

Министерство цифрового развития, связи и массовых коммуникаций
Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Сибирский государственный университет телекоммуникаций и информатики»
(СибГУТИ)

Кафедра _____ ТСиВС
Допустить к защите

Зав.каф _____ Дроздова В.Г.

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА БАКАЛАВРА

Разработка кроссплатформенного интернет-сервиса для предложения услуг.

Пояснительная записка

Студент _____ Ляшенко К. Е. /...../

Факультет _____ ИВТ _____ Группа _____ ИА-031

Руководитель _____ Моренкова О. И //

Новосибирск 2024 г.

Министерство цифрового развития, связи и массовых коммуникаций
Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Сибирский государственный университет телекоммуникаций и информатики»
(СибГУТИ)

КАФЕДРА

*Телекоммуникационных систем и вычислительных
средств*

ЗАДАНИЕ НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ БАКАЛАВРА

СТУДЕНТА Ляшенко. К. Е ГРУППЫ ИА-031

УТВЕРЖДАЮ

« ____ » _____ 20__ г.

Зав. кафедрой
_____ / В.Г.Дроздова /

Новосибирск 2024 г.

1. Тема выпускной квалификационной работы бакалавра

Разработка кроссплатформенного интернет-сервиса для предложения услуг.

утверждена приказом СибГУТИ от «____»____ 20__ г. № ____

2.Срок сдачи студентом законченной работы « ____ » _____ 20__ г.

3.Исходные данные к работе

1 Специальная литература

2 Материалы сети интернет

4.Содержание пояснительной записки (перечень подлежащих разработке вопросов)	Сроки выполнения по разделам

Дата выдачи задания «____» _____ 20__ г.

Руководитель _____
подпись

Задание принял к исполнению «____» _____ 20__ г.

Студент _____
подпись

АННОТАЦИЯ

Выпускная квалификационная работа _____
(Фамилия, И.О.)

по теме

«_____» Разработка
сайта для фирмы «ИП Филиппов»»

Объём работы - _____ страниц, на которых размещены _____ рисунков и
_____ таблиц. При написании работы использовалось _____ источников.

Ключевые слова:

Работа выполнена _____
(название предприятия, подразделения)

Основные результаты*

*В данном разделе должны быть отражены основные результаты исследования

Оглавление

ВВЕДЕНИЕ.....	5
1. ХАРАКТЕРИСТИКА ПРЕДМЕТНОЙ ОБЛАСТИ.....	7
1.1 Актуальность кроссплатформенных приложений	7
1.2 Актуальность баз данных	8
2. ОБЗОР И ОБОСНОВАНИЕ ВЫБОРА СРЕДСТВ РЕАЛИЗАЦИИ ПРОЕКТА	10
2.1 Платформы.....	10
2.2 Кроссплатформенные языки программирования.....	11
2.3 Базы данных.....	14
2.4 Среды разработки.....	16
2.4 Обоснование выбора средств реализации.....	17
3 Практическая часть	20
3.1 Общая структура приложения.....	20
3.2 Проектирование и создание базы данных.....	22
3.3 Программный код проекта.....	24
3.4 Результат работы приложения.....	42
ЗАКЛЮЧЕНИЕ	50
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ.....	51

ВВЕДЕНИЕ

В наше время каждый человек использует несколько электронных гаджетов, которые стали незаменимыми помощниками в различных сферах нашей жизни. Смартфоны, планшеты, ноутбуки и умные часы помогают нам оставаться на связи, работать, учиться, развлекаться и следить за своим здоровьем. Благодаря быстрому развитию технологий, эти устройства становятся все более мощными, функциональными и доступными. Они позволяют нам экономить время, упрощают доступ к информации и открывают новые возможности для самореализации и общения.

Для современных компаний доступность и распространённость электронных гаджетов является определяющим средством привлечь новую клиентскую базу и удержать старую. Электронные гаджеты предоставляют клиентам компании разнообразный и удобный способ взаимодействия с продуктами и услугами компании. Они могут обращаться к ним из любой точки мира и в любое время, что делает процесс использования продуктов более эффективным. Электронные гаджеты позволяют компаниям устанавливать более тесные связи с клиентами через приложения. Это способствует к лучшему взаимопониманию и удовлетворению потребностей клиентов.

Есть большая проблема для распространения приложения на разные электронные гаджеты. Каждый новый производитель гаджета ставит новое программное обеспечение, реализованное на определённом языке программирования. Чтобы приложение работало на нём приходится писать на том же языке программирования. Компании приходится нанимать больше программистов, что приводит к большим временным и денежным затратам. Из-за этих затрат уменьшается зарплата разработчика. Для решения этой проблемы многие компании переходят на кроссплатформенные языки.

Актуальность кроссплатформенных приложений для бизнеса не вызывает сомнений, а востребованность обусловлена простотой и удобством использования, ведь одной из ваших электронных устройств всегда рядом с вами.

Цель данной работы заключается в разработки кроссплатформенного приложения “Витрина объявлений” для предоставления и размещения пользователям услуг и товаров.

Для достижения поставленной цели необходимо выполнить следующие задачи:

1. Осуществить теоретический анализ научной и методической литературы по актуальности кроссплатформенных приложений;
2. Осуществить теоретический анализ научной и методической литературы по актуальности использования баз данных;

3. Произвести анализ средств реализации проекта;
4. Освоить основные компоненты и средства построения интерфейса среды программирования Flutter;
5. Реализовать базу данных и взаимодействие с ней
6. Спроектировать интерфейс программы и порядок ее взаимодействия с пользователем;
7. Разработать приложение, в котором пользователь сможет разместить свою услугу или воспользоваться чей-то.

1. ХАРАКТЕРИСТИКА ПРЕДМЕТНОЙ ОБЛАСТИ

1.1 Актуальность кроссплатформенных приложений

Растущая популярность и разнообразие электронных гаджетов на рынке создает серьезные проблемы для разработчиков программного обеспечения. Необходимость адаптации приложений под множество различных операционных систем, устройств и экранов приводит к значительным временным и финансовым затратам. Это становится серьезным препятствием для компаний, стремящихся охватить широкую аудиторию пользователей.

Кроссплатформенные приложения являются ключевым решением этой проблемы. Они позволяют создавать единое программное обеспечение, которое может работать на различных платформах без необходимости разработки отдельных версий. Это существенно упрощает и ускоряет процесс разработки, тестирования и развертывания приложений.

Одним из основных преимуществ кроссплатформенных решений является их универсальность и гибкость. Разработчики могут использовать единый код базы, что позволяет им быстро адаптировать приложение под новые устройства и операционные системы. Это дает возможность компаниям быстро выходить на рынок с обновленными версиями продуктов и оперативно реагировать на изменения в технологической среде.

Кроме того, кроссплатформенные приложения обеспечивают более последовательный и унифицированный пользовательский опыт. Независимо от используемого устройства, пользователи получают доступ к одинаковому функционалу и интерфейсу, что способствует повышению их лояльности и удовлетворенности.

Важным фактором является также снижение затрат на разработку и поддержку. Вместо необходимости нанимать отдельные команды программистов для каждой платформы, компании могут сосредоточиться на создании единого кода, который может быть развернут на различных устройствах. Это позволяет значительно оптимизировать расходы и повысить эффективность разработки.

Кроссплатформенные технологии, такие как HTML5, React Native, Flutter и Xamarin, активно развиваются и совершенствуются. Они предоставляют разработчикам мощный инструментарий для создания высококачественных, функциональных и адаптивных приложений, которые могут работать на широком спектре устройств.

В условиях быстро меняющегося технологического ландшафта и растущего разнообразия электронных гаджетов, кроссплатформенные

приложения становятся все более актуальными и необходимыми для успешного развития бизнеса. Они позволяют компаниям оптимизировать ресурсы, повысить эффективность разработки, расширить свое присутствие на рынке и предоставить пользователям последовательный и качественный опыт взаимодействия.

На сегодняшний день популярность кроссплатформенных языков возрастает на рынке товаров и услуг. Вот рейтинг ТЮВЕ кроссплатформенных языков на май 2024 года [1].

May 2024	May 2023	Change	Programming Language		Ratings	Change
1	1			Python	16.33%	+2.88%
4	3	▼		Java	8.69%	-3.53%
8	12	▲		Go	1.60%	+0.61%
26				Dart	0.68%	
5	5			C#	6.49%	-0.94%

Рисунок 1.1 – Рейтинг кроссплатформенных языков

1.2 Актуальность баз данных

Базы данных остаются неразрывной частью современного информационного общества и играют ключевую роль в различных областях, от бизнеса до науки и государственного управления. В контексте современных технологических тенденций и растущего объема данных, актуальность баз данных только увеличивается.

В современном мире организации и предприятия обрабатывают огромные объемы данных, которые требуют хранения, управления и анализа. Базы данных обеспечивают эффективное хранение и структурирование информации, что позволяет организациям быстро извлекать нужные данные для принятия бизнес-решений и управления процессами.

С развитием Интернета вещей (IoT), облачных технологий и мобильных приложений количество генерируемых данных продолжает расти. Базы данных становятся основой для сбора, хранения и анализа этих данных, обеспечивая возможность создания интеллектуальных систем, управления ресурсами и повышения эффективности бизнес-процессов.

В научных исследованиях и медицинской диагностике базы данных играют ключевую роль в хранении и анализе больших объемов данных, что позволяет исследователям и врачам извлекать ценную информацию, выявлять закономерности и разрабатывать новые методы лечения и диагностики.

Государственные учреждения также активно используют базы данных для хранения и обработки информации о населении, экономике, здравоохранении и других сферах, обеспечивая эффективное управление и принятие стратегических решений.

Кроме того, базы данных лежат в основе персонализации пользовательского опыта на веб-страницах. Они позволяют отслеживать предпочтения, поведение и историю взаимодействия каждого посетителя, чтобы предоставлять ему контент, рекомендации и функциональность, максимально соответствующие его интересам. Это повышает вовлеченность, лояльность и конверсию на веб-странице, делая ее более привлекательной и ценной для пользователя.

Не менее важна роль баз данных в обеспечении интеграции веб-страниц с другими системами и приложениями. Будь то синхронизация с учетными записями, обмен данными с мобильными приложениями, связь с внешними сервисами или любые другие виды интеграции - все это невозможно без надежного хранилища данных, доступного через API. Такая интеграция позволяет расширять функциональные возможности веб-страниц, делая их более полезными и удобными для пользователей.

Также, базы данных играют ключевую роль в обеспечении безопасности и целостности данных на веб-страницах. Они предоставляют механизмы контроля доступа, резервного копирования, восстановления и защиты от несанкционированных изменений, что критически важно для сохранности конфиденциальной информации и корректного функционирования веб-ресурсов.

Таким образом, базы данных остаются незаменимым инструментом для организации, анализа и управления данными в современном мире, и их актуальность будет продолжать возрастать в условиях динамичного развития информационных технологий и роста объемов данных.

На сегодняшний день список баз данных разнообразный и богатый. Вот рейтинг TOPDB на июнь 2024 годах [\[2\]](#).

Rank	Change	Database	Share	1-year trend
1		Oracle	28.74 %	+2.0 %
2		MySQL	16.91 %	-1.4 %
3		SQL Server	11.72 %	-0.6 %
4	↑	PostgreSQL	7.09 %	+0.5 %
5	↑	MongoDB	6.05 %	-0.0 %
6	↓↓	Microsoft Access	6.0 %	-1.1 %
7		Firebase	4.96 %	-0.1 %
8		Redis	3.43 %	+0.5 %
9		Splunk	2.53 %	+0.2 %
10	↑	SQLite	2.09 %	+0.2 %
11	↓	Elasticsearch	1.94 %	-0.1 %
12		MariaDB	1.47 %	+0.1 %
13	↑↑↑	SAP HANA	1.13 %	+0.2 %
14	↓	DynamoDB	1.11 %	-0.1 %
15		DB2	1.0 %	+0.0 %
16	↓↓	Apache Hive	0.76 %	-0.2 %
17		Neo4j	0.54 %	+0.1 %
18		FileMaker	0.45 %	+0.0 %
19		Solr	0.45 %	+0.0 %
20	↑	Firebird	0.28 %	+0.0 %

Рисунок 1.2 – Рейтинг баз данных

2. ОБЗОР И ОБОСНОВАНИЕ ВЫБОРА СРЕДСТВ РЕАЛИЗАЦИИ ПРОЕКТА

2.1 Платформы

2.1.1 Android

Платформа Android была создана 23 сентября 2008 году компанией Google для мобильных устройств. Данная система используется не только в смартфонах, но и в планшетах, телевизорах, умных часах, навигаторах. Платформа Android включает в себя операционную систему и инструменты разработки. Операционная система основана на ядре Linux и собственной реализации виртуальной машины Java компании Google.

Android имеет большую популярность в мире, благодаря своей открытой архитектуре и возможности изменения, по сравнению с iOS. Пользователю предоставляется огромный выбор приложений, которые легко установить с интернета или официальных магазинов приложений для Android (Google Play, AppGallery).

Android - прекрасная платформа для разработчиков приложений. Это связано с тем, что Android имеет большую долю рынка, около 70% на 2024 год, среди мобильных устройств и таким образом предоставляет огромную базу потенциальных клиентов для разработанного приложения [3].

Android предоставляет разработчикам доступ ко множеству API и библиотек, которые сильно облегчают создание приложений. Также разработчики могут использовать различные инструменты и языки программирования для создания приложений для устройств на базе Android, такие как Java, Kotlin, XML, Dart. Открытость Android позволяет разработчикам создавать обширные коллекции приложений, тестировать их на разных устройствах и, впоследствии, публиковать в Google Play.

Система Android является одной из самых популярных и широко используемых операционных систем для мобильных устройств в мире.

2.1.2 iOS

iOS - это мобильная операционная система, разработанная компанией Apple исключительно для своих устройств, таких как iPhone, iPad и iPod Touch. Её популярность обусловлена рядом факторов. Во-первых, iOS отличается интуитивно понятным и простым в использовании интерфейсом, что делает её доступной для широкого круга пользователей. Во-вторых, система работает плавно и быстро благодаря оптимизации под конкретное "железо" устройств

Apple. В-третьих, App Store предлагает огромный выбор качественных приложений, многие из которых появляются на iOS раньше, чем на других платформах. Кроме того, пользователи ценят регулярные обновления системы с новыми функциями и улучшениями безопасности, а также тесную интеграцию с другими устройствами экосистемы Apple, такими как Mac и Apple Watch. Немаловажную роль играет и приверженность компании защите конфиденциальности пользователей.

Однако iOS имеет и ряд отличий от других мобильных ОС, в частности от Android. Прежде всего, это закрытая экосистема, в которой Apple полностью контролирует как аппаратную, так и программную часть, в то время как Android является открытой системой, используемой многими производителями устройств. Из-за этого iOS предоставляет более ограниченные возможности кастомизации по сравнению с Android. Также в устройствах на базе iOS отсутствует слот для карты памяти и ограничено взаимодействие с файловой системой. Интерфейс и принципы взаимодействия в iOS также отличаются от Android, например, отсутствием привычной кнопки "Назад". Кроме того, приложения для iOS проходят более строгую проверку перед публикацией в App Store. С другой стороны, пользователи iOS традиционно получают новые версии ОС быстрее и могут рассчитывать на более долгую поддержку обновлениями своих устройств.

В целом, iOS привлекает пользователей своей простотой, стабильностью работы, качественной экосистемой приложений и сервисов, а также имиджем премиального продукта. Однако за это приходится платить меньшей свободой и ограниченными возможностями настройки системы под свои нужды по сравнению с более открытой и гибкой платформой Android.

2.2 Кроссплатформенные языки программирования

2.2.1 Java

Java — один из самых популярных кроссплатформенных языков программирования. Основной принцип Java — "Написано однажды, работает везде" (Write Once, Run Anywhere). Это достигается благодаря тому, что Java-код компилируется в байт-код, который исполняется на Java Virtual Machine (JVM). JVM доступна для большинства операционных систем, что позволяет запускать Java-программы на различных платформах без необходимости их переработки.

JVM — это виртуальная машина, которая исполняет Java-байт-код. Она абстрагирует операционную систему и аппаратное обеспечение, предоставляя единый уровень для исполнения Java-программ. Существуют реализации JVM

для различных операционных систем, включая Windows, macOS, Linux и другие.

Java предоставляет богатый набор стандартных библиотек и API, которые абстрагируют низкоуровневые операции, специфичные для конкретных операционных систем. Это упрощает разработку кроссплатформенных приложений, так как разработчики могут использовать стандартные средства для выполнения широкого спектра задач, не беспокоясь о совместимости.

Для случаев, когда необходимо взаимодействовать с специфическими функциями операционной системы, Java предлагает механизмы взаимодействия с нативным кодом, такие как Java Native Interface (JNI). Это позволяет разработчикам вызывать функции, написанные на других языках программирования, таких как C или C++, и тем самым расширять функциональность Java-программ.

Java поддерживает множество кроссплатформенных фреймворков и инструментов, таких как:

1. Spring Framework — используется для разработки серверных приложений.
2. Apache Hadoop — фреймворк для работы с большими данными.
3. Android SDK — основной инструмент для разработки приложений под Android.
4. JavaFX — для создания кроссплатформенных графических интерфейсов.

Java является хорошим выбором для кроссплатформенной разработки.

2.2.2 Flutter/Dart

Dart — это язык программирования, разработанный Google, который используется для разработки кроссплатформенных приложений. Он разработан Google и имеет следующие особенности:

1. Объектно-ориентированный: Dart - это объектно-ориентированный язык, поддерживающий классы, интерфейсы, наследование и другие ООП концепции
2. Строгая типизация: Dart использует строгую статическую типизацию, что помогает обнаруживать ошибки на этапе компиляции и повышает надежность кода.
3. Простой синтаксис: Синтаксис Dart похож на синтаксис других популярных языков, таких как Java и C++, что облегчает его изучение для разработчиков, знакомых с этими языками.
4. Поддержка асинхронного программирования: Dart предоставляет встроенную поддержку асинхронного программирования с помощью ключевых слов `async` и `await`, а также классов `Future` и `Stream`.

5. Компиляция: Dart может компилироваться как в нативный код (например, при использовании Flutter), так и в JavaScript (для веб-разработки).

Flutter — это UI-фреймворк, также разработанный Google, который использует Dart для создания кроссплатформенных приложений. Его основные особенности включают:

1. Единая кодовая база: С помощью Flutter можно создавать приложения для iOS и Android, используя один и тот же код. Это значительно сокращает время и усилия, необходимые для разработки и поддержки приложений на нескольких платформах.
2. Быстрая разработка: Flutter предоставляет возможность горячей перезагрузки (hot reload), что позволяет вносить изменения в код и сразу видеть результат в приложении без необходимости его полной перекомпиляции.
3. Богатый набор виджетов: Flutter включает в себя обширную библиотеку готовых виджетов (визуальных компонентов), которые можно использовать для создания пользовательского интерфейса. Эти виджеты адаптируются под дизайн платформы (iOS или Android), на которой запущено приложение.
4. Высокая производительность: Приложения, созданные на Flutter, компилируются в нативный код, что обеспечивает высокую производительность, сравнимую с нативными приложениями.
5. Поддержка различных IDE: Flutter поддерживается популярными интегрированными средами разработки (IDE), такими как Android Studio, Visual Studio Code и IntelliJ IDEA.
6. Сообщество и экосистема: Flutter имеет активное сообщество и богатую экосистему пакетов и плагинов, что облегчает интеграцию с различными сервисами и API.

Flutter в отличие от других UI-фреймворков имеет собственный рендеринг. Используется собственный графический движок Skia для рендеринга интерфейса, что позволяет ему обеспечивать высокую производительность и плавную анимацию.

Flutter выделяется среди других кроссплатформенных фреймворков благодаря высокопроизводительному рендерингу, собственному набору виджетов, единообразному интерфейсу на всех платформах и поддержке от Google. Это делает его отличным выбором для создания сложных и высококачественных кроссплатформенных приложений.

2.3 Базы данных

2.3.1 Реляционные базы данных

Реляционные базы данных (РБД) являются одной из наиболее широко используемых моделей управления данными. Они основаны на реляционной модели данных, предложенной Эдгаром Коддом в 1970 году. Реляционные базы данных хранят данные в виде таблиц (также называемых отношениями), что обеспечивает структурированное и организованное представление данных.

Основные понятия реляционных баз данных:

1. Таблица - представляет собой набор строк и столбцов, где каждая строка (называемая записью или кортежем) представляет отдельный объект или экземпляр данных, а каждый столбец (называемый атрибутом) представляет тип данных, присущий всем записям.
2. Строка (или запись) - представляет один экземпляр объекта данных. Например, в таблице "Пользователи" строка может содержать информацию о конкретном пользователе.
3. Столбец (или атрибут) - определяет тип данных для каждого объекта в таблице. Например, в таблице "Пользователи" могут быть столбцы "ID", "Имя", "Электронная почта" и т. д.
4. Первичный ключ — это уникальный идентификатор для каждой записи в таблице. Он гарантирует, что каждая строка может быть однозначно идентифицирована. Примером первичного ключа может быть "ID пользователя".
5. Внешний ключ - используется для установления связи между таблицами. Он ссылается на первичный ключ другой таблицы, создавая связь между данными.

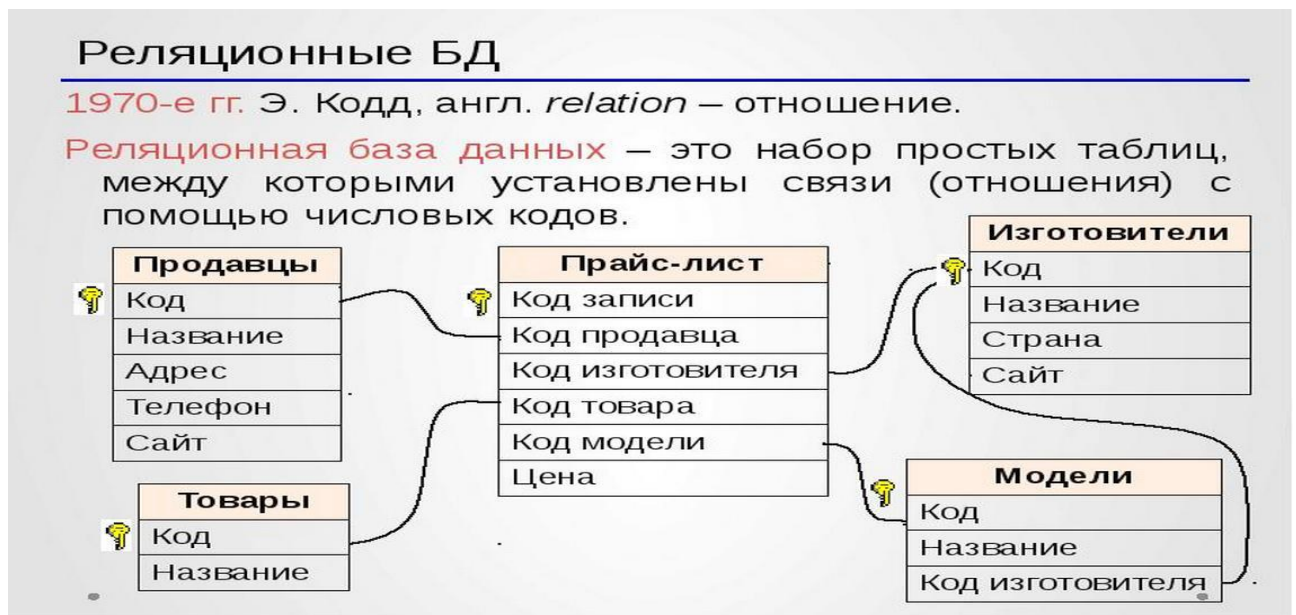


Рисунок 2.1 – Пример реляционной базы данных

Реляционные базы данных остаются важным инструментом для хранения, организации и управления структурированными данными. Их преимущества включают структурированность, целостность данных, мощный язык запросов и гибкость. Разработка и управление реляционными базами данных требуют понимания основ их работы, а также использования соответствующих инструментов и технологий.

2.3.2 Нереляционные базы данных

Нереляционные базы данных, также известные как NoSQL базы данных, предлагают альтернативные подходы к хранению и управлению данными, отличные от традиционных реляционных баз данных. Эти системы были разработаны для решения проблем масштабируемости и производительности, с которыми сталкиваются реляционные базы данных при работе с большими объемами данных или высоконагруженными приложениями.

Нереляционные базы данных бывают несколько типов:

1. Документные базы данных (Document Databases) - документные базы данных хранят данные в виде документов, обычно в формате JSON или BSON. Каждый документ может иметь свою структуру, что обеспечивает гибкость. Такие базы данных используются для хранения сложных, иерархических данных, таких как пользовательские профили, каталоги товаров и контент-менеджмент системы. Примером таких баз данных является MongoDB, CouchDB, Firebase Cloud Firestore.
2. Колонно-ориентированные базы данных (Column-family Stores) - данные хранятся в виде столбцов, сгруппированных по семействам столбцов. Это позволяет эффективно обрабатывать запросы, затрагивающие только небольшое количество столбцов. Такие базы данных подходят для аналитических приложений, обработки больших объемов данных и временных рядов.
3. Ключ-значение хранилища (Key-Value Stores) - данные хранятся в виде пар "ключ-значение". Это позволяет быстро получать доступ к данным по ключу. Такие базы данных используются для кэширования, управления сессиями, простых хранилищ данных и приложений с низкой задержкой. Примером таких баз данных является Redis, DynamoDB, Riak.
4. Графовые базы данных (Graph Databases) - данные представлены в виде графов с вершинами (узлами) и ребрами (связями). Это позволяет эффективно моделировать и выполнять запросы на графы. Такие базы данных подходят для социальных сетей, рекомендационных систем, управления правами доступа и любых приложений, где важны связи между объектами. Примером таких баз данных является Neo4j, ArangoDB

У нереляционных баз данных есть несколько преимуществ, которые отличает их от реляционных баз данных:

- Нереляционные базы данных не требуют фиксированной схемы, что позволяет легко изменять структуру данных.
- Нереляционные базы данных лучше масштабируются горизонтально (добавлением новых серверов), что делает их подходящими для работы с большими объемами данных и высокими нагрузками.
- Нереляционные базы данных часто обеспечивают высокую производительность и низкую задержку для определенных типов операций и рабочих нагрузок.
- Нереляционные базы данных могут эффективно работать с различными типами данных, такими как документы, графы и ключ-значение.

2.4 Среды разработки

2.4.1 Android Studio

Android Studio - это официальная интегрированная среда разработки (IDE) для создания приложений под управлением операционной системы Android. Она была разработана компанией Google на основе популярного IDE IntelliJ IDEA, и это наиболее используемый инструмент для Android-разработки.

Android Studio включает в себя инструменты, библиотеки и документацию для создания приложений для Android на языке программирования Java и Kotlin. IDE. К преимуществам данной среды разработки можно отнести:

- Расширенный редактор кода. Android Studio обеспечивает функции автодополнения кода, форматирования, рефакторинга и анализа кода, что упрощает процесс написания кода.
- Удобный интерфейс для разработки. IDE предоставляет доступ к значительному количеству инструментов и визуализаторов, которые помогают разработчику создавать отзывчивые и красивые интерфейсы.
- Интеграцию с Google Cloud и другими сервисами. Android Studio позволяет легко интегрировать приложение с сервисами Google Cloud, Firebase и другими платформами для быстрого создания и развертывания своего приложения.
- Поддержку многих устройств. IDE обеспечивает поддержку большинства устройств Android, включая смартфоны, планшеты, телевизоры и другие устройства.
- Обновления. Android Studio регулярно обновляется, что обеспечивает своевременную поддержку новых функций OS Android и новых возможностей Google.

Android Studio является бесплатной средой разработки, доступной для загрузки с официального сайта разработчика. Она используется многими разработчиками во всем мире для создания высококачественных приложений для Android.

2.4.2 Unity

Unity - это кросс-платформенная интегрированная среда разработки (IDE), используемая для создания игр и других интерактивных приложений для различных платформ, таких как iOS, Android, Windows, Mac, Xbox, PlayStation и многих других.

Unity вышла в 2005 году, и с тех пор стала одним из самых популярных инструментов для разработки игр. Она использует язык программирования C# и предоставляет много инструментов для создания реалистичных трехмерных игровых миров, анимации, физики, звука, света и других эффектов. Главные преимущества и функции среды разработки Unity:

- Графический интерфейс для создания игровых объектов, сцен и других элементов игры.
- Мощный движок рендеринга Unity, который позволяет создавать графику высокого качества.
- Инструменты для создания анимации и моделирования объектов.
- Инструменты для создания звуковых эффектов и музыки.
- Наличие готовых библиотек для различных задач и типов игр.
- Unity также предоставляет возможность создавать собственные инструменты и добавлять функциональность с помощью установок различных плагинов.

Unity имеет множество преимуществ, таких как быстрое проектирование и прототипирование, высокая производительность, удобство использования, поддержка многих платформ и готовые шаблоны игр.

Версия Unity доступна бесплатно на официальном сайте, однако для доступа ко всем функциональным возможностям требуется покупка лицензии или подписки.

2.4 Обоснование выбора средств реализации

2.4.1 Обоснования выбора языка программирования

Наиболее удобным языком программирования для создания кроссплатформенных приложений является Dart в связке UI-фреймворком Flutter. Это относительно простой в изучении и понимании язык программирования. Он имеет схожий синтаксис с другими языками программирования, такими как C++ или Java.

Flutter нам позволит легко создавать графический интерфейс для всех платформ с помощью визуальных объектов (виджетов). Dart нам позволит писать асинхронные методы для быстрой работоспособности приложения. Также Dart имеет в себе обширную библиотеку, содержащую большое количество полезных классов и методов, которые могут значительно упростить процесс разработки [4].

2.4.2 Обоснования выбора базы данных

Для моего проекта лучше всего подойдёт документная нереляционная база данных, а именно Firebase Cloud Firestore. У неё есть ряд преимуществ:

- **Гибкость:** Модель данных Cloud Firestore поддерживает гибкие иерархические структуры данных. Храните свои данные в документах, организованных в коллекции. Помимо подколлекций документы могут содержать сложные вложенные объекты.
- **Выразительные запросы:** В Cloud Firestore вы можете использовать запросы для получения отдельных, конкретных документов или для получения всех документов в коллекции, соответствующих параметрам вашего запроса. Ваши запросы могут включать в себя несколько связанных фильтров и сочетать фильтрацию и сортировку. Они также индексируются по умолчанию, поэтому производительность запросов пропорциональна размеру набора результатов, а не набора данных.
- **Обновления в реальном времени:** Cloud Firestore использует синхронизацию данных для обновления данных на любом подключенном устройстве. Однако он также предназначен для эффективного выполнения простых одноразовых запросов.
- **Оффлайн поддержка:** Cloud Firestore кэширует данные, которые активно использует ваше приложение, поэтому приложение может записывать, читать, прослушивать и запрашивать данные, даже если устройство находится в автономном режиме. Когда устройство снова подключается к сети, Cloud Firestore синхронизирует все локальные изменения обратно в Cloud Firestore.

Firebase Cloud Firestore бесплатный сервис, однако для доступа ко всем функциональным возможностям требуется покупка подписки.

2.4.3 Обоснования выбора среды разработки

Так как мы делаем кроссплатформенный проект на Flutter/Dart то выбираем среду разработки Android Studio. Кроме того, что она поддерживает данный язык, на ней можно установить эмуляторы разных платформ, которые помогут протестировать приложения во время разработки. Также Android Studio легко подключается к сервисам Firebase для добавления функционала.

Помимо вышеперечисленных преимуществ, среда разработки Android Studio, имеет и другие качества:

- Интеллектуальный редактор кода: Предоставляет расширенное автодополнение, рефакторинг и анализ кода.
- Редактор макетов: Визуальный редактор для проектирования макетов приложений с функцией перетаскивания.
- Анализатор APK: Инструмент для анализа содержимого APK и размера вашего приложения.
- Гибкая система сборки: Основана на Gradle и позволяет настраивать сборки.
- Профайлеры в реальном времени: Инструменты для мониторинга использования процессора, памяти и сети вашим приложением.

3 Практическая часть

Разработка кроссплатформенного приложения происходит в три этапа:

1. Разработка структуры и определение функций приложения;
2. Проектирование интерфейса кроссплатформенного приложения;
3. Тестирование приложения и регистрация результатов его работы.

3.1 Общая структура приложения

При разработке кроссплатформенного приложения в первую очередь необходимо создать структуру приложения, чтобы определить количество экранов и виджетов на них, а также сам функционал разрабатываемого приложения.

Для создания структуры кроссплатформенного приложения следует сделать следующие шаги:

1. Определить цели и задачи приложения. Необходимо определить, какую проблему должно решать приложение и какой функционал должен быть включён в приложение. В моём случае, это предоставления сервиса в котором каждый пользователь может разместить свою услугу или воспользоваться чужой.
2. Создать структуру экранов. Это необходимо, чтобы понимать, какой функционал и какие элементы будут присутствовать на каждом экране.
3. Создать интерфейс. Необходимо определить внешний вид приложения, оформить виджеты и прочее.
4. Определить количество экранов. Количество экранов зависит от целей и задач приложения, их можно определить на основе структуры экранов и внешнего вида.
5. Разработать функционал приложения. Разработать функционал на основе целей и задач.

В результате всех перечисленных выше этапов создается структура кроссплатформенного приложения, которая позволяет более эффективно разрабатывать и тестировать приложение.

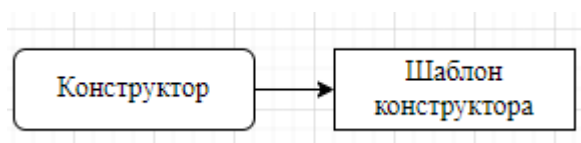


Рисунок 3.1- Структура экрана «Конструктор»

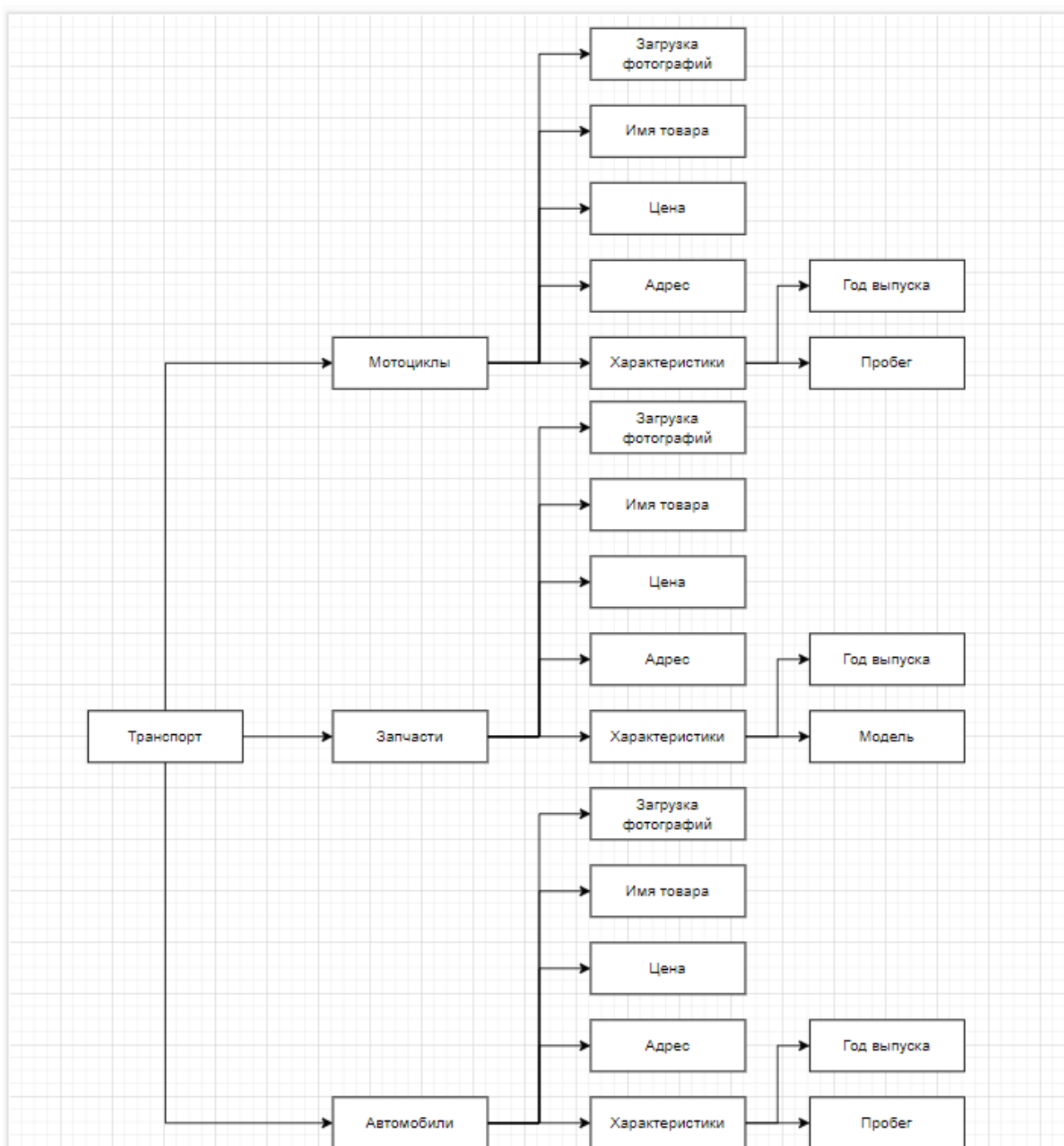


Рисунок 3.2 – Структура шаблона конструктора «Транспорт»

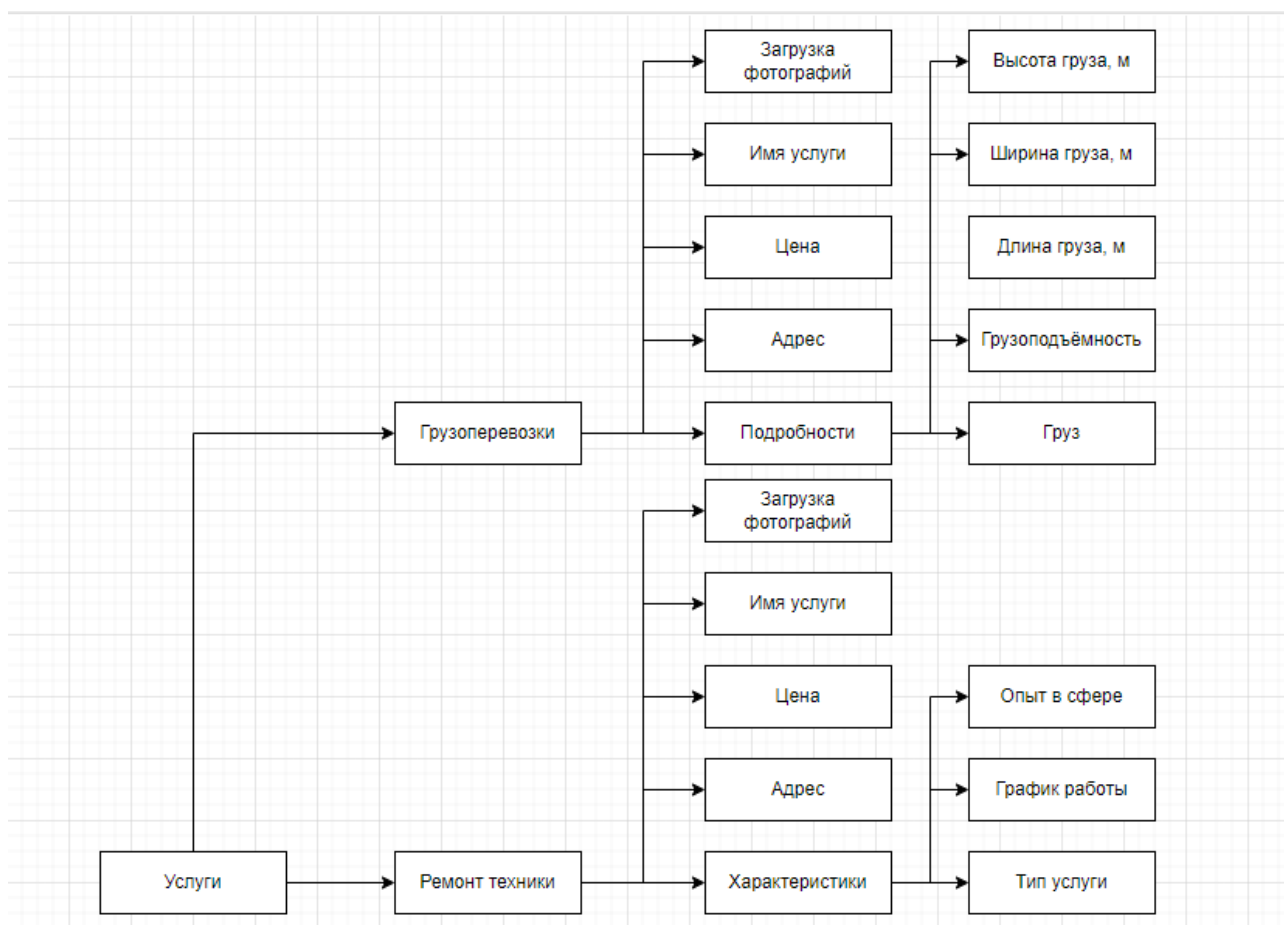


Рисунок 3.3 – Структура шаблона конструктора «Услуги»

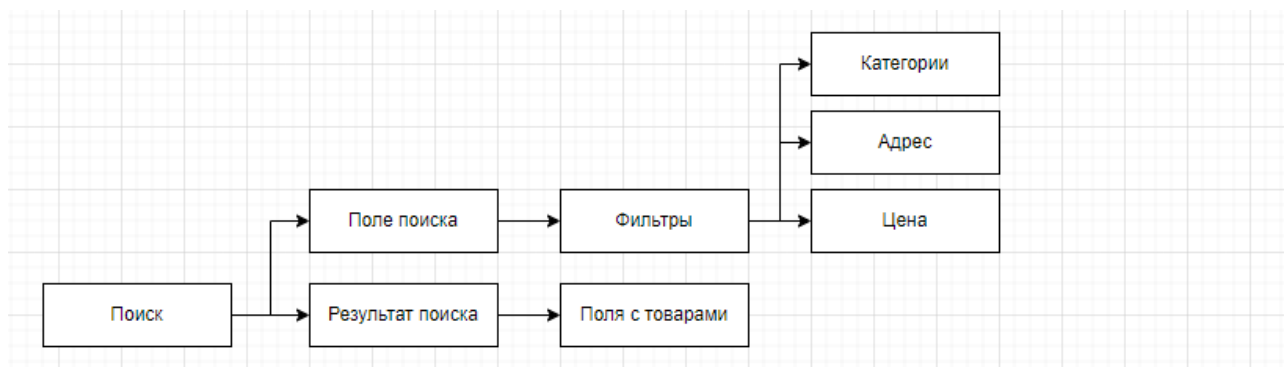


Рисунок 3.4 – Структура экрана «Поиск»

3.2 Проектирование и создание базы данных

Для хранения и обработки всей пользовательской информации понадобится База данных. В ней будет несколько коллекций:

1. “Ads” – В этой коллекции хранятся все созданные объявления для дальнейшей обработки. Так как это нереляционная база данных то в ней названия и типы данных полей документов могут отличаться друг от друга, зависимости от шаблона конструктора.

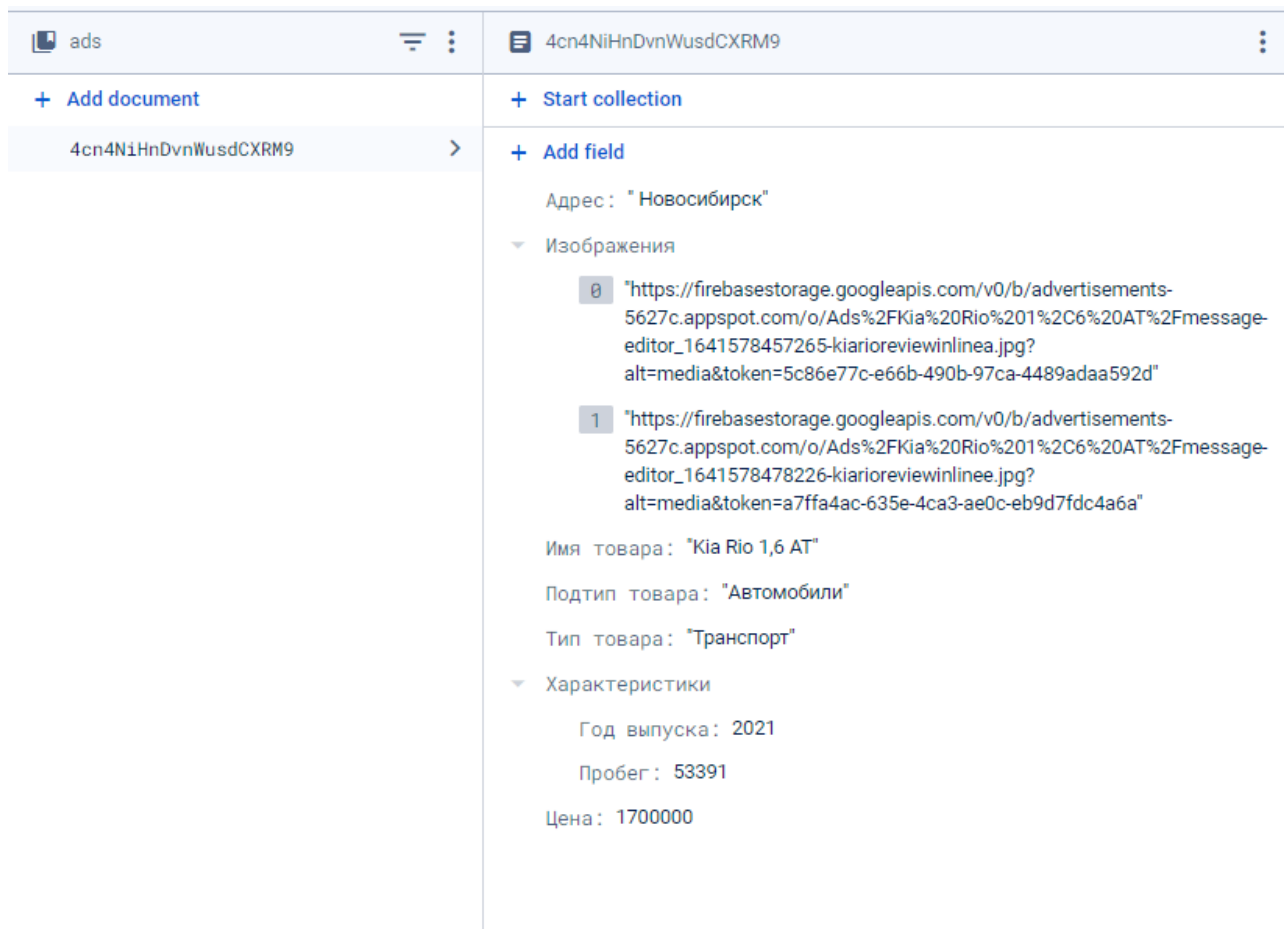


Рисунок 3.5 – Коллекция “Ads”

2. Users – В этой коллекции будут храниться данные о пользователях

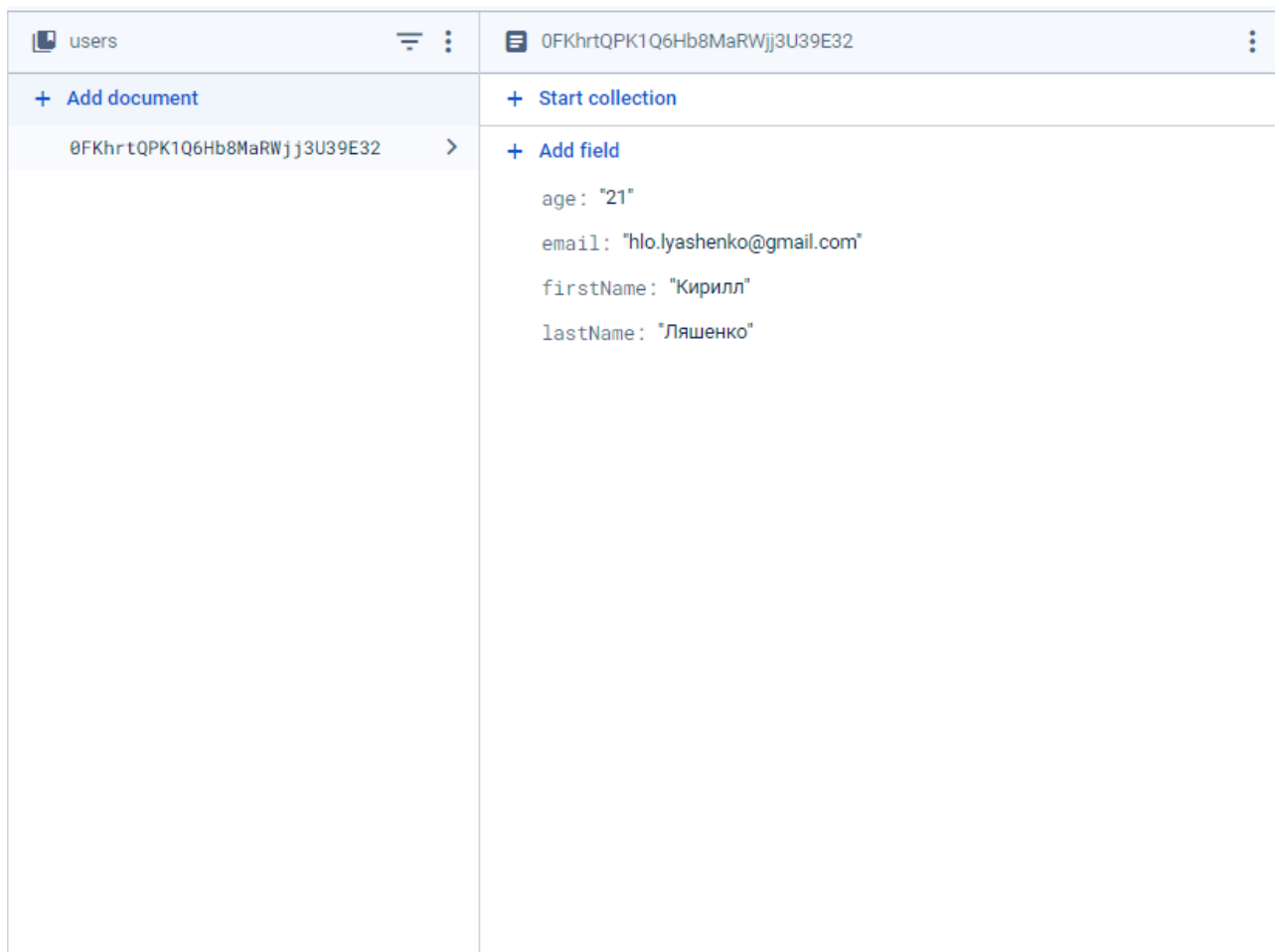


Рисунок 3.6 – Коллекция “Users”

3.3 Программный код проекта

При разработки приложения было создано несколько экранов.

1. Конструктор – экран для создания объявления доступный только зарегистрировавшимся пользователям. Он состоит из списка виджетов:
 - `DropDownButton` – кнопок с всплывающим списком для выбора типа и подтипа объявления. В зависимости от выбора появляются определённые виджеты.
 - `TextField` – текстовые поля для заполнения информации о объявлении (“Имя товара”, “Цена”, и т.д.).
 - `Container` – прожимаемые виджеты для загрузки изображения с устройства и отображения их. Также к ним привязаны маленькие иконки в виде крестика для удаления изображения.
 - `ElevatedButton` – кнопка завершающая создание объявления

Листинг 3.1 – `constructorAds.dart`

```
import 'dart:io';
import 'firebase.dart';
import 'package:cloud_firestore/cloud_firestore.dart';
import 'package:firebase_storage/firebase_storage.dart';
```

```

import 'package:flutter/material.dart';
import 'package:image_picker/image_picker.dart';
import 'package:path/path.dart' as Path;

class FirestoreDataWidget extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text('Firestore Data'),
      ),
      body: FirestoreDataList(),
    );
  }
}

class FirestoreDataList extends StatefulWidget {
  @override
  State<FirestoreDataList> createState() => _FirestoreDataListState();
}

class _FirestoreDataListState extends State<FirestoreDataList> {
  String? selectedTag;
  String? selectedTransport;
  List<File> imageFiles = []; // Список для хранения файлов изображений
  List<String> uploadedImageUrls =
    []; // Список для хранения URL загруженных изображений

  // Controllers for text fields
  final TextEditingController _nameController = TextEditingController();
  final TextEditingController _priceController = TextEditingController();
  final TextEditingController _addressController = TextEditingController();
  final TextEditingController _yearController = TextEditingController();
  final TextEditingController _mileageController = TextEditingController();

  Future<File?> getImageFromUser() async {
    final pickedFile =
      await ImagePicker().pickImage(source: ImageSource.gallery);
    if (pickedFile != null) {
      return File(pickedFile.path);
    }
    return null;
  }

  Future<String?> uploadImage(File imageFile, String adsname) async {
    try {
      Reference storageReference = FirebaseStorage.instance
        .ref()
        .child('Ads/${adsname}/${Path.basename(imageFile.path)}');
      UploadTask uploadTask = storageReference.putFile(imageFile);
      TaskSnapshot taskSnapshot = await uploadTask.whenComplete(() {});
      String downloadURL = await taskSnapshot.ref.getDownloadURL();
      return downloadURL;
    } catch (e) {
      print('Error uploading image: $e');
      return null;
    }
  }

  Future<void> deleteImage(String imageUrl) async {
    setState(() {

```

```

        uploadedImageUrls.remove(imageUrl);
    });
}

Future<void> uploadAllImages(String adsname) async {
    List<String> newUploadedUrls = [];
    for (File imageFile in imageFiles) {
        String? imageUrl = await uploadImage(imageFile, adsname);
        if (imageUrl != null) {
            newUploadedUrls.add(imageUrl);
        }
    }
    setState(() {
        uploadedImageUrls.addAll(newUploadedUrls);
    });
}

@override
Widget build(BuildContext context) {
    return StreamBuilder<QuerySnapshot>(
        stream: FirebaseFirestore.instance.collection('tags').snapshots(),
        builder: (context, snapshot) {
            if (snapshot.connectionState == ConnectionState.waiting) {
                return Center(child: CircularProgressIndicator());
            }
            if (snapshot.hasError) {
                return Center(child: Text('Error: ${snapshot.error}'));
            }
            if (!snapshot.hasData || snapshot.data!.docs.isEmpty) {
                return Center(child: Text('No data found'));
            }

            final documents = snapshot.data!.docs;
            final tags = documents.map((doc) => doc.id).toList();

            if (selectedTag == null && tags.isNotEmpty) {
                selectedTag = tags[0];
            }

            final data = documents.firstWhere((doc) => doc.id == selectedTag).data()
                as Map<String, dynamic>;
            final List<dynamic> arrayData = data['Категории транспорта'] ?? [];

            if (selectedTransport == null && arrayData.isNotEmpty) {
                selectedTransport = arrayData[0].toString();
            }

            return ListView(
                children: [
                    Padding(
                        padding: const EdgeInsets.all(8.0),
                        child: DropdownButton<String>(
                            value: selectedTag,
                            onChanged: (String? newValue) {
                                setState(() {
                                    selectedTag = newValue;
                                    selectedTransport =
                                        null; // Reset transport when tag changes
                                });
                            },
                            items: tags.map<DropdownMenuItem<String>>((String value) {

```

```

        return DropdownMenuItem<String>(
            value: value,
            child: Text(value),
        );
    }).toList(),
),
),
if (arrayData.isNotEmpty)
    Padding(
        padding: const EdgeInsets.all(8.0),
        child: DropdownButton<String>(
            value: selectedTransport,
            onChanged: (String? newValue) {
                setState(() {
                    selectedTransport = newValue;
                });
            },
            items:
                arrayData.map<DropdownMenuItem<String>>((dynamic value) {
                    return DropdownMenuItem<String>(
                        value: value.toString(),
                        child: Text(value.toString()),
                    );
                }).toList(),
        ),
    ),
if (selectedTag == 'Транспорт' &&
    selectedTransport == 'Автомобили') ...[
    Padding(
        padding: const EdgeInsets.all(8.0),
        child: TextField(
            controller: _nameController,
            decoration: InputDecoration(
                labelText: 'Имя товара',
            ),
        ),
    ),
    Padding(
        padding: const EdgeInsets.all(8.0),
        child: TextField(
            controller: _priceController,
            decoration: InputDecoration(
                labelText: 'Цена',
            ),
            keyboardType: TextInputType.number,
        ),
    ),
    Padding(
        padding: const EdgeInsets.all(8.0),
        child: TextField(
            controller: _addressController,
            decoration: InputDecoration(
                labelText: 'Адрес',
            ),
        ),
    ),
    Padding(
        padding: const EdgeInsets.all(8.0),
        child: Text('Характеристики'),
    ),
    Padding(

```


- `Future<void> addAdsFirestore() async` – создание документа и загрузка в него информацию об объявлении
- 2. `AuthScreen` – экран для аутентификация пользователя с двумя кнопками (Регистрация и логин)
- 3. `RegisterScreen` – экран для регистрации пользователя с помощью электронной почты. Он состоит списка виджетов:
 - `TextField` – 5 текстовых полей обязательных для заполнения (Email, Password, First Name, Last Name, Age).
 - `ElevatedButton` – кнопка для завершения регистрации.
- 4. `LoginScreen` – экран для авторизации пользователя. Он состоит списка виджетов:
 - `TextField` – 2 текстовых поля обязательных для заполнения (Email, Password)
 - `ElevatedButton` – кнопка для завершения авторизации.
- 5. `ProfileScreen` – экран для отображения информации о авторизированном пользователе. Он состоит списка виджетов:
 - `Text` – 3 строки в которых показывается информация о текущем пользователе

Листинг 3.2 profile.dart

```
import 'package:advertisements/main.dart';
import 'package:firebase_auth/firebase_auth.dart';
import 'package:cloud_firestore/cloud_firestore.dart';
import 'package:flutter/material.dart';

class AuthScreen extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text('Authentication'),
      ),
      body: Center(
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: [
            ElevatedButton(
              onPressed: () {
                Navigator.push(
                  context,
                  MaterialPageRoute(builder: (context) => LoginScreen()),
                );
              },
              child: Text('Login'),
            ),
            ElevatedButton(
              onPressed: () {
                Navigator.push(
                  context,
                  MaterialPageRoute(builder: (context) => RegisterScreen()),
                );
              },
            ),
          ],
        ),
      ),
    );
  }
}
```



```

        child: Text('Register'),
      ),
    ],
  ),
);
}
}

class LoginScreen extends StatefulWidget {
  @override
  _LoginScreenState createState() => _LoginScreenState();
}

class _LoginScreenState extends State<LoginScreen> {
  final FirebaseAuth _auth = FirebaseAuth.instance;
  final GlobalKey<FormState> _formKey = GlobalKey<FormState>();
  String _email = '';
  String _password = '';

  Future<void> _login() async {
    if (_formKey.currentState!.validate()) {
      _formKey.currentState!.save();
      try {
        await _auth.signInWithEmailAndPassword(email: _email, password:
        _password);
        ScaffoldMessenger.of(context).showSnackBar(SnackBar(content:
        Text('Logged in successfully')));
        Navigator.pushAndRemoveUntil(
          context,
          MaterialPageRoute(builder: (context) => MyHomePage()),
          (Route<dynamic> route) => false,
        );
      } catch (e) {
        ScaffoldMessenger.of(context).showSnackBar(SnackBar(content: Text('Login
        failed: $e')));
      }
    }
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text('Login'),
      ),
      body: Padding(
        padding: EdgeInsets.all(16.0),
        child: Form(
          key: _formKey,
          child: Column(
            children: [
              TextFormField(
                decoration: InputDecoration(labelText: 'Email'),
                keyboardType: TextInputType.emailAddress,
                validator: (value) {
                  if (value!.isEmpty) {
                    return 'Please enter your email';
                  }
                  return null;
                },
              ),
            ],
          ),
        ),
      ),
    );
  }
}

```

```

        onSave: (value) {
          _email = value!;
        },
      ),
      TextFormField(
        decoration: InputDecoration(labelText: 'Password'),
        obscureText: true,
        validator: (value) {
          if (value!.isEmpty) {
            return 'Please enter your password';
          }
          return null;
        },
        onSave: (value) {
          _password = value!;
        },
      ),
      SizedBox(height: 20),
      ElevatedButton(
        onPressed: _login,
        child: Text('Login'),
      ),
    ],
  ),
),
);
}
}

class RegisterScreen extends StatefulWidget {
  @override
  _RegisterScreenState createState() => _RegisterScreenState();
}

class _RegisterScreenState extends State<RegisterScreen> {
  final FirebaseAuth _auth = FirebaseAuth.instance;
  final FirebaseFirestore _firestore = FirebaseFirestore.instance;
  final GlobalKey<FormState> _formKey = GlobalKey<FormState>();
  String _email = '';
  String _password = '';
  String _firstName = '';
  String _lastName = '';
  String _age = '';

  Future<void> _register() async {
    if (_formKey.currentState!.validate()) {
      _formKey.currentState!.save();
      try {
        UserCredential userCredential = await
        _auth.createUserWithEmailAndPassword(email: _email, password: _password);
        User? user = userCredential.user;

        await _firestore.collection('users').doc(user!.uid).set({
          'firstName': _firstName,
          'lastName': _lastName,
          'age': _age,
          'email': _email,
        });

        ScaffoldMessenger.of(context).showSnackBar(SnackBar(content:

```

```

Text('Registered successfully')));
    Navigator.pushAndRemoveUntil(
      context,
      MaterialPageRoute(builder: (context) => MyHomePage()),
      (Route<dynamic> route) => false,
    );
  } catch (e) {
    ScaffoldMessenger.of(context).showSnackBar(SnackBar(content:
Text('Registration failed: $e')));
  }
}

}

@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
      title: Text('Register'),
    ),
    body: Padding(
      padding: EdgeInsets.all(16.0),
      child: Form(
        key: _formKey,
        child: SingleChildScrollView(
          child: Column(
            children: [
              TextFormField(
                decoration: InputDecoration(labelText: 'Email'),
                keyboardType: TextInputType.emailAddress,
                validator: (value) {
                  if (value!.isEmpty) {
                    return 'Please enter your email';
                  }
                  return null;
                },
                onSave: (value) {
                  _email = value!;
                },
              ),
              TextFormField(
                decoration: InputDecoration(labelText: 'Password'),
                obscureText: true,
                validator: (value) {
                  if (value!.isEmpty) {
                    return 'Please enter your password';
                  }
                  return null;
                },
                onSave: (value) {
                  _password = value!;
                },
              ),
              TextFormField(
                decoration: InputDecoration(labelText: 'First Name'),
                validator: (value) {
                  if (value!.isEmpty) {
                    return 'Please enter your first name';
                  }
                  return null;
                },
                onSave: (value) {

```

```

        _firstName = value!;
      },
    ),
    TextFormField(
      decoration: InputDecoration(labelText: 'Last Name'),
      validator: (value) {
        if (value!.isEmpty) {
          return 'Please enter your last name';
        }
        return null;
      },
      onSave: (value) {
        _lastName = value!;
      },
    ),
    TextFormField(
      decoration: InputDecoration(labelText: 'Age'),
      keyboardType: TextInputType.number,
      validator: (value) {
        if (value!.isEmpty) {
          return 'Please enter your age';
        }
        return null;
      },
      onSave: (value) {
        _age = value!;
      },
    ),
    SizedBox(height: 20),
    ElevatedButton(
      onPressed: _register,
      child: Text('Register'),
    ),
  ),
),
),
),
),
),
);
}
}

class ProfilePage extends StatelessWidget {
  final FirebaseAuth _auth = FirebaseAuth.instance;
  final FirebaseFirestore _firestore = FirebaseFirestore.instance;

  Future<Map<String, dynamic>?> _getUserData() async {
    User? user = _auth.currentUser;
    if (user != null) {
      DocumentSnapshot doc = await
        _firestore.collection('users').doc(user.uid).get();
      return doc.data() as Map<String, dynamic>;
    }
    return null;
  }

  Future<void> _logout(BuildContext context) async {
    await _auth.signOut();
    Navigator.of(context).popUntil((route) => route.isFirst);
  }
}

```

```

@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
      title: Text('Profile Page'),
      actions: [
        IconButton(
          icon: Icon(Icons.exit_to_app),
          onPressed: () => _logout(context),
        ),
      ],
    ),
    body: FutureBuilder<Map<String, dynamic>?>(
      future: _getUserData(),
      builder: (context, snapshot) {
        if (snapshot.connectionState == ConnectionState.waiting) {
          return Center(child: CircularProgressIndicator());
        } else if (snapshot.hasError) {
          return Center(child: Text('Error: ${snapshot.error}'));
        } else if (!snapshot.hasData) {
          return Center(child: Text('No user data found'));
        } else {
          var userData = snapshot.data!;
          return Center(
            child: Padding(
              padding: const EdgeInsets.all(16.0),
              child: Column(
                mainAxisAlignment: MainAxisAlignment.center,
                children: [
                  Text(
                    'Name: ${userData['firstName']} ${userData['lastName']}',
                    style: TextStyle(fontSize: 24),
                  ),
                  SizedBox(height: 10),
                  Text(
                    'Age: ${userData['age']}',
                    style: TextStyle(fontSize: 24),
                  ),
                  SizedBox(height: 10),
                  Text(
                    'Email: ${userData['email']}',
                    style: TextStyle(fontSize: 24),
                  ),
                  SizedBox(height: 20),
                  ElevatedButton(
                    onPressed: () => _logout(context),
                    child: Text('Logout'),
                  ),
                ],
              ),
            ),
          );
        }
      },
    ),
  );
}

```

Опишем основные методы:

- `Future<void> _login() async` - асинхронный метод авторизации пользователя. Метод проверяет есть ли такой пользователь в базе `Firebase Authentication`. В случае удачи авторизирует его, давая доступ пользователю использовать экраны, предназначенные только авторизованным пользователям. В случае провала возвращает текст (`Login failed`)
 - `Future<void> _register() async` - асинхронный метод регистрации пользователя. Метод создаёт пользователя в базе `Firebase Authentication` и документ в коллекцию `profile` с информацией об этом пользователе.
 - `Future<void> _logout(BuildContext context) async` - асинхронный метод выхода из профиля.
 - `Future<Map<String, dynamic>?> _getUserData() async` - асинхронный метод берущий информацию из документа, с `id` текущего пользователя, коллекции `profile`.
6. `SearchScreen` – экран для поиска объявлений. Он состоит списка виджетов:
- `TextField` – текстовое поле для реализации поиска.
 - `Column` – 2 столбца для отображения `Container`.
 - `Container` – “кликабельный” виджет для отображения объявления. Он состоит из 1 `Image` и 3 `Text` под ним.
 - `Image` – изображение объявления.
 - `Text` – 3 строки с содержанием (Название объявление, цена, адрес)

Листинг 3.3 – `search.dart`

```
import 'package:firebase_auth/firebase_auth.dart';
import 'package:cloud_firestore/cloud_firestore.dart';
import 'package:flutter/material.dart';
import 'detailsAds.dart';

class SearchScreen extends StatefulWidget {
  @override
  _SearchScreenState createState() => _SearchScreenState();
}

class _SearchScreenState extends State<SearchScreen> {
  final TextEditingController _searchController = TextEditingController();
  final FirebaseFirestore _firestore = FirebaseFirestore.instance;
  String _searchQuery = '';

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: TextField(
          controller: _searchController,
          decoration: InputDecoration(
            hintText: 'Search',
            border: InputBorder.none,
```

```

        hintStyle: TextStyle(color: Colors.white54),
      ),
      style: TextStyle(color: Colors.white),
      onChanged: (value) {
        setState(() {
          _searchQuery = value;
        });
      },
    ),
    backgroundColor: Colors.blue,
  ),
  body: StreamBuilder<QuerySnapshot>(
    stream: _firestore.collection('ads').snapshots(),
    builder: (context, snapshot) {
      if (!snapshot.hasData) {
        return Center(child: CircularProgressIndicator());
      }

      var items = snapshot.data!.docs;
      var filteredItems = items.where((item) {
        var title = item['Имя товара'].toString().toLowerCase();
        return title.contains(_searchQuery.toLowerCase());
      }).toList();

      return GridView.builder(
        gridDelegate: SliverGridDelegateWithFixedCrossAxisCount(
          crossAxisCount: 2,
          childAspectRatio: 3 / 4,
          crossAxisSpacing: 10,
          mainAxisSpacing: 10,
        ),
        padding: EdgeInsets.all(10),
        itemCount: filteredItems.length,
        itemBuilder: (context, index) {
          var item = filteredItems[index];
          var imageUrl = item['Изображения'][0];
          var itemName = item['Имя товара'];
          var itemPrice = item['Цена'];
          var itemAddress = item['Адрес'];

          return GestureDetector(
            onTap: () {
              Navigator.push(
                context,
                MaterialPageRoute(
                  builder: (context) => DetailScreen(item: item),
                ),
              );
            },
          ),
          child: Container(
            decoration: BoxDecoration(
              borderRadius: BorderRadius.circular(10),
              border: Border.all(color: Colors.grey),
            ),
            child: Column(
              crossAxisAlignment: CrossAxisAlignment.start,
              children: [
                Expanded(
                  child: Container(
                    width: double.infinity,
                    decoration: BoxDecoration(

```


Листинг 3.4 detailsAsd

```
import 'package:cloud_firestore/cloud_firestore.dart';
import 'package:flutter/material.dart';

class DetailScreen extends StatelessWidget {
  final QueryDocumentSnapshot item;

  DetailScreen({required this.item});

  @override
  Widget build(BuildContext context) {
    var images = item['Изображения'] as List<dynamic>;
    var fields = item.data() as Map<String, dynamic>;

    List<Widget> buildFieldWidgets() {
      List<Widget> fieldWidgets = [];
      fields.forEach((key, value) {
        if (key != 'Изображения') {
          if (value is Map<String, dynamic>) {
            value.forEach((subKey, subValue) {
              fieldWidgets.add(
                Padding(
                  padding: const EdgeInsets.symmetric(vertical: 4.0),
                  child: Text(
                    '$subKey: $subValue',
                    style: TextStyle(fontSize: 18),
                  ),
                ),
              );
            });
          } else {
            fieldWidgets.add(
              Padding(
                padding: const EdgeInsets.symmetric(vertical: 4.0),
                child: Text(
                  '$key: $value',
                  style: TextStyle(fontSize: 18),
                ),
              ),
            );
          }
        }
      });
      return fieldWidgets;
    }

    return Scaffold(
      appBar: AppBar(
        title: Text(fields['Имя товара'] ?? 'Подробная информация'),
      ),
      body: SingleChildScrollView(
        child: Column(
          crossAxisAlignment: CrossAxisAlignment.start,
          children: [
            Container(
              height: 300,
              child: PageView.builder(
                itemCount: images.length,
                itemBuilder: (context, index) {
                  return Image.network(images[index], fit: BoxFit.cover);
                },
              ),
            ),
            ...buildFieldWidgets(),
          ],
        ),
      ),
    );
  }
}
```

```

    ),
  ),
),
Padding(
  padding: EdgeInsets.all(16.0),
  child: Column(
    crossAxisAlignment: CrossAxisAlignment.start,
    children: buildFieldWidgets(),
  ),
),
],
),
);
}
}

```

Для обеспечения навигации по нашему приложению необходимо было разработать скрипт `main`, который отвечает за отображение и управление панелью с фрагментами информации и соответствующими кнопками для их выбора. Это позволит пользователю переключаться между различными разделами и изменять их цвет в зависимости от состояния.

Листинг 3.5 – `main.dart`

```

import 'package:cloud_firestore/cloud_firestore.dart';
import 'package:firebase_database/firebase_database.dart';
import 'package:firebase_database/ui/firebase_animated_list.dart';
import 'package:flutter/material.dart';
import 'package:firebase_core/firebase_core.dart';
import 'firebase_options.dart';
//import 'constructorAds.dart';
import 'constructor_ads.dart';
import 'firestore.dart';
import 'search.dart';
import 'profile.dart';
import 'package:firebase_auth/firebase_auth.dart';
void main() async {
  WidgetsFlutterBinding.ensureInitialized();
  await Firebase.initializeApp(
    options: DefaultFirebaseOptions.currentPlatform,
  );
  //var ford = Firestore('Автомобиль', 'Ford', 1000, 'пр.Ленина', 2002, 2000);
  //ford.addAdsFirestore();
  runApp(MyApp());
}

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Firestore Demo',
      theme: ThemeData(
        primarySwatch: Colors.blue,
      ),
      home: MyHomePage(),
    );
  }
}

```

```

class MyHomePage extends StatefulWidget {
  @override
  _MyHomePageState createState() => _MyHomePageState();
}

class _MyHomePageState extends State<MyHomePage> {
  int _selectedIndex = 0;

  static List<Widget> _widgetOptions = <Widget>[
    SearchScreen(),
    FirestoreDataWidget(),
    ProfilePage(),
  ];

  void isAuth(){
    if(FirebaseAuth.instance.currentUser == null)
      AuthScreen();
    else FirestoreDataWidget();
  }

  void _onItemTapped(int index) async {
    if (index == 1 || index==2) {
      // Check if the user is authenticated before showing the saved page
      if (FirebaseAuth.instance.currentUser == null) {
        // Redirect to login page
        Navigator.push(
          context,
          MaterialPageRoute(builder: (context) => AuthScreen()),
        );
      } else {
        setState(() {
          _selectedIndex = index;
        });
      }
    } else {
      setState(() {
        _selectedIndex = index;
      });
    }
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: IndexedStack(
        index: _selectedIndex,
        children: _widgetOptions,
      ),
      bottomNavigationBar: BottomNavigationBar(
        items: const <BottomNavigationBarItem>[
          BottomNavigationBarItem(
            icon: Icon(Icons.search),
            label: 'Поиск',
          ),
          BottomNavigationBarItem(
            icon: Icon(Icons.bookmark),
            label: 'Конструктор',
          ),
          BottomNavigationBarItem(
            icon: Icon(Icons.person),
            label: 'Профиль',
          ),
        ],
      ),
    );
  }
}

```

```

],
currentIndex: _selectedIndex,
selectedItemColor: Colors.blue,
onTap: _onItemTapped,
),
);
}
}
}

```

Методы:

- `void _onItemTapped(int index) async` – асинхронный метод меняющий текущий экран. В нём также есть проверка на авторизованность пользователя для доступа к определённым экранам. В случае если пользователь не авторизован, то вместо нужного экрана открывается экран авторизации.

3.4 Результат работы приложения

3.3.1 Экран “Поиск”

При запуске приложения пользователь встречает меню с поиском. На данном экране расположены все объявления, созданные пользователями. Вверху экрана расположено поле для поиска объявлений по их именам. В нижней панели расположена навигация по окнам приложения: "Услуги", "Конструктор", "Профиль".

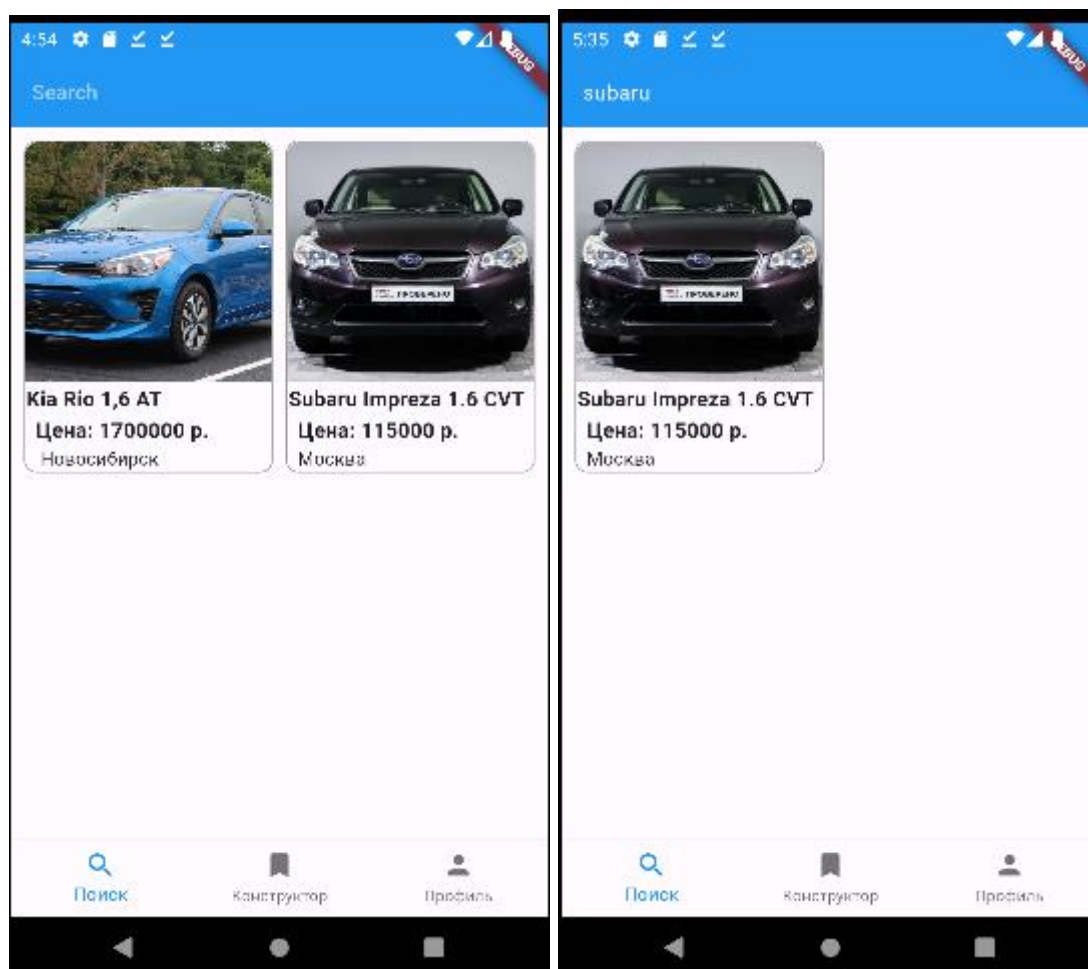


Рисунок 3.7 – “Поиск”

Каждое объявление имеет изображение, название объявления, цена, адрес. На каждое из них можно нажать. При нажатии всплывает экран с информацией о объявлении. Например, нажмём на объявление с Kio Rio 1,6 AT. При нажатии на объявление всплывает окно с изображениями товара и информацией: тип товара, цена, адрес, подтип товара, имя товара, год выпуска, пробег.

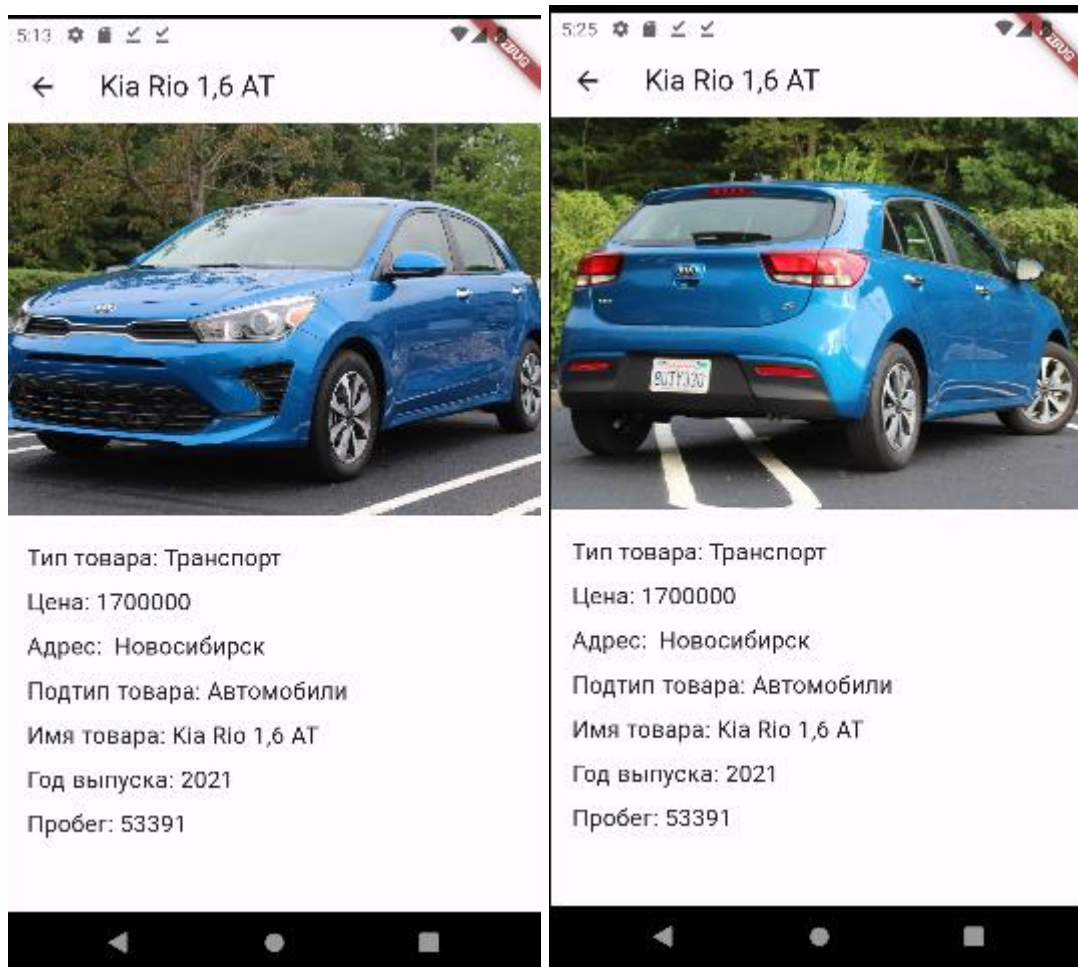


Рисунок 3.8 – “Информация об объявление”

3.3.2 Экран “Аутентификация”

Этот экран позволяет пользователю либо войти в аккаунт, либо зарегистрировать новый. На этом экране расположены 2 кнопки: “Login”, “Register”.

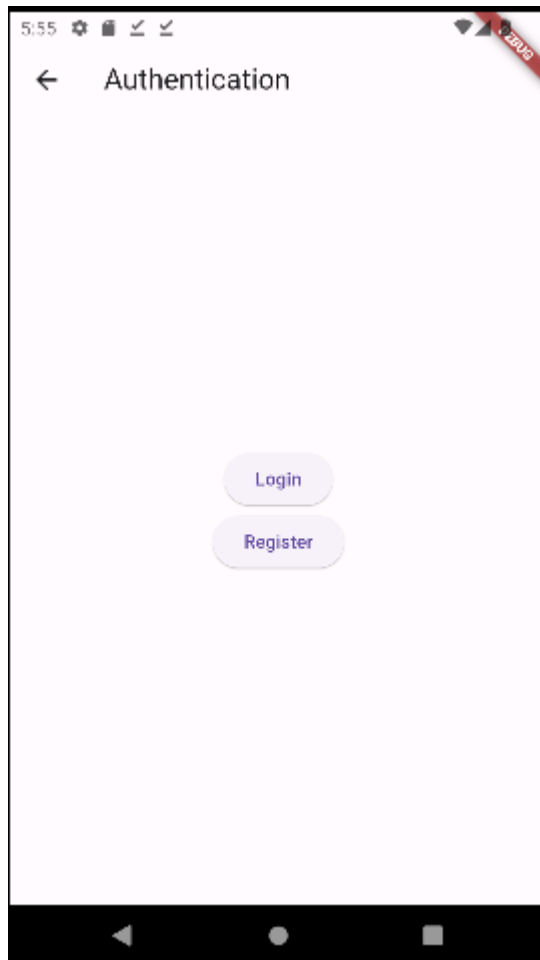
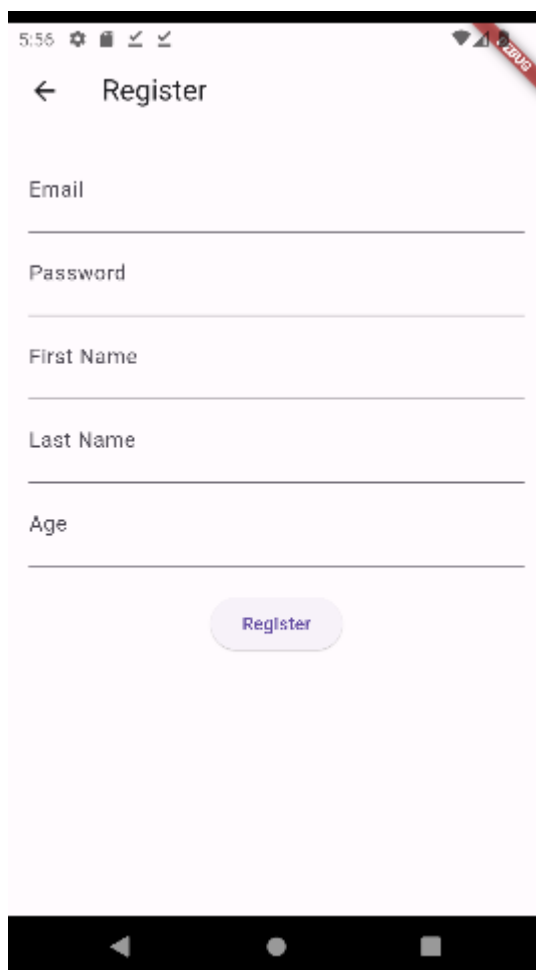


Рисунок 3.9 - Экран “Аутентификация”

При нажатии на кнопку “Register” появится экран регистрации. В нём требуется заполнить все поля: “Email”, “Password”, “First Name”, “Last Name”, “Age”. После чего нажать на кнопку “Register” для создания нового аккаунта.



The screenshot shows a mobile application interface for registration. At the top, there is a status bar with the time 5:56 and various icons. Below the status bar is a navigation bar with a back arrow and the title 'Register'. The main area contains five input fields: 'Email', 'Password', 'First Name', 'Last Name', and 'Age'. Each field has a horizontal line for text entry. At the bottom of the form is a rounded button labeled 'Register'. The entire form is set against a light pink background. A red 'BUG' sticker is visible in the top right corner of the screen.

Рисунок 3.10 – Экран “Регистрации”

После создания аккаунта можно авторизоваться с помощью экрана авторизации. В нём требуется заполнить все поля: “Email”, “Password”. После чего нажать на кнопку “Login” для входа в аккаунт.

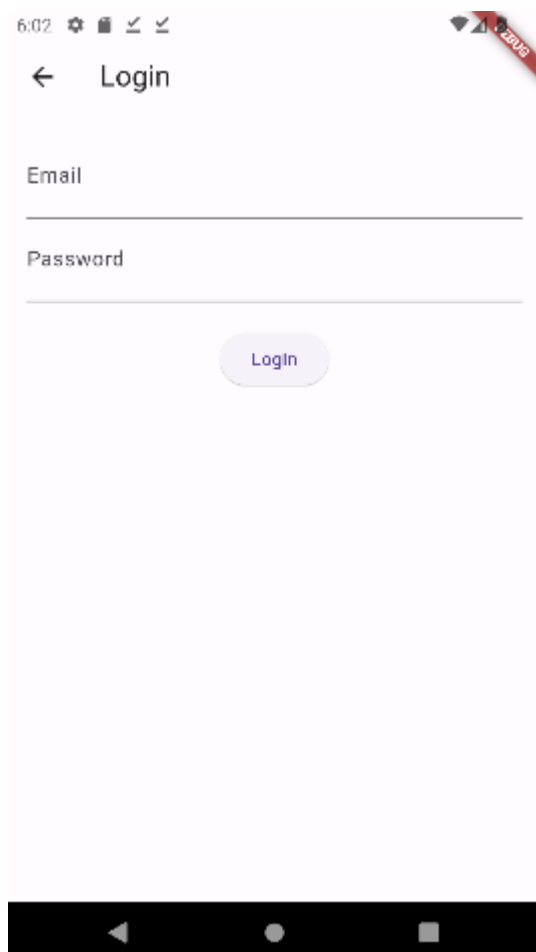


Рисунок 3.11 – Экран “Авторизации”

После успешной авторизации пользователю будут доступны экраны: “Конструктор”, “Профиль”.

3.3.3 Экран “Конструктор”

На этом экране пользователь может создавать свои объявления. Пользователю сначала позволяют выбрать тип объявления из списка, потом подтип объявления. В зависимости от выбора открываются поля обязательные для заполнения. Также позволяет загружать изображения по нажатию на иконку камеры с возможностью их будущего удаления. По нажатию на кнопку “Завершить” объявление создается и появляется на экране “Поиск”.

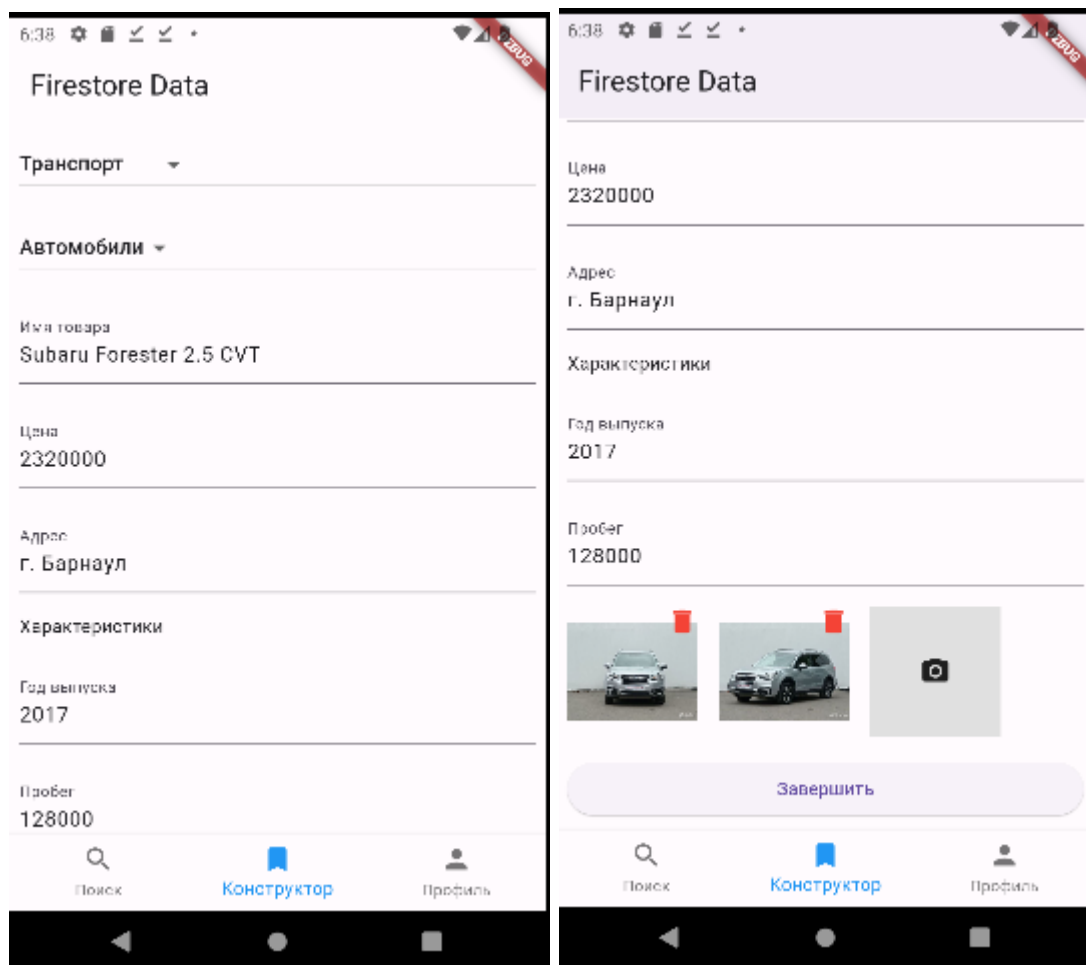


Рисунок 3.12 – Пример создания объявления в экране “Конструктор”



Рисунок 3.13 – Обновлённый экран “Поиск”

3.3.4 Экран “Профиль”

На экране “Профиль” написана информация о текущем пользователе и кнопка выхода из аккаунта.

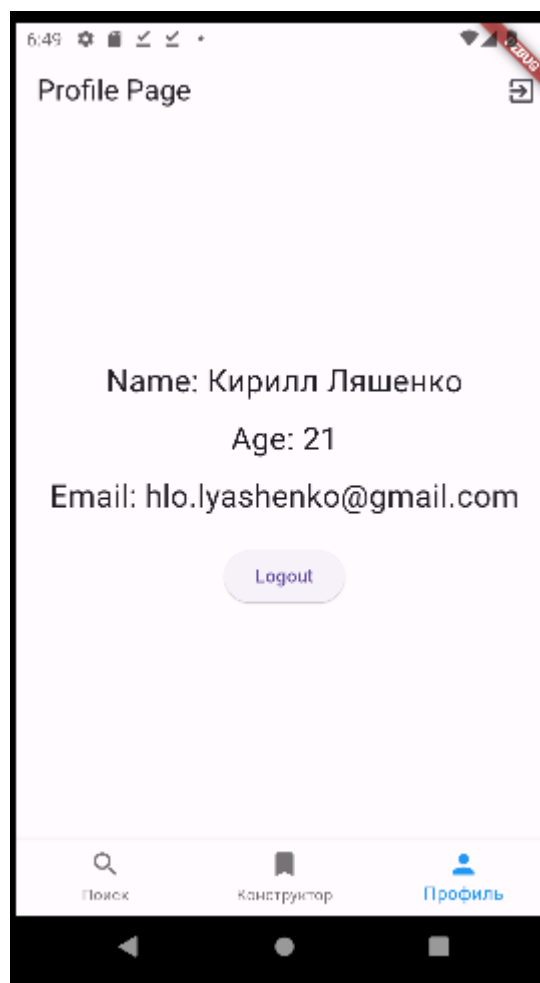


Рисунок 3.13 – Экран “Профиль”

ЗАКЛЮЧЕНИЕ

В данной дипломной работе был реализован проект кроссплатформенного приложения для просмотра и размещения объявлений. Приложение реализовано для устройств с операционной системой Android и iOS. Основные результаты зафиксированы и приведены выше в тексте пояснительной записки.

Для выполнения работы использовались современные инфокоммуникационные средства: персональный компьютер, смартфон, языки программирования высокого уровня, современное программное обеспечение и работа в сети Интернет. В первой главе проекта описывается характеристика предметной области, во второй - обзор и выбор средств реализации, а в третьей главе - структура проекта, его программный код и результаты работы.

Приложение позволяет пользователям смотреть объявления других пользователей, создавать их с помощью конструктора и искать нужные им объявления.

Подводя итоги проведенной работы, можно сказать, что создание кроссплатформенного приложения для предоставления услуг является актуальным и перспективным направлением. Они продолжают набирать популярность, а развитие технологий в этой сфере открывает новые возможности для пользователей и предпринимателей.

Разработка кроссплатформенного приложений для устройств, позволяет повысить доступность и удобство использования услуг, расширить аудиторию и улучшить качество обслуживания пользователей. Это может привести к экономическому росту и увеличению конкурентоспособности. Разработка приложений для мобильных устройств, позволяет повысить доступность и удобство использования услуг, расширить аудиторию и улучшить качество обслуживания пользователей. Это может привести к экономическому росту и увеличению конкурентоспособности сервисов для предоставления услуг на рынке.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Рейтинг языков программирования TIOBE [Электронный ресурс]. - Режим доступа: <https://www.tiobe.com/tiobe-index/>
2. Рейтинг баз данных TOPDB [Электронный ресурс]. - Режим доступа: <https://pypl.github.io/DB.html>
3. Доля рынка мобильных операционных систем в мире [Электронный ресурс]. - Режим доступа: <https://gs.statcounter.com/os-market-share/mobile/worldwide>
4. Flutter vs. Java: [Электронный ресурс]. - Режим доступа: <https://www.dhiwise.com/post/flutter-vs-java-breaking-down-the-best-framework>