эффективность алгоритма A* достигается за счет использования эвристической функции, которая в данной реализации представлена манхэттенским расстоянием. Эта функция быстро оценивает оставшуюся стоимость пути на основе абсолютных разностей координат вершин в трехмерном пространстве. Таким образом, алгоритм A* обеспечивает баланс между жадным

поиском и полным перебором, что позволяет находить оптимальные пути в сложных графах с минимальными вычислительными затратами.

Исходный код алгоритма A* представлен в листинге А.1 приложения А.

3.3. Разработка пользовательского интерфейса

На рисунке 12 представлено диалоговое окно для входа в приложение с использованием учетной записи Google. Пользователю предлагается выбрать один из представленных аккаунтов для авторизации. Выбор учетной записи непосредственно влияет на доступные пользователю функции, предоставляя доступ к соответствующим ресурсам и возможностям в зависимости от роли, а также данные, отображаемые на экране профиля.

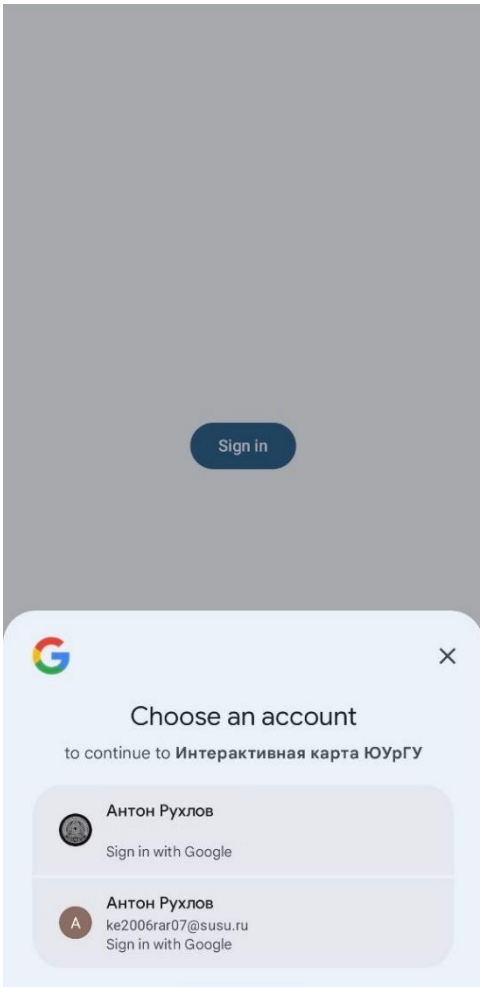

Рисунок 12 – Нижняя панель выбора аккаунта

На рисунке 13 представлено окно профиля пользователя в мобильном приложении.

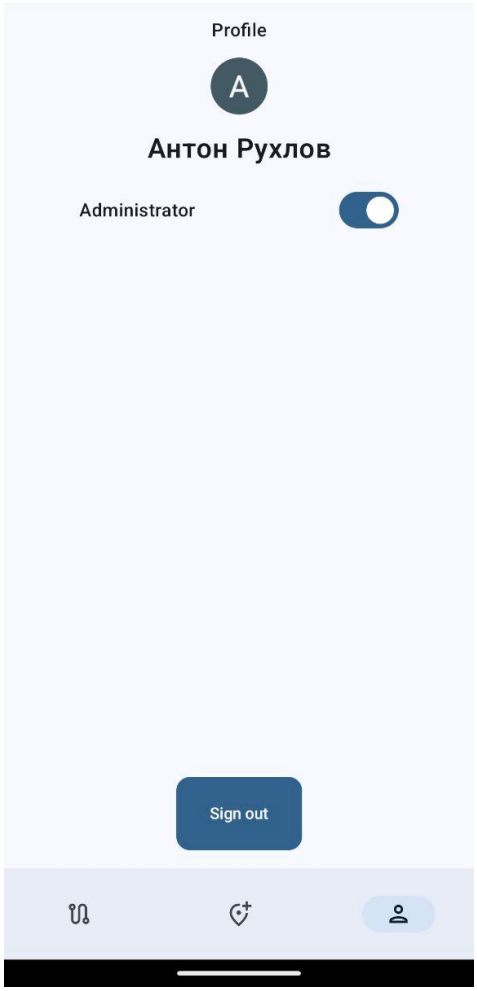

Рисунок 13 – Экран профиля пользователя

На экране отображается фотография и имя пользователя, а также его роль. Переключатель, находящийся в активированном состоянии (включен), указывает на то, что текущий пользователь обладает правами администратора и ему доступен экран с операциями над графом.

В нижней части экрана расположена кнопка «Sign out» («Выйти»), которая позволяет пользователю выйти из учетной записи.

Навигационная панель, расположенная в нижней части экрана, приложения состоит из трех вкладок, каждая из которых предназначена для доступа к различным разделам приложения. Первая вкладка представляет собой экран с выбором маршрута и отображением его в дополненной реальности. Вторая вкладка предоставляет доступ к экрану администратора, данный экран доступен только

пользователям с ролью администратора. Третья вкладка ведет к ранее представленному экрану профиля пользователя.

Экран администратора, изображенный на рисунке 14, представляет собой редактор графа с использованием дополненной реальности. На изображение, передаваемое с камеры мобильного устройства, накладывается компьютерная графика в виде точек (узлов) и прямых (ребер), соединяющих два узла между собой. Основные операции осуществляются жестами: одиночное нажатие на экран добавляет узел, двойное нажатие на узлы соединяет их ребром, долгое зажатие узла или ребра удаляет их.

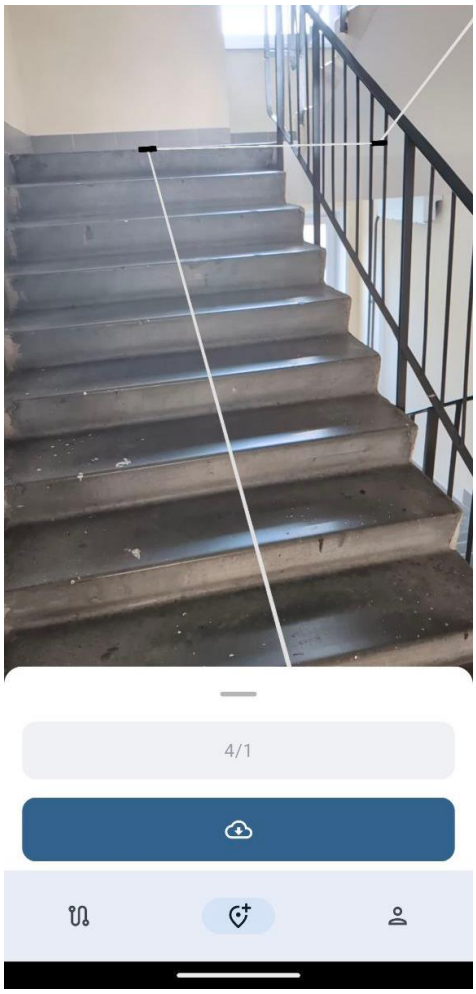

Рисунок 14 – Экран администратора

Помимо этого, экран администратора содержит диалоговое окно, которое можно свернуть, состоящее из поля, отображающего номер аудитории, которая будет добавлена в качестве узла, и кнопки. При нажатии на текстовое поле пользователь перенаправляется на экран со сканером, после возвращения на экран

администратора распознанный текст будет отображаться в текстовом поле. Нажатие на кнопку, находящуюся под текстовым полем, инициирует построение графа из данных, хранящихся в облачной базе данных.

Для запоминания местоположения узлов и ребер используется технология Cloud Anchor (облачный якорь) от ARCore. Для создания и размещения якоря ARCore использует 3D-карту пространства, окружающего этот якорь. Процесс получения такой карты включает создание карт окружения задней камерой устройства в центре интереса и вокруг него под разными углами обзора и с разных позиций до вызова операции запоминания. Затем API ARCore создает 3D-карту пространства и возвращает устройству уникальный идентификатор облачного якорь.

Распознавание ранее созданного узла происходит по следующему принципу: пользователь, находясь в том же окружении, наводит камеру своего устройства на область, где был размещен облачный якорь. Запрос на распознавание заставляет ARCore Cloud Anchor API периодически сравнивать визуальные особенности сцены с созданной 3D-картой характеристик. Это позволяет точно идентифицировать и восстановить ранее сохраненные узлы и ребра графа.

Исходный код экрана администратора представлен в листинге Б.1 приложения Б.

Экран, представленный на рисунке 15, демонстрирует функционал распознавания текста в мобильном приложении. На изображении видна табличка с номером «408», размещенная в центре экрана. На изображение, передаваемое с камеры мобильного устройства, наложена полупрозрачная серая рамка, определяющая зону распознавания текста. Эта зона служит для указания области, в пределах которой приложение будет производить сканирование и распознавание текста.

Рисунок 15 – Экран распознавания номера аудитории

В нижней части экрана размещена кнопка «Scan», предназначенная для запуска процесса сканирования. Пользователь направляет камеру на объект с текстом, который необходимо распознать, и нажимает кнопку «Scan». После успешного сканирования и распознавания, текст, попавший в выделенную серую зону, будет обработан и отображен в виде диалогового окна для подтверждения пользователем, как это показано на рисунке 16. Важно отметить, что для точного определения местоположения аудитории необходимо указать корпус, в котором она находится, поскольку из-за значительного разброса данных GPS и близкого расположения корпусов, точное определение корпуса только по координатам может быть затруднительно.

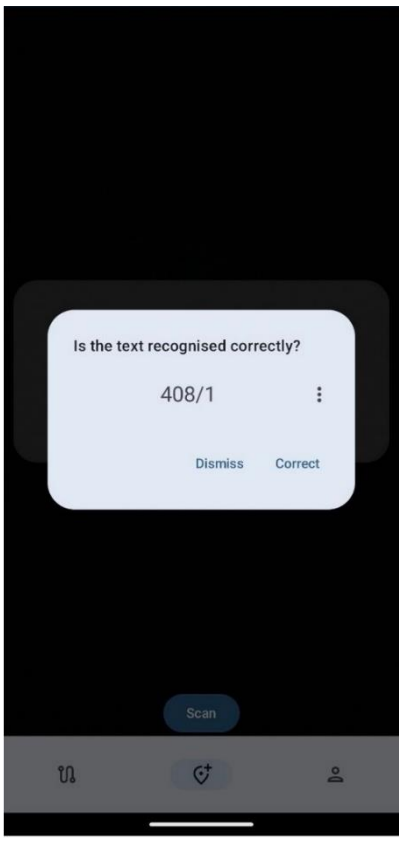

Рисунок 16 – Диалог подтверждения корректности распознанного текста

Для распознавания текста используется Google ML Kit — набор инструментов для машинного обучения, специально предназначенных для интеграции в мобильные приложения. Один из наиболее значимых компонентов этого набора — Text Recognition [51], который позволяет распознавать и извлекать текст из изображений в реальном времени. Технология распознавания текста обладает высокой точностью даже в сложных условиях, таких как плохое освещение или размытые изображения. Алгоритмы предобработки улучшают качество захваченных изображений, что способствует более точному распознаванию текста.

Процесс распознавания текста с помощью камеры мобильного телефона включает несколько этапов. Сначала камера устройства захватывает изображение текста, после чего алгоритмы предобработки улучшают его качество. Далее, Text Recognition анализирует изображение, извлекает текст и конвертирует его в машинно-читаемый формат. Этот процесс занимает всего около 100-300 миллисекунд, в зависимости от сложности и качества изображения. Исходный код

класса распознавания номера аудитории представлен в листинге В.1 приложения В.

На представленном рисунке 17 показан экран приложения, которое визуализирует маршрут в дополненной реальности.

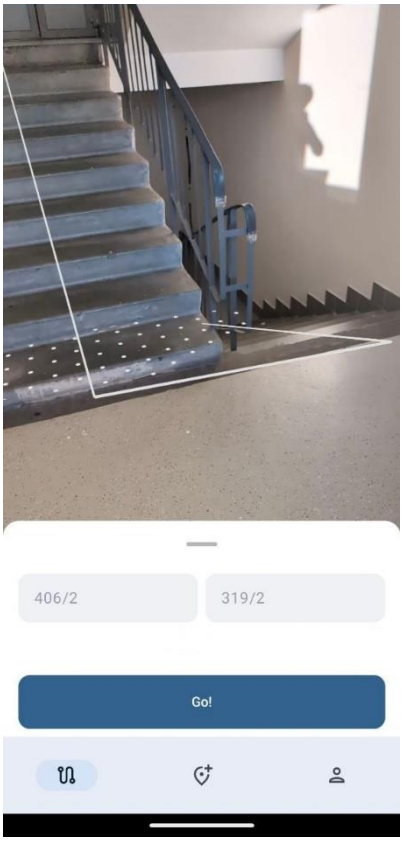

Рисунок 17 – Экран с построением маршрута в дополненной реальности

Для построения маршрута пользователю необходимо сначала нажать на левое текстовое поле, что перенаправит его на экран распознавания номера аудитории, где необходимо отсканировать номер ближайшей аудитории. Затем следует нажать на правое текстовое поле и выбрать из списка необходимую аудиторию, которая будет служить конечной точкой маршрута. После выбора начальной и конечной точек становится доступной кнопка «Go!», при нажатии на которую начинается построение маршрута. Линии, отображаемые на экране, указывают направление движения и помогают пользователю ориентироваться в пространстве, подсказывая, куда нужно идти. Исходный код экрана представлен в листинге Г.1 приложения Г.

Для выбора конечной точки пользователь перенаправляется на экран, изображенный на рисунке 18, где отображается список всех аудиторий, а также аудитории, которые ранее выбирались пользователем в качестве конечной точки.

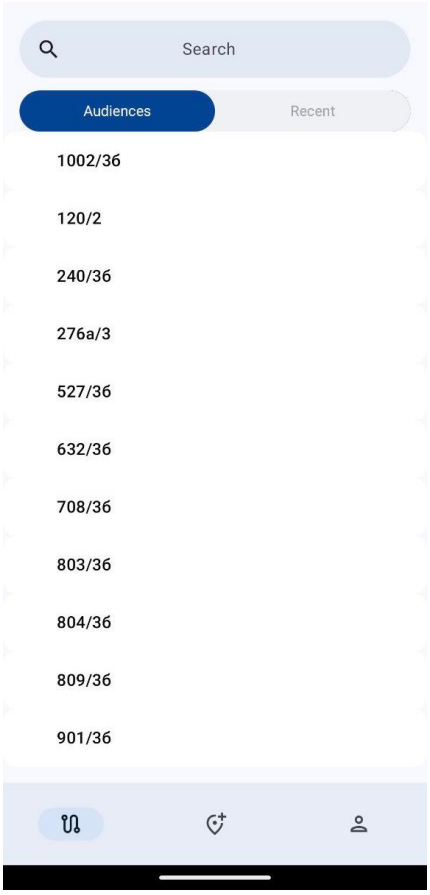

Рисунок 18 – Экран со списком аудиторий

Помимо этого, предоставляется возможность ввести часть номера аудитории в строку поиска, и система автоматически предложит подходящие варианты, что сокращает время на ручной поиск в длинном списке.

Вывод по разделу три

В данном разделе были разработаны: архитектура мобильного приложения, представленная в форме диаграммы классов; алгоритм поиска оптимального маршрута A*; а также пользовательский интерфейс, реализующий функциональные возможности распознавания номера аудитории, построение графа и визуализацию маршрута с применением дополненной реальности.

4. ТЕСТИРОВАНИЕ МОБИЛЬНОГО ПРИЛОЖЕНИЯ

4.1. Тестирование адаптивности пользовательского интерфейса

Тестирование адаптивности интерфейса включает проверку его визуальной целостности и функциональной корректности на различных типах устройств, таких как смартфоны, планшеты и устройства с крупными экранами. Основное внимание уделяется таким аспектам, как масштабируемость элементов интерфейса корректное отображение всех функциональных компонентов при изменении размеров экрана.

Для проведения тестирования использовались следующие устройства: Pixel 2 (5.0"), Pixel 8 (6.2"), Pixel Fold (7.6"), Pixel Tablet (10.95"). На каждом устройстве был запущен экран с построением маршрута в дополненной реальности, результаты представлены на рисунках 19-20.

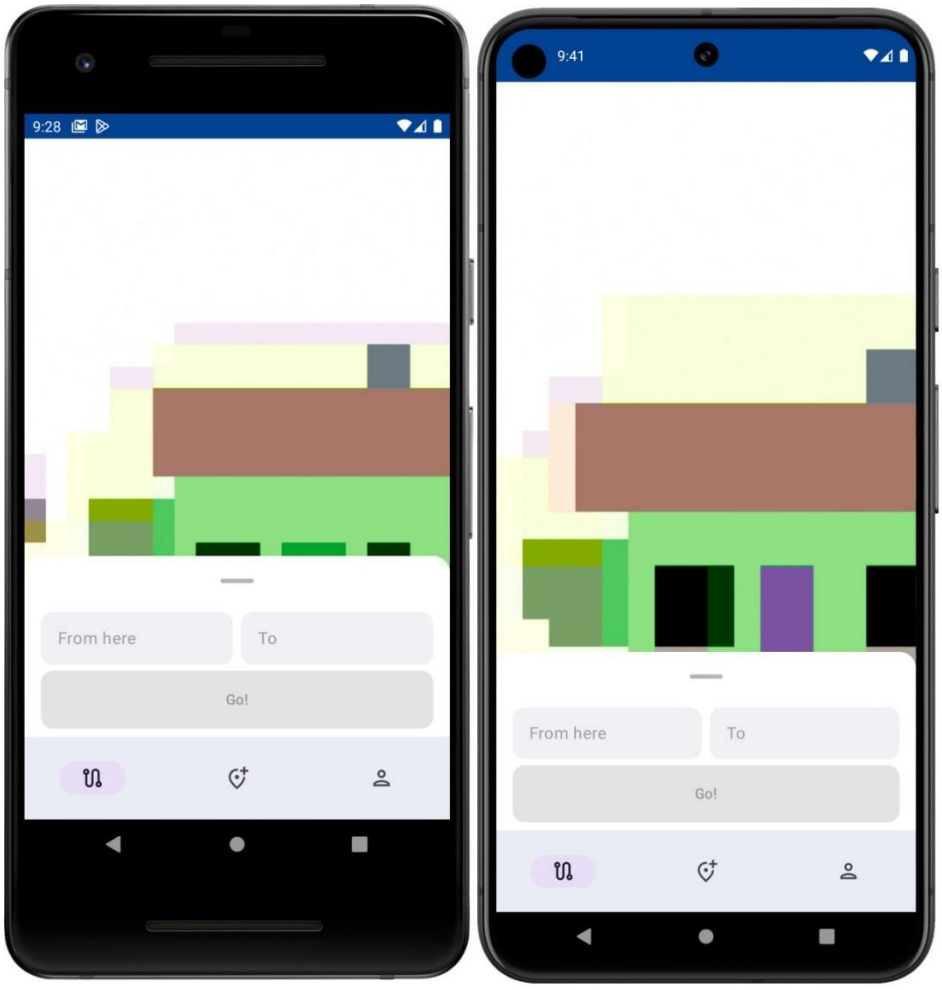

Рисунок 19 – Пользовательский интерфейс на Pixel 2 и 8 соответственно

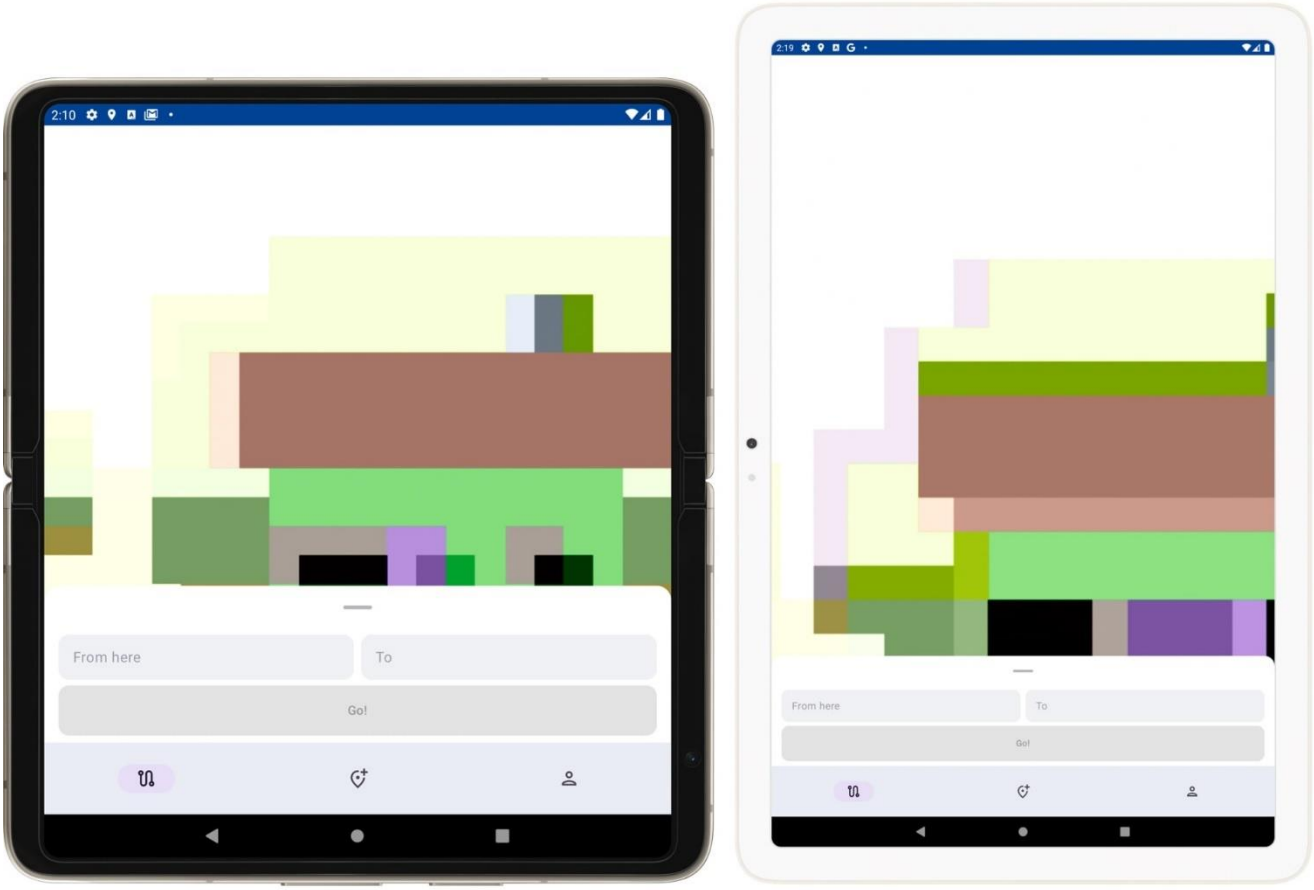

Рисунок 20 – Пользовательский интерфейс на Pixel Fold и Tablet соответственно

Результаты тестирования подтвердили, что все текстовые и графические элементы интерфейса отображаются корректно, вне зависимости от размеров и разрешений экранов. Масштабирование элементов происходило без искажений, сохраняя визуальную целостность и читаемость интерфейса. Это указывает на успешную реализацию адаптивного дизайна.

4.2. Тестирование приложения на ограниченные разрешения и несовместимость с ARCore

Работа мобильного приложения напрямую зависит от доступа к камере и наличия поддержки ARCore для реализации функций дополненной реальности. В связи с этим, важно провести тщательное тестирование приложения в условиях ограниченных разрешений на использование камеры и несовместимости с ARCore.

Приложение использует камеру для распознавания номеров аудиторий и отображения маршрутов с помощью дополненной реальности, что делает наличие доступа к камере и ARCore критически важным. Однако, в реальных условиях,

пользователи могут столкнуться с ситуациями, когда доступ к камере ограничен или устройство не поддерживает ARCore. Результаты тестирования представлены на рисунках 21-22.

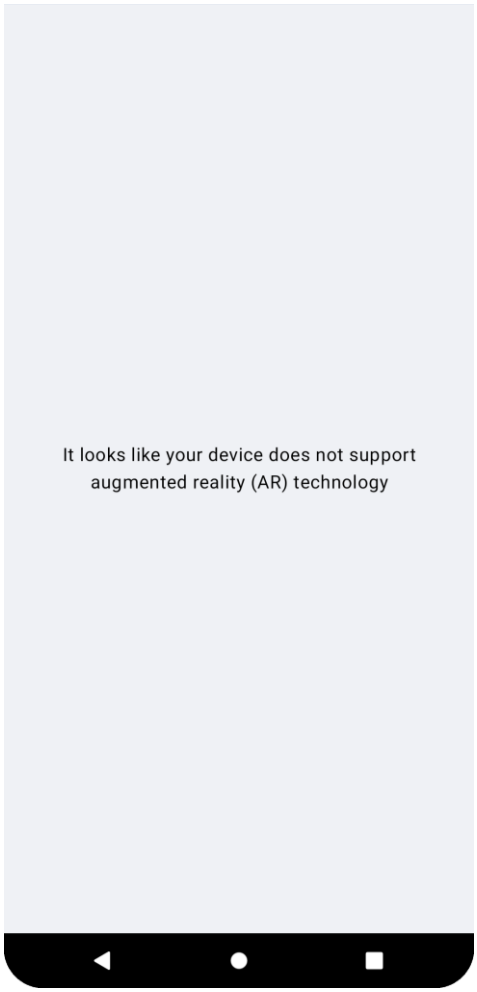

Рисунок 21 – Сообщение о том, что устройство не поддерживает технологию дополненной реальности (AR)

Тестирование мобильного приложения на устройстве, не поддерживающем ARCore, прошло успешно. В процессе тестирования было проверено, как приложение реагирует на отсутствие поддержки технологии дополненной реальности. В результате было установлено, что приложение корректно определяет отсутствие ARCore и выводит на экран соответствующее сообщение о том, что устройство не поддерживает технологию дополненной реальности и дальнейшее использование приложения невозможно.

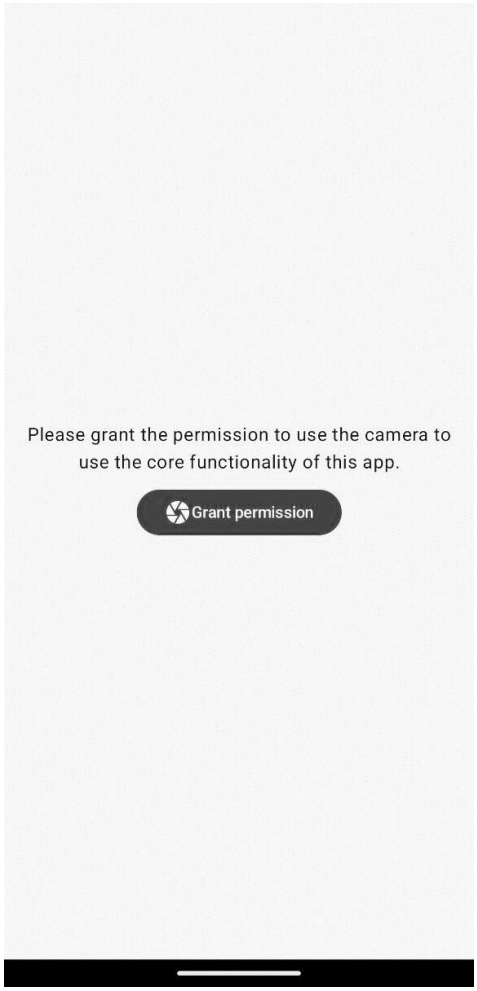

Рисунок 22 – Сообщение о том, что приложению необходимо предоставить доступ к камере устройства

Тестирование мобильного приложения в условиях отсутствия разрешения на доступ к камере прошло успешно. В ходе тестирования было проверено, как приложение реагирует на ситуацию, когда пользователь не предоставляет разрешение на использование камеры. Результаты показали, что приложение корректно определяет отсутствие данного разрешения и отображает соответствующее сообщение на экране.

При нажатии на кнопку «Grant permission» отобразится диалог с запросом у пользователя разрешения на доступ к камере. В случае получения разрешения приложение продолжит свою работу в штатном режиме, реализуя все заложенные функциональные возможности.

4.3. Автоматизированное тестирование основного функционала

Для проведения автоматизированного тестирования использовался фреймворк JUnit, позволяющий писать Unit-тесты, покрывающие классы, реализующий основной функционал мобильного приложения.

По результатам проведенного тестирования была составлена таблица 2.

Таблица 2 – Результаты автоматизированного тестирования

Тест	Описание	Результат
testAddVertexToDatabase()	Проверка функции добавления узла (вершины) в базу данных. Узел добавляется с указанными атрибутами в базу данных, а затем проверяется его наличие.	Пройден
testAddEdgeToDatabase()	Проверка функции добавления ребра в базу данных. Ребро добавляется между двумя узлами (вершинами), и проверяется его наличие в базе данных.	Пройден
testRemoveVertexFromDatabase()	Проверка функции удаления узла (вершины) из базы данных. Узел удаляется из базы данных, и проверяется его отсутствие в базе данных.	Пройден
testRemoveEdgeFromDatabase()	Проверка функции удаления ребра из базы данных. Ребро удаляется из базы данных, и проверяется его отсутствие в базе данных.	Пройден
testGetAllVerticesFromDatabase()	Проверка функции получения списка всех узлов (вершин) из базы данных. Возвращается список всех узлов в базе данных.	Пройден
testGetAllEdgesFromDatabase()	Проверка функции получения списка всех рёбер из базы данных. Возвращается список всех рёбер в базе данных.	Пройден
testGetEdgeFromDatabase()	Проверка функции получения ребра по начальному узлу (вершине) из базы данных. Для заданной начальной вершины возвращается соответствующее ей ребро.	Пройден
testGetVertexFromDatabase()	Проверка функции получения узла (вершины) по названию аудитории из базы данных. Возвращается узел, соответствующий заданному номеру аудитории.	Пройден

Продолжение таблицы 2

testGetAllAudiencesFromDatabase()	Проверка функции получения списка узлов (вершин) с номерами аудиторий из базы данных. Возвращается список всех узлов с номерами аудиторий.	Пройден
testAddRecordToDatabase()	Проверка функции добавления записи в базу данных. Запись добавляется в базу данных, а затем проверяется ее наличие.	Пройден
testRemoveRedordFromDatabase()	Проверка функции удаления записи из базы данных. Запись удаляется из базы данных, и проверяется, что ее больше нет в базе данных.	Пройден
testGetAllRecordsFromDatabase()	Проверка функции получения списка всех записей из базы данных. Возвращается список всех записей в базе данных.	Пройден
testRecognizeTextFromImage()	Проверка распознавания текста на изображении. Для этого теста было создано изображение с текстом, затем оно подавалось на вход алгоритму распознавания. Время, затраченное на распознавание текста, замерялось.	Пройден
testGetPath()	Проверка алгоритма нахождения оптимального маршрута. Возвращается список рёбер от начального узла до конечного.	Пройден

Как показано в таблице 2, все тесты были успешно пройдены. Особое внимание заслуживает тест, проверяющий распознавание текста, где время обработки не превышает 1,5 секунды.

Вывод по разделу четыре

В данном разделе было проведено тестирование мобильного приложения, результаты которого продемонстрировали соответствие разработанного функционала представленным требованиям.

ЗАКЛЮЧЕНИЕ

В рамках выпускной квалификационной работы было произведено проектирование и реализация мобильного приложения «Интерактивная карта ЮУрГУ» с использованием дополненной реальности.

В работе была затронута проблема навигации внутри крупных зданий, в частности, ЮУрГУ. Предлагаемый вариант решения позволит посетителям упростить перемещение по университету.

По ходу работы были решены следующие задачи:

– проведен аналитический обзор технической литературы по тематике работы, а также были найдены аналогичные решения поставленной цели;

– сформулированы требования к разрабатываемому мобильному приложению, а также выбраны среда и средства его разработки;

– разработан программный код мобильного приложения;

– проведено тестирование разработанного мобильного приложения.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Hassan, A.K. Universal Navigation on Smartphones. / A.K. Hassan. - New York: Springer New York, 2011. - 157 с.

2. Сибанбаева, С.Е. История, особенности и перспективы технологи дополненной реальности / С.Е. Сибанбаева, А.М. Муканова // Приоритетные научные направления: от теории к практике. - 2016. - №26 (1). - С. 194-199.

3. Что такое дополненная реальность, или AR? https://dynamics.microsoft.com/ru-ru/mixed-reality/guides/what-is-augmented-reality-ar/.

4. Разяпов, Р.В. Применение методов дополненной реальности в строительстве / Р.В. Разяпов // Экономика строительства. - 2021. - №5 (71). - С. 44-58.

5. Сайын, И.С. Анализ использования дополненной реальности в образовании / И.С. Сайын, Б.Б. Морозов // Вестник науки. - 2020. - №12 (33). - С. 23-26.

6. Даниленко, О.А. Опыт оперативного лечения пациентов с травмами и заболеваниями опорно-двигательного аппарата с использованием технологий дополненной реальности (AR) / О.А. Даниленко // Прикладная спортивная наука. - 2022. - №2 (16). - С. 77-85.

7. Юрлов, Д. Н. Принцип работы дополненной реальности в навигации / Д.Н. Юрлов // Academy. - 2019. - №1 (40). - С. 26-28.

8. Свищёв, А. В. Применение технологии дополненной реальности в сфере навигации внутри зданий / А. В. Свищёв, В. В Дульнев // Colloquium-journal. - 2020. - №10 (62). - С. 47-50.

9. Fumiaki, S. Indoor Navigation System Based on Augmented Reality Markers / S. Fumiaki // Innovative Mobile and Internet Services in Ubiquitous Computing. - Warsaw: Springer Cham, 2017. - С. 266-274.

10. Gotoh, Y. AR-Based Indoor Navigation: Hybrid Approach to Multi-floor Navigation / Y. Gotoh, H. Wang // Advances in Networked-based Information Systems. - Springer Cham, 2023. - С. 206-215.

11. Петрова, О.А. Метод определения текущего расположения в системах позиционирования и навигации внутри помещения. / О. А. Петрова, Г.В. Табунщик, Д.В. Мероде // Электротехнические и компьютерные системы. - 2017. - № 25. - С. 270–278.

12. Лось, Л.А. Использование Beacons для построения системы навигации внутри зданий / Л.А. Лось, Н.А. Волорова // BIG DATA and Advanced Analytics. - 2017. - №3. - С. 272-277.

13. Абашкина, С.С. Навигация в помещении с использованием дополненной реальности / С.С. Абашкина , Е.А. Карпова, М.М. Сизова, Б.Н. Седов // Аэрокосмическое приборостроение и эксплуатационные технологии. - 2020. - С. 204-212.

14. Фурсов, В.В. Анализ алгоритмов поиска пути для различных задач и их использование в мобильном приложении / В.В. Фурсов, С.Г. Самохвалова // Вестник Амурского государственного университета. Серия: Естественные и экономические науки. - 2017. - №77. - С. 58-64.

15. Delling, D. Engineering Route Planning Algorithms / D. Delling, P. Sanders, D. Schultes, D. Wagner // Algorithmics of Large and Complex Networks. Lecture Notes in Computer Science. - Berlin: Springer, 2009. - С. 117–139.

16. Амеличев, Г.Э. Сравнительный анализ алгоритмов поиска пути в помещении / Г.Э. Амеличев, В. С. Панина // E-Scio. - 2022. - №6 (69). - С. 224-231.

17. Сысоев, В.В. Итерационный алгоритм поиска кратчайшего пути в невзвешенном неориентированном графе / В.В. Сысоев // Современные информационные технологии и ИТ-образование. - 2021. - №17 (3). - С. 585-592.

18. Прихожий, А.А. Разнородный блочный алгоритм поиска кратчайших путей между всеми парами вершин графа / А.А. Прихожий, О.Н. Карасик // Системный анализ и прикладная информатика. - 2017. - №3. - С. 68-75.

19. Басараб, М.А. Алгоритмы решения задачи быстрого поиска пути на географических картах / М.А. Басараб, А.Б. Домрачева, В. М. Купляков // Инженерный журнал: наука и инновации. - 2013. - №11 (23). - С. 21-32.

20. Harabor, D. Online Graph Pruning for Pathfinding on Grid Maps / D. Harabor, A. Grastien // Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence. – 2011

21. Indoor maps. – https://yandex.com/support/mapeditor/indoor.html.

22. Indoor plans. – https://yandex.com/support/m-maps/indoor.html.

23. Go inside with indoor maps. – https://www.google.com/maps/about/partners/indoormaps.

24. IndoorAtlas. – https://www.indooratlas.com/get-started/.

25. Indoar. – https://www.viewar.com/products/indoar.

26. PolyNavi – расписание и навигация. –https://play.google.com/store/apps /details?id=com.starovoitov.polynavi&hl=en_GB.

27. РГГУ Навигатор. – https://play.google.com/store/apps/details?id= ru.rsuh.android&hl=en_GB.

28. Навигация Политеха AR. – https://play.google.com/store/apps/details?id= com.ARGeniuses.PolytechNavigationAR&hl=en_GB.

29. Мясоедова, В.А. К вопросу выбора оптимального языка программирования для разработки мобильного приложения / В.А. Мясоедова // E-Scio. - 2022. - №№11 (74). - С. 119-124.

30. Юдина, С.В. Кроссплатформенные и нативные мобильные приложения / С.В. Юдина, Ю.С. Артюхина // Теория и практика современной науки. - 2018. - №№5 (35). - С. 972-974.

31. Технологии разработки мобильных приложений / А.Н. Мытников, Е. А. Мытникова, Л.Н. Кузнецова, С.Ю. Солин // Теория и практика современной науки. - 2016. - №№4 (10). - С. 504-507.

32. Что такое Android. – https://www.android.com/intl/ru_ru/what-is-android/.

33. Погорелов, Д.В. Сравнение мобильных операционных систем Android и iOS / Д.В. Погорелов, Е.А. Колоколов, В.В. Ермолаева // Вестник науки. - 2022. - №№12 (57). - С. 118-127.

34. Mobile Operating System Market Share Worldwide. – https://gs.statcounter.com/os-market-share/mobile/worldwide.

35. What is Java?. – https://www.ibm.com/topics/java.

36. Comparison to Java. – https://kotlinlang.org/docs/comparison-to-java.html.

37. Android's Kotlin-first approach. – https://developer.android.com/kotlin/first.

38. Приходько, Н.А. Причины выбора Kotlin как основного языка мобильной разработки на Android / Н.А. Приходько, В.А. Сивченков, А.В. Панов // Международный журнал гуманитарных и естественных наук. - 2023. - № №11-4 (86). - С. 104-107.

39. Comparison to Java. – https://kotlinlang.org/docs/comparison-to-java.html#what-kotlin-has-that-java-does-not.

40. Layouts in Views. – https://developer.android.com/develop/ui/views/layout/declaring-layout.

41. Why adopt Compose. – https://developer.android.com/jetpack/compose/why-adopt.

42. Compare Compose and View metrics. –https://developer.android.com/jetpack/compose/migrate/compare-metrics.

43. Appropriate Uses For SQLite. – https://www.sqlite.org/whentouse.

44. Firebase Realtime Database. – https://firebase.google.com/docs/.

45. Edge Database + Sync for Embedded Devices in Mobile and IoT – offline-first / local-first. – https://objectbox.io/.

46. Android Studio. – https://developer.android.com/studio.

47. IntelliJ IDEA overview. – https://www.jetbrains.com/help/idea/discover-intellij-idea.html.

48. About the Eclipse Foundation. – https://www.eclipse.org/org/.

49. Guide to app. – https://developer.android.com/topic/architecture.

50. Cloud Anchors allow different users to share AR experiences. – https://developers.google.com/ar/develop/cloud-anchors.

51. Text recognition v2. – https://developers.google.com/ml-kit/vision/text-recognition/v2.

ПРИЛОЖЕНИЯ

Приложение А

Листинг А.1 — Исходный код алгоритма поиска оптимального маршрута A*

```
class AStar @Inject constructor() : Pathfinding {
    override fun findPath(
        start: Vertex, //Начальная точка маршрута
        finish: Vertex, //Конечная точка маршрута
        graph: Map<Vertex, List<Edge>>, // Граф, хранящий все маршруты
    ): List<Edge> {
        val fScore = mutableMapOf<Vertex, Long>().withDefault { Long.MAX_VALUE }
        val gScore = mutableMapOf<Vertex, Long>().withDefault { Long.MAX_VALUE }
        val openSet = PriorityQueue<Vertex>(compareBy { fScore[it] ?:
Long.MAX_VALUE }).apply {
            add(start)
        }
        val cameFrom = mutableMapOf<Vertex, Edge>()
        gScore[start] = 0
        fScore[start] = heuristicCostEstimate(start, finish)
        while (openSet.isNotEmpty()) {
            val current = openSet.poll() ?: break
            if (current == finish) {
                return reconstructPath(cameFrom, current)//Маршрут найден
            }
            openSet.remove(current)
            for (edge in graph[current] ?: emptyList()) {
                val neighbor = edge.destination
                val tentativeGScore = (gScore[current] ?: Long.MAX_VALUE) +
edge.weight
                if (tentativeGScore < (gScore[neighbor] ?: Long.MAX_VALUE)) {
                    cameFrom[neighbor] = edge
                    gScore[neighbor] = tentativeGScore
                    fScore[neighbor] = tentativeGScore +
heuristicCostEstimate(neighbor, finish)
                    if (!openSet.contains(neighbor)) {
                        openSet.add(neighbor)
                    }
                }
            }
        }
        return listOf() // Маршрут не найден, возвращается пустой список
    }
    private fun reconstructPath(cameFrom: Map<Vertex, Edge>, current: Vertex):
List<Edge> {
        val path = mutableListOf<Edge>()
        var currentVertex: Vertex? = current
        while (currentVertex != null && cameFrom.containsKey(currentVertex)) {
            val edge = cameFrom[currentVertex] ?: break
            path.add(edge)
            currentVertex = edge.source
        }
        return path.reversed()
    }
      //Функция расчета для эвристической оценки
    private fun heuristicCostEstimate(start: Vertex, finish: Vertex): Long {
        return abs(start.coordinates.x - finish.coordinates.x).toLong() +
                abs(start.coordinates.y - finish.coordinates.y).toLong() +
                abs(start.coordinates.z - finish.coordinates.z).toLong()
    }
}
```

Приложение Б

Листинг Б.1 — Исходный код экрана администратора

```
@Composable
fun AdminModeScreen(audience: String, onAddAudience: () -> Unit) {
    val viewModel: AdminModeViewModel = hiltViewModel()
    ARNavigationTheme {
        AdminModeContent(
            audience = audience,
            onAddAudience = onAddAudience,
            onCreateVertex = viewModel::createVertex,
            onRemoveVertex = viewModel::removeVertex,
            onCreateEdge = viewModel::createEdge,
            onRemoveEdge = viewModel::removeEdge,
            onObserveGraph = viewModel.graphState
        )
    }
}
//Содержимое экрана администратора
@OptIn(ExperimentalMaterial3Api::class)
@Composable
private fun AdminModeContent(
    audience: String, //Номер аудитории
    modifier: Modifier = Modifier,
    drawerHelper: DrawerHelper = DrawerHelper(), //Класс для отрисовки в AR
    onObserveGraph: StateFlow<ViewModelState<Map<Vertex, List<Edge>>>>,
    onAddAudience: () -> Unit, //Функция для добавления новой аудитории
    onCreateVertex: (Vertex) -> Unit, //Функция для добавления нового узла
    onCreateEdge: (Edge) -> Unit, //Функция для добавления нового ребра
    onRemoveVertex: (Vertex) -> Unit, //Функция для удаления узла
    onRemoveEdge: (Edge) -> Unit, //Функция для удаления ребра
) {
    var audienceNumber by remember { mutableStateOf(audience) }

    val lifecycleOwner = LocalLifecycleOwner.current
    val context = LocalContext.current

    val engine = rememberEngine()
    val modelLoader = rememberModelLoader(engine)
    val cameraNode = rememberCameraNode2(engine)
    val childNodes = rememberNodesMap()
    val view = rememberView(engine)
    val collisionSystem = rememberCollisionSystem(view)
    var frame by remember { mutableStateOf<Frame?>(null) }
    val modelInstances = remember { mutableListOf<ModelInstance>() }
    var trackingFailureReason by remember
{ mutableStateOf<TrackingFailureReason?>(null) }
    var clickedNode by remember { mutableStateOf<Node?>(null) }

    val graphState by onObserveGraph.collectAsState()

    val scaffoldState = rememberBottomSheetScaffoldState()
    val displayRotationHelper = DisplayRotationHelper(context)

    BottomSheetScaffold(
        modifier = Modifier.fillMaxSize(),
        scaffoldState = scaffoldState,
        sheetPeekHeight = 48.dp,
        sheetContainerColor = Color.White,
        sheetShape = RoundedCornerShape(topStart = 16.dp, topEnd = 16.dp),
        sheetContent = {
```

```
            BottomSheetContent(
                audienceNumber = audienceNumber,
                modifier = modifier,
                cameraNode = cameraNode,
                drawerHelper = drawerHelper,
                graphState = graphState,
                context = context,
                engine = engine,
                modelInstances = modelInstances,
                modelLoader = modelLoader,
                childNodes = childNodes,
                onAddAudience = onAddAudience
            )
        }
    ) { innerPadding ->
        ARScene(
            modifier = modifier
                .fillMaxSize()
                .padding(innerPadding),
            childNodes = childNodes.keys.toList(),
            engine = engine,
            view = view,
            lifecycle = lifecycleOwner.lifecycle,
            activity = LocalContext.current as MainActivity,
            modelLoader = modelLoader,
            collisionSystem = collisionSystem,
            sessionConfiguration = { session, config ->
                config.focusMode = Config.FocusMode.FIXED
                config.depthMode = if
(session.isDepthModeSupported(Config.DepthMode.AUTOMATIC)) {
                    Config.DepthMode.AUTOMATIC
                } else {
                    Config.DepthMode.DISABLED
                }
                config.instantPlacementMode =
Config.InstantPlacementMode.LOCAL_Y_UP
                config.lightEstimationMode = Config.LightEstimationMode.DISABLED
                config.cloudAnchorMode = Config.CloudAnchorMode.ENABLED
            },
            cameraNode = cameraNode,
            planeRenderer = true,
            onTrackingFailureChanged = {
                trackingFailureReason = it
            },
            onSessionResumed = {
                displayRotationHelper.onResume()
                cameraNode.onTrackingStateChanged(TrackingState.TRACKING)
            },
            onSessionPaused = {
                displayRotationHelper.onPause()
                cameraNode.onTrackingStateChanged(TrackingState.PAUSED)
                it.close()
            },
            onSessionUpdated = { _, updatedFrame ->
                frame = updatedFrame
            },
            onGestureListener = rememberOnGestureListener(
                onSingleTapConfirmed = { motionEvent, node ->
                    handleSingleTap(
                        node = node,
```

```
                        motionEvent = motionEvent,
                        frame = frame,
                        cameraNode = cameraNode,
                        context = context,
                        engine = engine,
                        modelLoader = modelLoader,
                        modelInstances = modelInstances,
                        childNodes = childNodes,
                        audienceNumber = audienceNumber,
                        onCreateVertex = onCreateVertex
                    )

                    if (audienceNumber.isNotBlank()) {
                        audienceNumber = ""
                    }
                },
                onLongPress = { _, node ->
                    handleLongPress(node, context, childNodes, onRemoveVertex,
onRemoveEdge)
                },
                onDoubleTap = { _, node ->
                    if (node == null) {
                        return@rememberOnGestureListener
                    }

                    if (clickedNode == null) {
                        clickedNode = node
                        Toast.makeText(context, "Source point is defined",
Toast.LENGTH_SHORT)
                            .show()
                        return@rememberOnGestureListener
                    }

                    if (clickedNode != node) {
                        Toast.makeText(context, "Destination point is defined",
Toast.LENGTH_SHORT)
                            .show()

                        val sourceVertex =
                            (childNodes[clickedNode] ?:
childNodes[clickedNode?.parent]) as Vertex
                        val destinationVertex =
                            (childNodes[node] ?: childNodes[node.parent]) as
Vertex

                        val edgeNode = drawerHelper.drawEdge(
                            destinationVertex = node,
                            sourceVertex = clickedNode!!,
                            engine = engine,
                        )

                        if (cameraNode.trackingState == TrackingState.TRACKING) {
                            cameraNode.session?.let { currentSession ->
                                frame?.getUpdatedPlanes()
                                    ?.firstOrNull { it.type ==
Plane.Type.HORIZONTAL_UPWARD_FACING }
                                    ?.let
{ it.createAnchorOrNull(it.centerPose) }?.let { anchor ->
                                        Toast.makeText(
                                            context,
                                            "Placing an edge",
```

```
                                        Toast.LENGTH_SHORT
                                    ).show()
                                    //Запоминание местоположения ребра
                                    CloudAnchorNode(
                                        engine = engine, anchor = anchor
                                    ).apply {
                                        host(
                                            session = currentSession,
                                            ttlDays = 1
                                        ) { cloudAnchorId, state ->
                                            if (state ==
Anchor.CloudAnchorState.SUCCESS && cloudAnchorId != null) {
                                                edgeNode.addChildNode(this)

                                                childNodes[edgeNode] = Edge(
                                                    id = Random.nextLong(),
                                                    source = sourceVertex,
                                                    destination =
destinationVertex,
                                                    cloudAnchorId =
cloudAnchorId,
                                                    weight =
drawerHelper.calculateWeight(
                                                        sourceVertex,
                                                        destinationVertex
                                                    )
                                                ).also(onCreateEdge)

                                                Toast.makeText(
                                                    context,
                                                    "Edge has been added",
                                                    Toast.LENGTH_SHORT
                                                ).show()
                                            } else {
                                                childNodes.remove(edgeNode)
                                                edgeNode.destroy()
                                            }
                                        }
                                    }
                                }
                            }
                            clickedNode = null
                        }
                    }
                )
            )
        }
    }
    //Содержимое диалогового окна
    @Composable
    private fun BottomSheetContent(
        audienceNumber: String,
        modifier: Modifier,
        cameraNode: ARCameraNode,
        drawerHelper: DrawerHelper,
        graphState: ViewModelState<Map<Vertex, List<Edge>>>,
        context: Context,
        engine: Engine,
        modelInstances: MutableList<ModelInstance>,
```

```
    modelLoader: ModelLoader,
    childNodes: SnapshotStateMap<Node, Any>,
    onAddAudience: () -> Unit,
) {
    val audienceField by remember { mutableStateOf(audienceNumber) }

    Column(
        verticalArrangement = Arrangement.spacedBy(16.dp),
        horizontalAlignment = Alignment.CenterHorizontally
    ) {
        Row(
            modifier = modifier
                .fillMaxWidth()
                .padding(horizontal = 16.dp),
            horizontalArrangement = Arrangement.spacedBy(8.dp)
        ) {
            SmallTextField(
                modifier = Modifier.clickable { onAddAudience() },
                placeholder = audienceField.ifBlank { stringResource(id =
R.string.click_to_add_an_audience_text_placeholder) },
                textAlign = TextAlign.Center
            )
        }
//Кнопка загрузки графа из облачной базы данных
        Button(
            modifier = Modifier
                .fillMaxWidth()
                .height(64.dp)
                .padding(start = 16.dp, end = 16.dp, bottom = 8.dp),
            shape = RoundedCornerShape(12.dp),
            onClick = {
                if (cameraNode.trackingState == TrackingState.TRACKING) {
                    childNodes.destroy()

                    when (graphState) {
                        is ViewModelState.Error -> showToast(
                            context,
                            "An error occurred during graph initialization"
                        )

                        is ViewModelState.Loading -> showToast(context, "Loading
the graph...")
                        is ViewModelState.None -> showToast(context, "None. ?:_)")
                        is ViewModelState.Success -> {
                            cameraNode.session?.let { currentSession ->
                                drawerHelper.loadGraph(
                                    graphState.data,
                                    context,
                                    engine,
                                    currentSession,
                                    modelInstances,
                                    modelLoader,
                                    childNodes
                                )
                            }
                        }
                    }
                }
            }
        ) {
            Image(
```

```
                imageVector = Icons.Outlined.CloudDownload,
                contentDescription = null,
                colorFilter = ColorFilter.tint(Color.White)
            )
        }
    }
}

@Composable
fun rememberNodesMap(creator: MutableMap<Node, Any>.() -> Unit = {}) = remember {
    buildMap(creator).toList().toMutableStateMap()
}.also { nodes ->
    DisposableEffect(nodes) {
        onDispose {
            nodes.forEach { it.key.destroy() }
            nodes.clear()
        }
    }
}

fun SnapshotStateMap<Node, Any>.destroy() {
    this.forEach { (node, _) ->
        node.destroy()
    }
    clear()
}
//Функция отображения уведомлений на экране
private fun showToast(context: Context, message: String) {
    Toast.makeText(context, message, Toast.LENGTH_SHORT).show()
}
//Функция реагирования на одиночное нажатие по экрану (добавление узла)
private fun handleSingleTap(
    node: Node?,
    motionEvent: MotionEvent,
    frame: Frame?,
    cameraNode: ARCameraNode,
    context: Context,
    engine: Engine,
    modelLoader: ModelLoader,
    modelInstances: MutableList<ModelInstance>,
    childNodes: SnapshotStateMap<Node, Any>,
    audienceNumber: String,
    onCreateVertex: (Vertex) -> Unit,
) {
    if (node == null) {
        val hitResults = frame?.hitTest(motionEvent.x, motionEvent.y)
        hitResults?.firstOrNull { it.isValid(depthPoint = true, point = false) }
            ?.createAnchorOrNull()?.let { anchor ->
            val vertexNode = ModelNode(
                modelInstance = modelInstances.apply {
                    if (isEmpty()) {
                        this +=
modelLoader.createInstancedModel("models/cylinder.glb", 1)
                    }
                }.removeLast(), scaleToUnits = 0.1f
            )

            if (cameraNode.trackingState == TrackingState.TRACKING) {
                showToast(context, "Placing vertex")

                cameraNode.session?.let { currentSession ->
```

```
                    if (!currentSession.canHostCloudAnchor(cameraNode)) {
                        showToast(context, "Can't host cloud node")
                        vertexNode.destroy()
                        return
                    }
                //Запоминание местоположения узла
                    CloudAnchorNode(engine = engine, anchor = anchor).apply {
                        host(session = currentSession, ttlDays = 1)
{ cloudAnchorId, state ->
                            if (state == Anchor.CloudAnchorState.SUCCESS &&
cloudAnchorId != null) {
                                vertexNode.parent = this
                                childNodes[vertexNode] = Vertex(
                                    id = Random.nextLong(),
                                    data = audienceNumber.ifBlank
{ audienceNumber },
                                    cloudAnchorId = cloudAnchorId,
                                    coordinates = vertexNode.worldPosition
                                ).also(onCreateVertex)
                                showToast(context, "Vertex has been added")
                            } else {
                                childNodes.remove(vertexNode)
                                vertexNode.destroy()
                            }
                        }
                    }
                }
            }
        }
    }
}
//Функция реагирования на долгое нажатие по экрану (удаление элемента)
private fun handleLongPress(
    node: Node?,
    context: Context,
    childNodes: SnapshotStateMap<Node, Any>,
    onRemoveVertex: (Vertex) -> Unit,
    onRemoveEdge: (Edge) -> Unit,
) {
    if (node == null) return
    childNodes[node]?.let { element ->
        when (element) {
            is Edge -> {
                onRemoveEdge(element)
                childNodes.remove(node)
                showToast(context, "Edge has been removed")
                node.destroy()
                node.parent?.destroy()
            }
            is Vertex -> {
                onRemoveVertex(element)
                childNodes.remove(node)
                showToast(context, "Vertex has been removed")
                node.destroy()
                node.parent?.destroy()
            }
            else -> {
                return
            }
        }
    }}
```

Приложение В

Листинг В.1 — Исходный код класса распознания номера аудитории

```
class TextRecognitionAnalyzer @Inject constructor(
    private val onDetectedTextUpdated: (String) -> Unit,
) : ImageAnalysis.Analyzer {
    private val coroutineScope = CoroutineScope(Dispatchers.IO +
SupervisorJob())
    private val textRecognizer: TextRecognizer =
        TextRecognition.getClient(TextRecognizerOptions.DEFAULT_OPTIONS)
    private var imageCropPercentages: Pair<Int, Int> =
        Pair(DESIRED_HEIGHT_CROP_PERCENT, DESIRED_WIDTH_CROP_PERCENT)
    @OptIn(ExperimentalGetImage::class)
    override fun analyze(image: ImageProxy) {
        coroutineScope.launch {
            val mediaImage = image.image ?: return@launch
            val rotationDegrees = image.imageInfo.rotationDegrees
            val imageHeight = mediaImage.height //Высота изображения
            val imageWidth = mediaImage.width // Ширина изображения
            val actualAspectRatio = imageWidth / imageHeight
            val convertImageToBitmap =
ImageUtils.convertYuv420888ImageToBitmap(mediaImage)
            val cropRect = Rect(0, 0, imageWidth, imageHeight) //Зона анализа
            val currentCropPercentages = imageCropPercentages
            if (actualAspectRatio > 3) {
                val originalHeightCropPercentage=currentCropPercentages.first
                val originalWidthCropPercentage=currentCropPercentages.second
                imageCropPercentages =
                    Pair(originalHeightCropPercentage / 2,
originalWidthCropPercentage)
            }
            val cropPercentages = imageCropPercentages
            val heightCropPercent = cropPercentages.first
            val widthCropPercent = cropPercentages.second
            val (widthCrop, heightCrop) = when (rotationDegrees) {
                90, 270 -> Pair(heightCropPercent / 100f, widthCropPercent /
100f)
                else -> Pair(widthCropPercent / 100f, heightCropPercent /
100f)
            }
            cropRect.inset(
                (imageWidth * widthCrop / 2).toInt(),
                (imageHeight * heightCrop / 2).toInt()
            )
            val croppedBitmap =ImageUtils.rotateAndCrop(convertImageToBitmap,
rotationDegrees, cropRect)
            recognizeTextOnDevice(InputImage.fromBitmap(croppedBitmap,
0)).addOnCompleteListener {
                image.close()
            }
        }
    }
    private fun recognizeTextOnDevice(
        image: InputImage,
    ) = return textRecognizer.process(image)
            .addOnSuccessListener { visionText ->
                onDetectedTextUpdated(visionText.text)
            }
            .addOnFailureListener { exception ->
                error(exception)
            }
    }}
```

Приложение Г

Листинг В.1 — Исходный код экрана с построением маршрута

```
@Composable
fun RouteScreen(
    source: String, //Начальная точка маршрута
    destination: String, //Конечная точка маршрута
    navController: NavController, //Класс для навигации между экранами приложения
) {
    val viewModel: RouteViewModel = hiltViewModel()
    ARNavigationTheme {
        RouteScreenContent(
            source = source,
            destination = destination,
            viewModel = viewModel,
            onChooseSource = { navController.navigate(Screen.Scanner.route) },
            onChooseDestination =
{ navController.navigate("${Screen.Search.route}?$SOURCE_PARAM_KEY=$source") }
        )
    }
}
//Содержимое экрана
@OptIn(ExperimentalMaterial3Api::class)
@Composable
private fun RouteScreenContent(
    source: String,
    destination: String,
    viewModel: RouteViewModel,
    onChooseSource: () -> Unit,//Функция для реагирования на нажатие кнопки выбора
начальной точки
    onChooseDestination: () -> Unit, ,//Функция для реагирования на нажатие кнопки
выбора конечной точки
) {
    val context = LocalContext.current
    val sourceTextField by remember { mutableStateOf(source) }
    val destinationTextField by remember { mutableStateOf(destination) }
    val sourceVertex by produceState<ViewModelState<Vertex>>(
        initialValue = ViewModelState.None(),
        source
    ) {
        value = viewModel
            .getVertex(source)
            .await()
    }
    val destinationVertex by produceState<ViewModelState<Vertex>>(
        initialValue = ViewModelState.None(),
        destination
    ) {
        value = viewModel
            .getVertex(destination)
            .await()
    }
    val lifecycleOwner = androidx.lifecycle.compose.LocalLifecycleOwner.current
    val engine = rememberEngine()
    val modelLoader = rememberModelLoader(engine)
    val cameraNode = rememberCameraNode2(engine)
    val childNodes = rememberNodesMap()
    val view = rememberView(engine)
    val modelInstances = remember { mutableListOf<ModelInstance>() }
    val collisionSystem = rememberCollisionSystem(view)
    var frame by remember { mutableStateOf<Frame?>(null) }
```

```
    var trackingFailureReason by remember {
        mutableStateOf<TrackingFailureReason?>(null)
    }
    val scaffoldState = rememberBottomSheetScaffoldState()
    val minutes by remember { mutableIntStateOf(0) }
    val graphState = viewModel.graph.collectAsState().value
    val drawerHelper = DrawerHelper()
    val displayRotationHelper = DisplayRotationHelper(context)
    BottomSheetScaffold(
        sheetPeekHeight = 48.dp,
        scaffoldState = scaffoldState,
        sheetContainerColor = Color.White,
        sheetShape = RoundedCornerShape(topStart = 16.dp, topEnd = 16.dp),
        sheetContent = {
            BottomSheetTextFields(
                onChooseSource,
                sourceTextField,
                onChooseDestination,
                destinationTextField,
                minutes
            )
            Button(
                modifier = Modifier
                    .fillMaxWidth()
                    .height(64.dp)
                    .padding(start = 16.dp, end = 16.dp, bottom = 8.dp),
                shape = RoundedCornerShape(12.dp),
                enabled = sourceTextField.isNotBlank() &&
destinationTextField.isNotBlank(),
                onClick = {
                    when (graphState) {
                        is ViewModelState.Error -> Toast.makeText(
                            context,
                            "Error",
                            Toast.LENGTH_SHORT
                        ).show()
                        is ViewModelState.Loading -> Toast.makeText(
                            context,
                            "Loading",
                            Toast.LENGTH_SHORT
                        ).show()
                        is ViewModelState.None -> {}
                        is ViewModelState.Success -> {
                            if (cameraNode.trackingState ==
TrackingState.TRACKING) {
                                childNodes.destroy()
                                val path = viewModel.findPath( //Поиск маршрута
                                    start = (sourceVertex as
ViewModelState.Success<Vertex>).data,
                                    destination = (destinationVertex as
ViewModelState.Success<Vertex>).data,
                                    graph = graphState.data
                                )
                                if (cameraNode.trackingState ==
TrackingState.TRACKING) {
                                    cameraNode.session?.let { currentSession ->
                                        drawerHelper.drawPath(
                                            path = path,
                                            context = context,
                                            engine = engine,
                                            session = currentSession,
```

```
                                            modelInstances = modelInstances,
                                            modelLoader = modelLoader,
                                            childNodes = childNodes
                                        )
                                    }
                                }
                            }
                        }
                    }
                ) {
                    Text(text = stringResource(id = R.string.go_text_button))
                }
            }) { innerPadding ->
            ARScene(
                modifier = Modifier
                    .fillMaxSize()
                    .padding(innerPadding),
                childNodes = childNodes.keys.toList(),
                lifecycle = lifecycleOwner.lifecycle,
                engine = engine,
                view = view,
                modelLoader = modelLoader,
                collisionSystem = collisionSystem,
                sessionConfiguration = { session, config ->
                    config.depthMode =
                        when
(session.isDepthModeSupported(Config.DepthMode.AUTOMATIC)) {
                            true -> Config.DepthMode.AUTOMATIC
                            else -> Config.DepthMode.DISABLED
                        }
                    config.instantPlacementMode =
Config.InstantPlacementMode.LOCAL_Y_UP
                    config.lightEstimationMode = Config.LightEstimationMode.DISABLED
                    config.cloudAnchorMode = Config.CloudAnchorMode.ENABLED
                },
                cameraNode = cameraNode,
                planeRenderer = true,
                onTrackingFailureChanged = {
                    trackingFailureReason = it
                },
                onSessionResumed = {
                    displayRotationHelper.onResume()
                    cameraNode.onTrackingStateChanged(TrackingState.TRACKING)
                },
                onSessionPaused = {
                    displayRotationHelper.onPause()
                    cameraNode.onTrackingStateChanged(TrackingState.PAUSED)
                    it.close()
                },
                onSessionUpdated = { _, updatedFrame ->
                    frame = updatedFrame
                },
            )
        }
    }
    @Composable
    private fun BottomSheetTextFields(
        onChooseSource: () -> Unit,
        sourceTextField: String,
        onChooseDestination: () -> Unit,
```

```
    destinationTextField: String,
    minutes: Int,
) {
    Row(
        modifier = Modifier
            .fillMaxWidth()
            .padding(horizontal = 16.dp, vertical = 6.dp),
        horizontalArrangement = Arrangement.spacedBy(8.dp),
    ) {
        SmallTextField(
            modifier = Modifier
                .weight(0.5f)
                .clickable {
                    onChooseSource()
                },
            placeholder = sourceTextField.ifBlank {
                stringResource(
                    id = R.string.from_here_placeholder_text
                )
            },
        )
        SmallTextField(
            modifier = Modifier
                .weight(0.5f)
                .clickable {
                    onChooseDestination()
                },
            placeholder = destinationTextField.ifBlank {
                stringResource(
                    id = R.string.to_placeholder_text
                )
            }
        )
    }
    if (sourceTextField.isNotBlank() && destinationTextField.isNotBlank()) {
        TravelTimeBar(minutes = minutes)
                                            }
}
```