Содержание

[Условное обозначение и сокращение. 7](#_Toc169268364)

[Введение 8](#_Toc169268365)

[1.Анализ предметной области 1](#_Toc169268366)1

[1.1.Общая характеристика предметной области «Аренда автомобилей» 1](#_Toc169268367)1

[1.1.1.Сущности и участники предметной области 1](#_Toc169268368)1

[1.1.2.Основные бизнес-процессы 1](#_Toc169268369)2

[1.2.Классификация сервисов аренды автомобилей 1](#_Toc169268372)3

[1.3.Проблемы существующих подходов к аренде автомомбилей 1](#_Toc169268373)4

[1.4.Анализ аналогов 1](#_Toc169268374)5

[1.5.Технологический фундамент и выбор платформы разработки 1](#_Toc169268375)9

[1.5.1.Требования к технологическому стеку для мобильного приложения 2](#_Toc169268375)0

[1.5.2.Обзор современных подходов к разработке мобильных приложений 2](#_Toc169268376)1

[1.5.3.Обзор современных подходов к бекенд-разработке 2](#_Toc169268377)1

[1.6.Выбор инструментальных средств для разработки проекта 2](#_Toc169268378)2

[1.6.1.Среда разработки Android studio 2](#_Toc169268379)3

[1.6.2.Язык программирования Kotlin 2](#_Toc169268380)4

[1.6.3.Бекенд-сервис Supabase 25](#_Toc169268381)

[2.Анализ требований 2](#_Toc169268382)6

[2.1.Анализ функциональных требований 2](#_Toc169268383)6

[2.2.Анализ нефункциональных требований 3](#_Toc169268384)0

[2.2.1.Интерфейсы пользователей 3](#_Toc169268385)0

[2.2.2.Интерфейсы программного обеспечения 3](#_Toc169268386)0

[2.2.2.Требования к безопасности 3](#_Toc169268386)0

[2.2.3.Требования к производительности 3](#_Toc169268387)1

[2.2.4.Атрибуты качества ПО 3](#_Toc169268388)1

[3.Проектирование 35](#_Toc169268389)

[3.1.Проектирование программных средств 35](#_Toc169268390)

[3.1.1.Базовая архитектура игрового приложения 35](#_Toc169268391)

[3.1.2.Проектирование игрового приложения 36](#_Toc169268392)

[3.2.Проектирование пользовательского интерфейса 40](#_Toc169268393)

[4.Реализация 46](#_Toc169268394)

[4.1.Реализация программного игрового приложения 46](#_Toc169268395)

[4.2.Реализация графических объектов 46](#_Toc169268396)

[5.Контроль качества программного обеспечения 50](#_Toc169268397)

[5.1.Функциональное тестирование 50](#_Toc169268398)

[5.2.Модульное тестирование 51](#_Toc169268399)

[6.Планирование разработки и оценка бюджета 52](#_Toc169268400)

[6.1.Оценка трудоемкости. 52](#_Toc169268401)

[6.2.Организационно-экономическое обоснование проекта 53](#_Toc169268402)

[6.3.Расчет себестоимости 53](#_Toc169268403)

[6.4.Расчет затрат на оплату труда разработчика программного продукта 54](#_Toc169268404)

[6.5.Расчет общепроизводственных расходов 56](#_Toc169268405)

[6.6.Расчёт себестоимости программного продукта 57](#_Toc169268406)

[6.7.Расчёт цены программного изделия 57](#_Toc169268407)

[Заключение 59](#_Toc169268408)

[Список использованных источников 60](#_Toc169268409)

[Приложение А 61](#_Toc169268410)

[Техническое задание 61](#_Toc169268411)

[Приложение Б 64](#_Toc169268412)

[Программный код 64](#_Toc169268413)

# Условное обозначение и сокращение.

ТЗ – техническое задание.

Ритм игра — жанр компьютерных игр, где во главу ставится музыкальная составляющая, а от игрока требуется наличие чувства ритма. Игры данного жанра берут за основу танцы или исполнение группой музыкальных композиций.

Kotlin (Ко́тлин) — кроссплатформенный, статически типизированный, объектно-ориентированный язык программирования, работающий поверх Java Virtual Machine и разрабатываемый компанией JetBrains. Также компилируется в JavaScript и в исполняемый код ряда платформ через инфраструктуру LLVM. Авторы ставили целью создать язык более лаконичный и типобезопасный, чем Java, и более простой, чем Scala[4]. Следствием упрощения по сравнению со Scala стали также более быстрая компиляция и лучшая поддержка языка в IDE[5]. Язык полностью совместим с Java, что позволяет Java-разработчикам постепенно перейти к его использованию; в частности, язык также встраивается в Android, что позволяет для существующего Android-приложения внедрять новые функции на Kotlin без переписывания приложения целиком.

Android Studio — [интегрированная среда разработки](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D0%B3%D1%80%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D0%B0%D1%8F_%D1%81%D1%80%D0%B5%D0%B4%D0%B0_%D1%80%D0%B0%D0%B7%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%BA%D0%B8) (IDE) для работы с платформой [Android](https://ru.wikipedia.org/wiki/Android" \o "Android), анонсированная [16 мая](https://ru.wikipedia.org/wiki/16_%D0%BC%D0%B0%D1%8F) [2013 года](https://ru.wikipedia.org/wiki/2013_%D0%B3%D0%BE%D0%B4) на конференции [Google I/O](https://ru.wikipedia.org/wiki/Google_I/O). В последней версии Android Studio поддерживается [Android 4.1](https://ru.wikipedia.org/wiki/Android_Jelly_Bean" \o "Android Jelly Bean) и выше. Данная IDE находилась в свободном доступе начиная с версии 0.1, опубликованной в мае 2013, а затем перешла в стадию бета-тестирования, начиная с версии 0.8, которая была выпущена в июне 2014 года. Первая стабильная версия 1.0 была выпущена в декабре 2014 года, тогда же прекратилась поддержка плагина Android Development Tools (ADT) для [Eclipse](https://ru.wikipedia.org/wiki/Eclipse_(%D1%81%D1%80%D0%B5%D0%B4%D0%B0_%D1%80%D0%B0%D0%B7%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%BA%D0%B8)).

Android Studio, основанная на программном обеспечении [IntelliJ IDEA](https://ru.wikipedia.org/wiki/IntelliJ_IDEA" \o "IntelliJ IDEA) от компании [JetBrains](https://ru.wikipedia.org/wiki/JetBrains" \o "JetBrains), — официальное средство разработки Android приложений[[4]](https://ru.wikipedia.org/wiki/Android_Studio#cite_note-4). Данная среда разработки доступна для [Windows](https://ru.wikipedia.org/wiki/Windows), [macOS](https://ru.wikipedia.org/wiki/MacOS" \o "MacOS) и [GNU/Linux](https://ru.wikipedia.org/wiki/Linux)[[5]](https://ru.wikipedia.org/wiki/Android_Studio#cite_note-autogenerated1-5). 17 мая 2017, на ежегодной конференции Google I/O, [Google](https://ru.wikipedia.org/wiki/Google_(%D0%BA%D0%BE%D0%BC%D0%BF%D0%B0%D0%BD%D0%B8%D1%8F)) анонсировал поддержку языка [Kotlin](https://ru.wikipedia.org/wiki/Kotlin" \o "Kotlin), используемого в Android Studio, как официального [языка программирования](https://ru.wikipedia.org/wiki/%D0%AF%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F) для платформы Android в дополнение к [Java](https://ru.wikipedia.org/wiki/Java) и [C++](https://ru.wikipedia.org/wiki/C%2B%2B).

Supabase – Это реляционная база данных на основе тех же технологий, что лежат в PostgreSQL – одной из самых популярных и надежных БД в мире. Но такого описания для Supabase явно недостаточно, ведь это крупный проект, включающий в себя куда больше интересных решений, чем может показаться на первый взгляд. Supabase – это бесплатный аналог Firebase, полифункциональная платформа, объединяющая в себе несколько важных программных решений и упрощающая их реализацию до предельно примитивного уровня, чтобы даже новички в разработке могли спокойно добавить в свои приложения или сайты такие функции, как авторизация, хранилище файлов, обновление контента на сайте в реальном времени и т.п.

# Введение

Современный мир характеризуется ускоренным темпом жизни и ростом мобильности населения. В условиях ограниченного времени и необходимости быстрого перемещения, услуги по аренде автомобилей становятся все более востребованными как для деловых поездок, так и для личных нужд. Традиционные методы аренды, часто связанные с посещением офисов, заполнением бумажных документов и ограниченным временем работы, становятся менее удобными по сравнению с цифровыми сервисами.

Стремительное развитие мобильных технологий и проникновение смартфонов во все сферы жизни создают благоприятные условия для перехода услуг аренды автомобилей в онлайн-формат. Мобильные приложения предлагают пользователям беспрецедентное удобство: возможность найти, забронировать и оплатить автомобиль в любое время и в любом месте, получить актуальную информацию о наличии и ценах, а также управлять своими бронированиями прямо с телефона.

Актуальность данной темы обусловлена возрастающей потребностью рынка в эффективных, удобных и технологичных приложениях. Разработка такого приложения позволяет не только удовлетворить потребности конечных пользователей, но и предоставить компаниям, занимающимся арендой, современный инструмент для оптимизации бизнес-процессов и расширения клиентской базы. Исходя из вышеизложенного было принято решение создать мобильное приложение с использованием современных сред и способов разработки.

Для достижения поставленной цели необходимо решить следующие задачи:

1. Провести анализ предметной области "аренда автомобилей", изучить существующие бизнес-процессы и выявить ключевые сущности и их взаимосвязи.

2. Исследовать существующие мобильные приложения и платформы для аренды автомобилей, провести анализ их функционала, преимуществ и недостатков.

3. Сформулировать функциональные и нефункциональные требования к разрабатываемому мобильному приложению.

4. Разработать архитектуру мобильного приложения и структуру базы данных.

5. Спроектировать пользовательский интерфейс (UI) и пользовательский опыт (UX) приложения, обеспечив его интуитивность и удобство.

6. Разработать мобильное приложение на языке Kotlin с использованием Android Studio.

7. Интегрировать приложение с бэкендом для хранения данных, аутентификации пользователей и выполнения серверных операций. Основные этапы разработки игры:

1. Планирование и концепция.
2. Выбор IDE и языка для разработки.
3. Прототипирование. Создание основных функций и их тестирование.
4. Начало разработки.

# Анализ предметной области

# Общая характеристика предметной области "Аренда автомобилей"

Предметная область "Аренда автомобилей" представляет собой сегмент рынка услуг, предоставляющий возможность временного использования транспортных средств физическим и юридическим лицам без необходимости их приобретения в собственность. Исторически эта услуга предоставлялась через стационарные офисы компаний, однако с развитием информационных технологий и, в частности, мобильных устройств, происходит активная цифровизация процессов аренды, что значительно повышает доступность и удобство сервиса.

# Сущности и участники предметной области

В рамках предметной области "Аренда автомобилей" можно выделить следующих ключевых участников и сущности:

• Арендатор (Клиент): Физическое или юридическое лицо, желающее временно использовать автомобиль. Осуществляет поиск, бронирование, оплату и использование автомобиля.

• Арендодатель (Компания или Частное лицо): Владелец автопарка (компания) или отдельного автомобиля (в случае P2P аренды), предоставляющий транспортные средства в аренду. Отвечает за состояние автомобилей, оформление документов, выдачу и прием транспорта.

• Автомобиль: Основной объект аренды. Характеризуется маркой, моделью, годом выпуска, типом кузова, трансмиссии, топлива, классом, стоимостью аренды, доступностью и текущим состоянием.

• Бронирование: Процесс предварительного резервирования конкретного автомобиля на определенный период времени. Включает выбор дат, времени, места получения/возврата и дополнительных опций.

• Договор аренды: Юридический документ, фиксирующий условия предоставления автомобиля в аренду, права и обязанности сторон, стоимость, срок аренды и ответственность.

• Пункт выдачи/возврата: Локация, где происходит передача автомобиля арендатору и его последующий возврат. Может быть офисом компании, парковкой, аэропортом и т.д.

• Оплата: Процесс расчета за услуги аренды, страхования и дополнительные опции. Осуществляется, как правило, безналичным способом.

В 90-е переняли эстафету Интернет-игры. Ранее упомянутая «Quake», разрешала организовывать многопользовательские баталии в Интернете, на ровне с такими играми, как «StarCraft», «EverQuest, «Ultima Online», «Age of Empires». Стали разрабатываться игры на Macromedia Flash. Подобные игры обычно являлись различными реализациями простых, но атмосферных и увлекательных концепций, которые популярны и сегодня. Главная особенность Flash-игр заключалась в том, что они запускались в окне WEB- браузера, оснащенного соответствующими плагинами от компании Adobe.

* + 1. Основные бизнес-процессы

Типовой цикл взаимодействия в рамках предметной области, поддерживаемый цифровыми сервисами, включает следующие основные бизнес-процессы:

1. Регистрация и Авторизация: Пользователь создает учетную запись в системе или входит под существующей. Может потребоваться верификация документов (паспорт, водительское удостоверение).

2. Поиск и Выбор Автомобиля: Арендатор указывает желаемые параметры (даты, место получения/возврата, класс авто и др.) и просматривает доступные варианты.

3. Просмотр Деталей: Пользователь изучает подробную информацию о выбранном автомобиле (характеристики, фото, условия аренды, стоимость, рейтинг).

4. Бронирование: Арендатор подтверждает выбор, указывает дополнительные услуги (навигатор, детское кресло и др.) и переходит к оформлению.

5. Оформление Договора и Оплата: Система формирует электронный договор или предлагает ознакомиться с его условиями. Производится расчет стоимости и осуществляется оплата (полная или частичная).

6. Выдача Автомобиля: Арендатор прибывает в пункт выдачи, предъявляет документы. Происходит осмотр автомобиля, подписание акта приема-передачи (в электронном виде в мобильных сервисах).

7. Использование Автомобиля: Период фактической аренды.

8. Возврат Автомобиля: Арендатор возвращает автомобиль в указанный пункт в оговоренное время. Происходит повторный осмотр, сверка показаний (пробег, уровень топлива), подписание акта возврата.

9. Финальный Расчет: Производится окончательный расчет с учетом фактически пройденного расстояния, расхода топлива (если не по полному баку), штрафов (при наличии).

# Классификация сервисов аренды автомобилей

Сервисы аренды автомобилей можно классифицировать по различным признакам, в первую очередь по длительности и модели предоставления услуги:

• Классическая (традиционная) аренда: Предоставляется на срок от одних суток до нескольких месяцев. Как правило, требует посещения офиса для оформления документов и получения/возврата автомобиля. Ориентирована на путешествия, деловые поездки, замену собственного авто на время ремонта.

• Каршеринг (Carsharing): Краткосрочная аренда с поминутной или почасовой тарификацией. Автомобили обычно рассредоточены по городу, бронирование, получение и завершение аренды происходит через мобильное приложение без прямого контакта с сотрудником компании. Ориентирован на короткие поездки в пределах города.

• Peer-to-Peer (P2P) аренда: Платформы, позволяющие частным владельцам сдавать свои автомобили в аренду другим частным лицам. Платформа выступает посредником, обеспечивая бронирование, оплату, страхование и поддержку.

Разрабатываемое мобильное приложение ориентировано преимущественно на модель Peer-to-Peer (P2P), автоматизируя и упрощая процессы взаимодействия между арендатором и арендодателем (компанией), но с возможностью адаптации для поддержки элементов других моделей в будущем.

# Проблемы существующих подходов к аренде автомобилей

Несмотря на развитие рынка, существующие подходы к аренде автомобилей имеют ряд проблем и ограничений, снижающих удобство для пользователей и эффективность для компаний:

1. Бумажная волокита и бюрократия: Традиционные методы часто требуют заполнения множества бумажных документов, что занимает время и увеличивает вероятность ошибок.
2. Ограниченная доступность: Офисы компаний имеют фиксированный график работы, что создает неудобства для клиентов, особенно в нерабочее время. Процесс получения/возврата вне часов работы офиса может быть затруднен или невозможен.
3. Отсутствие прозрачности информации: В некоторых случаях клиентам сложно получить полную и актуальную информацию о доступных автомобилях, их реальном состоянии, полной стоимости аренды со всеми скрытыми платежами и страховками без прямого запроса или посещения офиса.
4. Сложность сравнения предложений: Сравнение цен и условий аренды у разных компаний требует посещения множества сайтов или звонков, что неудобно.
5. Устаревшие или неудобные цифровые решения: Некоторые существующие онлайн-сервисы или мобильные приложения имеют сложный, перегруженный интерфейс, низкую скорость работы, ограниченный функционал или неоптимизированы для мобильных устройств.
6. Проблемы с верификацией и безопасностью: Ручная проверка документов может быть медленной, а отсутствие цифровых следов затрудняет разрешение спорных ситуаций.

Разрабатываемое мобильное приложение призвано решить эти проблемы путем предоставления удобного, быстрого и прозрачного инструмента для всех этапов процесса аренды, от поиска до возврата автомобиля, с использованием современных технологий для автоматизации и цифровизации взаимодействия.

# Анализ аналогов

Для определения ключевых функций, пользовательских сценариев и выявления лучших практик, а также потенциальных недостатков, был проведен анализ существующих на рынке мобильных приложений и онлайн-платформ для аренды автомобилей.

Первым в списке было приложение Едем.рф. Основными функциями являются поиск попутных автомобилей и покупка билетов на автобус. Сразу стоит отметить что такое приложение для аренды автомобилей не подходит

* Преимущества: Поиск попутных автомобилей.
* Недостатки: нет необходимых функций для аренды автомобиля
* Вывод для данного проекта: функционал не подходит для реализации в моем проекте

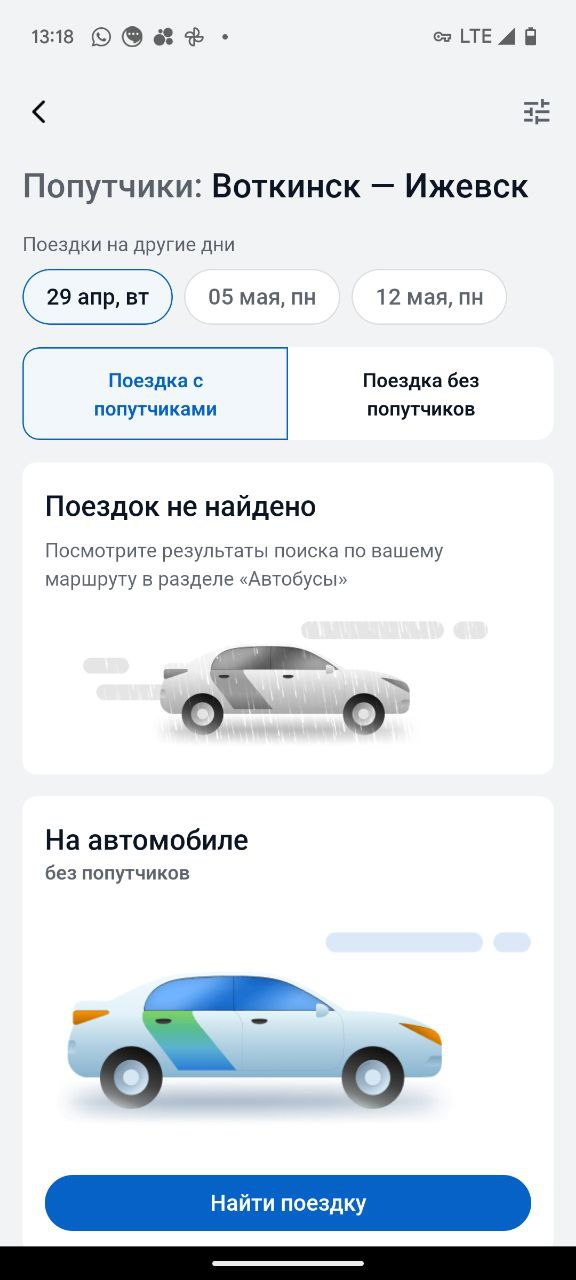


Рисунок 1 – Едем.рф

# 

Рисунок 2 – Едем.рф

Вторым в очереди было приложение “Прокат авто”. Оно очень похоже на “Едем.рф”, функционал точно такой же, поиск попутных автомобилей и покупка билетов на автобус и самолет.

* Преимущества: Поиск попутных автомобилей.
* Недостатки: нет необходимых функций для аренды автомобиля
* Вывод для данного проекта: функционал не подходит для реализации в моем проекте

# 

Рисунок 3 – Прокат авто

# Следующим было приложение “RentRide”.

# Преимущества: В объявлении автомобиля можно посмотреть фотографии, отзывы, характеристики автомобиля и условия аренды.

* Недостатки: Все автомобили в Москве и в регионах их взять не получиться.
* Вывод для данного проекта: можно использовать для сравнительного анализа и интеграции реализованных функций в свой проект

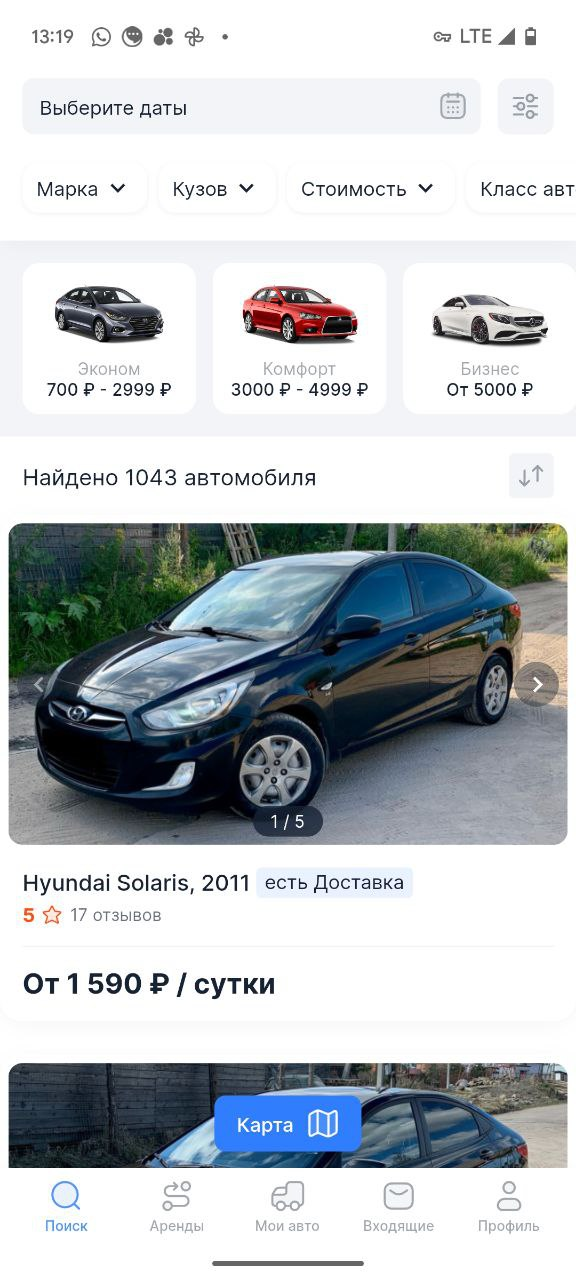


Рисунок 4 – RentRide

# 

Рисунок 5 – RentRide



Рисунок 6 – RentRide

# 1.5 Технологический фундамент и выбор платформы разработки

Выбор технологического стека является критически важным этапом, определяющим возможности, производительность, масштабируемость и удобство поддержки разрабатываемого программного продукта.

# Требования к технологическому стеку для мобильного приложения аренды

Для успешной реализации функционала мобильного приложения для аренды автомобилей необходим надежный и эффективный технологический фундамент, включающий:

* Клиентская часть (Frontend): Нативное мобильное приложение для платформы Android, обеспечивающее интуитивно понятный пользовательский интерфейс, быстрый доступ к функциям, возможность работы с данными (отображение списков авто, карт, форм ввода) и взаимодействие с бэкендом.
* Серверная часть (Backend): Система, обеспечивающая хранение и управление данными (информация об авто, пользователях, бронированиях), обработку бизнес-логики (расчет стоимости, проверка доступности), аутентификацию пользователей, обработку платежей и предоставление API для взаимодействия с клиентским приложением.
* База данных: Надежное хранилище для структурированных данных.
* Файловое хранилище: Для хранения медиафайлов, таких как фотографии автомобилей, сканы документов пользователей.
* Система аутентификации и авторизации: Для безопасного управления учетными записями пользователей и контроля доступа к данным.

# Обзор современных подходов к разработке мобильных приложений (Android)

Существуют два основных подхода к разработке мобильных приложений:

* Нативная разработка: Использование официальных языков и инструментов, предоставляемых разработчиками платформы (Java/Kotlin для Android, Swift/Objective-C для iOS). Обеспечивает максимальную производительность, полный доступ ко всем API устройства и лучшую интеграцию с операционной системой. Требует отдельной разработки для каждой платформы.
* Кросс-платформенная разработка: Использование фреймворков (React Native, Flutter, Xamarin), позволяющих писать код, который может быть скомпилирован и запущен на нескольких платформах (Android и iOS) из единой кодовой базы. Ускоряет разработку для двух платформ, но может иметь ограничения по производительности, доступу к специфическим API и интеграции с нативными элементами.

Для данного проекта выбрана нативная разработка под платформу Android с использованием языка Kotlin. Такое решение обосновано следующими причинами:

* Необходимость достижения максимальной производительности и отзывчивости интерфейса, что критично для приложений с активным взаимодействием с пользователем и обработкой данных (например, отображение карты).
* Полный и беспрепятственный доступ ко всем возможностям и API платформы Android, что может потребоваться для реализации специфического функционала (например, интеграция с платежными системами, работа с геолокацией в фоновом режиме).
* Kotlin является современным, безопасным и высокопроизводительным языком, официально поддерживаемым Google для Android-разработки, что обеспечивает долгосрочную актуальность и доступ к обширной документации и сообществу.
* Проект фокусируется на разработке качественного приложения для одной из наиболее распространенных платформ, что соответствует поставленным целям.

# Обзор современных подходов к бэкенд-разработке и облачным сервисам (BaaS)

Для реализации серверной части проекта также существуют различные подходы:

* Традиционный бэкенд: Самостоятельная разработка серверной части с использованием фреймворков (например, Spring Boot, Node.js/Express, Django, Ruby on Rails), развертывание и управление серверами, разработка API и работа с базой данных. Обеспечивает полный контроль, но требует значительных временных и ресурсных затрат на разработку инфраструктуры.
* Backend as a Service (BaaS): Использование облачных платформ (например, Firebase, AWS Amplify, Supabase), предоставляющих готовые к использованию компоненты бэкенда (база данных, аутентификация, файловое хранилище, API) через API или SDK. Значительно ускоряет разработку, снижает затраты на управление инфраструктурой, обеспечивает масштабируемость "из коробки".

Для данного проекта выбран подход с использованием Backend as a Service, а именно платформы Supabase. Обоснование выбора:

* Ускорение разработки: Supabase предоставляет готовые компоненты (база данных, аутентификация, хранилище), что позволяет разработчику мобильного приложения сосредоточиться на клиентской части, минимизируя время на создание бэкенд-инфраструктуры с нуля.
* PostgreSQL Database: Supabase основан на мощной и надежной реляционной базе данных PostgreSQL, что обеспечивает гибкость в моделировании данных и поддержку сложных запросов.
* Автоматическое API (PostgREST): Supabase автоматически генерирует RESTful API из схемы базы данных, что упрощает взаимодействие клиентского приложения с данными.
* Встроенная аутентификация: Готовая система управления пользователями с поддержкой различных методов входа.
* Файловое хранилище (Storage): Удобное решение для хранения и управления файлами, такими как фотографии автомобилей.
* Реальное время (Realtime): Возможность подписки на изменения данных в базе данных, что может быть полезно для реализации динамических функций.
* Открытый исходный код: Supabase является проектом с открытым исходным кодом, что повышает прозрачность и снижает риски привязки к проприетарным решениям

Таким образом, комбинация нативной разработки на Kotlin для Android и использования BaaS-платформы Supabase для бэкенда выбрана как оптимальная для быстрого и эффективного создания функционального и масштабируемого мобильного приложения для аренды автомобилей в рамках данного дипломного проекта.

# Выбор инструментальных средств для разработки проекта

Для реализации проекта был выбран набор специализированных инструментов, обеспечивающих эффективный процесс разработки, от написания кода до развертывания.

# Среда разработки Android Studio

Android Studio является официальной интегрированной средой разработки (IDE) для платформы Android, разработанной компанией Google. Она основана на IntelliJ IDEA и предоставляет все необходимые инструменты для разработки нативных Android-приложений, включая:

* Продвинутый редактор кода с поддержкой Kotlin и Java, автодополнением, рефакторингом и анализом кода.
* Инструменты для проектирования пользовательского интерфейса (Layout Editor).
* Мощный отладчик.
* Эмулятор Android для тестирования приложений на различных виртуальных устройствах.
* Инструменты для сборки (Gradle), профилирования производительности, анализа памяти и энергопотребления.
* Интеграция с системой контроля версий (Git).

Использование Android Studio является стандартом де-факто для Android-разработки и обеспечивает наиболее комфортные и эффективные условия для создания высококачественных приложений под данную платформу. Главное окно программы представлено на рисунке 7.

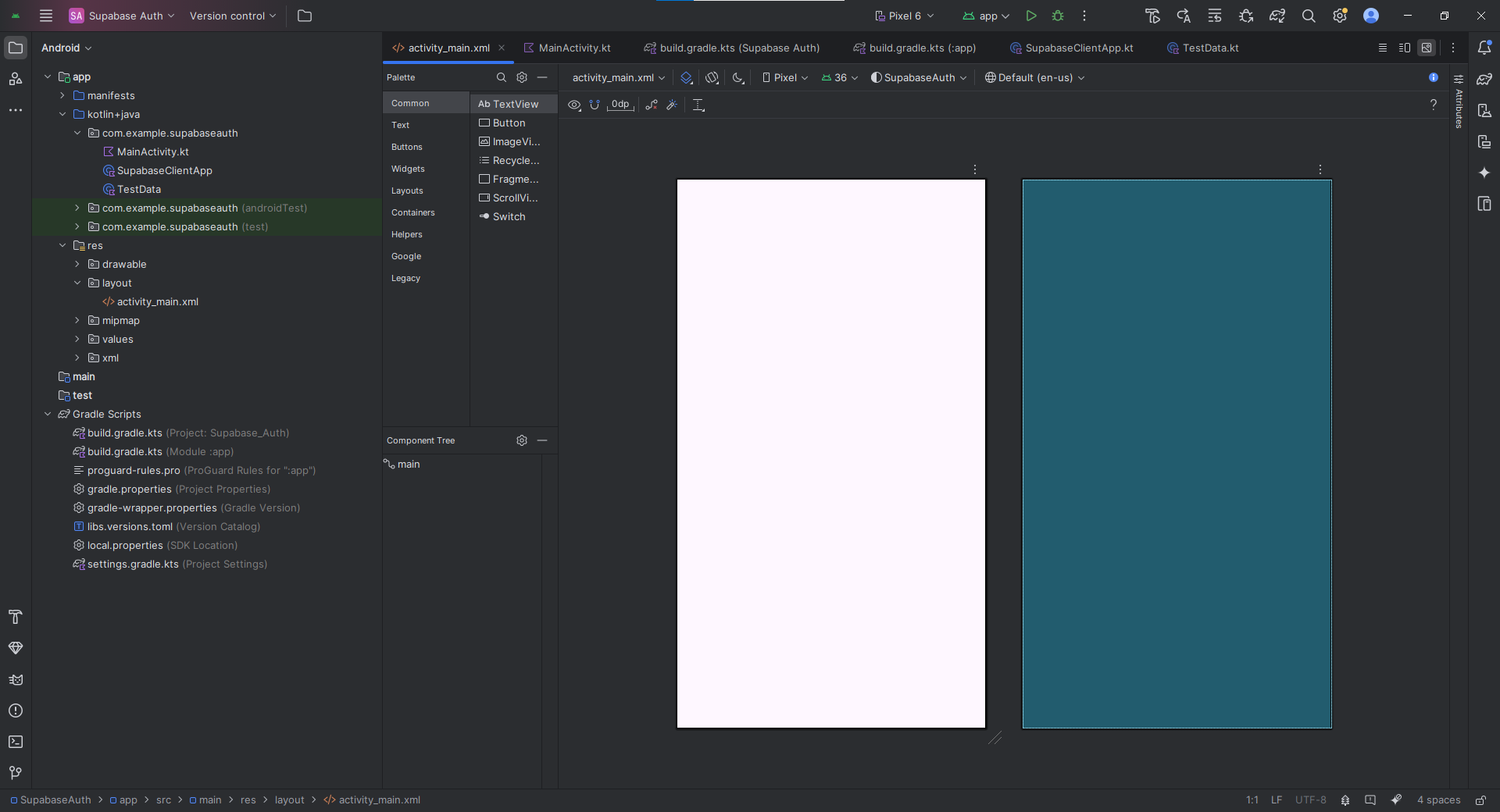


Рисунок 7 – Программное пространство Android studio

# Язык программирования Kotlin

Kotlin – это статически типизированный язык программирования, разработанный компанией JetBrains. В 2017 году Google объявила Kotlin официальным языком для разработки Android-приложений. Основные преимущества Kotlin, определившие его выбор для данного проекта:

* Лаконичный синтаксис: Позволяет писать меньше кода для выполнения тех же задач по сравнению с Java, повышая производительность разработчика.
* Безопасность: Встроенная поддержка null safety помогает избежать ошибок, связанных с нулевыми ссылками (NullPointerException), которые являются распространенной проблемой в Java.
* Современные возможности: Поддержка корутин для асинхронного программирования, расширяющие функции, делегирование свойств и другие современные концепции.
* Полная совместимость с Java: Kotlin полностью интероперабелен с существующим Java-кодом и библиотеками, что позволяет использовать всю обширную экосистему Android.
* Официальная поддержка: Активная поддержка со стороны Google и большое, растущее сообщество разработчиков.

Использование Kotlin позволяет создать более надежное и поддерживаемое приложение, сократив при этом время на написание кода.

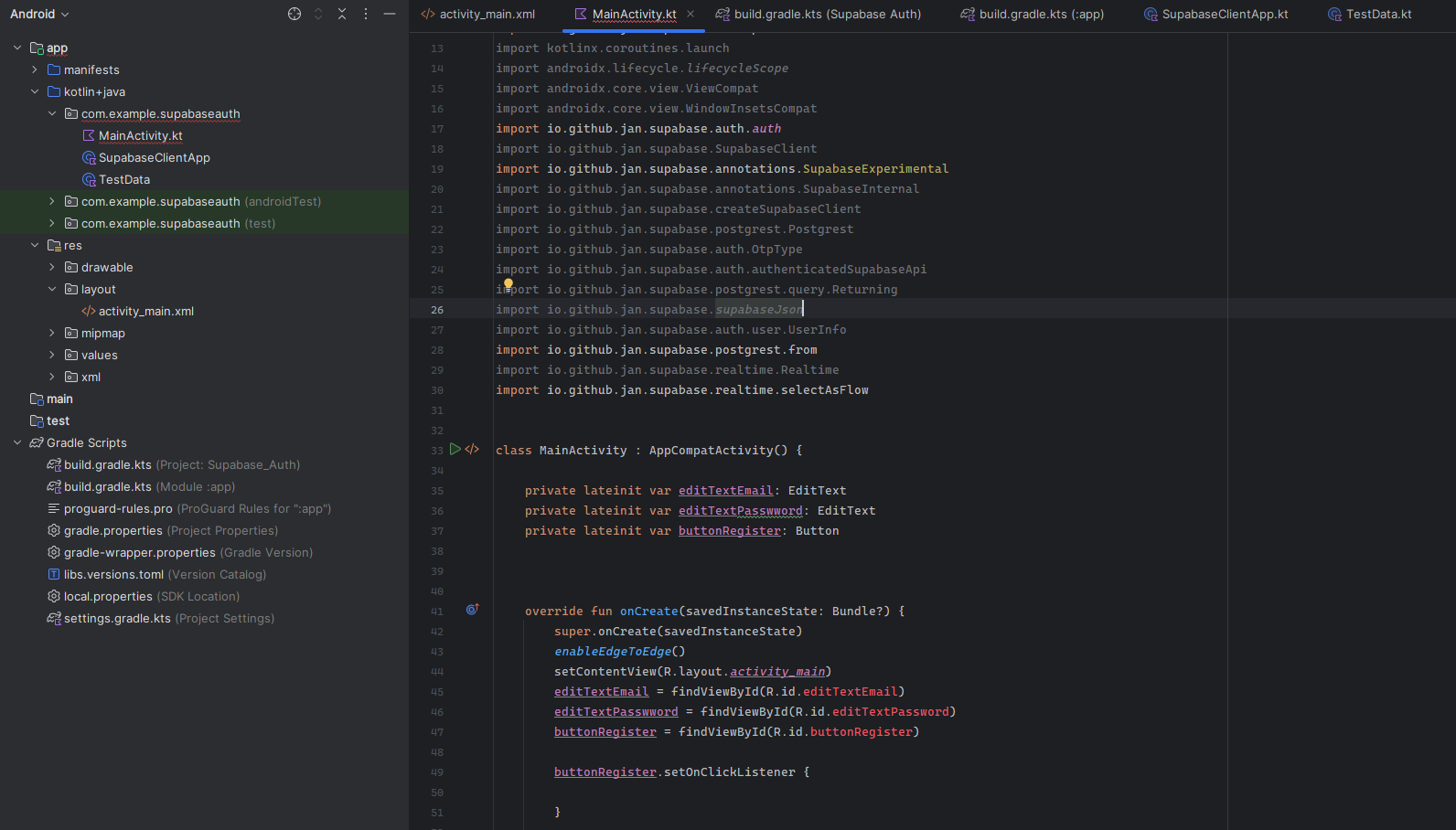


Рисунок 8 – Пространство языка программирования Kotlin

# Бэкенд-сервис Supabase

Supabase используется в качестве основного инструмента для предоставления бэкенд-инфраструктуры для мобильного приложения. Как уже упоминалось в разделе 1.5.3, Supabase предоставляет набор облачных сервисов, интегрированных вокруг базы данных PostgreSQL. В рамках проекта будут активно использоваться следующие компоненты:

* PostgreSQL Database: Служит центральным хранилищем всех данных приложения (пользователи, автомобили, бронирования и т.д.).
* Authentication: Предоставляет готовые API и SDK для регистрации, входа в систему и управления пользователями.
* Storage: Используется для хранения неструктурированных данных, таких как фотографии автомобилей.
* PostgREST API: Автоматически сгенерированный RESTful API, который позволяет мобильному приложению взаимодействовать с базой данных, выполняя операции CRUD (Create, Read, Update, Delete) над данными.

Использование Supabase позволяет получить мощную и масштабируемую бэкенд-инфраструктуру без необходимости ее самостоятельной разработки и администрирования, что является оптимальным решением для проекта с ограниченными сроками реализации.

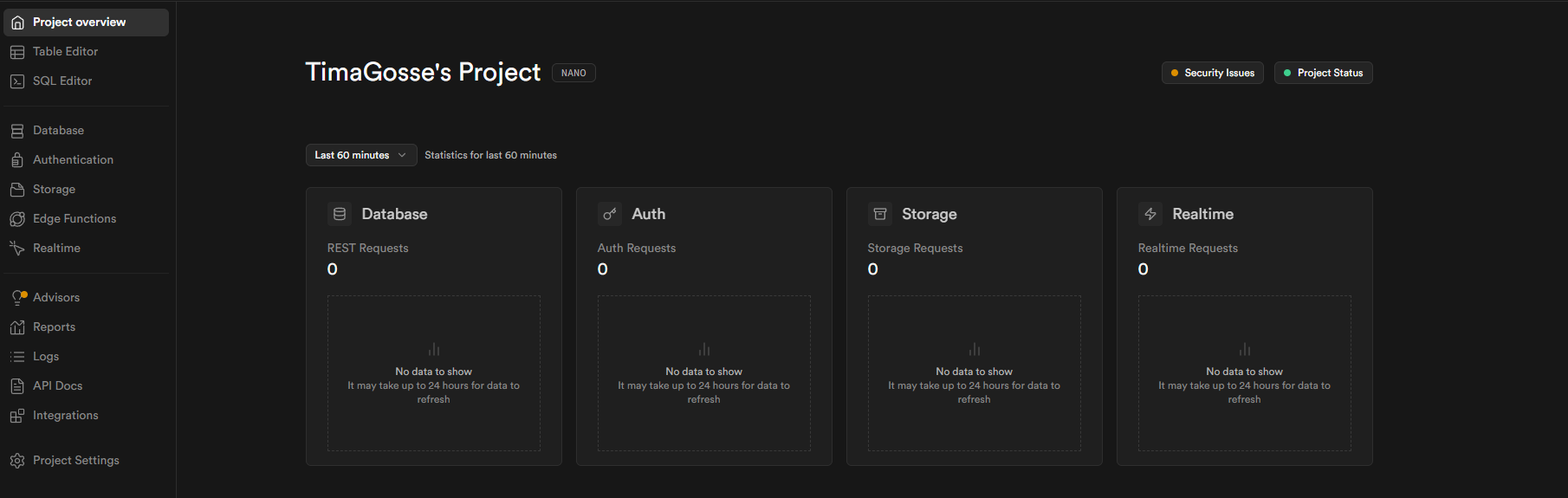


Рисунок 13 – Пространство работы с Supabase

# Анализ требований

Требование можно определить, как «подробное описание того, что должно быть реализовано». Существует два основных типа требований:

* функциональные требования – какое поведение должна предлагать система;
* нефункциональные требования – особое свойство или ограничение, накладываемое на систему.

# Анализ функциональных требований

В качестве пользователей разрабатываемого приложения представлены два типа пользователей «Арендатор» и «Арендодатель». Их классы и характеристики представлены в таблице 1.

Таблица – 1

Классы и характеристики пользователя

|  |  |
| --- | --- |
| Класс пользователей | Характеристика пользователей |
| Арендатор | Арендатор имеет возможность просмотра размещенных в аренду автомобилей, изменения параметров поиска (Марка, Модель, Цена, Тип кузова), а также непосредственно возможность аренды и последующего получения машины |
| Арендодатель | Арендодатель имеет возможность размещения автомобиля в аренду (добавление характеристик, фотографий, цены и описания). И непосредственного размещения в ленту объявлений. При выборе арендатором автомобиля, арендодатель должен провести встречу с выдачей автомобиля, а при сдаче провести осмотр. |

Исходя из этого была построена модель прецедентов предметной области для класса Арендодателя и Арендатора (Рисунок 14).



Рисунок 14 – Диаграмма вариантов использования

Для наглядного представления поведения прецедентов наилучшим вариантом будет построение диаграмм деятельности. Диаграммы деятельности, представлены на рисунках 15 и 16, построена для прецедентов «Арендатор» и «Арендодатель» ввиду их наибольшей значимости.

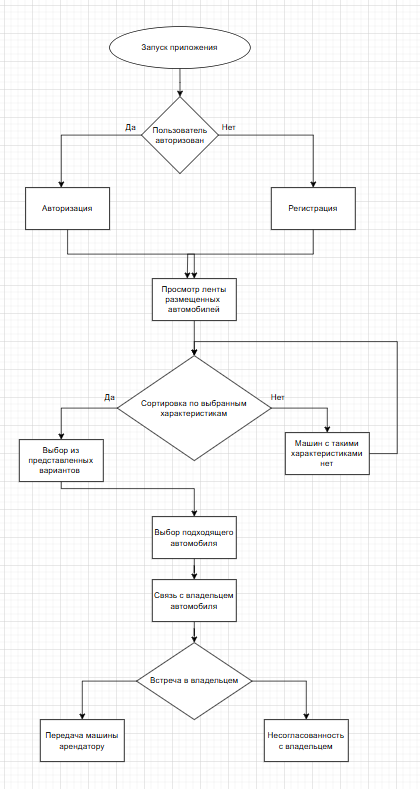


Рисунок 15 – Диаграмма деятельности прецедента «Арендатор»

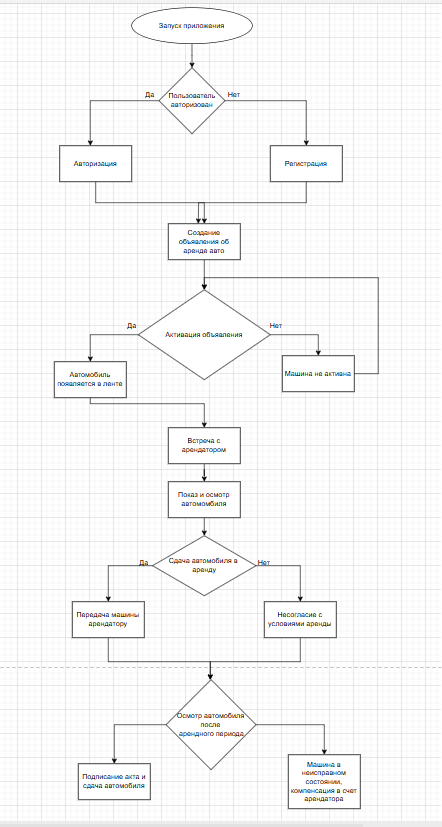


Рисунок 16 – Диаграмма деятельности прецедента «Арендатор»

# Анализ нефункциональных требований

# Интерфейсы пользователей

Арендатора при запуске приложения встречает страница авторизации представляющая собой приятный и понятный интерфейс, не создающий никаких сложностей для пользователя. С её помощью можно авторизоваться и зайти в приложение, а если учетной записи нет, то её можно создать. Следом открывается главное окно, с лентой размещенных в аренду автомобилей, с помощью фильтров можно отсортировать автомобили по нужным критериям, таким как: цена, тип кузова, класс. Нажав на понравившуюся машину открывается её страница, на которой можно детальнее всё изучить и связаться с арендодателем

# Арендодатель при запуске приложения при запуске приложения встречает страница авторизации представляющая собой приятный и понятный интерфейс, не создающий никаких сложностей для пользователя. С её помощью можно авторизоваться и зайти в приложение, а если учетной записи нет, то её можно создать. Следом открывается главное окно, с лентой размещенных в аренду автомобилей, нажав на кнопку добавить автомобиль открывается окно в котором необходимо заполнить данные о машине и добавить её изображения, после чего его автомобиль добавиться в ленту размещенных автомобилей.

# Требования к безопасности

1. Авторизация и регистрация происходит с помощью встроенного сервиса Supabase Auth. При регистрации используется двухфакторная аутентификация по электронной почте, также используется bcrypt — сильная функция хэширования паролей. Для дополнительной безопасности к каждому хэшу добавляется случайно сгенерированный параметр соли.
2. Все таблицы в базе данных имеют RLS политики безопасностиНекоторые возможности политик безопасности**:**
   * + 1. Публичный доступ. Ведра можно сделать публичными, и тогда читать файлы сможет кто угодно без аутентификации.
       2. Приватный доступ. Для чувствительных данных ведра можно сделать закрытыми, и тогда для доступа к файлам потребуется правильная аутентификация.
       3. Безопасность на уровне строк (RLS). С её помощью можно создавать гибкие политики доступа, которые позволяют ограничивать доступ в зависимости от бизнес-потребностей.

# Требования к производительности

# Приложение должно быть стабильным и устойчивым к сбоям. Система должна корректно обрабатывать ошибки и исключительные ситуации, предоставляя пользователю информативные сообщения. Доступность сервиса (uptime) должна быть на высоком уровне

# Атрибуты качества ПО

Для готового приложения не требуется никаких дополнительных атрибутов. Достаточно установить приложение на своё мобильное устройство, иметь стабильное подключение к интернету и иметь активный адрес электронной почты для регистрации и последующего подтверждения учетной записи

# Проектирование

# Проектирование программных средств

# Базовая архитектура игрового приложения

Диаграмма развертывания, показанная на рисунке 17, демонстрирует физическую архитектуру мобильного приложения. Диаграмма показывает, что развертывание предельно просто: на смартфоне пользователя установлено мобильное приложение, которое при помощи Http запросов и интернета отправляет и получает данные на сервер приложения, на котором установлены все необходимые библиотеки для работы с «Android Studio». Сервер приложения также при помощи Http запросов и интернета взаимодействует с сервером базы данных «Supabase» служащий для хранения, передачи и обработки данных, хранения изображений и авторизации.

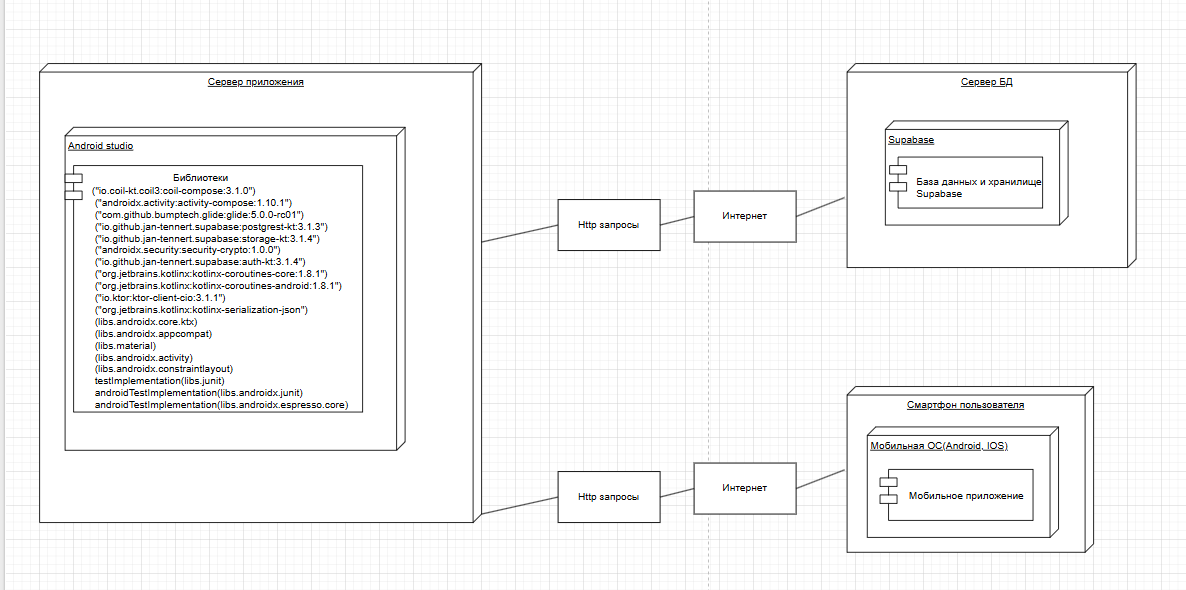


Рисунок 17 – Диаграмма развертывания

# Проектирование мобильного приложения

В ходе проектирования было выявлено 19 классов – сущностей, описание которых приведено в таблице 2.

Таблица – 2

Описание классов игрового приложения

|  |  |
| --- | --- |
| Класс | Назначение |
| AuthManager | Класс авторизации и регистрации, служит для получения и сохранения сессий, входа и выхода из системы. |
| CarAdapter | Класс адаптер, служит для загрузки изображений для конкретного автомобиля из базы данных и хранилища |
| CarData | Data класс содержит классы с переменными для их последующего использования в других классах. |
| GetCar | Класс использующийся для страницы добавления нового автомобиля. Для этого нужно заполнить все необходимые поля, после чего данные обрабатываются и передаются в базу данных |

Продолжение таблицы 2.

|  |  |
| --- | --- |
| Класс | Назначение |
| ImageCarActivity | Класс для страницы добавления фотографий добавляемого автомобиля, имеет функции открытия камеры, галереи, и вызов класса адаптера |
| ImageAdapter | Класс перемещения выбранных фотографий в хранилище |
| ImageCarActivity | Класс для выбора фотографий из галереи и выбора необходимых для добавления их к объявлению автомобиля |
| ListCar | Класс отображения ленты объявлений, по макету item\_car\_list |
| LoginActivity | Класс для авторизации зарегистрированных пользователей |
| MyApplication | Класс удобного создания Supabase Client и подключения к нему из любого класса проекта |
| ProfileActivity | Класс для регистрации новых пользователей, и последующего добавления данных в Supabase |
| Набор классов игрового интерфейса (Рисунок 21) | Набор классов, отвечающий за отображения всего визуального интерфейса игры и декораций. |

Диаграмма классов для просмотра ленты размещенных объявлений представлена на рисунке 17.

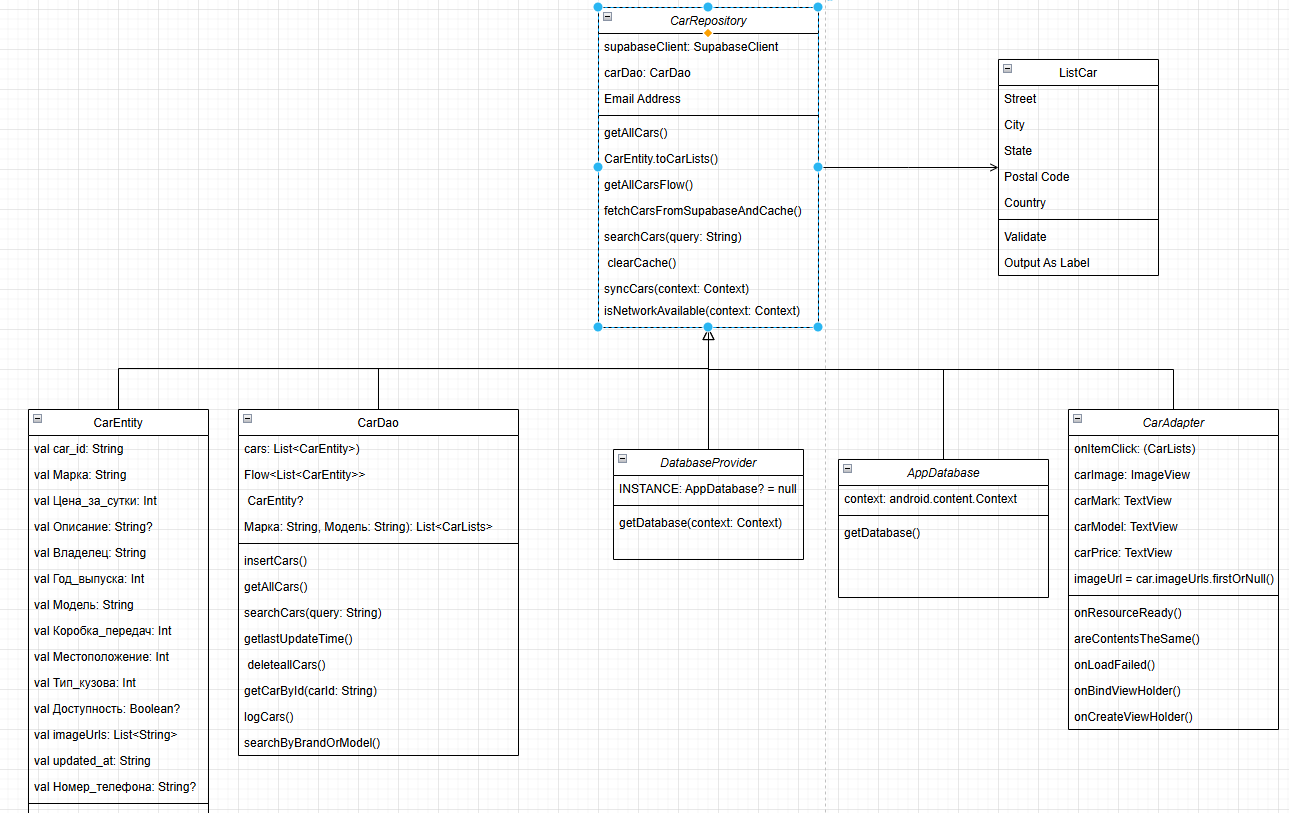


Рисунок 17 – Диаграмма классов (просмотр ленты)

Диаграмма классов для добавления новых автомобилей в базу данных представлена на рисунке 18.

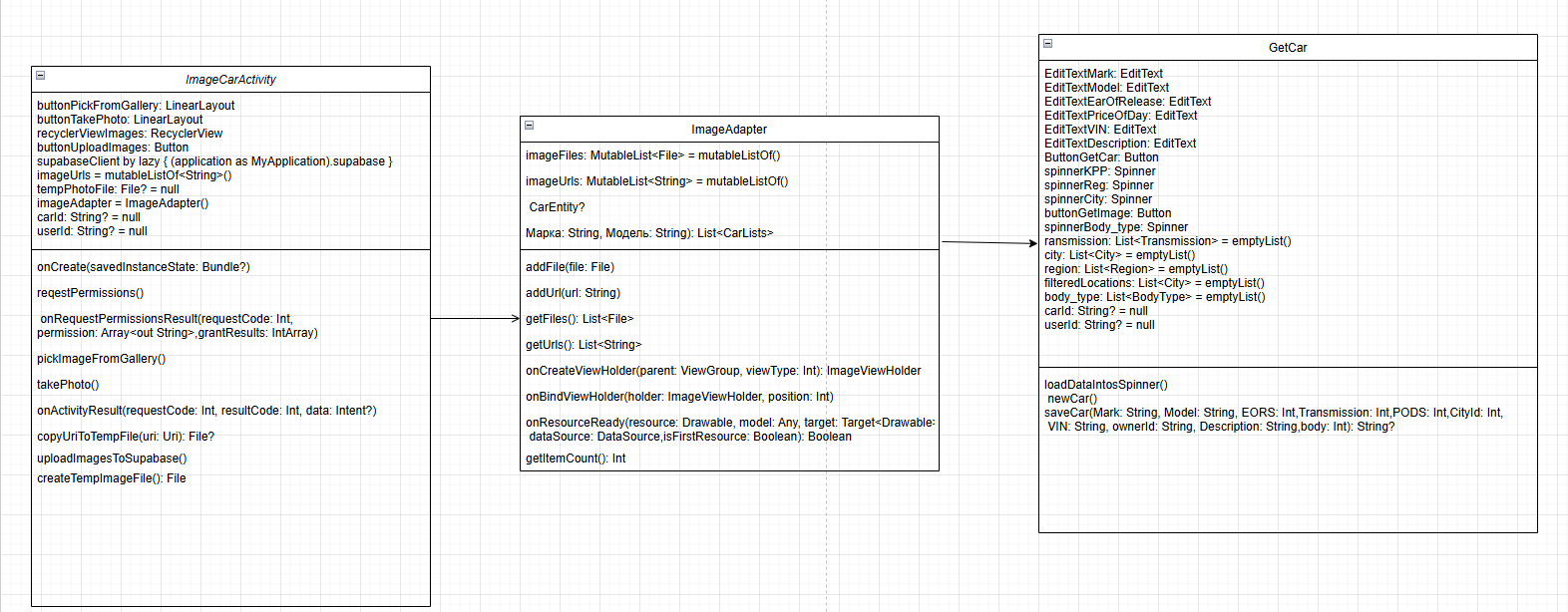


Рисунок 18 – Диаграмма классов (добавление автомобиля)

Диаграмма классов для регистрации и заполнения профиля представлена на рисунке 19.

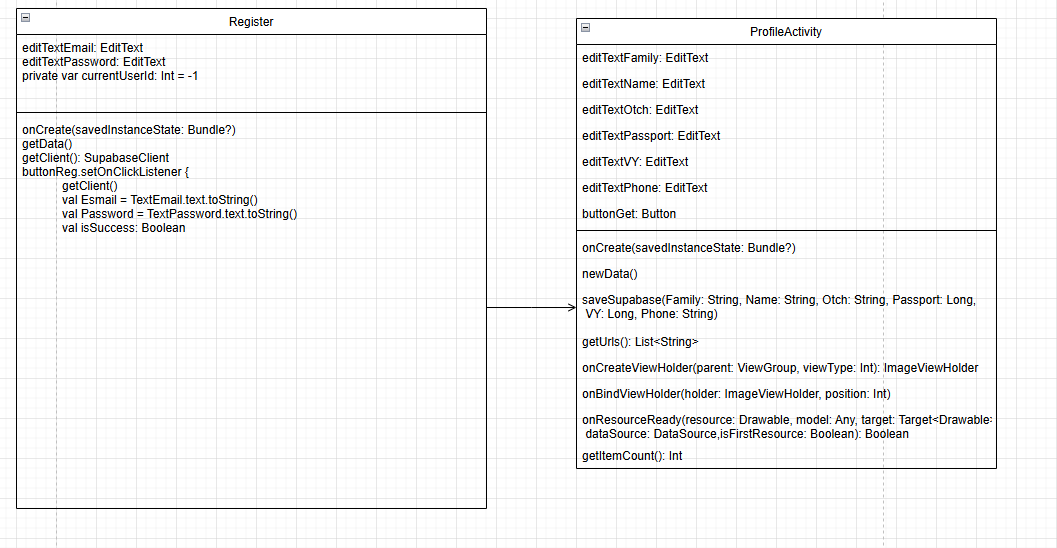


Рисунок 19 – Диаграмма классов (регистрация)

Диаграмма классов для просмотра объявлений представлена на рисунке 20.

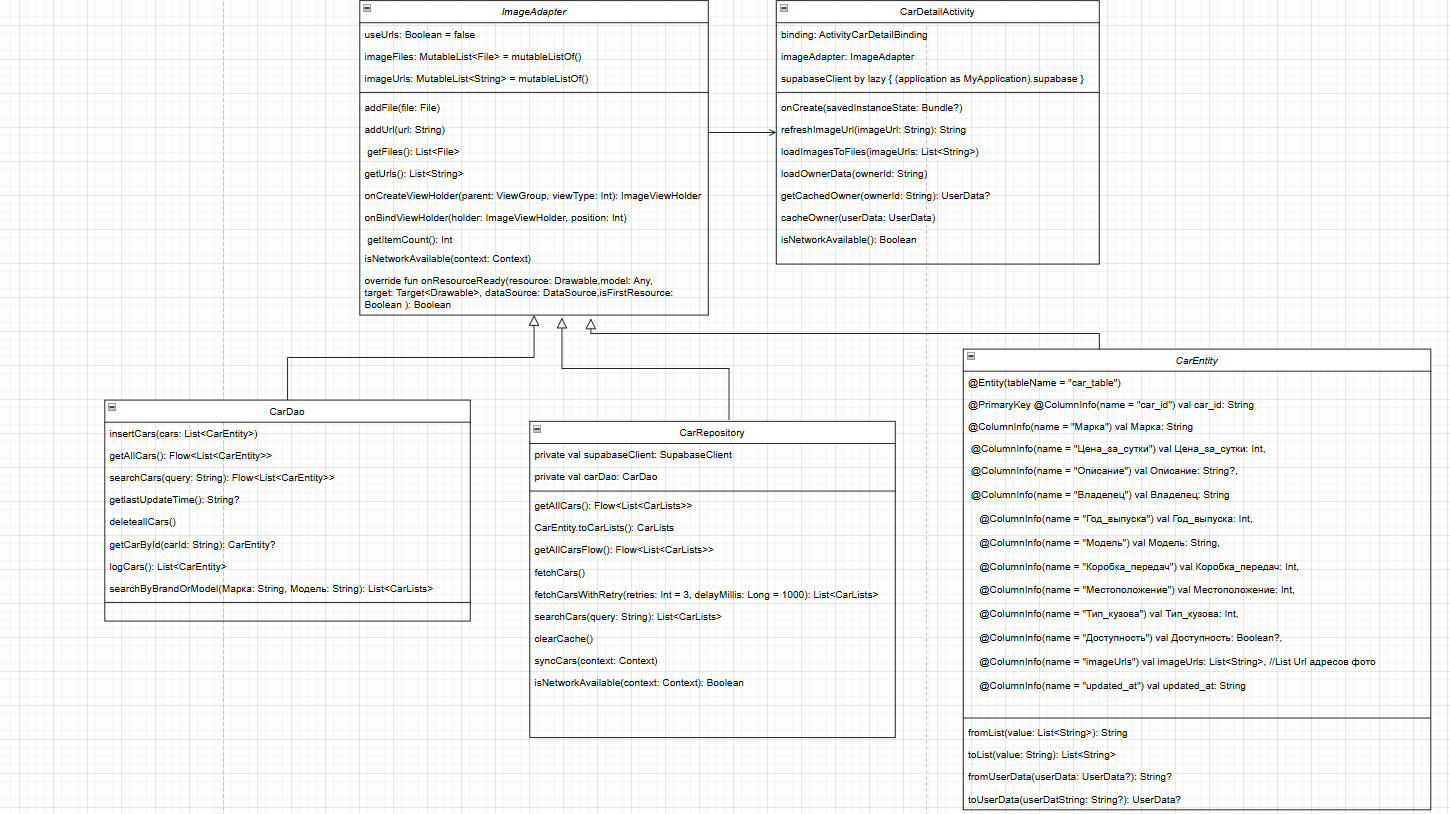


Рисунок 20 – Диаграмма классов (игровой интерфейс)

# Проектирование пользовательского интерфейса

Исходя из анализа требований и модели анализа, были спроектированы интуитивно понятные графические пользовательские интерфейсы, представленные на рисунках 21 – 30. Описание компонентов пользовательских интерфейсов приведено в таблицах 3.

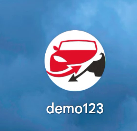


Рисунок 21 – Логотип приложения

Данный логотип отображается на дисплее телефона пользователя

.

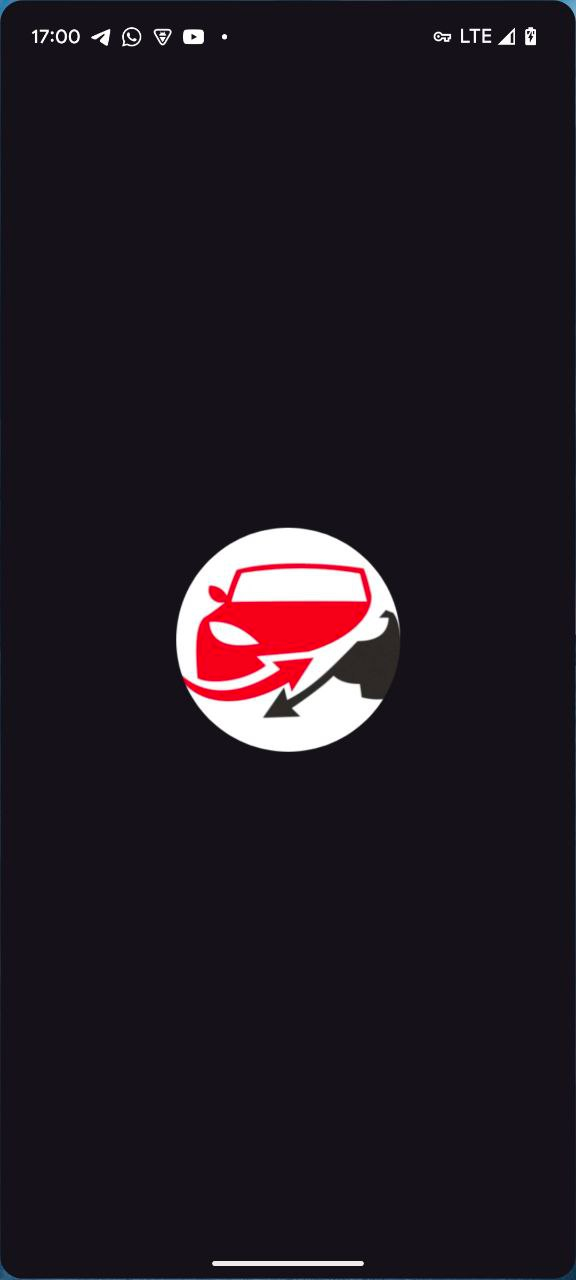


Рисунок 22 – Заставка приложения

Эта заставка появляется при запуске приложения

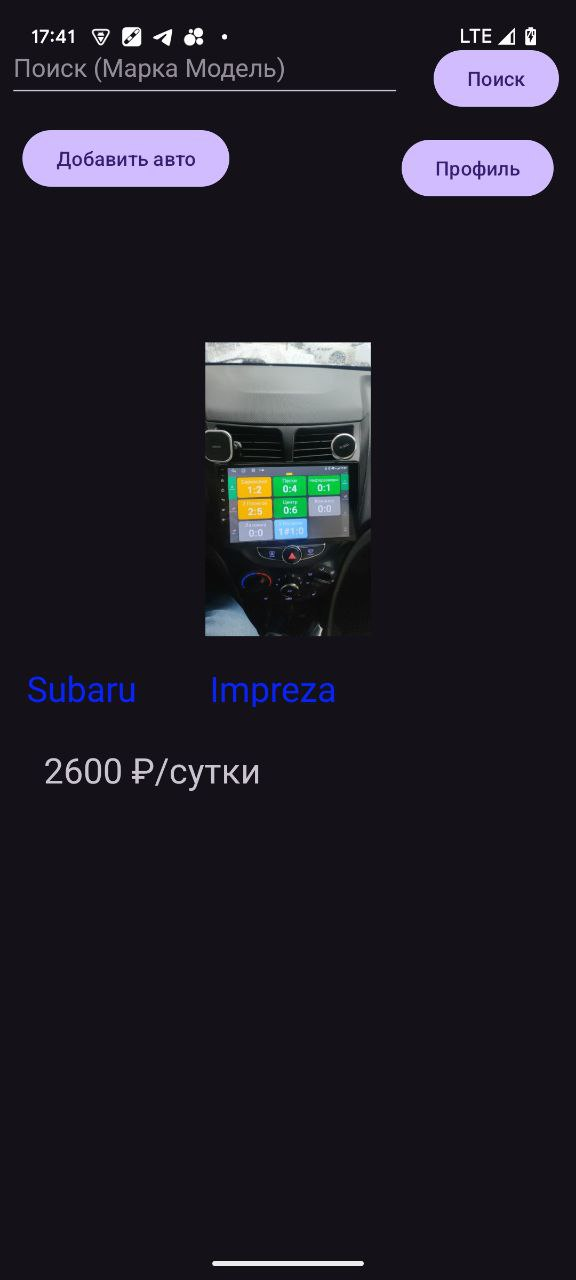


Рисунок 23 – Лента объявлений

При запуске приложения отображается лента с объявлениями размещенных в аренду автомобилей.



Рисунок 24 – Страница автомобиля

При нажатии на объявление открывается страница с информацией об автомобиле и его фотографиями.

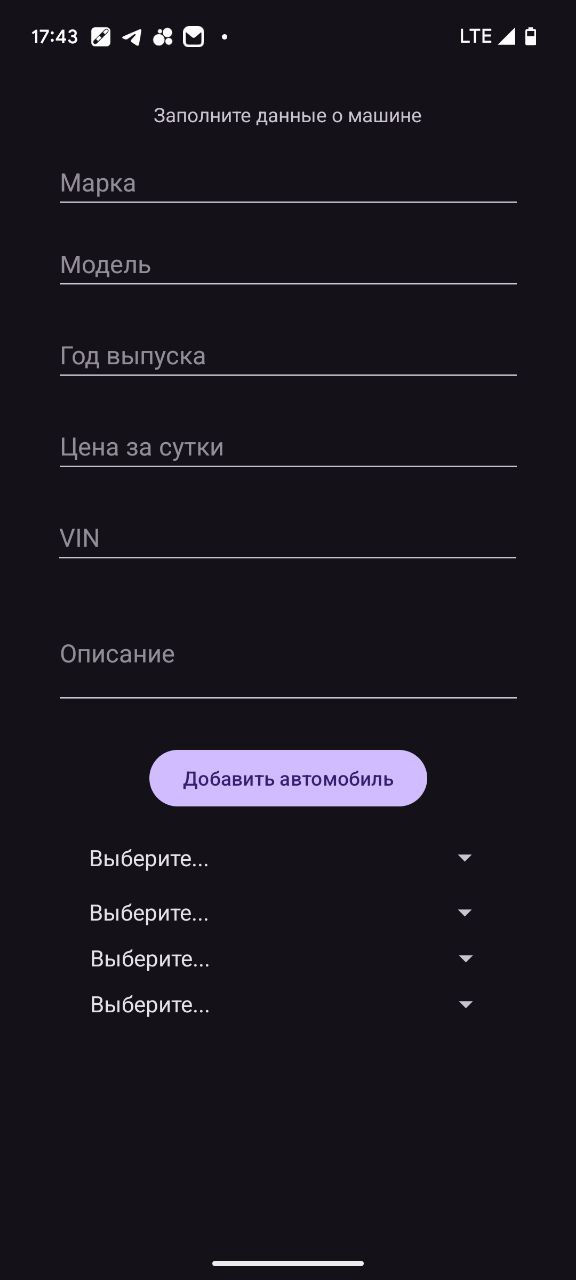


Рисунок 25 – Страница добавления автомобиля

При нажатии на кнопку «Добавить автомобиль» открывается соответствующее окно, с полями для заполнения данных об автомобиле.

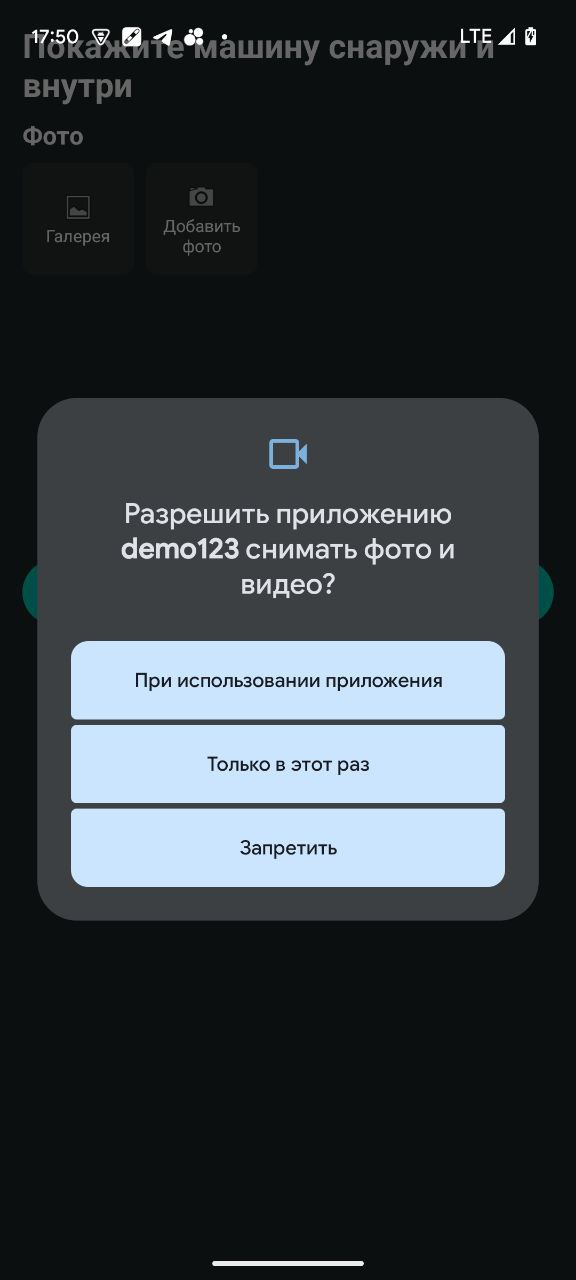


Рисунок 26 – Окно разрешения на доступ к камее

На экране отображается окно для предоставления приложению доступа к камере устройства



Рисунок 27 – Окно разрешения на доступ к фото и видео

На экране отображается окно для предоставления приложению доступа к галерее устройства

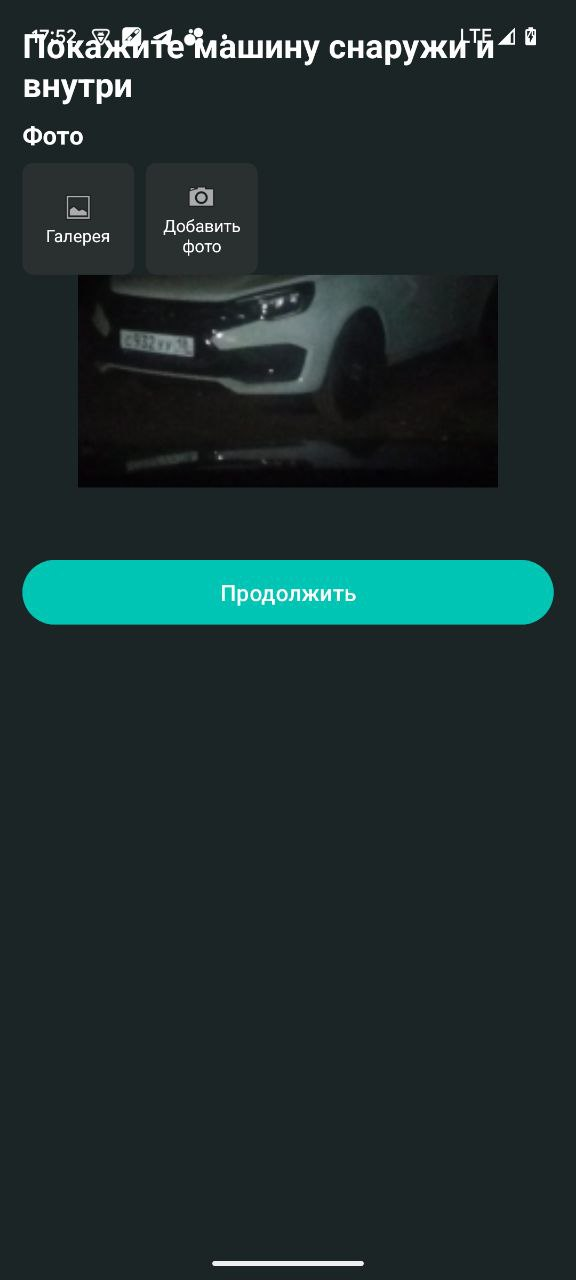


Рисунок 28 – Окно добавления изображений автомобиля

На странице можно выбрать изображения добавляемого автомобиля из галереи устройства или использовать для этого камеру

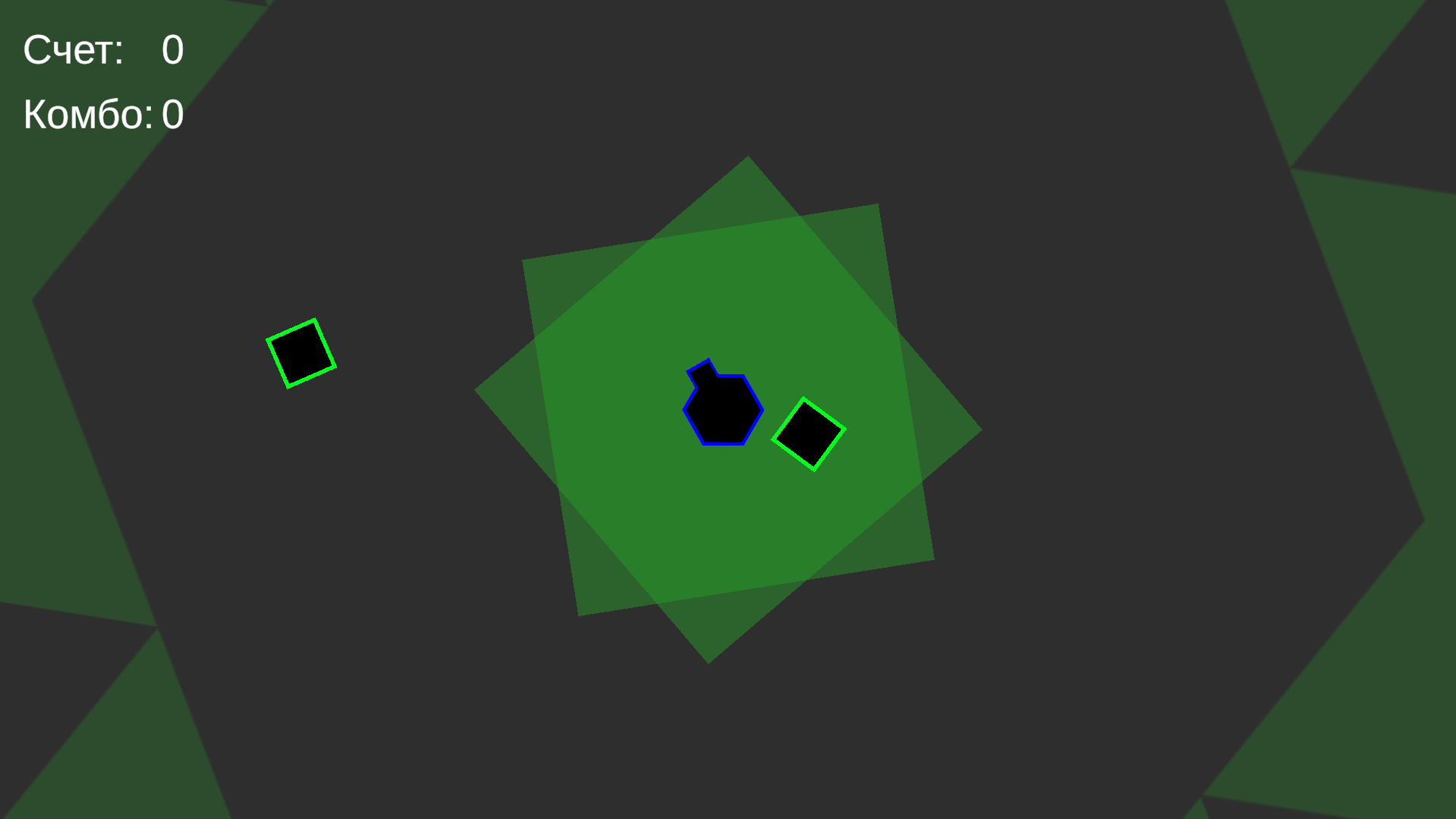


Рисунок 28 – Сцена бесконечного режима

На сцене бесконечного режима располагается игрок, которому необходимо сбивать летящее в него объекты нужными цветами и набивать как можно более большой счет.

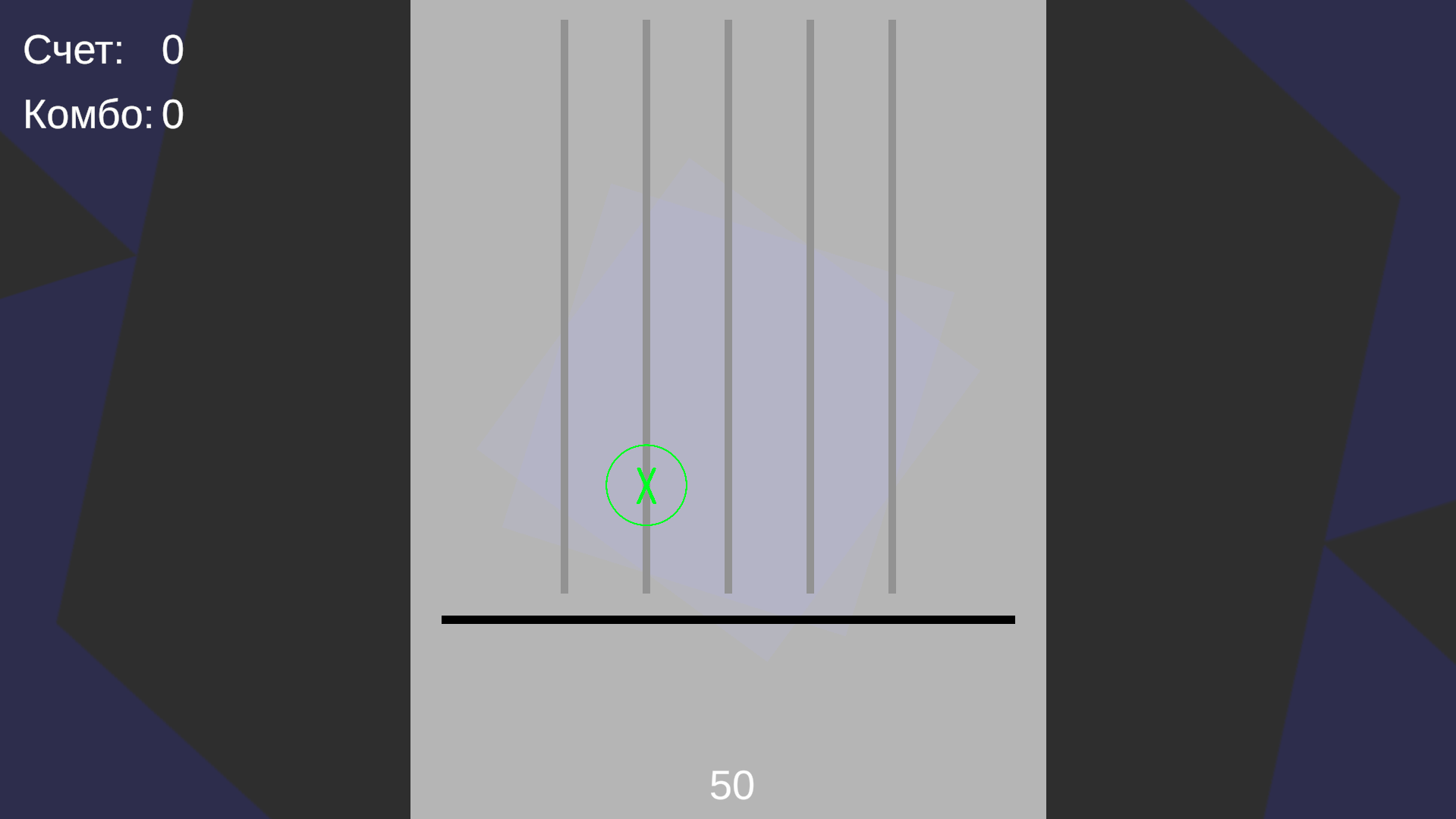


Рисунок 29 – Сцена ритмичного режима

На сцене ритмичного режима располагаются направления, в даль которых появляться ноты определенного цвета и двигаться в направление нижнего края экрана, где располагается полоса, на которой надо нажимать ноты.

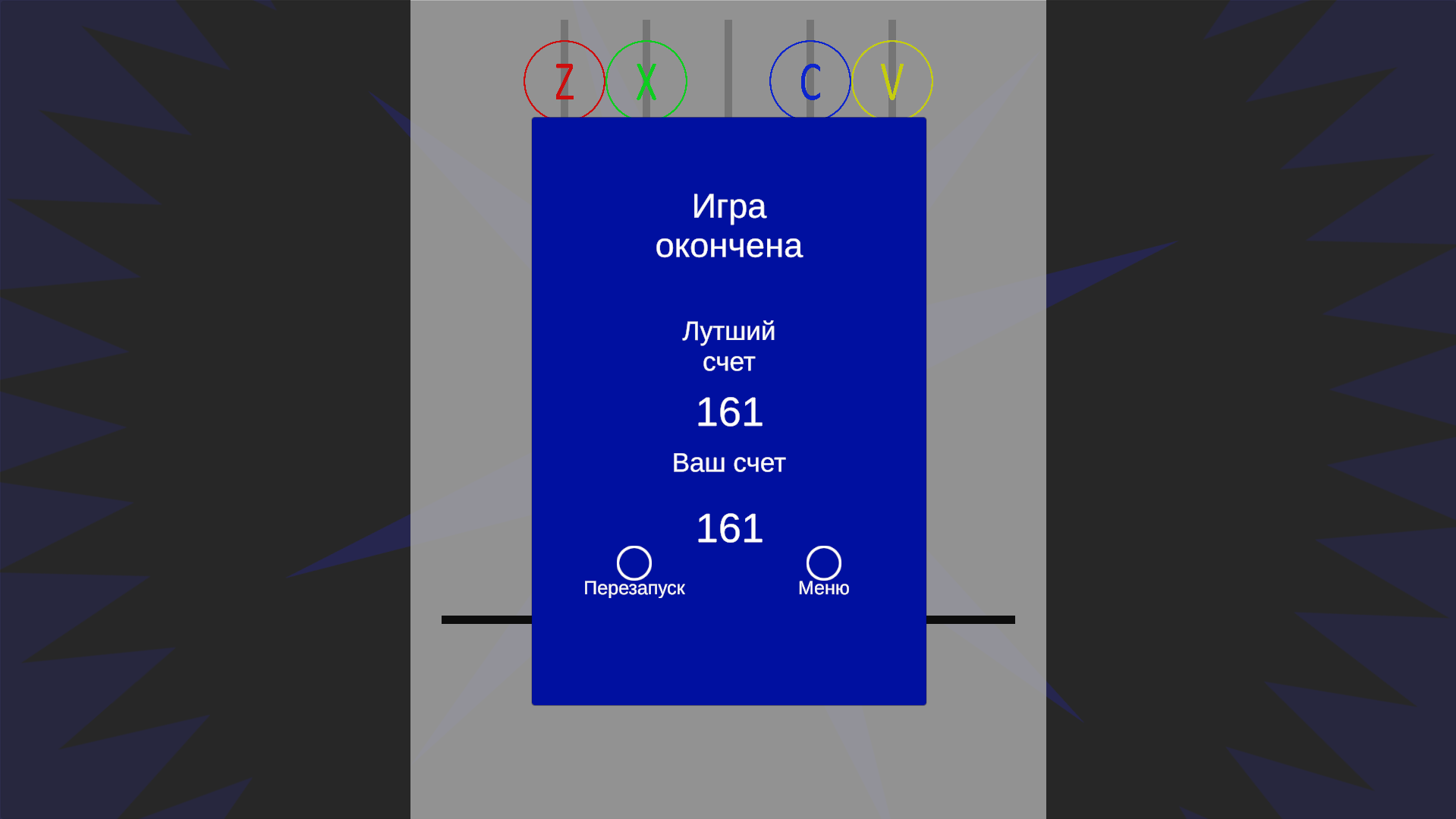


Рисунок 30 – Окно окончание игры

Окно окончание игры появляется при проигрыше или прохождение уровня до конца, на нем можно увидеть лучший и пекущий счет.

В случае поражения игроку предоставляется возможность выйти из игры или же продолжить ее, т.е. начать заново. Для этого в новом открывшемся окне надо либо закрыть его, либо нажать на кнопку «Перезапуск». Интерфейс окна окончания игры представлен на рисунке 30.

Таким образом был спроектирован удобный интерфейс, отвечающий, как всем программным аспектам, так и параметрам разработчиков игр.

# Реализация

# Реализация программного игрового приложения

Реализованная структура программного приложения полностью соответствует структуре, определенной на этапе анализа. В таблице 3 приведено описание основных компонентов программного игрового приложения.

Таблица – 3

Описание компонентов игрового приложения

|  |  |
| --- | --- |
| Компонент | Название |
| Главное меню | Экран, из которого мы можем перейти в настройки и перейти к выбору режима игры. |
| Настройки | Экран, на котором мы можем изменить громкость музыки и звуков, включить или отключить декорации и сбросить весь прогресс игры. |
| Предупреждение | Окно предупреждения появляется при попытке сбросить прогресс игры во избежание случайного удаления. |
| Режим игры | Экран для выбора нужного нам режима игры. |
| Выбор уровня | Экран, на котором мы можем выбрать понравившийся уровень и просмотреть лучей счет и количество попыток. |
| Уровень | Сцена с игровым процессом. |

# Реализация графических объектов

Для создания полноценного графического интерфейса недостаточно обыкновенных форм, поэтому было принято решения создания художественных объектов игры в соответствии со стилистикой игры.

Используемые кнопки, объекты и другое представлены на рисунках 32-39.

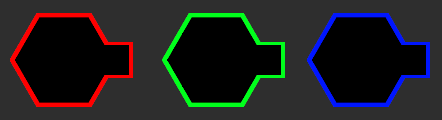


Рисунок 32 – Цветовая схема игрока

Данная цветовая схема используешься для отображения игрока.



Рисунок 33 – Типы боеприпасов

Типы боеприпасов используются при выстреле игроком.

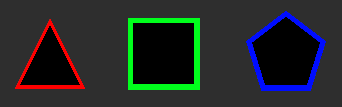


Рисунок 34 – Объекты летящие в игрока

Эти фигуры выступают в качестве объектов, которые должен сбивать игрок.

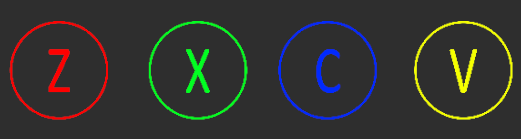


Рисунок 35 – Типы нот

Данные ноты показывают игроку какую кнопку необходимо нажимать.



Рисунок 36 – Кнопка

Приведённые выше кнопки используются в интерфейсе игры.

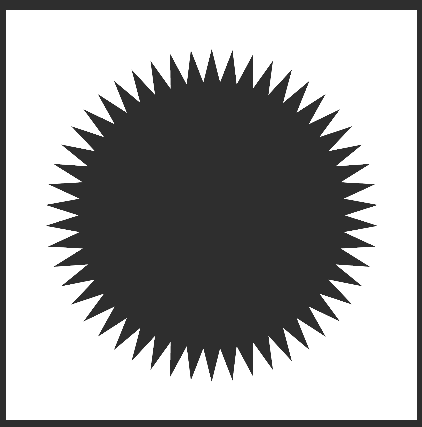
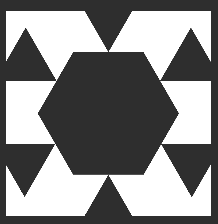


Рисунок 37 – Элементы фона

Эти элементы выступают в качестве боковых декораций.



Рисунок 38 – Элемент фона

Представленные элементы выступают в качестве центральной декорации.

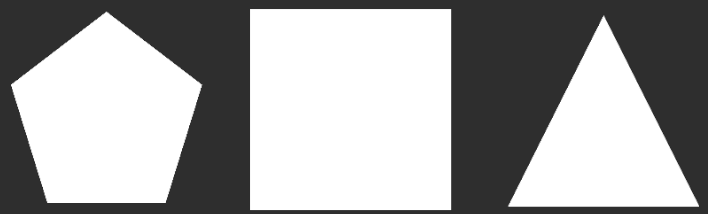


Рисунок 39 – Элементы фона

Игровое приложение помимо названия имеет свой собственную иконку, оформленную в своей стилистике (рисунок 40).

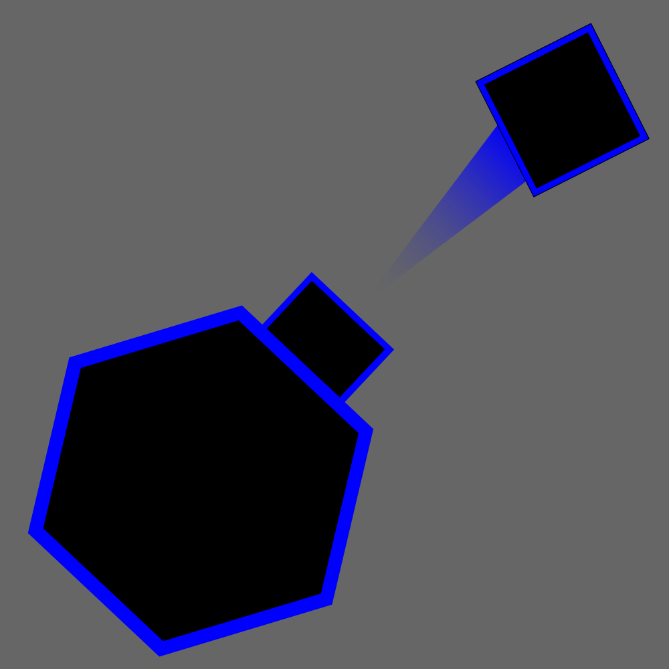


Рисунок 40 – Иконка программного игрового приложения

Иконки, представленные на рисунке 32 - 33, имеют три различных состояния. Они меняться при нажатии на определённые кнопки. Иконки изображённые на рисунке 34 используются для создания объектов, сменяющихся в случайном порядке в бесконечном режиме. Ноты изображённые на рисунке 35 используются для создания нот в ритмичном режиме. Далее показаны художественные объекты (рисунки 37-39), которые имеют так же три состояния. Они изменяются с учетом происходящего в игре.

# Контроль качества программного обеспечения

Тестирование – этап разработки программного обеспечения, позволяющий выявить ситуации, в которых поведение программы не соответствует спецификации. В данной выпускной квалификационной работы были выбраны две методологии тестирования: функциональное и модульное.

# Функциональное тестирование

Функциональное тестирование предполагает составление плана тестирование, который строится на основе вариантов использования и включает в себя описание тестов и их результаты.

Функциональное тестирование позволяет оценить работоспособность каждой из функций разработанного программного обеспечения, не заглядывая в особенности реализации, то есть фактически производится тестирование методом черного ящика. План тестирования приведен в таблице 4.

Таблица – 4

План тестирования

|  |  |  |  |
| --- | --- | --- | --- |
| № | Вариант использования | Тест | Результаты |
| 1 | Отображение интерфейсов | Запуск начального окна | Тест выполнен |
| 2 | Запуск окна настроек | Тест выполнен |
| 3 | Запуск окна предупреждения | Тест выполнен |
| 4 | Запуск окна выбора режима | Тест выполнен |
| 5 | Запуск окна выбора уровня | Тест выполнен |
| 6 | Запуск игровой сцены | Тест выполнен |
| 7 | Вызов окна паузы | Тест выполнен |
| 8 | Вызов окна конца игры | Тест выполнен |
| 9 | Ритмичный режим | Нажатие правильной кнопки в такт музыки | Тест выполнен |
| 10 | Нажатие неправильной кнопки в такт музыки | Тест выполнен |
| 11 | Нажатие правильной кнопки не в такт музыки | Тест выполнен |
| 12 | Игнорирование нажатий | Тест выполнен |
| 13 | Проигрыш уровня | Тест выполнен |
| 14 | Завершение уровня | Тест выполнен |

Продолжение таблицы 4.

|  |  |  |  |
| --- | --- | --- | --- |
| № | Вариант использования | Тест | Результаты |
| 15 | Бесконечный режим | Выстрел правильным цветом в летящий объект | Тест выполнен |
| 16 | Выстрел неправильным цветом в летящий объект | Тест выполнен |
| 17 | Выстрел за приделы игрового пространства | Тест выполнен |
| 18 | Игнорирование выстрела | Тест выполнен |
| 19 | Выбор параметров игры | Включение декораций | Тест выполнен |
| 20 | Выключение декораций | Тест выполнен |
| 21 | Изменение громкости музыки | Тест выполнен |
| 22 | Изменение громкости звуков | Тест выполнен |
| 23 | Сохранение параметров игры | Тест выполнен |

На основе анализа результатов тестирования можно сделать вывод о том, что разработанное программное игровое приложение с использованием мультиплатфоременной среды, работает корректно.

# 5.2. Модульное тестирование

Модульное или unit-тестирование позволяет провести более глубокое исследование и анализ работоспособности отдельных модулей программного обеспечения. При разработке модульных тестов был использован программный модуль FlexUnit, результаты тестирования игрового приложения представлены на рисунке 41.

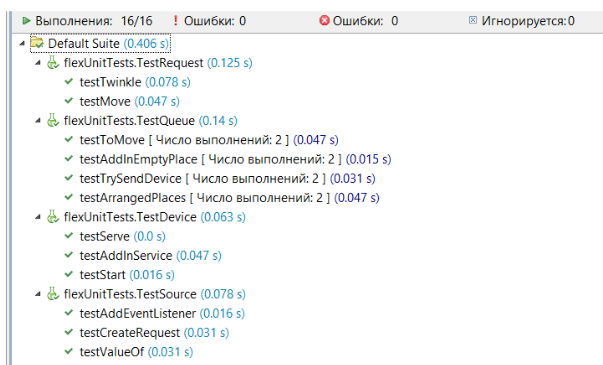


Рисунок 41 – Результаты модульного тестирования

# Планирование разработки и оценка бюджета

Управление проектами – область деятельности, в ходе которой определяются и достигаются цели проекта при балансировании между объемом работ, ресурсами (такими как люди, деньги, труд, материалы, энергия, пространство и др.), временем, качеством и рисками. Ключевым фактором успеха проектного управления является наличие четкого заранее определенного плана.

# Оценка трудоемкости.

Трудоемкость характеризуется перечнем основных этапов и видов работ, которые должны быть выполнены в проекте. В ходе планирования проекта было выделено 29 задач. Расчет трудоемкости проекта, (виды работ и длительность) приведены в таблице 5.

Таблица – 5

Расчет трудоемкости

|  |  |  |  |
| --- | --- | --- | --- |
| № | Название | Вид Задачи | Продолжительность |
| 1. | Начало реализации проекта | Веха | - |
| 2. | Анализ предметной области | Задача | 4 ч. |
| 3. | Анализ классификаций игр | Задача | 2 ч. |
| 4. | Анализ сред разработки игровых приложений | Задача | 2 ч. |
| 5. | Анализ выбранной методологии разработки | Задача | 2 ч. |
| 6. | Анализ программных средств | Задача | 2 ч. |
| 7. | Анализ предметной области завершен | Веха | - |
| 8. | Анализ требований | Фаза | - |
| 9. | Анализ функциональных требований | Задача | 3 ч. |
| 10. | Анализ не функциональных требований | Задача | 3 ч. |
| 11. | Анализ требований завершен | Веха | - |
| 12. | Проектирование | Фаза | - |
| 14. | Проектирование художественных объектов | Задача | 6 ч. |
| 15. | Проектирование интерфейса | Задача | 6 ч. |
| 16. | Проектирование завершено | Веха | - |
| 17. | Реализация | Фаза | - |

Продолжение таблицы 5.

|  |  |  |  |
| --- | --- | --- | --- |
| № | Название | Вид Задачи | Продолжите  льность |
| 18. | Реализация интерфейса | Задача | 20 ч. |
| 19. | Реализация логики | Задача | 40 ч. |
| 20. | Контроль качества ПО | Веха | - |
| 21. | Функциональное тестирование | Задача | 3 ч. |
| 22. | Модульное тестирование | Задача | 3 ч. |
| 23. | Оценка качества ПО | Задача | 6 ч. |
| 24. | Контроль качества завершен | Веха | - |
| 25. | Завершение проекта | Веха | - |
| 26. | Составление программной документации | Задача | 12 ч. |
| 27. | Проект завершен | Веха | - |
| 28. | Конец проекта | Веха | - |
|  | Итог |  | 114 ч. |

Определим необходимые ресурсы для выполнения каждой из задач данного проекта. Ресурсы и затраты приведены в таблице 4. Распределение ресурсов по задачам приведено в таблице 5.

# Организационно-экономическое обоснование проекта

Возможными потребителями данного программного продукта станут игроки, которые будут использовать его для развлечения.

Преимуществом данного проекта послужит уникальный игровой опыт, увеличение потребности ритм игр и малое количество конкурентов.

# Расчет себестоимости

Разработка любого устройства (программного продукта) требует определенных материальных, временных и трудовых затрат, а, следовательно, должна соответственно окупаться.

Себестоимость продукции – это денежное выражение непосредственных затрат предприятия на производство и реализацию продукции.

Себестоимость показывает, во что обходится предприятию выпускаемая им продукция. В себестоимость включаются перенесенные на продукцию затраты прошлого труда (амортизация основных фондов, стоимость сырья, материалов, топлива и других материальных ресурсов) и расходы на оплату труда работников предприятия (заработная плата).

Данный проект потребовал массу усилий на его разработку и создание. В общей сумме ушло порядка 114 часов.

Так же на создание проекта ушли материальные затраты, такие как:

* Электричество;
* Износ личного персонального компьютера;
* Интернет.

# 6.4. Расчет затрат на оплату труда разработчика программного продукта

Заработная плата – это денежная компенсация, которую работник получает в обмен за свой труд.

Фонд заработной платы определяется по формуле (1):

= + П + + Отч руб., (1)

где: – фонд заработной платы;

– основная заработная плата 35 625 руб.;

П – премия;

– уральский коэффициент;

Отч – отчисления в государственные фонды;

Основная заработная плата – это заработная плата, начисленная рабочим и служащим за фактически проработанное время и выполнение работы на предприятии.

Основная заработная плата рассчитывается по формуле (2):

= Т \* , (руб.) (2)

где: Т – трудозатраты;

– тарифная ставка;

=114 \* 312,5 = 35 625 руб.

Премия – это одна из форм поощрения за выдающиеся результаты, достигнутые в какой-либо области деятельности. Премия начисляется в размере 30% от основной заработной платы.

Премия определяется по формуле (3):

П = \* 0,3, (руб.) (3)

П = 35 625 \* 0,3 =10 687,5 руб.

Уральский коэффициент – это надбавка к заработной плате, который устанавливается для работников, трудящихся в районах с особыми климатическими условиями. Уральский коэффициент начисляется в размере 15% от основной заработной платы и премии.

Формула для расчета уральского коэффициента (4):

= ( + П) \* 0,15 руб., (4)

где: – основная заработная плата;

П – премия;

= (35 625 + 10 687,5) \* 0,15 = 6 946,87 руб.

Отчисления на социальные нужды - обязательные отчисления по нормам, установленным законодательством государственного социального страхования, в Фонд социального страхования РФ, Пенсионный фонд РФ, фонды обязательного медицинского страхования от затрат на оплату труда работников, включаемых в себестоимость продукции.

Отчисления на социальные нужды определяются по формуле (5):

Отч = ( + П + ) \* 0,302, (руб.) (5)

где: – уральский коэффициент;

Отч = (35 625 + 10 687,5 + 6 946,87) \* 0,302 = 16 084,33 руб.

Итого, фонд заработной платы составляет:

= 35 625 + 10 687,5 + 6 946,87 + 16 084,33 = 69 343,7 руб.

Заработная плата разработчика определяется по формуле (6):

= + П + , (руб.) (6)

где: – общая заработная плата;

= 35 625 + 10 687,5 + 6 946,87 = 53 259,37 руб.

# 6.5. Расчет общепроизводственных расходов

Общепроизводственные расходы – это расходы на электроэнергию, потребляемую компьютером и принтером, а также амортизационные отчисления.

Затраты на электроэнергию представлены в таблице 6.

Таблица – 6

Затраты на потребляемую электроэнергию

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Вид оборудования | Мощность, КВт | Стоимость  1 КВт, руб. | Время, час | Общая сумма затрат, руб. |
| Компьютер | 0,3 | 3,42 | 114 | 116,96 |
| Итого: |  |  |  | 116,96 |

Итого, затраты на электроэнергию составили 116,96 руб.

Амортизация — перенесение по частям стоимости основных средств и нематериальных активов по мере их физического или морального износа на стоимость производимой продукции.

Норма амортизации – часть стоимости основных средств, выраженная в процентном соотношении годовой суммы амортизации к первоначальной стоимости основных фондов.

Амортизационные отчисления — это денежные средства, предназначенные для возмещения износа предметов, относящихся к основным средствам предприятия. Затраты на амортизационные отчисления представлены в таблице 7.

Таблица – 7

Затраты на амортизационные отчисления

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Вид оборудования | Первоначальная стоимость, руб. | Норма аморт., % | Время работы, час | Аморт. отчисления, руб. |
| Компьютер | 30000 | 13 | 114 | 251,06 |
| Итого: |  |  |  | 251,06 |

Амортизационные отчисления определяются по формуле (7):

АО = ( \* НА/100 \* ) / руб., (7)

где: АО – амортизационные отчисления;

– первоначальная стоимость;

НА – норма амортизации;

– время работы;

– годовой фонд рабочего времени, равный 1972ч.

АО = (30000 \* 13/100 \* 114) / 1972 = 225,46 руб.

Общепроизводственные расходы рассчитываются исходя из затрат на электроэнергию и суммы амортизационных отчислений (8):

= + АО руб., (8)

где: – общепроизводственные расходы;

– затраты на электроэнергию;

= 116,96 + 251,06 = 368,02 руб.

Итого, сумма общепроизводственных расходов составила 368,02 руб.

# Расчёт себестоимости программного продукта

Себестоимость — стоимостная оценка текущих затрат предприятия на производство и реализацию продукции.

Себестоимость рассчитывается по формуле (9):

С/С = + руб. (9)

где: С/С – себестоимость;

– общая заработная плата;

– общепроизводственные расходы;

С/С = 53 259,37 + 368,02 = 53 627,39 руб.

# Расчёт цены программного изделия

Прибыль на программное изделие составляет 25%. НДС на программу составляет 20%.

Прибыль Пр, руб., рассчитывается по формуле (10):

Пр = С/С \* 25% руб. (10)

Пр = 53 627,39 \* 25% = 13 406,85 руб.

Цену расчетную , руб., рассчитывают по формуле (11):

= С/С + Пр, (руб.) (11)

= 53 627,39 + 13 406,85 = 67 033,97 руб.

НДС, руб., рассчитывают по формуле (12):

НДС = \* 20%, (руб.) (12)

НДС = 67 033,97 \* 20% = 13 406,79 руб.

Цена отпускная , руб., рассчитывают по формуле (13):

= + НДС руб. (13)

= 67 033,97 + 13 406,79 = 80 440,76 руб.

# Расчёт отпускной цены продукта.

|  |  |  |
| --- | --- | --- |
| № | Элементы затрат | Сумма |
| 1 | Основная заработная плата | 35 625 руб. |
| 2 | Отчисления в государственные фонды | 16 084,33 руб. |
| 3 | Общепроизводственные расходы | 368,02 руб. |
| 4 | Себестоимость | 53 627,39 руб. |
| 5 | Прибыль | 13 406,85 руб. |
| 6 | Расчетная цена | 67 033,97 руб. |
| 7 | НДС | 13 406,79 руб. |
|  | Отпускная цена | 80 440,76 руб. |

Отпускная цена на изделие составила 80 440,76 руб.

# Заключение

В ходе выполнения выпускной квалификационной работы было разработано программное игровое приложение с использованием мультиплатформенной среды Unity.

Был проведен анализ существующих игровых приложений, схожих с созданным, и поиск готовых решений для его построения. Также было проведено проектирование, позволившее определить окончательную структуру игрового приложения, а затем был спроектирован простой и удобный пользовательский интерфейс.

Программная игра реализована средствами мультиплатформенного движка Unity, языка программирования C#, а графические объекты рисовались в графическом редакторе Adobe Photoshop CC.

На этапе контроля качества программного обеспечения было проведено функциональное тестирование. На основе анализа результатов тестирования можно сделать вывод о том, что разработанное игровое приложение и все входящие в его состав объекты работают корректно. Также был проведен анализ качества программного кода, показавший хороший результат.

В ходе выполнения работы также было выполнено планирование проекта. Длительность выполнения проекта равна 114 часам, а отпускная цена проекта составила 80 440,76 руб.

Таким образом, в результате выполнения выпускной квалификационной работы были разработаны программные средства, отвечающие всем требованиям технического задания.

# Список использованных источников

1. Могилев, А.В. Информатика. / Н.И. Пак, Е.К. Хённер – М.: Академия, 2004. – 848 c.

2. Шеллинг, Т. Стратегия конфликта. / М.: ИРИСЭН, 2007. – 376 с.

3. Диксит, А. Теория игр: искусство стратегического мышления в бизнесе и жизни/ Б. Нейлбафф, – М.: Манн, Иванов и Фербер, 2014. – 464 с.

4. Мураховский, В.И. Компьютерная графика – М.: АСТ-ПРЕСС 2002 – 638 с.

5. Корабельникова, Г.Б. Adobe Photoshop в теории и на практике. / Ю.А.Гурский – М.: Новое Знание, 2002. – 528 с.

6. Макарова, Н. В. Информатика / Л.А. Матвеев, В.Л. Бройдо – М.: Финансы и Статистика, 2002. – 768 с.

7. Пауэль, Т.А. Справочник программиста. / Т.А. Пауэль, Д. Уитворт – М.: АСТ, 2005. – 470 с.

8. Симонович, С.В. Практическая информатика. / Г.А. Евсеев – М.: АСТ-ПРЕСС КНИГА, 2005. – 480 с.

9. Морелос-Сарагоса, Р. Искусство помехоустойчивого кодирования. Методы, алгоритмы, применение – М.: ТЕХНОСФЕРА, 2006. – 320 с.

10. Блэкманн, С. Моделирование с помощью Unity – М.: АСТ- ПРЕСС, 2011. – 147 с.

11. Голдстон, В. Предметы первой необходимости для создания игр на Unity – М.: Академия, 2009 – 201 с.

12. Игошев, Б. ЭВТ: знакомимся, делаем, играем. / М. Галагузова, Д. Комский – М.:Молодая гвардия, 1989. – 156 с.

13. Лысовский, А. Как сибирские программисты заставили весь мир играть. / А.Егошин, А.Бочкарев - М.:Русский Репортер №10, 2012. - 61с.

# Приложение А

## Техническое задание

А.1. Общие сведения

А.1.1. Полное наименование системы «Разработка игры в жанре ритм на Unity»

А.1.2. Условное наименование системы «Pulce»

А.1.3. Разработчик

ФИО: Зыков Александр Анатольевич

Телефон: 8 (912)-767-42-09

Электронная почта: aa.zykov228@gmail.com

А.1.4. Заказчик

ФИО: Морозов Артём Михайлович

Организация: «Администрация муниципального образования Воткинский район»

Телефон: 7 (919) 459 65 55

Электронная почта: morozov\_am@vot.udmr.ru

А.2. Назначение и цели создания системы

Система предназначена для получения уникального игрового опыта. Система должна предоставлять автоматизацию организационно-управленческих процессов, хранения информации и её представление в удобном пользователю виде.

Ритм игра будет представлять несколько режимов игры в первом из которых необходимо нажимать кнопки в такт музыке, а во втором стрелять в летящие на игрока геометрические объекты.

Целями создания системы являются:

* Предоставление уникального игрового опыта;
* Развитие чувства ритма
* Улучшение реакции;

А.3. Характеристика объекта автоматизации

Деятельность игрока при использовании продукта предполагает следующие действия:

* Рыбор режима игры;
* Влияние на игровое пространство.

А.4. Требования к системе

А.4.1. Перечень подсистем, их назначение и основные характеристики

А.4.4. Требование к эргономике интерфейса

Система должна предоставлять удобный, интуитивно понятный пользовательский интерфейс. Все действия должны легко запоминаться и не требовать выполнения дополнительных команд.

Интерфейс не должен утомлять зрение. Он должен быть рассчитан на длительную работу пользователей с приложением в течение рабочего дня.

Формы должны быть оптимально загружены элементами управления

Все страницы Системы должны быть оформлены исходя из одного общего стиля. Интерфейс Системы должен обеспечивать:

* Быструю и удобную навигацию;
* Визуальное структурирование информации;
* Визуальное отображение информации.

А.4.5. Требования к видам обеспечения

А.4.5.1. Техническое обеспечение

Для работоспособности пользователю необходимо обладать мышкой и клавиатурой для управления, а также для наилучшего игрового опыта колонки для воспроизведения музыки и звуков.

Таблица – А.1

Необходимое техническое обеспечение

|  |  |
| --- | --- |
| Компонент | Требование |
| Процессор | Процессор с тактовой частотой 500 МГц или выше |
| Оперативная память | ОЗУ объёмом 256 МБ или больше |
| Физическая память | 2 ГБ свободного дискового пространства |
| Операционная система | Операционные системы Windows XP (32-разрядная), WindowsVista с пакетом обновления 1, WindowsServer 2003 R2 с установленным MSXML 6.0, WindowsServer 2008 (32- или 64-разрядная), Windows 7 или более поздних версий. |

А.5. Состав и содержание работ по созданию системы

А.5.1. Изучение предметной области

А.5.2. Составление концептуальной схемы в соответствии с заданными требованиями к предметной области

А.5.3. Разработка дизайна сайта, макетирование

А.5.4. Разработка программного продукта

А.5.4.1. Разработка элементов интерфейса

А.5.4.2. Разработка C#-скриптов

А.5.5. Тестирование программного продукта

А.5.6. Подготовка документации

А.5.7. Создание отчёта

А.6. Требования к документированию

Техническое задание составлено в соответствии со стандартом ГОСТ 34.602-89 (состав и содержание) и ГОСТ 19.201-78 (техническое задание – требования к содержанию и оформлению)

Руководитель дипломного проекта \_\_\_\_\_\_\_\_\_ Кузьминых А.С.

# Приложение Б

## Программный код

Б.1. Lane

public class Lane : MonoBehaviour

{

public Melanchall.DryWetMidi.MusicTheory.NoteName noteRestriction;

public KeyCode input;

public GameObject notePrefab;

List<Note> notes = new List<Note>();

public List<double> timeStamps = new List<double>();

int spawnIndex = 0;

int inputIndex = 0;

public GameObject InfParticleBlue;

public GameObject InfParticleGrin;

public GameObject InfParticleRed;

public GameObject InfParticleYellow;

public static int colorCart;

void Start()

{

colorCart = 2;

}

public void SetTimeStamps(Melanchall.DryWetMidi.Interaction.Note[] array)

{

foreach (var note in array)

{

if (note.NoteName == noteRestriction)

{

var metricTimeSpan = TimeConverter.ConvertTo<MetricTimeSpan>(note.Time, SongManager.midiFile.GetTempoMap());

timeStamps.Add((double)metricTimeSpan.Minutes \* 60f + metricTimeSpan.Seconds + (double)metricTimeSpan.Milliseconds / 1000f);

}

}

}

void Update()

{

if (spawnIndex < timeStamps.Count)

{

if (SongManager.GetAudioSourceTime() >= timeStamps[spawnIndex] - SongManager.Instance.noteTime)

{

var note = Instantiate(notePrefab, transform);

notes.Add(note.GetComponent<Note>());

note.GetComponent<Note>().assignedTime = (float)timeStamps[spawnIndex];

spawnIndex++;

}

}

if (inputIndex < timeStamps.Count)

{

double timeStamp = timeStamps[inputIndex];

double marginOfError = SongManager.Instance.marginOfError;

double audioTime = SongManager.GetAudioSourceTime() - (SongManager.Instance.inputDelayInMilliseconds / 1000.0);

if (Input.GetKeyDown(input))

{

if (Math.Abs(audioTime - timeStamp) < marginOfError)

{

Hit();

if (notes[inputIndex].gameObject.tag == "red")

{

Instantiate(InfParticleRed, transform.position, Quaternion.identity);

colorCart = 0;

}

if (notes[inputIndex].gameObject.tag == "blue")

{

Instantiate(InfParticleBlue, transform.position, Quaternion.identity);

colorCart = 2;

}

if (notes[inputIndex].gameObject.tag == "grin")

{

Instantiate(InfParticleGrin, transform.position, Quaternion.identity);

colorCart = 1;

}

if (notes[inputIndex].gameObject.tag == "yellow")

{

Instantiate(InfParticleYellow, transform.position, Quaternion.identity);

colorCart = 3;

}

Destroy(notes[inputIndex].gameObject);

inputIndex++;

}

else

{

Miss();

}

}

if (timeStamp + marginOfError <= audioTime)

{

Miss();

inputIndex++;

}

}

}

private void Hit()

{

ScoreManager.Hit();

}

private void Miss()

{

ScoreManager.Miss();

}

}

Б.2. Note

public class Note : MonoBehaviour

{

double timeInstantiated;

public float assignedTime;

void Start()

{

timeInstantiated = SongManager.GetAudioSourceTime();

}

// Update is called once per frame

void Update()

{

double timeSinceInstantiated = SongManager.GetAudioSourceTime() - timeInstantiated;

float t = (float)(timeSinceInstantiated / (SongManager.Instance.noteTime \* 2));

GetComponent<SpriteRenderer>().enabled = true;

if (t > 1)

{

Destroy(gameObject);

}

else

{

transform.localPosition = Vector3.Lerp(Vector3.up \* SongManager.Instance.noteSpawnY, Vector3.up \* SongManager.Instance.noteDespawnY, t);

GetComponent<SpriteRenderer>().enabled = true;

}

}

}

Б.3. ScoreManager

public class ScoreManager : MonoBehaviour

{

public static ScoreManager Instance;

public AudioSource hitSFX;

public AudioSource missSFX;

public AudioSource \_currentAudio;

public TMPro.TextMeshPro comboText;

public TMPro.TextMeshPro scoreText;

public TMPro.TextMeshPro bestScoreText;

public TMPro.TextMeshPro hpText;

public TMPro.TextMeshPro scoreGameOverText;

public GameObject GameOverMenuUI;

public GameObject ScorsMenuUI;

public int NamderLevlId;

static int comboScore;

static int Score;

static int hp;

public static int gameOvee;

private JsonFile1 \_jsonFile1 = new JsonFile1();

private string \_path;

void Start()

{

Instance = this;

comboScore = 0;

Score = 0;

hp = 50;

gameOvee = 0;

AddProgresAttempt();

}

public static void Hit()

{

if (comboScore <8)

{

comboScore += 1;

}

if (hp < 100)

{

hp += 5;

}

Score += comboScore;

Instance.hitSFX.Play();

}

public static void Miss()

{

comboScore = 0;

hp -= 10;

Instance.missSFX.Play();

}

private void Update()

{

if (hp <= 0)

{

if (gameOvee == 0)

{

GameOver();

}

}

comboText.text = comboScore.ToString();

scoreText.text = Score.ToString();

hpText.text = hp.ToString();

if (SongManager.GetAudioSourceTime() > \_currentAudio.clip.length)

{

if (gameOvee == 0)

{

GameEnd();

}

}

}

public void GameOver()

{

scoreGameOverText.text = Score.ToString();

GameOverMenuUI.SetActive(true);

ScorsMenuUI.SetActive(false);

\_currentAudio?.Stop();

gameOvee = 1;

AddProgresScore();

}

public void GameEnd()

{

scoreGameOverText.text = Score.ToString();

GameOverMenuUI.SetActive(true);

ScorsMenuUI.SetActive(false);

\_currentAudio?.Stop();

gameOvee = 1;

AddProgresScore();

}

public void AddProgresScore()

{

\_path = Application.streamingAssetsPath + "/" + "DateRitm.json";

\_jsonFile1 = JsonUtility.FromJson<JsonFile1>(File.ReadAllText(\_path));

if (NamderLevlId == 1)

{

if (\_jsonFile1.BestScorLevl1 <= Score)

{

\_jsonFile1.BestScorLevl1 = Score;

}

bestScoreText.text = \_jsonFile1.BestScorLevl1.ToString();

}

if (NamderLevlId == 2)

{

if (\_jsonFile1.BestScorLevl2 <= Score)

{

\_jsonFile1.BestScorLevl2 = Score;

}

bestScoreText.text = \_jsonFile1.BestScorLevl2.ToString();

}

File.WriteAllText(\_path, JsonUtility.ToJson(\_jsonFile1));

}

public void AddProgresAttempt()

{

\_path = Application.streamingAssetsPath + "/" + "DateRitm.json";

\_jsonFile1 = JsonUtility.FromJson<JsonFile1>(File.ReadAllText(\_path));

if (NamderLevlId == 1)

{

\_jsonFile1.AttemptLevl1 += 1;

}

if (NamderLevlId == 2)

{

\_jsonFile1.AttemptLevl2 += 1;

}

File.WriteAllText(\_path, JsonUtility.ToJson(\_jsonFile1));

}

public class JsonFile1

{

public int BestScorLevl1;

public int AttemptLevl1;

public int BestScorLevl2;

public int AttemptLevl2;

}

}

Б.4. SongManager

public class SongManager : MonoBehaviour

{

public static SongManager Instance;

public AudioSource audioSource;

public Lane[] lanes;

public float songDelayInSeconds;

public double marginOfError; // in seconds

public int inputDelayInMilliseconds;

public string fileLocation;

public float noteTime;

public float noteSpawnY;

public float noteTapY;

public float noteDespawnY

{

get

{

return noteTapY - (noteSpawnY - noteTapY);

}

}

public static MidiFile midiFile;

// Start is called before the first frame update

[Obsolete]

void Start()

{

Instance = this;

if (Application.streamingAssetsPath.StartsWith("http://") || Application.streamingAssetsPath.StartsWith("https://"))

{

StartCoroutine(ReadFromWebsite());

}

else

{

ReadFromFile();

}

}

[Obsolete]

private IEnumerator ReadFromWebsite()

{

using (UnityWebRequest www = UnityWebRequest.Get(Application.streamingAssetsPath + "/" + fileLocation))

{

yield return www.SendWebRequest();

if (www.isNetworkError || www.isHttpError)

{

Debug.LogError(www.error);

}

else

{

byte[] results = www.downloadHandler.data;

using (var stream = new MemoryStream(results))

{

midiFile = MidiFile.Read(stream);

GetDataFromMidi();

}

}

}

}

private void ReadFromFile()

{

midiFile = MidiFile.Read(Application.streamingAssetsPath + "/" + fileLocation);

GetDataFromMidi();

}

public void GetDataFromMidi()

{

var notes = midiFile.GetNotes();

var array = new Melanchall.DryWetMidi.Interaction.Note[notes.Count];

notes.CopyTo(array, 0);

foreach (var lane in lanes) lane.SetTimeStamps(array);

Invoke(nameof(StartSong), songDelayInSeconds);

}

public void StartSong()

{

audioSource.Play();

}

public static double GetAudioSourceTime()

{

return (double)Instance.audioSource.timeSamples / Instance.audioSource.clip.frequency;

}

void Update()

{

}

}