МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ  
Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ   
ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных технологий

Кафедра Информационных систем и технологий

Специальность 1-98 01 03 «Программное обеспечение информационной безопасности мобильных систем»

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА КУРСОВОГО ПРОЕКТА**

по дисциплине «Программирование мобильных систем»

Тема «Мобильное приложение для заказа такси»

**Исполнитель**

студент 3 курса 8 группы \_\_\_\_\_\_\_\_\_\_\_\_\_\_ Е.А. Демидюк

подпись, дата

**Руководитель**

ассистент \_\_\_\_\_\_\_\_\_\_\_\_\_\_ Н. И. Уласевич

должность, учен. степень, ученое звание подпись, дата

Допущен(а) к защите \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

дата, подпись

Курсовой проект защищен с оценкой

Руководитель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_ Н. И. Уласевич

подпись дата инициалы и фамилия

**Содержание**

[Введение 3](#_Toc166629113)

[1. Постановка задачи 4](#_Toc166629114)

[2. Аналитический обзор аналогов 7](#_Toc166629115)

[2.1 Программное средство «taxi.yandex.by» 7](#_Toc166629116)

[2.2 Программное средство «135.by» 8](#_Toc166629117)

[2.3 Программное средство «taximaxim.by» 9](#_Toc166629118)

[2.4 Вывод анализа требований 9](#_Toc166629119)

[3. Проектирование приложения 10](#_Toc166629120)

[3.1 Проектирование базы данных 10](#_Toc166629121)

[3.2 Общая структура 12](#_Toc166629122)

[3.3 Вывод по разделу 13](#_Toc166629123)

[4. Реализация приложения 14](#_Toc166629124)

[4.1 Технические средства разработки 14](#_Toc166629125)

[4.2 Разработка мобильного приложения 15](#_Toc166629126)

[5. Тестирование приложения 16](#_Toc166629127)

[5.1 Вывод по разделу 18](#_Toc166629128)

[6. Краткое описание приложения для демонстрации 19](#_Toc166629129)

[6.1 Вывод по разделу 19](#_Toc166629130)

[7. Руководство пользователя 20](#_Toc166629131)

[7.1 Вывод по разделу 24](#_Toc166629132)

[Заключение 25](#_Toc166629133)

# **Введение**

В современном мире мобильные приложения для заказа такси становятся неотъемлемой частью повседневной жизни, обеспечивая пользователям удобство, безопасность и эффективность в перемещениях. В этом контексте разработка качественного программного обеспечения играет ключевую роль в успешной оперативности и управлении таксопарком.

Выбором для разработки нашего проекта станет фреймворк Flutter, предоставляющий широкие возможности для создания кроссплатформенных мобильных приложений. Сочетание гибкости и производительности Flutter делает его идеальным инструментом для создания инновационного приложения для управления таксопарком.

Цель данного курсового проекта заключается в разработке мобильного приложения для управления таксопарком с использованием фреймворка Flutter. Мы сосредоточимся на проектировании и реализации функционала, который включает в себя учет водителей и автомобилей, систему заказа такси, а также инструменты аналитики и отчетности.

В результате выполнения проекта мы получим готовое мобильное приложение, отвечающее современным стандартам дизайна и функциональности, а также удовлетворяющее потребностям современных таксопарков. Этот проект позволит нам не только приобрести ценный опыт в разработке мобильных приложений с использованием Flutter, но и предоставит практический опыт в создании комплексных программных решений для сферы такси.

1. **Постановка задачи**

Целью данного проекта является разработка кроссплатформенного мобильного приложения для управления таксопарком на основе фреймворка Flutter. Приложение должно обеспечивать эффективное управление водителями и автомобилями, а также удобный заказ такси для клиентов.

Для достижения поставленной цели необходимо выполнить следующие задачи:

* Провести анализ потребностей и требований таксопарков и их клиентов;
* изучить особенности рынка такси и конкурентное окружение;
* разработать структуру базы данных для хранения информации о водителях, автомобилях и заказах;
* спроектировать пользовательский интерфейс приложения с учетом удобства использования для всех категорий пользователей;
* написать код приложения на языке программирования Dart с использованием фреймворка Flutter;
* реализовать функции регистрации и авторизации пользователей, управления данными о водителях и автомобилях, а также систему заказа такси;
* провести тестирование приложения на различных устройствах и операционных системах;
* выявить и устранить возможные ошибки и недочеты в работе приложения;
* подготовить приложение к выпуску в продакшн;
* обеспечить поддержку и обновления приложения после его запуска.

Ожидаемыми результатами выполнения проекта являются:

* Разработанное мобильное приложение для управления таксопарком, соответствующее всем поставленным требованиям;
* полноценно функционирующая система, позволяющая водителям эффективно управлять своей работой и клиентам удобно заказывать такси;
* успешное внедрение приложения на рынок и его популяризация среди пользователей.

Важным аспектом разработки таксопарка является математическое обоснование ключевых операций, таких как расчет стоимости поездки, управление данными водителей и автомобилей, а также анализ статистики. Это включает в себя формулы для расчета стоимости поездки, алгоритмы управления данными и другие математические аспекты.

На основе теоретического анализа и математического обоснования проведем моделирование технических объектов системы. Это включает в себя диаграммы классов, диаграммы последовательности и другие модели, отражающие взаимодействие между различными компонентами системы.

Рассмотрим ключевые функции программного средства, обеспечивающие взаимодействие администратора и клиента с таксопарком.

Функции администратора таксопарка:

* Просмотр информации о водителях и клиентах таксопарка;
* блокировка водителя или клиента таксопарка;
* просмотр истории заказов таксопарка.

Функции клиента/пользователя:

* Регистрация и авторизация: Пользователи должны иметь возможность создавать аккаунт и входить в систему, чтобы воспользоваться услугами такси и просмотреть историю своих заказов;
* восстановление пароля от аккаунта;
* заказ такси: Клиентам предоставляется функционал для заказа такси указав адрес;
* просмотр истории заказов: Пользователи должны иметь доступ к истории своих заказов, чтобы контролировать свои поездки и расходы;
* редактирование своего профиля: Клиентам предоставляется возможность редактировать свои личные данные, в том числе контактную информацию, для удобства использования сервиса.

Функции водителя:

* Регистрация и авторизация: Водители должны иметь возможность создавать аккаунт и входить в систему, чтобы воспользоваться услугами такси и просмотреть историю своих заказов;
* восстановление пароля от аккаунта;
* редактирование своего профиля: Водителям предоставляется возможность редактировать свои личные данные, в том числе контактную информацию, для удобства использования сервиса.
* просмотр истории заказов: Водители должны иметь доступ к истории своих заказов, чтобы контролировать свои поездки и расходы;
* выполнение заказов.

Диаграмма вариантов использования для администратора должна включать в себя действия связанные с контролем самого приложения и иметь наивысшие уровни действия над другими. Пользователь или гость имеет ограничение по действиям связанные с конкретной тематикой приложения.

Диаграммы вариантов использования для администратора, пользователя, гостя можно увидеть на рисунке 1.1.

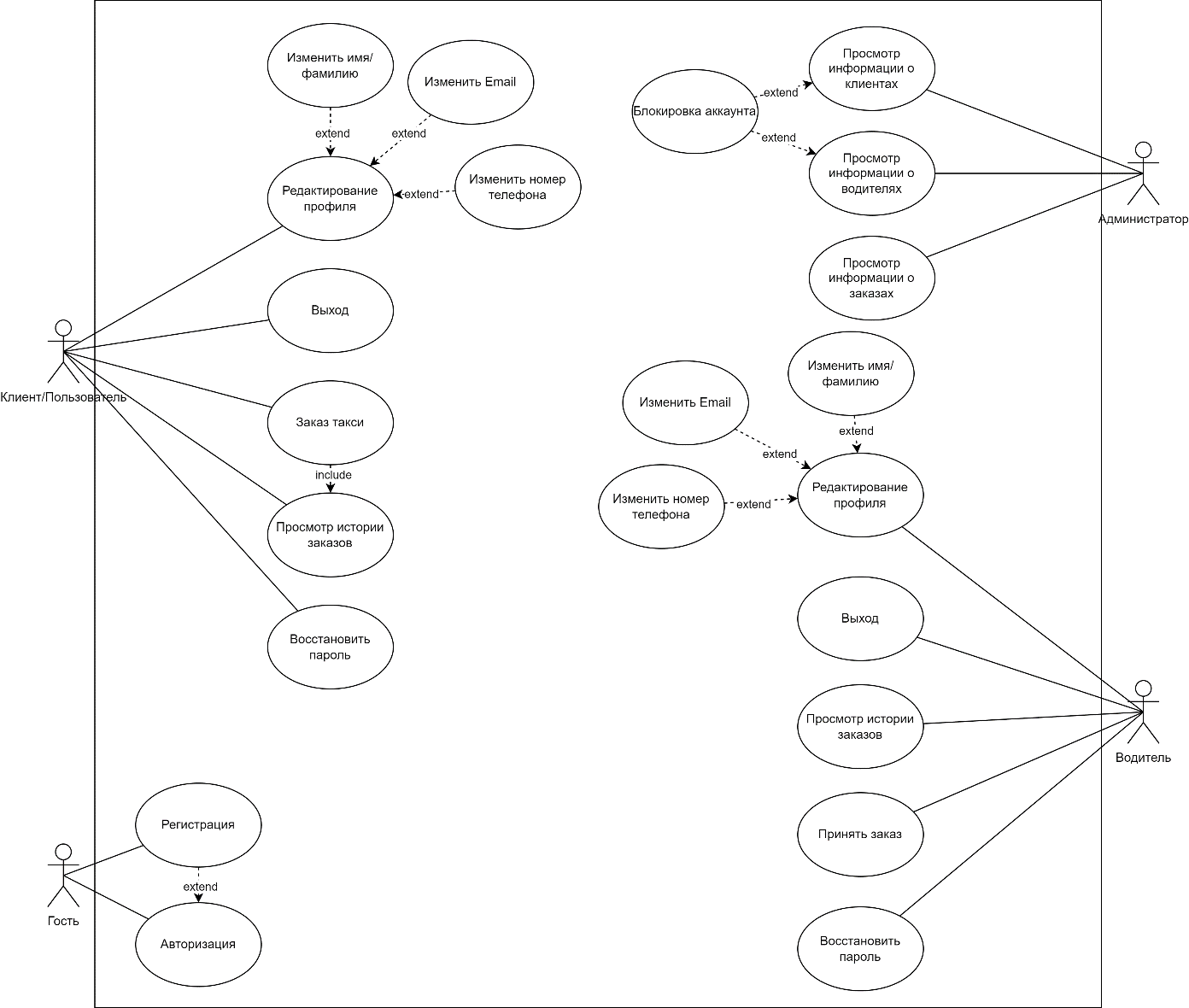


Рисунок 1.1 – Диаграмма вариантов использования

**Вывод по постановке задачи:**

В результате анализа требований к программному средству были определены основные функциональные и нефункциональные требования к мобильному приложению для управления таксопарком на основе фреймворка Flutter. Это включает в себя не только основные функции приложения, такие как регистрация пользователей, управление данными о водителях и автомобилях, система заказа такси и отслеживание заказов, но и необходимость обеспечения высокой производительности, безопасности и удобства использования.

1. **Аналитический обзор аналогов**

В наши дни множество подобных программ можно найти на сайтах с тематикой по данному профилю. Перед тем как приступить к работе требовалось провести анализ и прочитать соответствующие статьи по данной теме. В ходе поиска данной литературы мною были найдены прототипы компаний таксопарков.

* 1. **Программное средство «taxi.yandex.by»**

На рисунке 2.1 предствлен скриншот веб-приложения «taxi.yandex.by».

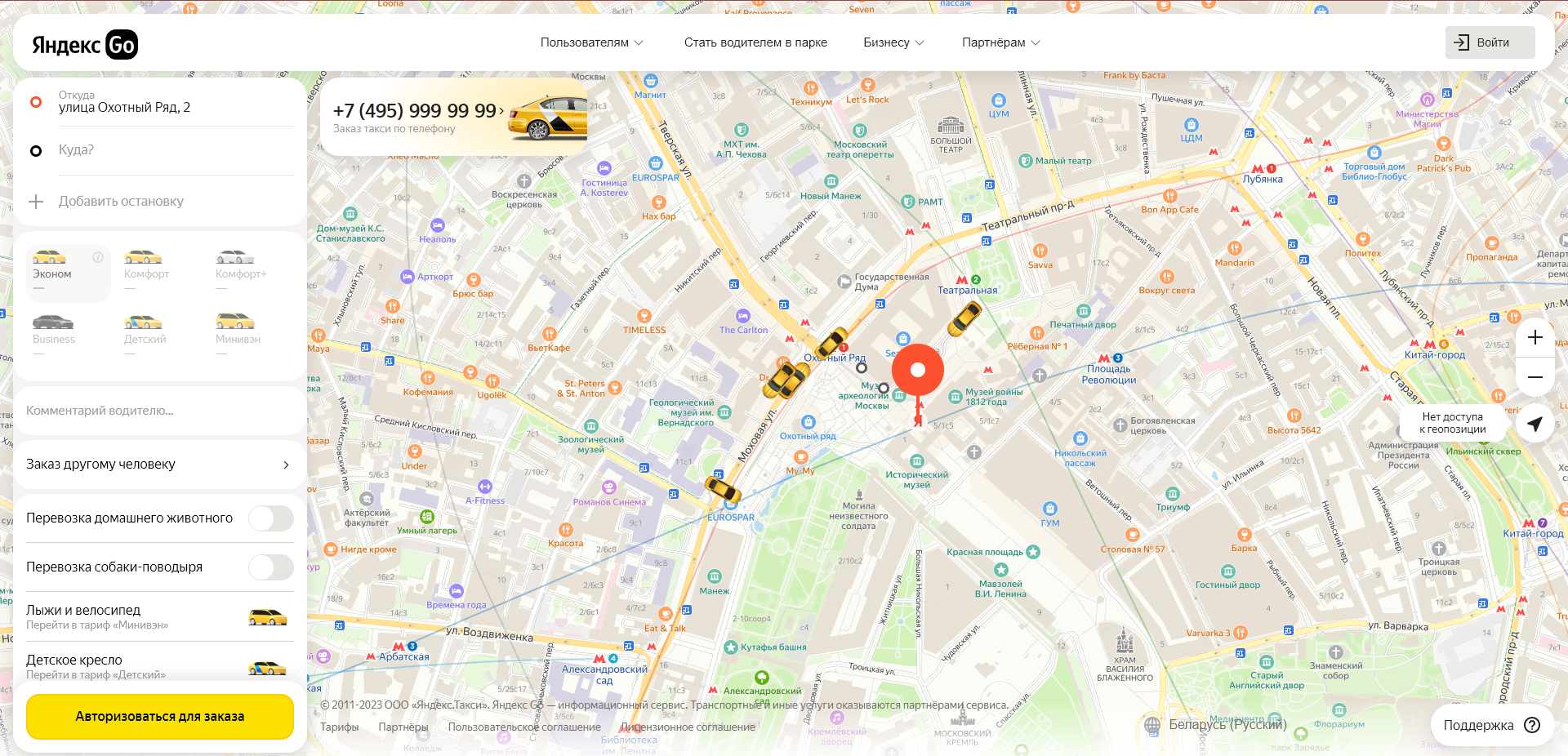


Рисунок 2.1 – Скриншот аналога веб-приложения «taxi.yandex.by»

На этом сайте можно заказать необходимую машину по введенным пользователем данным. У каждого пользователя есть доступ к личному кабинету, в котором он может просмотреть историю поездок и информацию аккаунта. [1]

Проанализировав «taxi.yandex.by», можно выделить некоторые плюсы и минусы программного средства.

Основные плюсы:

* возможность выбрать точку подачи с помощью карты;
* возможность выбрать предпочитаемый тариф поездки;
* возможность просмотреть историю поездок.

Основные минусы:

* много пустого пространства на панели меню;
* нет возможности выбрать предпочитаемую машину;
* слишком светлые тона.
  1. **Программное средство «135.by»**

На рисунке 2.2 предствлен скриншот веб-приложения «135.by».

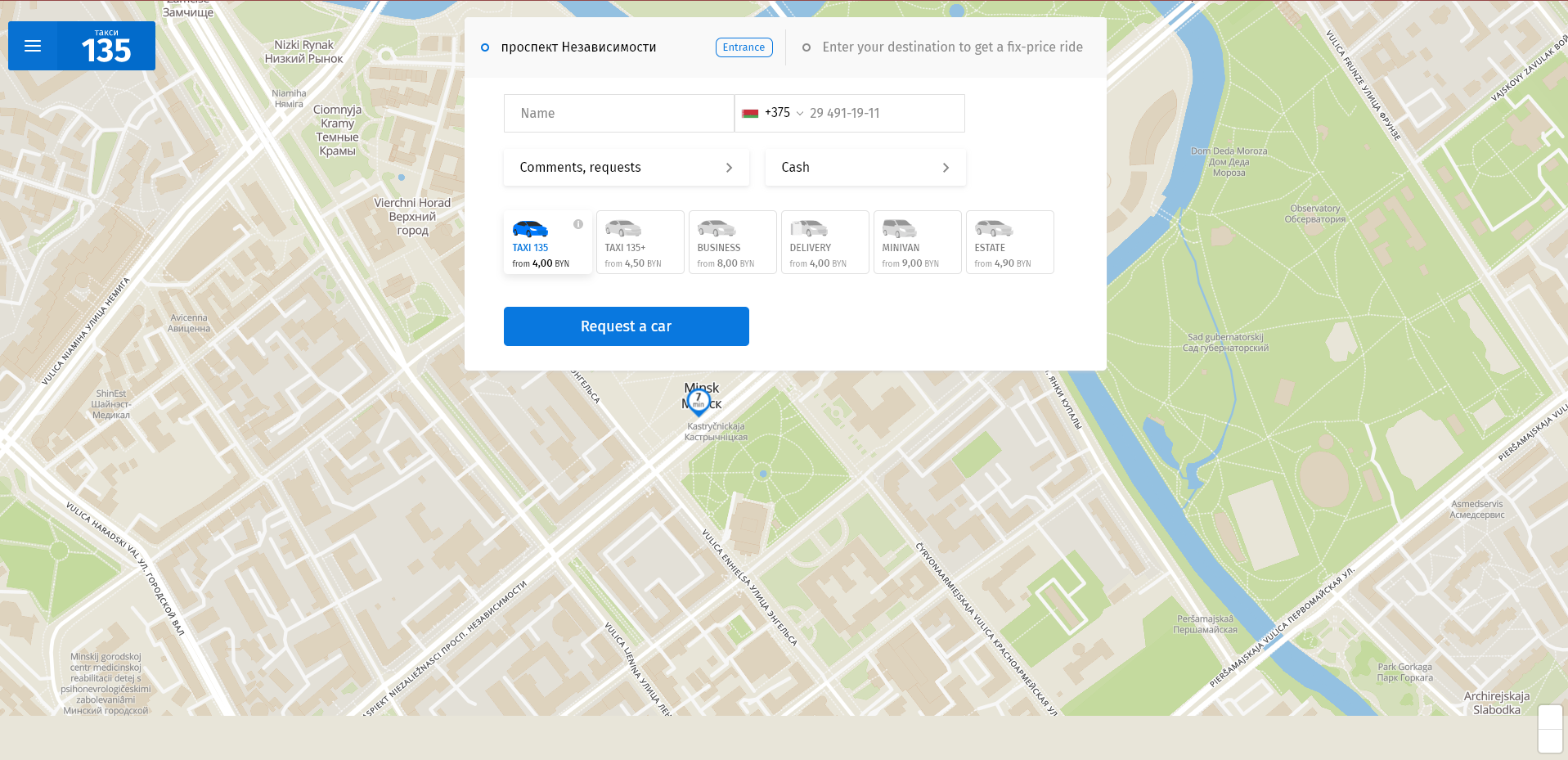


Рисунок 2.2 – Скриншот аналога веб-приложения «135.by»

«135.by» — приложение, похожее на «taxi.yandex.by». На данном сайте можно заказать такси используя поля для ввода адреса. Нету возможности завести личный профиль пользователя. Присутствует карта на заднем фоне, которая покажет маршрут поездки. [2]

Проанализировав «135.by», можно выделить некоторые плюсы и минусы программного средства.

Основные плюсы приложения:

* возможность просмотреть маршрут поездки;
* возможность просмотреть историю поездок;
* возможность выбора тарифа поездки;
* отзывчивый интерфейс.

Основные минусы:

* отсутствует возможность выбора точки подачи на карте;
* отсутствие возможности оставить отзыв о поездке;
* отсутствие личного профиля пользователя.
  1. **Программное средство «taximaxim.by»**

На рисунке 2.3 представлен скриншот веб-приложения «taximaxim.by».

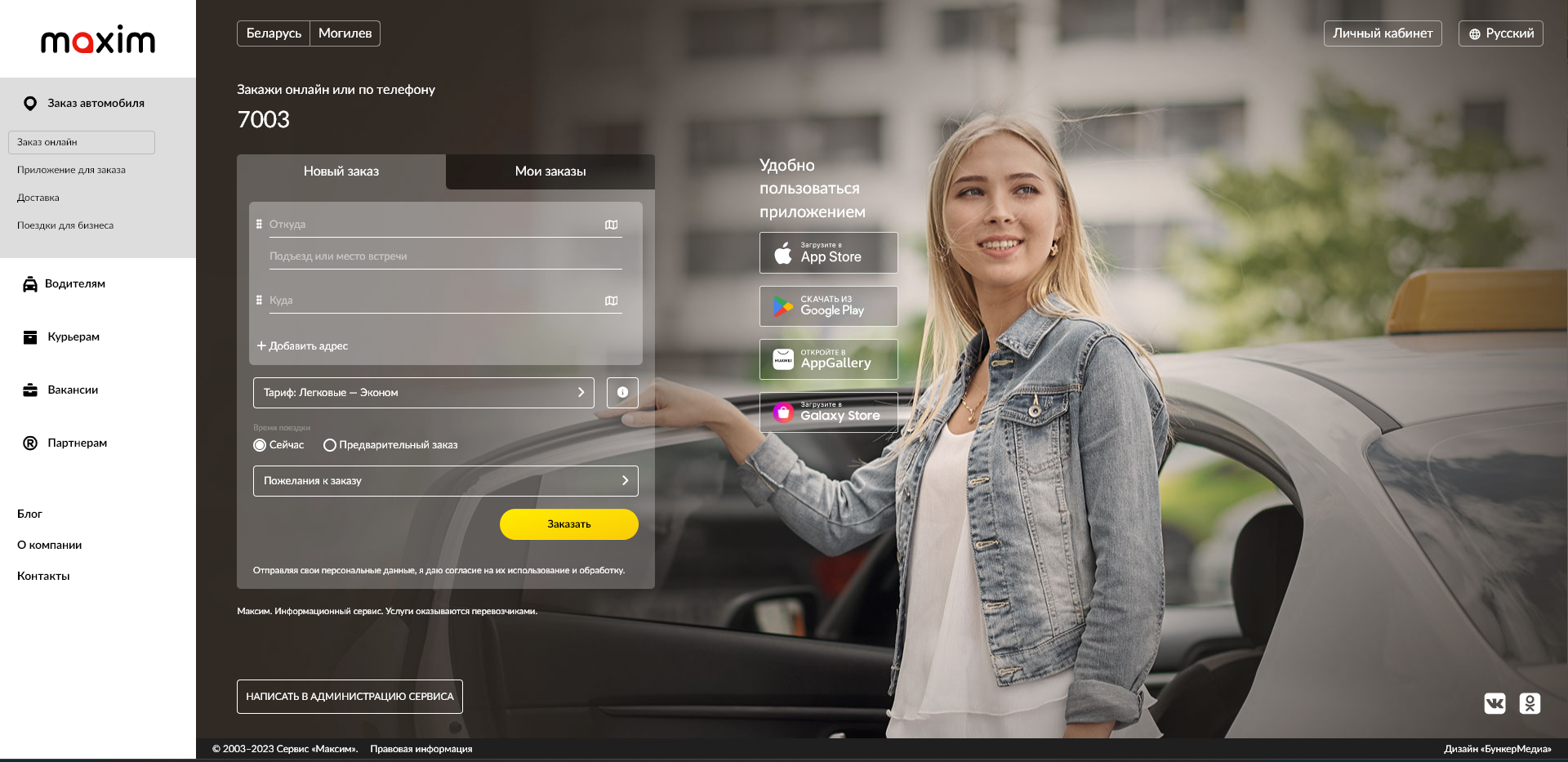


Рисунок 2.3 – Скриншот аналога веб-приложения «taximaxim.by»

На этом сайте можно заказать такси по введенным пользователем данным. Пользовать может завести свой личный аккаунт и просматривать историю своих заказов. Отсутствует функциональная карта, следовательно нельзя будет выбрать точку подачи и просмотреть сам маршрут поездки. [3]

Проанализировав «taximaxim.by», можно выделить некоторые плюсы и минусы программного средства.

Основные плюсы:

* отзывчивый интерфейс;
* возможность зарегистрировать аккаунт пользователя;
* возможность просмотреть историю поездок.

Основные минусы:

* отсутствие функциональной карты;
* низкое качество фонового фота на главной странице.

**2.4 Вывод анализа требований**

При анализе трех ведущих веб-приложений для такси были выявлены их уникальные функции, а также плюсы и минусы. Ни одно из приложений не является идеальным: «taxi.yandex.by» имеет сложный дизайн, «135.by» не предлагает личного профиля пользователя, а «taximaxim.by» не имеет функциональной карты. На основе этих выводов, был сформирован вывод основанный на необходимости наличия функциональной карты в приложении, отзывчивого интерфейса и личного кабинета пользователя.

1. **Проектирование приложения**
2. **Проектирование базы данных**

Первым этапом курсового проекта будет проектирование логически взаимосвязанных таблиц. Чтобы составить визуальную взаимосвязанную структуру базы данных, нам необходимо продумать, какая информация будет храниться в этих таблицах, после этого спроектировать связи с помощью первичных и внешних ключей.

Диаграмма базы данных, спроектированной в ходе разработки показана на рисунке 3.1.

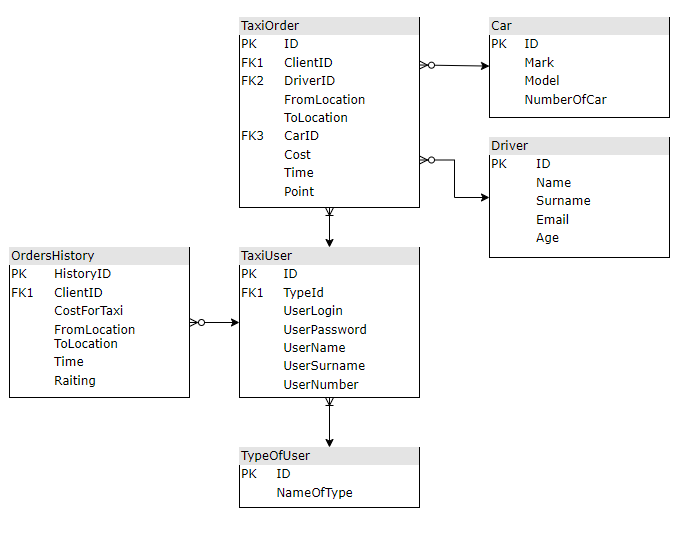


Рисунок 3.1 – Диаграмма базы данных

База данных, основанная на реляционной модели данных. В ее структуру входят следующие таблицы: TypeOfUser, TaxiUser, OrdersHistory,\_TaxiOrder, Driver, Car. Таблицы хранят в себе данные необходимые для описания и представления возможности заказа такси. А также информацию о самих клиентах и администраторе. База данных и таблицы были спроектированы на основе диаграммы вариантов использования, разработанной для гостя, клиента и администратора.

При разработке приложения для курсового проекта была использована realtime database Firebase.

В базе данных представлены 6 таблиц: TypeOfUser, TaxiUser, OrdersHistory, TaxiOrder, Driver, Car.

Таблица TypeOfUser хранит типы пользователей, состоит из столбцов (таблица 3.1):

Таблица 3.1 – Столбцы таблицы TypeOfUser

|  |  |  |
| --- | --- | --- |
| Наименование | Описание | Тип |
| Typeid | Идентификатор типа, первичный ключ | Integer |
| Nameoftype | Имя типа | Text |

Таблица TaxiUser хранит список пользователей и их персональные данные, состоит из столбцов (таблица 3.2):

Таблица 3.2 – Столбцы таблицы TaxiUser

|  |  |  |
| --- | --- | --- |
| Наименование | Описание | Тип |
| Id | Идентификатор пользователя, первичный ключ | Integer |
| Typeid | Идентификатор типа, внешний ключ | Integer |
| Userlogin | Логин пользователя | Text |
| Userpassword | Пароль пользователя | Text |
| Username | Имя пользователя | Text |
| Usersurname | Фамилия пользователя | Text |
| UserInteger | Номер телефона пользователя | Text |

Таблица OrdersHistory хранит историю заказов пользователя, таблица состоит из столбцов (таблица 3.3):

Таблица 3.3 – Столбцы таблицы OrdersHistory

|  |  |  |
| --- | --- | --- |
| Наименование | Описание | Тип |
| Historyid | Идентификатор истории, первичный ключ | Integer |
| Clientid | Идентификатор пользователя, внешний ключ | Integer |
| Costfortaxi | Стоимость поездки | Integer |
| Tolocation | Точка подачи | Text |
| Fromlocation | Точка завершения поездки | Text |
| Time | Время начала поездки | Text |
| Raiting | Оценка | Integer |

Таблица TaxiOrder представляет список заказов такси, состоит из столбцов (таблица 3.4):

Таблица 3.4 – Столбцы таблицы TaxiOrder

|  |  |  |
| --- | --- | --- |
| Наименование | Описание | Тип |
| Id | Идентификатор заказа, первичный ключ | Integer |
| Clientid | Идентификатор пользователя, внешний ключ | Integer |
| Driverid | Идентификатор водителя, внешний ключ | Integer |
| Carid | Идентификатор машины, внешний ключ | Integer |
| Fromlocation | Точка подачи | Text |
| Tolocation | Точка завершения поездки | Text |
| Cost | Стоимость поездки | Integer |
| Time | Время начала поездки | Text |
| Point | Оценка | Integer |

Таблица Driver представляет список водителей таксопарка, состоит из столбцов (таблица 3.5):

Таблица 3.5 – Столбцы таблицы Orders

|  |  |  |
| --- | --- | --- |
| Наименование | Описание | Тип |
| Id | Идентификатор водителя, первичный ключ | Integer |
| Name | Имя водителя | Text |
| Surname | Фамилия водителя | Text |
| Email | Email водителя | Text |
| Age | Возраст водителя | Integer |

Таблица Car представляет список машин таксопарка. Данная таблица состоит из столбцов (таблица 3.6):

Таблица 3.6 – Столбцы таблицы Car

|  |  |  |
| --- | --- | --- |
| Наименование | Описание | Тип |
| Id | Идентификатор машины, первичный ключ | Integer |
| Mark | Марка машины | Text |
| Model | Модель машины | Text |
| Numberofcar | Номер машины | Integer |

**3.2 Общая структура**

Реализация приложения для заказа такси на базе Flutter представляет собой сбалансированную архитектуру, обеспечивающую эффективное взаимодействие с пользователями и обработку операций заказа. В приложении используются различные ключевые пакеты Flutter для реализации различных функций и возможностей.

Использование пакета firebase\_core позволяет интегрировать ваше приложение с Firebase, обеспечивая основу для работы с аутентификацией, базой данных, облачным хранилищем и другими сервисами Firebase. Пакет firebase\_auth обеспечивает безопасную аутентификацию пользователей, что позволяет пользователям создавать учетные записи и входить в приложение.

Firebase Database и Firebase Storage используются для хранения и обработки данных о водителях, автомобилях и заказах, обеспечивая реальном времени доступ к информации и эффективное управление данными.

Google Maps Flutter интегрируется для отображения карты с местоположением машин, маршрутов и других объектов, связанных с заказами такси.

Пакеты geolocator и geocoding используются для работы с геолокацией, что позволяет определять текущее местоположение пользователей, а также преобразовывать адреса в географические координаты и наоборот.

Пакеты permission\_handler и url\_launcher обеспечивают удобный доступ к различным разрешениям устройства и возможность открывать URL-адреса во внешних приложениях.

Также, используя пакет provider, ваше приложение может эффективно управлять состоянием приложения, обеспечивая отзывчивость интерфейса и логику взаимодействия с пользователем.

Реализация приложения отлично сбалансирована по функциональным возможностям и обеспечивает удобство использования как для водителей, так и для клиентов, что делает его привлекательным для широкого круга пользователей.

**3.3 Вывод по разделу**

В данном разделе представлен детальный обзор процесса проектирования базы данных в среде Firebase, а также описаны компоненты, составляющие приложение, и подробная структура файлов проекта. Был Осуществлен анализ и определение структуры базы данных с использованием Firebase.

Реализация приложения для заказа такси на базе Flutter с использованием описанных пакетов обеспечивает эффективное функционирование приложения и удовлетворяет основные требования пользователей.

1. **Реализация приложения**

Реализация программного средства — это процесс преобразования детального плана или концепции программного продукта, разработанного в процессе проектирования, в функциональный продукт.

**4.1 Технические средства разработки**

Для реализации курсового проекта будет использован язык программирования flutter.

Flutter — это открытая платформа разработки, созданная компанией Google, которая позволяет разработчикам создавать высокопроизводительные мобильные приложения для различных платформ, таких как Android и iOS, а также веб-приложения и приложения для настольных компьютеров из одного кодовой базы.

Одной из ключевых особенностей Flutter является его использование одного и того же кода для разных платформ. Разработчики могут написать приложение на языке программирования Dart, который является основным языком разработки для Flutter, и затем скомпилировать его в нативный код для каждой платформы. Это позволяет достичь высокой производительности и единообразного пользовательского интерфейса на разных устройствах.

База данных была разработана c помощью Firebase.

Firebase — это платформа разработки приложений в облаке, предоставляемая компанией Google. Она предлагает набор инструментов и сервисов, которые позволяют разработчикам создавать и управлять приложениями для мобильных устройств, веб-приложений и серверов без необходимости заботиться о сложностях инфраструктуры.

Основные компоненты Firebase включают:

1. Firebase Authentication: Сервис аутентификации, который обеспечивает безопасную регистрацию и аутентификацию пользователей в приложении. Он поддерживает различные методы аутентификации, включая электронную почту, социальные сети и многое другое.

2. Cloud Firestore: Гибкая и масштабируемая база данных в реальном времени, которая позволяет хранить и синхронизировать данные между клиентскими устройствами и сервером. Firestore предлагает мощные функции запросов, обеспечивает мгновенные обновления данных и поддерживает автоматическую синхронизацию.

3. Firebase Storage: Сервис хранения файлов, который позволяет загружать и загружать файлы, такие как изображения, видео и документы, в облачное хранилище. Он обеспечивает простой доступ к файлам и предлагает интеграцию с другими сервисами Firebase.

4. Firebase Cloud Messaging: Сервис рассылки push-уведомлений, который позволяет отправлять уведомления и сообщения на мобильные устройства пользователей. Он обеспечивает надежную доставку уведомлений и поддерживает настраиваемые сегменты аудитории.

5. Firebase Hosting: Сервис размещения веб-сайтов и статических веб-приложений, который позволяет разработчикам развертывать и обслуживать свои приложения прямо из облака Google. Он обеспечивает быструю и надежную доставку контента всему миру.

6. Firebase Analytics: Инструмент аналитики, который предоставляет подробную информацию о поведении пользователей, использовании приложений и эффективности маркетинговых кампаний. Он помогает разработчикам принимать информированные решения, оптимизировать свои приложения и улучшать пользовательский опыт.

Firebase также предлагает множество других сервисов, таких как машинное обучение, тестирование, анализ ошибок, аутентификация социальных сетей и многое другое. Он интегрируется с различными платформами разработки, включая Android, iOS, веб и серверы, что делает его мощным инструментом для создания современных и масштабируемых приложений.

Для данного приложения используются Firebase Authentication, Firebase Cloud и Firebase Storage.

**4.2 Разработка мобильного приложения**

Мобильное приложение содержит в себе несколько папок. Каждая из перечисленных папок в мобильном приложении выполняет определенную функцию:

* utils представляет классы-помощники, которые используются во всем приложении и содержат некоторую логику.
* pages содержит классы, отвечающие за взаимодействие с пользовательским интерфейсом.
* models содержит классы, которые описывают данные и бизнес-логику приложения. Здесь находятся классы моделей данных, которые отображают структуру данных приложения.

Со всей кодовой структурой проекта можно ознакомиться в приложении А.

**4.3 Вывод по разделу**

В результате раздела была описана реализация приложения. Выбор Flutter для разработки и Firebase для облачных сервисов обеспечивает современный и эффективный подход к созданию мобильных приложений. Firebase Authentication гарантирует безопасность, а Firestore и Storage обеспечивают гибкое хранение данных и мультимедиа. Организация приложения по функциональным папкам упрощает его разработку и поддержку.

1. **Тестирование приложения**

В момент авторизации и регистрации возможна ситуация, когда пользователь вводит некорректные данные, например, неверный пароль, незарегистрированный логин, некорректный адрес электронной почты или номер телефона. Такие исключения обрабатываются программным средством с помощью показа пользователю всплывающих сообщений с текстом ошибки. Примеры обработки разных видов исключительных ситуаций представлены на рисунках 5.1 – 5.2.

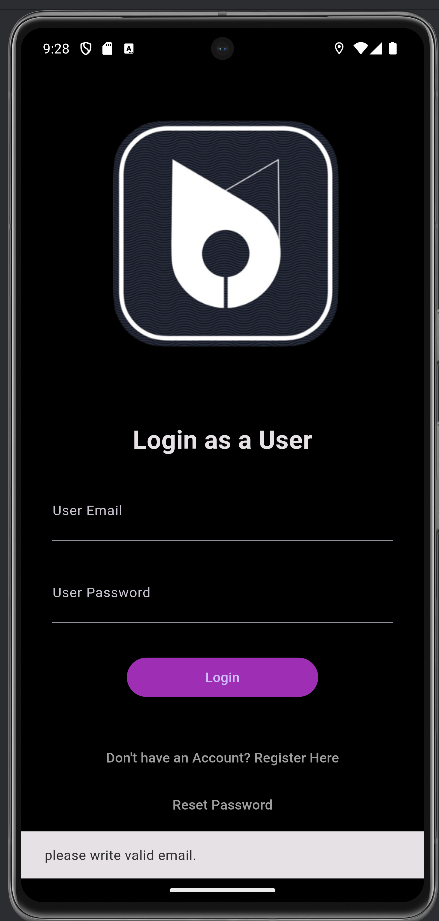


Рисунок 5.1 – Валидация окна входа

На рисунке ниже представлен пример валидации окна регистрации.

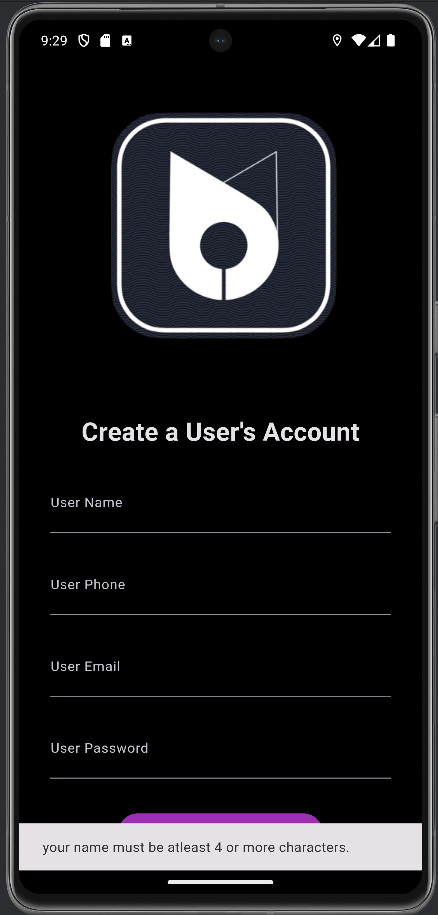


Рисунок 5.2 – Валидация окна регистрации

Кроме рассмотренных возможностей приложения, обработаны возможные ошибки при редактировании профиля. Например, нельзя поменять информацию на точно такую же.

На рисунке 5.3 можно увидеть, что геолокация отображается корректно.

Процесс получения геолокации на телефоне обычно включает следующие шаги:

Запрос разрешения: Приложение, которое нуждается в доступе к геолокации, отправляет запрос на получение разрешения у пользователя. Это обеспечивает конфиденциальность данных пользователя и соответствует требованиям безопасности.

Включение служб геолокации: Пользователь включает службы геолокации на своем устройстве, такие как GPS или Wi-Fi. Это обеспечивает точность определения местоположения.

Использование API: Приложение использует API, предоставленный операционной системой (например, Android Location API или Core Location Framework в iOS), для запроса текущего местоположения устройства.

Определение местоположения: Устройство использует доступные методы, такие как GPS, Wi-Fi или сетевой сигнал, для определения своего местоположения с учетом точности и энергопотребления.

Передача данных: Полученные координаты местоположения передаются обратно в приложение, где могут быть использованы для различных целей, таких как отображение на карте, поиск ближайших объектов и т. д.

Важно отметить, что точность определения местоположения может зависеть от множества факторов, включая наличие сигнала GPS, доступность Wi-Fi и сети мобильной связи, а также настроек приватности и безопасности, установленных пользователем на своем устройстве.

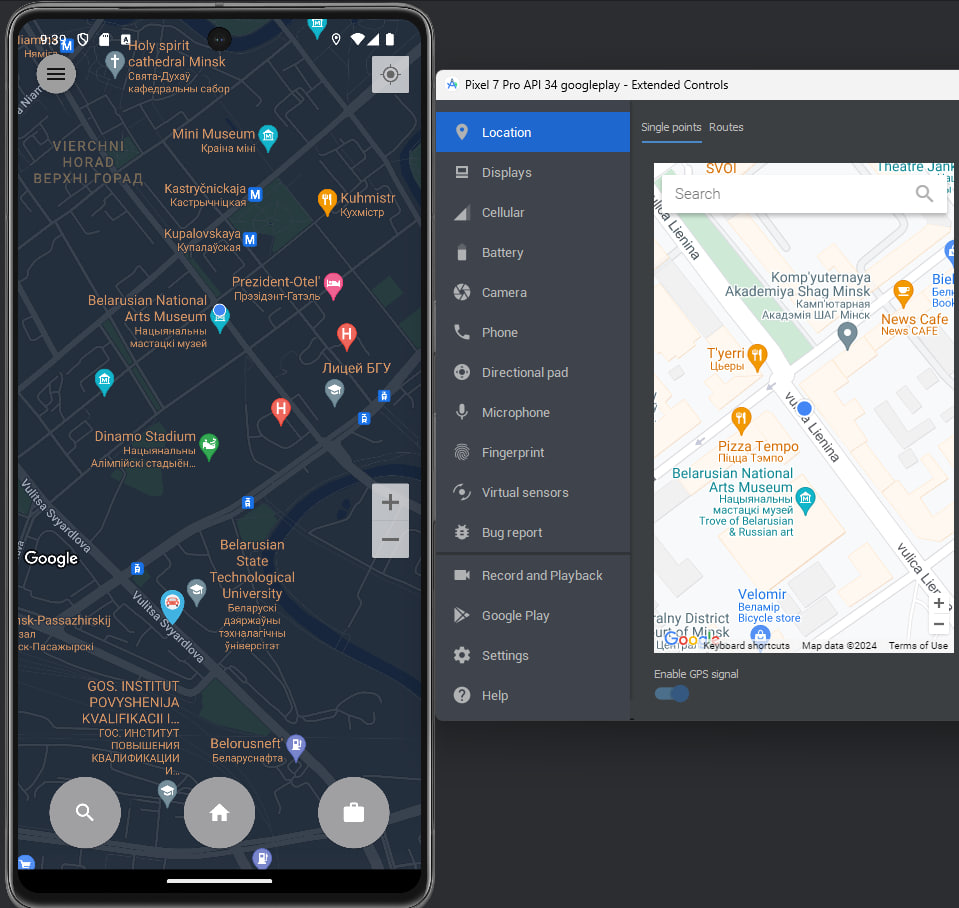


Рисунок 5.3 – Отображение корректной геолокации на телефоне

**5.1 Вывод по разделу**

В данном разделе был представлен обзор процесса тестирования и оценки работоспособности разработанного приложения. Проведение тестов позволило выявить и устранить различные ошибки и недочеты в функционировании приложения. Таким образом, процесс тестирования и оценки работоспособности приложения позволил не только выявить и устранить возможные проблемы, но и удостовериться в его соответствии заявленным требованиям, обеспечивая стабильное и эффективное функционирование.

1. **Краткое описание приложения для демонстрации**

Приложение нацелено на область работы таксопарка, т.е. возможность принимать и выполнять заказы по перевозке пассажиров, так же возможность рассматривать историю заказов. Сам по себе данный проект состоит из 3 приложений: для водителя, для клиента и контролирующего администратора. Все приложения взаимодействую между собой, производя запросы к БД FireBase и получение уведомлений от клиента/водителя. Приложения имеют отзывчивый, понятный дизайн. Удовлетворяют потребностям в личном аккаунте, восстановление пароля от аккаунта, в случае потери/забытии его. Как клиент, так и водитель имеют доступ к просмотру своих поездок.

Наше приложение для таксопарка разработано с использованием Flutter - мощной и универсальной платформы разработки от Google. Flutter позволяет создавать красивые и высокопроизводительные мобильные приложения для различных платформ, включая Android и iOS, используя один и тот же код. Это упрощает процесс разработки и поддержки, а также обеспечивает единообразный пользовательский интерфейс на всех устройствах. Благодаря Flutter мы можем быстро развернуть новые функции и обновления, обеспечивая приложению высокую степень надежности и производительности.

Flutter также предоставляет разработчикам широкие возможности для создания красивого и адаптивного дизайна благодаря своим гибким инструментам для создания пользовательского интерфейса. Это позволяет нам создавать интерактивные и привлекательные пользовательские интерфейсы, которые соответствуют современным требованиям и ожиданиям пользователей. Благодаря гибкости и производительности Flutter, наше приложение для таксопарка может быть легко адаптировано к различным устройствам и разрешениям экрана, обеспечивая отличный пользовательский опыт на всех платформах.

**6.1 Вывод по разделу**

В данном разделе представлено краткое описание приложения для демонстрации, представляющего собой таксопарк с приложениями для водителя/клиента и администратора. Описано, что приложение предоставляет удобный функционал для каждой из указанных ролей, включая возможность заказа такси, управление заказами, навигацию, оплату поездок, а также административные функции. Также отмечено, что приложение разработано с использованием Flutter, что обеспечивает красивый дизайн, высокую производительность и возможность адаптации к различным устройствам и платформам.

1. **Руководство пользователя**

При запуске приложения для клиента вы увидите окно входа в аккаунт пользователя. Под кнопкой «Login», есть возможность зарегистрировать аккаунт, или восстановить пароль от него. Само окно можно увидеть на рисунке 7.1.

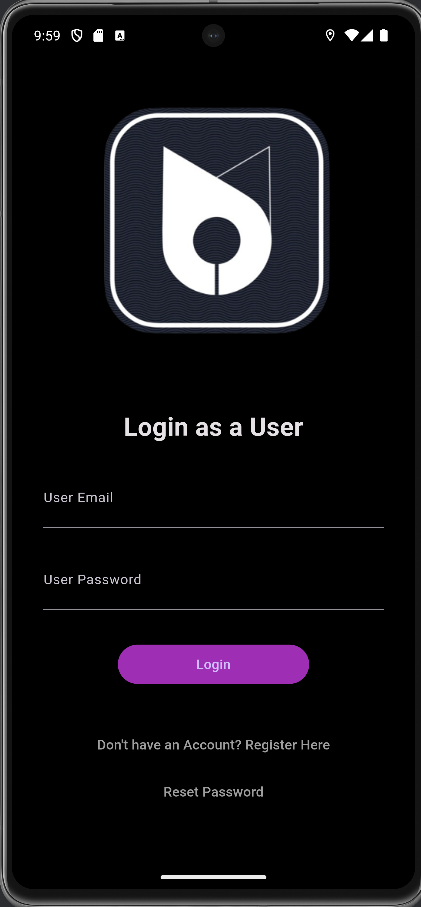


Рисунок 7.1 – Окно входа в аккаунт клиента

Восстановить пароль вы можете по средством валидной почты, на которую придет сообщение со сбросом пароля для данной учетной записи.

Регистрация потребует от вас ФИО, email, номер телефона и пароль.

После входа в аккаунт вы окажитесь на главной странице с картой. Главную страницу можно увидеть на рисунке 7.2.

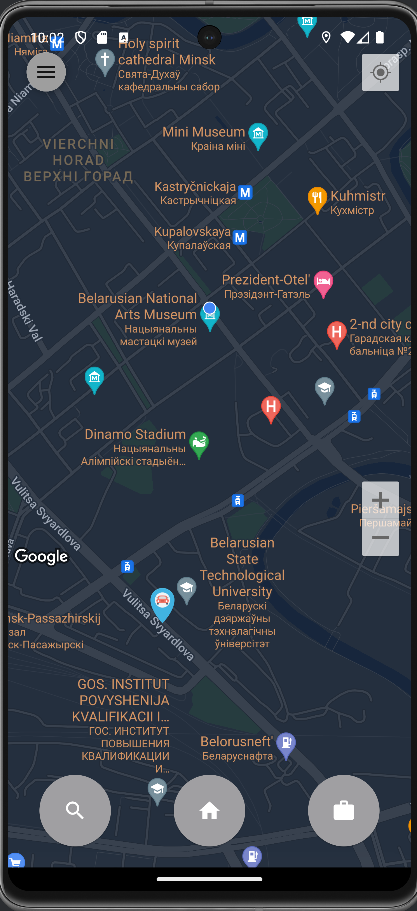


Рисунок 7.2 – Изображение главной страницы приложения для клиента

При нажатии на кнопку меню сверху-слева можно увидеть меню с возможностью увидеть историю заказов, изменить данные пользователя, либо выйти из аккаунта.

Меню можно увидеть на рисунке 7.3.

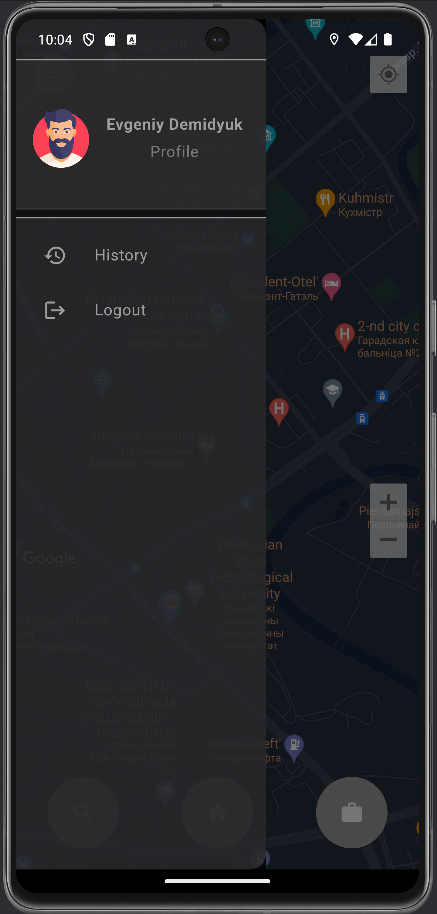


Рисунок 7.3 – Изображение меню в приложении клиента

При нажатие на кнопку лупы на главной странице появится окно с вводом адресов. После выбора адреса подачи и доставки появится стоимость поездки и возможность создать заказ.

Для водителя так же имеется страница для входа/регистрации и восстановления пароля. На главной странице так же находится карта. Через меню снизу можно посмотреть историю заказов, редактировать информацию профиля, либо общую прибыль.

Главную страницу приложение водителя можно увидеть на рисунке 7.4.

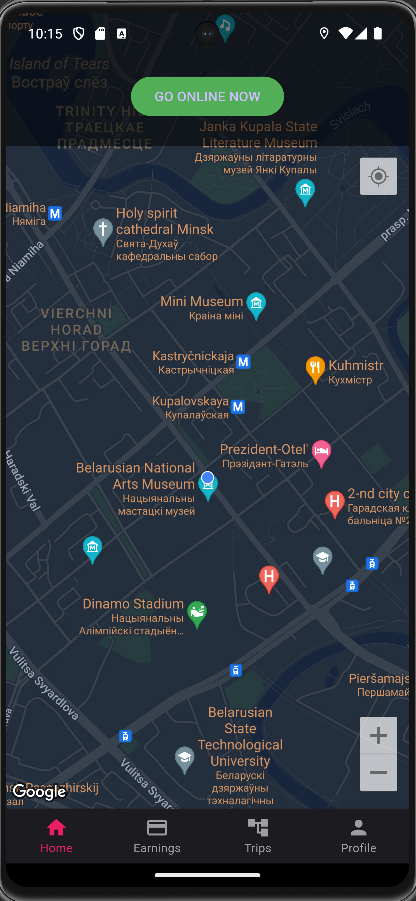


Рисунок 7.4 – Главная страница приложения водителя

Так же с помощью кнопки сверху «Go online now» водитель выходить в онлайн режим и клиенты видят его местоположение, т.к. он готов к заказам.

Приложение для администратора в основном носит информационный характер, администратор может посмотреть каждого водителя/клиента. Заблокировать их, либо посмотреть историю заказов таксопарка. Главную страницу приложения администратора можно увидеть на рисунке 7.5.

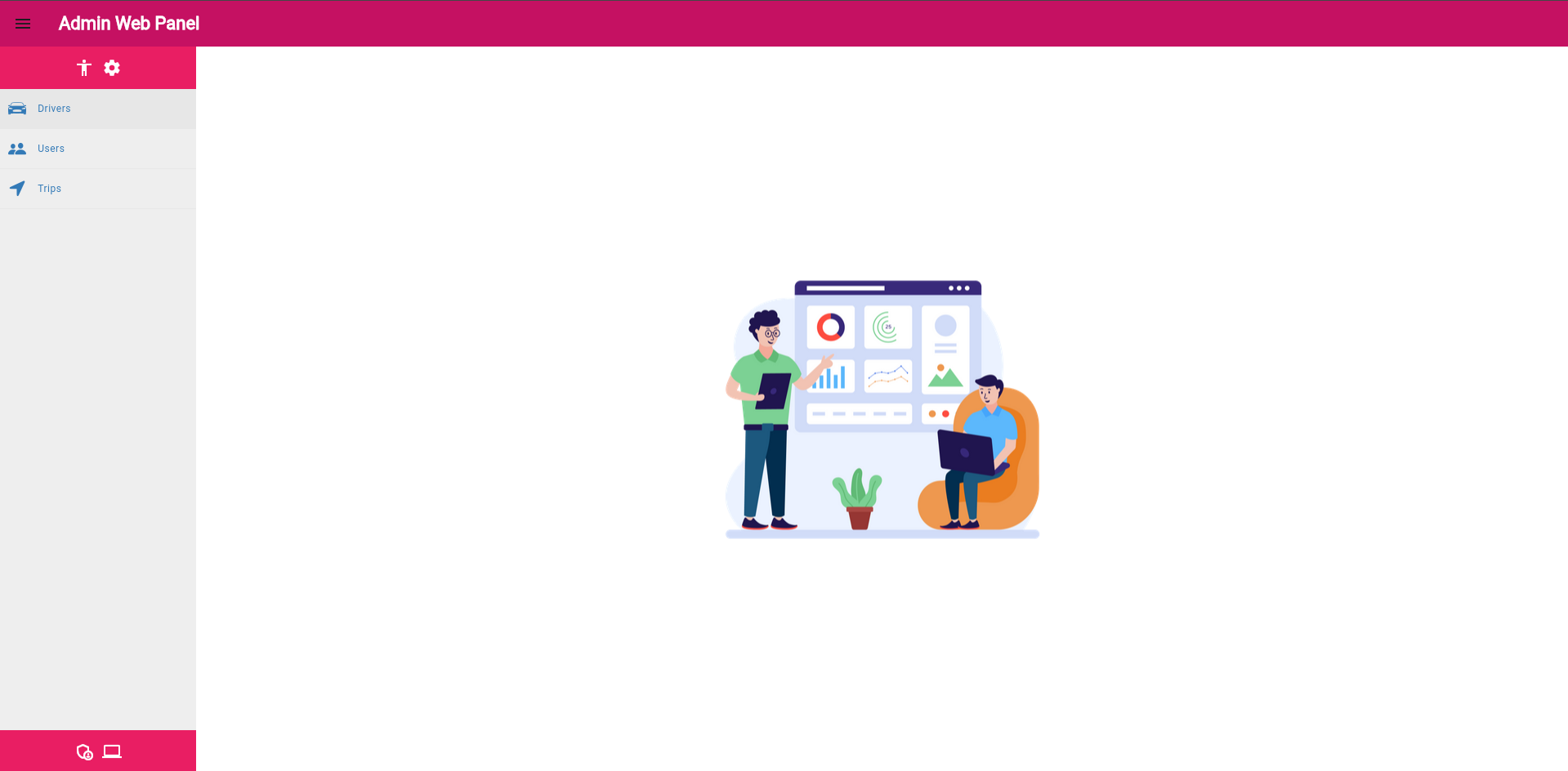


Рисунок 7.5 – Главная страница приложения администратора

Подробнее посмотреть каждую страницу проекта можно в приложении Б.

**7.1 Вывод по разделу**

В результате раздела была описана функциональность проекта, описаны возможности приложения клиента, водителя, администратора. Приложения выполняют поставленные возможности и обеспечивают беспрерывное функционирование. Пользователь может зарегистрироваться, авторизоваться, восстановить пароль, заказать такси, просмотреть свою историю заказов и изменить данные в профиле. Водитель может так же регистрироваться, авторизовываться, восстанавливать пароль, совершать поездки и изменять свою информацию. Администратор ведет контроль над водителями и клиентами.

**Заключение**

Целью данного курсового проекта является разработка кроссплатформенного мобильного приложения для управления таксопарком на базе фреймворка Flutter. Приложение должно обеспечивать эффективное управление водителями и автомобилями, а также удобный заказ такси для клиентов.

Для достижения этой цели были выполнены следующие задачи: проведен анализ потребностей и требований таксопарков и их клиентов, изучены особенности рынка такси и конкурентное окружение, разработана структура базы данных, спроектирован пользовательский интерфейс, написан код приложения на языке Dart с использованием Flutter, реализованы функции регистрации и авторизации пользователей, управления данными о водителях и автомобилях, а также системы заказа такси. После этого приложение было протестировано на различных устройствах и операционных системах, обнаруженные ошибки были устранены.

Ожидаемыми результатами выполнения проекта являются следующие: разработанное мобильное приложение для управления таксопарком, полноценно функционирующая система, успешное внедрение на рынок и его популяризация среди пользователей.

Процесс разработки таксопарка включал математическое обоснование ключевых операций, таких как расчет стоимости поездки и анализ статистики. Также было проведено моделирование технических объектов системы, включающее в себя диаграммы классов, диаграммы последовательности и другие модели, отражающие взаимодействие между различными компонентами системы.

Функции приложения для администратора, клиента и водителя были четко определены, обеспечивая удобство использования для всех категорий пользователей. Для администратора предусмотрены функции управления данными и мониторинга заказов, для клиента - заказ такси и просмотр истории заказов, для водителя - выполнение заказов и управление профилем.

Таким образом, разработанное приложение для таксопарка на основе Flutter представляет собой универсальное и удобное средство для организации работы таксопарка и обеспечения удобства клиентов.

**Список используемых источников**

1. Yandex Taxi [Электронный ресурс] – Режим доступа: https://taxi.yandex.by – Дата доступа: 15.05.2024
2. 135 Taxi [Электронный ресурс] – Режим доступа: https://135.by – Дата доступа: 15.05.2024
3. Taxi Maxim [Электронный ресурс] – Режим доступа: https://taximaxim.by – Дата доступа: 20.10.2023
4. Flutter [Электронный ресурс] – Режим доступа: https://flutter.dev/ – Дата доступа: 15.05.2024
5. Metanit [Электронный ресурс] – Режим доступа: https://metanit.com/ –Дата доступа: 15.05.2024

**Приложение А**

class LoginScreen extends StatefulWidget {

const LoginScreen({super.key});

@override

State<LoginScreen> createState() => \_LoginScreenState();

}

class \_LoginScreenState extends State<LoginScreen>

{

TextEditingController emailTextEditingController = TextEditingController();

TextEditingController passwordTextEditingController = TextEditingController();

CommonMethods cMethods = CommonMethods();

checkIfNetworkIsAvailable()

{

cMethods.checkConnectivity(context);

signInFormValidation();

}

signInFormValidation()

{

if(!emailTextEditingController.text.contains("@"))

{

cMethods.displaySnackBar("please write valid email.", context);

}

else if(passwordTextEditingController.text.trim().length < 5)

{

cMethods.displaySnackBar("your password must be atleast 6 or more characters.", context);

}

else

{

signInUser();

}

}

signInUser() async

{

showDialog(

context: context,

barrierDismissible: false,

builder: (BuildContext context) => LoadingDialog(messageText: "Allowing you to Login..."),

);

final User? userFirebase = (

await FirebaseAuth.instance.signInWithEmailAndPassword(

email: emailTextEditingController.text.trim(),

password: passwordTextEditingController.text.trim(),

).catchError((errorMsg)

{

Navigator.pop(context);

cMethods.displaySnackBar(errorMsg.toString(), context);

})

).user;

if(!context.mounted) return;

Navigator.pop(context);

if(userFirebase != null)

{

DatabaseReference usersRef = FirebaseDatabase.instance.ref().child("users").child(userFirebase.uid);

await usersRef.once().then((snap)

{

if(snap.snapshot.value != null)

{

if((snap.snapshot.value as Map)["blockStatus"] == "no")

{

userName = (snap.snapshot.value as Map)["name"];

userPhone = (snap.snapshot.value as Map)["phone"];

Navigator.push(context, MaterialPageRoute(builder: (c)=> HomePage()));

}

else

{

FirebaseAuth.instance.signOut();

cMethods.displaySnackBar("you are blocked. Contact admin: alizeb875@gmail.com", context);

}

}

else

{

FirebaseAuth.instance.signOut();

cMethods.displaySnackBar("your record do not exists as a User.", context);

}

});

}

}

@override

Widget build(BuildContext context) {

return Scaffold(

body: SingleChildScrollView(

child: Padding(

padding: const EdgeInsets.all(10),

child: Column(

children: [

Image.asset(

"assets/images/logo.png"

),

const Text(

"Login as a User",

style: TextStyle(

fontSize: 26,

fontWeight: FontWeight.bold,

),

),

//text fields + button

Padding(

padding: const EdgeInsets.all(22),

child: Column(

children: [

TextField(

controller: emailTextEditingController,

keyboardType: TextInputType.emailAddress,

decoration: const InputDecoration(

labelText: "User Email",

labelStyle: TextStyle(

fontSize: 14,

),

),

style: const TextStyle(

color: Colors.grey,

fontSize: 15,

),

),

const SizedBox(height: 22,),

TextField(

controller: passwordTextEditingController,

obscureText: true,

keyboardType: TextInputType.text,

decoration: const InputDecoration(

labelText: "User Password",

labelStyle: TextStyle(

fontSize: 14,

),

),

style: const TextStyle(

color: Colors.grey,

fontSize: 15,

),

),

const SizedBox(height: 32,),

ElevatedButton(

onPressed: ()

{

checkIfNetworkIsAvailable();

},

style: ElevatedButton.styleFrom(

backgroundColor: Colors.purple,

padding: const EdgeInsets.symmetric(horizontal: 80, vertical: 10)

),

child: const Text(

"Login"

),

),

],

),

),

const SizedBox(height: 12,),

//textbutton

TextButton(

onPressed: ()

{

Navigator.push(context, MaterialPageRoute(builder: (c)=> SignUpScreen()));

},

child: const Text(

"Don\'t have an Account? Register Here",

style: TextStyle(

color: Colors.grey,

),

),

),

TextButton(

onPressed: ()

{

Navigator.push(

context,

MaterialPageRoute(builder: (context) => ResetPasswordScreen()),

);

},

child: const Text(

"Reset Password",

style: TextStyle(

color: Colors.grey,

),

),

),

],

),

),

),

);

}

}

class ResetPasswordScreen extends StatefulWidget {

@override

\_ResetPasswordScreenState createState() => \_ResetPasswordScreenState();

}

class \_ResetPasswordScreenState extends State<ResetPasswordScreen> {

TextEditingController emailController = TextEditingController();

CommonMethods cMethods = CommonMethods();

@override

Widget build(BuildContext context) {

return Scaffold(

appBar: AppBar(

title: Text('Reset Password'),

),

body: Padding(

padding: const EdgeInsets.all(20),

child: Column(

crossAxisAlignment: CrossAxisAlignment.stretch,

children: [

TextField(

controller: emailController,

decoration: InputDecoration(

labelText: 'Enter your email',

),

),

SizedBox(height: 20),

ElevatedButton(

onPressed: () {

resetPassword();

},

child: Text('Send Reset Email'),

),

],

),

),

);

}

void resetPassword() async {

String email = emailController.text.trim();

if (email.isNotEmpty && email.contains('@')) {

showDialog(

context: context,

barrierDismissible: false,

builder: (BuildContext context) =>

LoadingDialog(messageText: "Sending reset email..."),

);

try {

await FirebaseAuth.instance.sendPasswordResetEmail(email: email);

Navigator.pop(context);

cMethods.displaySnackBar("Password reset email sent!", context);

} catch (error) {

Navigator.pop(context);

cMethods.displaySnackBar(error.toString(), context);

}

} else {

cMethods.displaySnackBar("Please enter a valid email.", context);

}

}

}

class SignUpScreen extends StatefulWidget {

const SignUpScreen({super.key});

@override

State<SignUpScreen> createState() => \_SignUpScreenState();

}

class \_SignUpScreenState extends State<SignUpScreen>

{

TextEditingController userNameTextEditingController = TextEditingController();

TextEditingController userPhoneTextEditingController = TextEditingController();

TextEditingController emailTextEditingController = TextEditingController();

TextEditingController passwordTextEditingController = TextEditingController();

CommonMethods cMethods = CommonMethods();

checkIfNetworkIsAvailable()

{

cMethods.checkConnectivity(context);

signUpFormValidation();

}

signUpFormValidation()

{

if(userNameTextEditingController.text.trim().length < 3)

{

cMethods.displaySnackBar("your name must be atleast 4 or more characters.", context);

}

else if(userPhoneTextEditingController.text.trim().length < 7)

{

cMethods.displaySnackBar("your phone number must be atleast 8 or more characters.", context);

}

else if(!emailTextEditingController.text.contains("@"))

{

cMethods.displaySnackBar("please write valid email.", context);

}

else if(passwordTextEditingController.text.trim().length < 5)

{

cMethods.displaySnackBar("your password must be atleast 6 or more characters.", context);

}

else

{

registerNewUser();

}

}

registerNewUser() async

{

showDialog(

context: context,

barrierDismissible: false,

builder: (BuildContext context) => LoadingDialog(messageText: "Registering your account..."),

);

final User? userFirebase = (

await FirebaseAuth.instance.createUserWithEmailAndPassword(

email: emailTextEditingController.text.trim(),

password: passwordTextEditingController.text.trim(),

).catchError((errorMsg)

{

Navigator.pop(context);

cMethods.displaySnackBar(errorMsg.toString(), context);

})

).user;

if(!context.mounted) return;

Navigator.pop(context);

DatabaseReference usersRef = FirebaseDatabase.instance.ref().child("users").child(userFirebase!.uid);

Map userDataMap =

{

"name": userNameTextEditingController.text.trim(),

"email": emailTextEditingController.text.trim(),

"phone": userPhoneTextEditingController.text.trim(),

"id": userFirebase.uid,

"blockStatus": "no",

};

usersRef.set(userDataMap);

Navigator.push(context, MaterialPageRoute(builder: (c)=> HomePage()));

}

@override

Widget build(BuildContext context)

{

return Scaffold(

body: SingleChildScrollView(

child: Padding(

padding: const EdgeInsets.all(10),

child: Column(

children: [

Image.asset(

"assets/images/logo.png"

),

const Text(

"Create a User\'s Account",

style: TextStyle(

fontSize: 26,

fontWeight: FontWeight.bold,

),

),

//text fields + button

Padding(

padding: const EdgeInsets.all(22),

child: Column(

children: [

TextField(

controller: userNameTextEditingController,

keyboardType: TextInputType.text,

decoration: const InputDecoration(

labelText: "User Name",

labelStyle: TextStyle(

fontSize: 14,

),

),

style: const TextStyle(

color: Colors.grey,

fontSize: 15,

),

),

const SizedBox(height: 22,),

TextField(

controller: userPhoneTextEditingController,

keyboardType: TextInputType.text,

decoration: const InputDecoration(

labelText: "User Phone",

labelStyle: TextStyle(

fontSize: 14,

),

),

style: const TextStyle(

color: Colors.grey,

fontSize: 15,

),

),

const SizedBox(height: 22,),

TextField(

controller: emailTextEditingController,

keyboardType: TextInputType.emailAddress,

decoration: const InputDecoration(

labelText: "User Email",

labelStyle: TextStyle(

fontSize: 14,

),

),

style: const TextStyle(

color: Colors.grey,

fontSize: 15,

),

),

const SizedBox(height: 22,),

TextField(

controller: passwordTextEditingController,

obscureText: true,

keyboardType: TextInputType.text,

decoration: const InputDecoration(

labelText: "User Password",

labelStyle: TextStyle(

fontSize: 14,

),

),

style: const TextStyle(

color: Colors.grey,

fontSize: 15,

),

),

const SizedBox(height: 32,),

ElevatedButton(

onPressed: ()

{

checkIfNetworkIsAvailable();

},

style: ElevatedButton.styleFrom(

backgroundColor: Colors.purple,

padding: const EdgeInsets.symmetric(horizontal: 80, vertical: 10)

),

child: const Text(

"Sign Up"

),

),

],

),

),

const SizedBox(height: 12,),

//textbutton

TextButton(

onPressed: ()

{

Navigator.push(context, MaterialPageRoute(builder: (c)=> LoginScreen()));

},

child: const Text(

"Already have an Account? Login Here",

style: TextStyle(

color: Colors.grey,

),

),

),

],

),

),

),

);

}

}

class CommonMethods

{

checkConnectivity(BuildContext context) async

{

var connectionResult = await Connectivity().checkConnectivity();

if(connectionResult != ConnectivityResult.mobile && connectionResult != ConnectivityResult.wifi)

{

if(!context.mounted) return;

displaySnackBar("your Internet is not Available. Check your connection. Try Again.", context);

}

}

displaySnackBar(String messageText, BuildContext context)

{

var snackBar = SnackBar(content: Text(messageText));

ScaffoldMessenger.of(context).showSnackBar(snackBar);

}

static sendRequestToAPI(String apiUrl) async

{

http.Response responseFromAPI = await http.get(Uri.parse(apiUrl));

try

{

if(responseFromAPI.statusCode == 200)

{

String dataFromApi = responseFromAPI.body;

var dataDecoded = jsonDecode(dataFromApi);

return dataDecoded;

}

else

{

return "error";

}

}

catch(errorMsg)

{

return "error";

}

}

///Reverse GeoCoding

static Future<String> convertGeoGraphicCoOrdinatesIntoHumanReadableAddress(Position position, BuildContext context) async

{

String humanReadableAddress = "";

String apiGeoCodingUrl = "https://maps.googleapis.com/maps/api/geocode/json?latlng=${position.latitude},${position.longitude}&key=$googleMapKey";

var responseFromAPI = await sendRequestToAPI(apiGeoCodingUrl);

if(responseFromAPI != "error")

{

humanReadableAddress = responseFromAPI["results"][0]["formatted\_address"];

AddressModel model = AddressModel();

model.humanReadableAddress = humanReadableAddress;

model.placeName = humanReadableAddress;

model.longitudePosition = position.longitude;

model.latitudePosition = position.latitude;

Provider.of<AppInfo>(context, listen: false).updatePickUpLocation(model);

}

return humanReadableAddress;

}

///Directions API

static Future<DirectionDetails?> getDirectionDetailsFromAPI(LatLng source, LatLng destination) async

{

String urlDirectionsAPI = "https://maps.googleapis.com/maps/api/directions/json?destination=${destination.latitude},${destination.longitude}&origin=${source.latitude},${source.longitude}&mode=driving&key=$googleMapKey";

var responseFromDirectionsAPI = await sendRequestToAPI(urlDirectionsAPI);

if(responseFromDirectionsAPI == "error")

{

return null;

}

DirectionDetails detailsModel = DirectionDetails();

detailsModel.distanceTextString = responseFromDirectionsAPI["routes"][0]["legs"][0]["distance"]["text"];

detailsModel.distanceValueDigits = responseFromDirectionsAPI["routes"][0]["legs"][0]["distance"]["value"];

detailsModel.durationTextString = responseFromDirectionsAPI["routes"][0]["legs"][0]["duration"]["text"];

detailsModel.durationValueDigits = responseFromDirectionsAPI["routes"][0]["legs"][0]["duration"]["value"];

detailsModel.encodedPoints = responseFromDirectionsAPI["routes"][0]["overview\_polyline"]["points"];

return detailsModel;

}

calculateFareAmount(DirectionDetails directionDetails)

{

double distancePerKmAmount = 0.4;

double durationPerMinuteAmount = 0.3;

double baseFareAmount = 2;

double totalDistanceTravelFareAmount = (directionDetails.distanceValueDigits! / 1000) \* distancePerKmAmount;

double totalDurationSpendFareAmount = (directionDetails.durationValueDigits! / 60) \* durationPerMinuteAmount;

double overAllTotalFareAmount = baseFareAmount + totalDistanceTravelFareAmount + totalDurationSpendFareAmount;

return overAllTotalFareAmount.toStringAsFixed(1);

}

}

class ManageDriversMethods

{

static List<OnlineNearbyDrivers> nearbyOnlineDriversList = [];

static void removeDriverFromList(String driverID)

{

int index = nearbyOnlineDriversList.indexWhere((driver) => driver.uidDriver == driverID);

if(nearbyOnlineDriversList.length > 0)

{

nearbyOnlineDriversList.removeAt(index);

}

}

static void updateOnlineNearbyDriversLocation(OnlineNearbyDrivers nearbyOnlineDriverInformation)

{

int index = nearbyOnlineDriversList.indexWhere((driver) => driver.uidDriver == nearbyOnlineDriverInformation.uidDriver);

nearbyOnlineDriversList[index].latDriver = nearbyOnlineDriverInformation.latDriver;

nearbyOnlineDriversList[index].lngDriver = nearbyOnlineDriverInformation.lngDriver;

}

}

class PushNotificationService

{

static sendNotificationToSelectedDriver(String deviceToken, BuildContext context, String tripID) async

{

String dropOffDestinationAddress = Provider.of<AppInfo>(context, listen: false).dropOffLocation!.placeName.toString();

String pickUpAddress = Provider.of<AppInfo>(context, listen: false).pickUpLocation!.placeName.toString();

Map<String, String> headerNotificationMap =

{

"Content-Type": "application/json",

"Authorization": serverKeyFCM,

};

Map titleBodyNotificationMap =

{

"title": "NET TRIP REQUEST from $userName",

"body": "PickUp Location: $pickUpAddress \nDropOff Location: $dropOffDestinationAddress",

};

Map dataMapNotification =

{

"click\_action": "FLUTTER\_NOTIFICATION\_CLICK",

"id": "1",

"status": "done",

"tripID": tripID,

};

Map bodyNotificationMap =

{

"notification": titleBodyNotificationMap,

"data": dataMapNotification,

"priority": "high",

"to": deviceToken,

};

await http.post(

Uri.parse("https://fcm.googleapis.com/fcm/send"),

headers: headerNotificationMap,

body: jsonEncode(bodyNotificationMap),

);

}

}

class HomePage extends StatefulWidget

{

const HomePage({super.key});

@override

State<HomePage> createState() => \_HomePageState();

}

class \_HomePageState extends State<HomePage>

{

final Completer<GoogleMapController> googleMapCompleterController = Completer<GoogleMapController>();

GoogleMapController? controllerGoogleMap;

Position? currentPositionOfUser;

GlobalKey<ScaffoldState> sKey = GlobalKey<ScaffoldState>();

CommonMethods cMethods = CommonMethods();

double searchContainerHeight = 276;

double bottomMapPadding = 0;

double rideDetailsContainerHeight = 0;

double requestContainerHeight = 0;

double tripContainerHeight = 0;

DirectionDetails? tripDirectionDetailsInfo;

List<LatLng> polylineCoOrdinates = [];

Set<Polyline> polylineSet = {};

Set<Marker> markerSet = {};

Set<Circle> circleSet = {};

bool isDrawerOpened = true;

String stateOfApp = "normal";

bool nearbyOnlineDriversKeysLoaded = false;

BitmapDescriptor? carIconNearbyDriver;

DatabaseReference? tripRequestRef;

List<OnlineNearbyDrivers>? availableNearbyOnlineDriversList;

StreamSubscription<DatabaseEvent>? tripStreamSubscription;

bool requestingDirectionDetailsInfo = false;

makeDriverNearbyCarIcon()

{

if(carIconNearbyDriver == null)

{

ImageConfiguration configuration = createLocalImageConfiguration(context, size: Size(0.5, 0.5));

BitmapDescriptor.fromAssetImage(configuration, "assets/images/tracking.png").then((iconImage)

{

carIconNearbyDriver = iconImage;

});

}

}

void updateMapTheme(GoogleMapController controller)

{

getJsonFileFromThemes("themes/dark\_style.json").then((value)=> setGoogleMapStyle(value, controller));

}

Future<String> getJsonFileFromThemes(String mapStylePath) async

{

ByteData byteData = await rootBundle.load(mapStylePath);

var list = byteData.buffer.asUint8List(byteData.offsetInBytes, byteData.lengthInBytes);

return utf8.decode(list);

}

setGoogleMapStyle(String googleMapStyle, GoogleMapController controller)

{

controller.setMapStyle(googleMapStyle);

}

getCurrentLiveLocationOfUser() async

{

Position positionOfUser = await Geolocator.getCurrentPosition(desiredAccuracy: LocationAccuracy.bestForNavigation);

currentPositionOfUser = positionOfUser;

LatLng positionOfUserInLatLng = LatLng(currentPositionOfUser!.latitude, currentPositionOfUser!.longitude);

CameraPosition cameraPosition = CameraPosition(target: positionOfUserInLatLng, zoom: 15);

controllerGoogleMap!.animateCamera(CameraUpdate.newCameraPosition(cameraPosition));

await CommonMethods.convertGeoGraphicCoOrdinatesIntoHumanReadableAddress(currentPositionOfUser!, context);

await getUserInfoAndCheckBlockStatus();

await initializeGeoFireListener();

}

getUserInfoAndCheckBlockStatus() async

{

DatabaseReference usersRef = FirebaseDatabase.instance.ref()

.child("users")

.child(FirebaseAuth.instance.currentUser!.uid);

await usersRef.once().then((snap)

{

if(snap.snapshot.value != null)

{

if((snap.snapshot.value as Map)["blockStatus"] == "no")

{

setState(() {

userName = (snap.snapshot.value as Map)["name"];

userPhone = (snap.snapshot.value as Map)["phone"].toString();

});

}

else

{

FirebaseAuth.instance.signOut();

Navigator.push(context, MaterialPageRoute(builder: (c)=> LoginScreen()));

cMethods.displaySnackBar("you are blocked. Contact admin: alizeb875@gmail.com", context);

}

}

else

{

FirebaseAuth.instance.signOut();

Navigator.push(context, MaterialPageRoute(builder: (c)=> LoginScreen()));

}

});

}

displayUserRideDetailsContainer() async

{

///Directions API

await retrieveDirectionDetails();

setState(() {

searchContainerHeight = 0;

bottomMapPadding = 240;

rideDetailsContainerHeight = 242;

isDrawerOpened = false;

});

}

retrieveDirectionDetails() async

{

var pickUpLocation = Provider.of<AppInfo>(context, listen: false).pickUpLocation;

var dropOffDestinationLocation = Provider.of<AppInfo>(context, listen: false).dropOffLocation;

var pickupGeoGraphicCoOrdinates = LatLng(pickUpLocation!.latitudePosition!, pickUpLocation.longitudePosition!);

var dropOffDestinationGeoGraphicCoOrdinates = LatLng(dropOffDestinationLocation!.latitudePosition!, dropOffDestinationLocation.longitudePosition!);

showDialog(

barrierDismissible: false,

context: context,

builder: (BuildContext context) => LoadingDialog(messageText: "Getting direction..."),

);

///Directions API

var detailsFromDirectionAPI = await CommonMethods.getDirectionDetailsFromAPI(pickupGeoGraphicCoOrdinates, dropOffDestinationGeoGraphicCoOrdinates);

setState(() {

tripDirectionDetailsInfo = detailsFromDirectionAPI;

});

Navigator.pop(context);

//draw route from pickup to dropOffDestination

PolylinePoints pointsPolyline = PolylinePoints();

List<PointLatLng> latLngPointsFromPickUpToDestination = pointsPolyline.decodePolyline(tripDirectionDetailsInfo!.encodedPoints!);

polylineCoOrdinates.clear();

if(latLngPointsFromPickUpToDestination.isNotEmpty)

{

latLngPointsFromPickUpToDestination.forEach((PointLatLng latLngPoint)

{

polylineCoOrdinates.add(LatLng(latLngPoint.latitude, latLngPoint.longitude));

});

}

polylineSet.clear();

setState(() {

Polyline polyline = Polyline(

polylineId: const PolylineId("polylineID"),

color: Colors.pink,

points: polylineCoOrdinates,

jointType: JointType.round,

width: 4,

startCap: Cap.roundCap,

endCap: Cap.roundCap,

geodesic: true,

);

polylineSet.add(polyline);

});

//fit the polyline into the map

LatLngBounds boundsLatLng;

if(pickupGeoGraphicCoOrdinates.latitude > dropOffDestinationGeoGraphicCoOrdinates.latitude

&& pickupGeoGraphicCoOrdinates.longitude > dropOffDestinationGeoGraphicCoOrdinates.longitude)

{

boundsLatLng = LatLngBounds(

southwest: dropOffDestinationGeoGraphicCoOrdinates,

northeast: pickupGeoGraphicCoOrdinates,

);

}

else if(pickupGeoGraphicCoOrdinates.longitude > dropOffDestinationGeoGraphicCoOrdinates.longitude)

{

boundsLatLng = LatLngBounds(

southwest: LatLng(pickupGeoGraphicCoOrdinates.latitude, dropOffDestinationGeoGraphicCoOrdinates.longitude),

northeast: LatLng(dropOffDestinationGeoGraphicCoOrdinates.latitude, pickupGeoGraphicCoOrdinates.longitude),

);

}

else if(pickupGeoGraphicCoOrdinates.latitude > dropOffDestinationGeoGraphicCoOrdinates.latitude)

{

boundsLatLng = LatLngBounds(

southwest: LatLng(dropOffDestinationGeoGraphicCoOrdinates.latitude, pickupGeoGraphicCoOrdinates.longitude),

northeast: LatLng(pickupGeoGraphicCoOrdinates.latitude, dropOffDestinationGeoGraphicCoOrdinates.longitude),

);

}

else

{

boundsLatLng = LatLngBounds(

southwest: pickupGeoGraphicCoOrdinates,

northeast: dropOffDestinationGeoGraphicCoOrdinates,

);

}

controllerGoogleMap!.animateCamera(CameraUpdate.newLatLngBounds(boundsLatLng, 72));

//add markers to pickup and dropOffDestination points

Marker pickUpPointMarker = Marker(

markerId: const MarkerId("pickUpPointMarkerID"),

position: pickupGeoGraphicCoOrdinates,

icon: BitmapDescriptor.defaultMarkerWithHue(BitmapDescriptor.hueGreen),

infoWindow: InfoWindow(title: pickUpLocation.placeName, snippet: "Pickup Location"),

);

Marker dropOffDestinationPointMarker = Marker(

markerId: const MarkerId("dropOffDestinationPointMarkerID"),

position: dropOffDestinationGeoGraphicCoOrdinates,

icon: BitmapDescriptor.defaultMarkerWithHue(BitmapDescriptor.hueYellow),

infoWindow: InfoWindow(title: dropOffDestinationLocation.placeName, snippet: "Destination Location"),

);

setState(() {

markerSet.add(pickUpPointMarker);

markerSet.add(dropOffDestinationPointMarker);

});

//add circles to pickup and dropOffDestination points

Circle pickUpPointCircle = Circle(

circleId: const CircleId('pickupCircleID'),

strokeColor: Colors.blue,

strokeWidth: 4,

radius: 14,

center: pickupGeoGraphicCoOrdinates,

fillColor: Colors.pink,

);

Circle dropOffDestinationPointCircle = Circle(

circleId: const CircleId('dropOffDestinationCircleID'),

strokeColor: Colors.blue,

strokeWidth: 4,

radius: 14,

center: dropOffDestinationGeoGraphicCoOrdinates,

fillColor: Colors.pink,

);

setState(() {

circleSet.add(pickUpPointCircle);

circleSet.add(dropOffDestinationPointCircle);

});

}

resetAppNow()

{

setState(() {

polylineCoOrdinates.clear();

polylineSet.clear();

markerSet.clear();

circleSet.clear();

rideDetailsContainerHeight = 0;

requestContainerHeight = 0;

tripContainerHeight = 0;

searchContainerHeight = 276;

bottomMapPadding = 300;

isDrawerOpened = true;

status = "";

nameDriver = "";

photoDriver = "";

phoneNumberDriver = "";

carDetailsDriver = "";

tripStatusDisplay = 'Driver is Arriving';

});

}

cancelRideRequest()

{

//remove ride request from database

tripRequestRef!.remove();

setState(() {

stateOfApp = "normal";

});

}

displayRequestContainer()

{

setState(() {

rideDetailsContainerHeight = 0;

requestContainerHeight = 220;

bottomMapPadding = 200;

isDrawerOpened = true;

});

//send ride request

makeTripRequest();

}

updateAvailableNearbyOnlineDriversOnMap()

{

setState(() {

markerSet.clear();

});

Set<Marker> markersTempSet = Set<Marker>();

for(OnlineNearbyDrivers eachOnlineNearbyDriver in ManageDriversMethods.nearbyOnlineDriversList)

{

LatLng driverCurrentPosition = LatLng(eachOnlineNearbyDriver.latDriver!, eachOnlineNearbyDriver.lngDriver!);

Marker driverMarker = Marker(

markerId: MarkerId("driver ID = " + eachOnlineNearbyDriver.uidDriver.toString()),

position: driverCurrentPosition,

icon: carIconNearbyDriver!,

);

markersTempSet.add(driverMarker);

}

setState(() {

markerSet = markersTempSet;

});

}

initializeGeoFireListener()

{

Geofire.initialize("onlineDrivers");

Geofire.queryAtLocation(currentPositionOfUser!.latitude, currentPositionOfUser!.longitude, 22)!

.listen((driverEvent)

{

if(driverEvent != null)

{

var onlineDriverChild = driverEvent["callBack"];

switch(onlineDriverChild)

{

case Geofire.onKeyEntered:

OnlineNearbyDrivers onlineNearbyDrivers = OnlineNearbyDrivers();

onlineNearbyDrivers.uidDriver = driverEvent["key"];

onlineNearbyDrivers.latDriver = driverEvent["latitude"];

onlineNearbyDrivers.lngDriver = driverEvent["longitude"];

ManageDriversMethods.nearbyOnlineDriversList.add(onlineNearbyDrivers);

if(nearbyOnlineDriversKeysLoaded == true)

{

//update drivers on google map

updateAvailableNearbyOnlineDriversOnMap();

}

break;

case Geofire.onKeyExited:

ManageDriversMethods.removeDriverFromList(driverEvent["key"]);

//update drivers on google map

updateAvailableNearbyOnlineDriversOnMap();

break;

case Geofire.onKeyMoved:

OnlineNearbyDrivers onlineNearbyDrivers = OnlineNearbyDrivers();

onlineNearbyDrivers.uidDriver = driverEvent["key"];

onlineNearbyDrivers.latDriver = driverEvent["latitude"];

onlineNearbyDrivers.lngDriver = driverEvent["longitude"];

ManageDriversMethods.updateOnlineNearbyDriversLocation(onlineNearbyDrivers);

//update drivers on google map

updateAvailableNearbyOnlineDriversOnMap();

break;

case Geofire.onGeoQueryReady:

nearbyOnlineDriversKeysLoaded = true;

//update drivers on google map

updateAvailableNearbyOnlineDriversOnMap();

break;

}

}

});

}

makeTripRequest()

{

tripRequestRef = FirebaseDatabase.instance.ref().child("tripRequests").push();

var pickUpLocation = Provider.of<AppInfo>(context, listen: false).pickUpLocation;

var dropOffDestinationLocation = Provider.of<AppInfo>(context, listen: false).dropOffLocation;

Map pickUpCoOrdinatesMap =

{

"latitude": pickUpLocation!.latitudePosition.toString(),

"longitude": pickUpLocation.longitudePosition.toString(),

};

Map dropOffDestinationCoOrdinatesMap =

{

"latitude": dropOffDestinationLocation!.latitudePosition.toString(),

"longitude": dropOffDestinationLocation.longitudePosition.toString(),

};

Map driverCoOrdinates =

{

"latitude": "",

"longitude": "",

};

Map dataMap =

{

"tripID": tripRequestRef!.key,

"publishDateTime": DateTime.now().toString(),

"userName": userName,

"userPhone": userPhone,

"userID": userID,

"pickUpLatLng": pickUpCoOrdinatesMap,

"dropOffLatLng": dropOffDestinationCoOrdinatesMap,

"pickUpAddress": pickUpLocation.placeName,

"dropOffAddress": dropOffDestinationLocation.placeName,

"driverID": "waiting",

"carDetails": "",

"driverLocation": driverCoOrdinates,

"driverName": "",

"driverPhone": "",

"driverPhoto": "",

"fareAmount": "",

"status": "new",

};

tripRequestRef!.set(dataMap);

tripStreamSubscription = tripRequestRef!.onValue.listen((eventSnapshot) async

{

if(eventSnapshot.snapshot.value == null)

{

return;

}

if((eventSnapshot.snapshot.value as Map)["driverName"] != null)

{

nameDriver = (eventSnapshot.snapshot.value as Map)["driverName"];

}

if((eventSnapshot.snapshot.value as Map)["driverPhone"] != null)

{

phoneNumberDriver = (eventSnapshot.snapshot.value as Map)["driverPhone"];

}

if((eventSnapshot.snapshot.value as Map)["driverPhoto"] != null)

{

photoDriver = (eventSnapshot.snapshot.value as Map)["driverPhoto"];

}

if((eventSnapshot.snapshot.value as Map)["carDetails"] != null)

{

carDetailsDriver = (eventSnapshot.snapshot.value as Map)["carDetails"];

}

if((eventSnapshot.snapshot.value as Map)["status"] != null)

{

status = (eventSnapshot.snapshot.value as Map)["status"];

}

if((eventSnapshot.snapshot.value as Map)["driverLocation"] != null)

{

double driverLatitude = double.parse((eventSnapshot.snapshot.value as Map)["driverLocation"]["latitude"].toString());

double driverLongitude = double.parse((eventSnapshot.snapshot.value as Map)["driverLocation"]["longitude"].toString());

LatLng driverCurrentLocationLatLng = LatLng(driverLatitude, driverLongitude);

if(status == "accepted")

{

//update info for pickup to user on UI

//info from driver current location to user pickup location

updateFromDriverCurrentLocationToPickUp(driverCurrentLocationLatLng);

}

else if(status == "arrived")

{

//update info for arrived - when driver reach at the pickup point of user

setState(() {

tripStatusDisplay = 'Driver has Arrived';

});

}

else if(status == "ontrip")

{

//update info for dropoff to user on UI

//info from driver current location to user dropoff location

updateFromDriverCurrentLocationToDropOffDestination(driverCurrentLocationLatLng);

}

}

if(status == "accepted")

{

displayTripDetailsContainer();

Geofire.stopListener();

//remove drivers markers

setState(() {

markerSet.removeWhere((element) => element.markerId.value.contains("driver"));

});

}

if(status == "ended")

{

if((eventSnapshot.snapshot.value as Map)["fareAmount"] != null)

{

double fareAmount = double.parse((eventSnapshot.snapshot.value as Map)["fareAmount"].toString());

var responseFromPaymentDialog = await showDialog(

context: context,

builder: (BuildContext context) => PaymentDialog(fareAmount: fareAmount.toString()),

);

if(responseFromPaymentDialog == "paid")

{

tripRequestRef!.onDisconnect();

tripRequestRef = null;

tripStreamSubscription!.cancel();

tripStreamSubscription = null;

resetAppNow();

Restart.restartApp();

}

}

}

});

}

displayTripDetailsContainer()

{

setState(() {

requestContainerHeight = 0;

tripContainerHeight = 291;

bottomMapPadding = 281;

});

}

updateFromDriverCurrentLocationToPickUp(driverCurrentLocationLatLng) async

{

if(!requestingDirectionDetailsInfo)

{

requestingDirectionDetailsInfo = true;

var userPickUpLocationLatLng = LatLng(currentPositionOfUser!.latitude, currentPositionOfUser!.longitude);

var directionDetailsPickup = await CommonMethods.getDirectionDetailsFromAPI(driverCurrentLocationLatLng, userPickUpLocationLatLng);

if(directionDetailsPickup == null)

{

return;

}

setState(() {

tripStatusDisplay = "Driver is Coming - ${directionDetailsPickup.durationTextString}";

});

requestingDirectionDetailsInfo = false;

}

}

updateFromDriverCurrentLocationToDropOffDestination(driverCurrentLocationLatLng) async

{

if(!requestingDirectionDetailsInfo)

{

requestingDirectionDetailsInfo = true;

var dropOffLocation = Provider.of<AppInfo>(context, listen: false).dropOffLocation;

var userDropOffLocationLatLng = LatLng(dropOffLocation!.latitudePosition!, dropOffLocation.longitudePosition!);

var directionDetailsPickup = await CommonMethods.getDirectionDetailsFromAPI(driverCurrentLocationLatLng, userDropOffLocationLatLng);

if(directionDetailsPickup == null)

{

return;

}

setState(() {

tripStatusDisplay = "Driving to DropOff Location - ${directionDetailsPickup.durationTextString}";

});

requestingDirectionDetailsInfo = false;

}

}

noDriverAvailable()

{

showDialog(

context: context,

barrierDismissible: false,

builder: (BuildContext context) => InfoDialog(

title: "No Driver Available",

description: "No driver found in the nearby location. Please try again shortly.",

)

);

}

searchDriver()

{

if(availableNearbyOnlineDriversList!.length == 0)

{

cancelRideRequest();

resetAppNow();

noDriverAvailable();

return;

}

var currentDriver = availableNearbyOnlineDriversList![0];

//send notification to this currentDriver - currentDriver means selected driver

sendNotificationToDriver(currentDriver);

availableNearbyOnlineDriversList!.removeAt(0);

}

sendNotificationToDriver(OnlineNearbyDrivers currentDriver)

{

//update driver's newTripStatus - assign tripID to current driver

DatabaseReference currentDriverRef = FirebaseDatabase.instance

.ref()

.child("drivers")

.child(currentDriver.uidDriver.toString())

.child("newTripStatus");

currentDriverRef.set(tripRequestRef!.key);

//get current driver device recognition token

DatabaseReference tokenOfCurrentDriverRef = FirebaseDatabase.instance

.ref()

.child("drivers")

.child(currentDriver.uidDriver.toString())

.child("deviceToken");

tokenOfCurrentDriverRef.once().then((dataSnapshot)

{

if(dataSnapshot.snapshot.value != null)

{

String deviceToken = dataSnapshot.snapshot.value.toString();

//send notification

PushNotificationService.sendNotificationToSelectedDriver(

deviceToken,

context,

tripRequestRef!.key.toString()

);

}

else

{

return;

}

const oneTickPerSec = Duration(seconds: 1);

var timerCountDown = Timer.periodic(oneTickPerSec, (timer)

{

requestTimeoutDriver = requestTimeoutDriver - 1;

//when trip request is not requesting means trip request cancelled - stop timer

if(stateOfApp != "requesting")

{

timer.cancel();

currentDriverRef.set("cancelled");

currentDriverRef.onDisconnect();

requestTimeoutDriver = 20;

}

//when trip request is accepted by online nearest available driver

currentDriverRef.onValue.listen((dataSnapshot)

{

if(dataSnapshot.snapshot.value.toString() == "accepted")

{

timer.cancel();

currentDriverRef.onDisconnect();

requestTimeoutDriver = 20;

}

});

//if 20 seconds passed - send notification to next nearest online available driver

if(requestTimeoutDriver == 0)

{

currentDriverRef.set("timeout");

timer.cancel();

currentDriverRef.onDisconnect();

requestTimeoutDriver = 20;

//send notification to next nearest online available driver

searchDriver();

}

});

});

}

@override

Widget build(BuildContext context)

{

makeDriverNearbyCarIcon();

return Scaffold(

key: sKey,

drawer: Container(

width: 255,

color: Colors.black87,

child: Drawer(

backgroundColor: Colors.white10,

child: ListView(

children: [

const Divider(

height: 1,

color: Colors.grey,

thickness: 1,

),

//header

Container(

color: Colors.black54,

height: 160,

child: GestureDetector(

onTap: () {

Navigator.push(

context,

MaterialPageRoute(builder: (context) => ProfileDetailsPage()),

);

},

child: DrawerHeader(

decoration: const BoxDecoration(

color: Colors.white10,

),

child: Row(

children: [

Image.asset(

"assets/images/avatarman.png",

width: 60,

height: 60,

),

const SizedBox(width: 16,),

Column(

mainAxisAlignment: MainAxisAlignment.center,

children: [

Text(

userName,

style: const TextStyle(

fontSize: 16,

color: Colors.grey,

fontWeight: FontWeight.bold,

),

),

const SizedBox(height: 4,),

const Text(

"Profile",

style: TextStyle(

color: Colors.white38,

),

),

],

),

],

),

),

),

),

const Divider(

height: 1,

color: Colors.grey,

thickness: 1,

),

const SizedBox(height: 10,),

//body

GestureDetector(

onTap: ()

{

Navigator.push(context, MaterialPageRoute(builder: (c)=> TripsHistoryPage()));

},

child: ListTile(

leading: IconButton(

onPressed: () {},

icon: const Icon(Icons.history, color: Colors.grey,),

),

title: const Text("History", style: TextStyle(color: Colors.grey),),

),

),

GestureDetector(

onTap: ()

{

FirebaseAuth.instance.signOut();

Navigator.push(context, MaterialPageRoute(builder: (c)=> LoginScreen()));

},

child: ListTile(

leading: IconButton(

onPressed: (){},

icon: const Icon(Icons.logout, color: Colors.grey,),

),

title: const Text("Logout", style: TextStyle(color: Colors.grey),),

),

),

],

),

),

),

body: Stack(

children: [

///google map

GoogleMap(

padding: EdgeInsets.only(top: 26, bottom: bottomMapPadding),

mapType: MapType.normal,

myLocationEnabled: true,

polylines: polylineSet,

markers: markerSet,

circles: circleSet,

initialCameraPosition: googlePlexInitialPosition,

onMapCreated: (GoogleMapController mapController)

{

controllerGoogleMap = mapController;

updateMapTheme(controllerGoogleMap!);

googleMapCompleterController.complete(controllerGoogleMap);

setState(() {

bottomMapPadding = 300;

});

getCurrentLiveLocationOfUser();

},

),

///drawer button

Positioned(

top: 36,

left: 19,

child: GestureDetector(

onTap: ()

{

if(isDrawerOpened == true)

{

sKey.currentState!.openDrawer();

}

else

{

resetAppNow();

}

},

child: Container(

decoration: BoxDecoration(

color: Colors.white,

borderRadius: BorderRadius.circular(20),

boxShadow: const

[

BoxShadow(

color: Colors.black26,

blurRadius: 5,

spreadRadius: 0.5,

offset: Offset(0.7, 0.7),

),

],

),

child: CircleAvatar(

backgroundColor: Colors.grey,

radius: 20,

child: Icon(

isDrawerOpened == true ? Icons.menu : Icons.close,

color: Colors.black87,

),

),

),

),

),

///search location icon button

Positioned(

left: 0,

right: 0,

bottom: -80,

child: Container(

height: searchContainerHeight,

child: Row(

mainAxisAlignment: MainAxisAlignment.spaceAround,

children: [

ElevatedButton(

onPressed: () async

{

var responseFromSearchPage = await Navigator.push(context, MaterialPageRoute(builder: (c)=> SearchDestinationPage()));

if(responseFromSearchPage == "placeSelected")

{

displayUserRideDetailsContainer();

}

},

style: ElevatedButton.styleFrom(

backgroundColor: Colors.grey,

shape: const CircleBorder(),

padding: const EdgeInsets.all(24)

),

child: const Icon(

Icons.search,

color: Colors.white,

size: 25,

),

),

ElevatedButton(

onPressed: () {},

style: ElevatedButton.styleFrom(

backgroundColor: Colors.grey,

shape: const CircleBorder(),

padding: const EdgeInsets.all(24)

),

child: const Icon(

Icons.home,

color: Colors.white,

size: 25,

),

),

ElevatedButton(

onPressed: () {},

style: ElevatedButton.styleFrom(

backgroundColor: Colors.grey,

shape: const CircleBorder(),

padding: const EdgeInsets.all(24)

),

child: const Icon(

Icons.work,

color: Colors.white,

size: 25,

),

),

],

),

),

),

///ride details container

Positioned(

left: 0,

right: 0,

bottom: 0,

child: Container(

height: rideDetailsContainerHeight,

decoration: const BoxDecoration(

color: Colors.black54,

borderRadius: BorderRadius.only(topLeft: Radius.circular(15), topRight: Radius.circular(15)),

boxShadow:

[

BoxShadow(

color: Colors.white12,

blurRadius: 15.0,

spreadRadius: 0.5,

offset: Offset(.7, .7),

),

],

),

child: Padding(

padding: const EdgeInsets.symmetric(vertical: 18),

child: Column(

mainAxisAlignment: MainAxisAlignment.center,

children: [

Padding(

padding: const EdgeInsets.only(left: 16, right: 16),

child: SizedBox(

height: 200,

child: Card(

elevation: 10,

child: Container(

width: MediaQuery.of(context).size.width \* .70,

color: Colors.black45,

child: Padding(

padding: const EdgeInsets.only(top: 8, bottom: 8),

child: Column(

mainAxisAlignment: MainAxisAlignment.center,

children: [

Padding(

padding: const EdgeInsets.only(left: 8, right: 8),

child: Row(

mainAxisAlignment: MainAxisAlignment.spaceBetween,

children: [

Text(

(tripDirectionDetailsInfo != null) ? tripDirectionDetailsInfo!.distanceTextString! : "",

style: const TextStyle(

fontSize: 16,

color: Colors.white70,

fontWeight: FontWeight.bold,

),

),

Text(

(tripDirectionDetailsInfo != null) ? tripDirectionDetailsInfo!.durationTextString! : "",

style: const TextStyle(

fontSize: 16,

color: Colors.white70,

fontWeight: FontWeight.bold,

),

),

],

),

),

GestureDetector(

onTap: ()

{

setState(() {

stateOfApp = "requesting";

});

displayRequestContainer();

//get nearest available online drivers

availableNearbyOnlineDriversList = ManageDriversMethods.nearbyOnlineDriversList;

//search driver

searchDriver();

},

child: Image.asset(

"assets/images/uberexec.png",

height: 122,

width: 122,

),

),

Text(

(tripDirectionDetailsInfo != null) ? "\$ ${(cMethods.calculateFareAmount(tripDirectionDetailsInfo!)).toString()}" : "",

style: const TextStyle(

fontSize: 18,

color: Colors.white70,

fontWeight: FontWeight.bold,

),

),

],

),

),

),

),

),

),

],

),

),

),

),

///request container

Positioned(

left: 0,

right: 0,

bottom: 0,

child: Container(

height: requestContainerHeight,

decoration: const BoxDecoration(

color: Colors.black54,

borderRadius: BorderRadius.only(topLeft: Radius.circular(16), topRight: Radius.circular(16)),

boxShadow:

[

BoxShadow(

color: Colors.black26,

blurRadius: 15.0,

spreadRadius: 0.5,

offset: Offset(

0.7,

0.7,

),

),

],

),

child: Padding(

padding: const EdgeInsets.symmetric(horizontal: 24, vertical: 18),

child: Column(

crossAxisAlignment: CrossAxisAlignment.center,

children: [

const SizedBox(height: 12,),

SizedBox(

width: 200,

child: LoadingAnimationWidget.flickr(

leftDotColor: Colors.greenAccent,

rightDotColor: Colors.pinkAccent,

size: 50,

),

),

const SizedBox(height: 20,),

GestureDetector(

onTap: ()

{

resetAppNow();

cancelRideRequest();

},

child: Container(

height: 50,

width: 50,

decoration: BoxDecoration(

color: Colors.white70,

borderRadius: BorderRadius.circular(25),

border: Border.all(width: 1.5, color: Colors.grey),

),

child: const Icon(

Icons.close,

color: Colors.black,

size: 25,

),

),

),

],

),

),

),

),

///trip details container

Positioned(

left: 0,

right: 0,

bottom: 0,

child: Container(

height: tripContainerHeight,

decoration: const BoxDecoration(

color: Colors.black87,

borderRadius: BorderRadius.only(topLeft: Radius.circular(16), topRight: Radius.circular(16)),

boxShadow:

[

BoxShadow(

color: Colors.white24,

blurRadius: 15.0,

spreadRadius: 0.5,

offset: Offset(

0.7,

0.7,

),

),

],

),

child: Padding(

padding: const EdgeInsets.symmetric(horizontal: 24, vertical: 18),

child: Column(

crossAxisAlignment: CrossAxisAlignment.start,

children: [

const SizedBox(height: 5,),

//trip status display text

Row(

mainAxisAlignment: MainAxisAlignment.center,

children: [

Text(

tripStatusDisplay,

style: const TextStyle(fontSize: 19, color: Colors.grey,),

),

],

),

const SizedBox(height: 19,),

const Divider(

height: 1,

color: Colors.white70,

thickness: 1,

),

const SizedBox(height: 19,),

//image - driver name and driver car details

Row(

mainAxisAlignment: MainAxisAlignment.center,

crossAxisAlignment: CrossAxisAlignment.center,

children: [

ClipOval(

child: Image.network(

photoDriver == ''

? "https://firebasestorage.googleapis.com/v0/b/flutter-uber-clone-with-admin.appspot.com/o/avatarman.png?alt=media&token=7a04943c-a566-45d3-b820-d33da3b105c7"

: photoDriver,

width: 60,

height: 60,

fit: BoxFit.cover,

),

),

const SizedBox(width: 8,),

Column(

mainAxisAlignment: MainAxisAlignment.start,

crossAxisAlignment: CrossAxisAlignment.start,

children: [

Text(nameDriver, style: const TextStyle(fontSize: 20, color: Colors.grey,),),

Text(carDetailsDriver, style: const TextStyle(fontSize: 14, color: Colors.grey,),),

],

),

],

),

const SizedBox(height: 19,),

const Divider(

height: 1,

color: Colors.white70,

thickness: 1,

),

const SizedBox(height: 19,),

//call driver btn

Row(

mainAxisAlignment: MainAxisAlignment.center,

children: [

GestureDetector(

onTap: ()

{

launchUrl(Uri.parse("tel://$phoneNumberDriver"));

},

child: Column(

crossAxisAlignment: CrossAxisAlignment.center,

children: [

Container(

height: 50,

width: 50,

decoration: BoxDecoration(

borderRadius: const BorderRadius.all(Radius.circular(25)),

border: Border.all(

width: 1,

color: Colors.white,

),

),

child: const Icon(

Icons.phone,

color: Colors.white,

),

),

const SizedBox(height: 11,),

const Text("Call", style: TextStyle(color: Colors.grey,),),

],

),

),

],

),

],

),

),

),

),

],

),

);

}

}

class ProfileDetailsPage extends StatefulWidget {

const ProfileDetailsPage({Key? key}) : super(key: key);

@override

State<ProfileDetailsPage> createState() => \_ProfileDetailsPageState();

}

class \_ProfileDetailsPageState extends State<ProfileDetailsPage> {

final TextEditingController nameController = TextEditingController();

final TextEditingController phoneController = TextEditingController();

final TextEditingController emailController = TextEditingController();

@override

void initState() {

super.initState();

retrieveCurrentUserInfo();

setUserInfo();

}

void setUserInfo() {

nameController.text = userName;

phoneController.text = userPhone;

emailController.text = FirebaseAuth.instance.currentUser!.email.toString();

}

// Получаем данные пользователя из базы данных и отображаем их на странице

void retrieveCurrentUserInfo() async {

await FirebaseDatabase.instance.ref()

.child("users")

.child(FirebaseAuth.instance.currentUser!.uid)

.once().then((snap)

{

userName = (snap.snapshot.value as Map)["name"];

userPhone = (snap.snapshot.value as Map)["phone"];

userEmail = (snap.snapshot.value as Map)["email"];

});

}

// Обновляем данные пользователя в базе данных

void \_saveUserData() {

DatabaseReference \_userRef = FirebaseDatabase.instance.ref().child("users").child(FirebaseAuth.instance.currentUser!.uid);

\_userRef.update({

'name': nameController.text,

'phone': phoneController.text,

'email': emailController.text,

}).then((\_) {

ScaffoldMessenger.of(context).showSnackBar(const SnackBar(content: Text('Changes saved')));

}).catchError((error) {

ScaffoldMessenger.of(context).showSnackBar(const SnackBar(content: Text('Failed to save changes')));

});

}

@override

Widget build(BuildContext context) {

return Scaffold(

appBar: AppBar(

title: const Text('Edit Profile'),

actions: [

IconButton(

onPressed: \_saveUserData,

icon: const Icon(Icons.save),

),

],

),

body: Padding(

padding: const EdgeInsets.all(16.0),

child: Column(

crossAxisAlignment: CrossAxisAlignment.start,

children: [

TextField(

controller: nameController,

decoration: const InputDecoration(labelText: 'Name'),

),

TextField(

controller: phoneController,

decoration: const InputDecoration(labelText: 'Phone'),

),

TextField(

controller: emailController,

decoration: const InputDecoration(labelText: 'Email'),

),

],

),

),

);

}

}

class SearchDestinationPage extends StatefulWidget {

const SearchDestinationPage({super.key});

@override

State<SearchDestinationPage> createState() => \_SearchDestinationPageState();

}

class \_SearchDestinationPageState extends State<SearchDestinationPage>

{

TextEditingController pickUpTextEditingController = TextEditingController();

TextEditingController destinationTextEditingController = TextEditingController();

List<PredictionModel> dropOffPredictionsPlacesList = [];

bool isElectricCar = false;

bool isCombustionCar = false;

///Places API - Place AutoComplete

searchLocation(String locationName) async

{

if(locationName.length > 1)

{

String apiPlacesUrl = "https://maps.googleapis.com/maps/api/place/autocomplete/json?input=$locationName&key=$googleMapKey";

var responseFromPlacesAPI = await CommonMethods.sendRequestToAPI(apiPlacesUrl);

if(responseFromPlacesAPI == "error")

{

return;

}

if(responseFromPlacesAPI["status"] == "OK")

{

var predictionResultInJson = responseFromPlacesAPI["predictions"];

var predictionsList = (predictionResultInJson as List).map((eachPlacePrediction) => PredictionModel.fromJson(eachPlacePrediction)).toList();

setState(() {

dropOffPredictionsPlacesList = predictionsList;

});

}

}

}

@override

Widget build(BuildContext context)

{

String userAddress = Provider.of<AppInfo>(context, listen: false).pickUpLocation!.humanReadableAddress ?? "";

pickUpTextEditingController.text = userAddress;

return Scaffold(

body: SingleChildScrollView(

child: Column(

children: [

Card(

elevation: 10,

child: Container(

height: 335,

decoration: const BoxDecoration(

color: Colors.black12,

boxShadow:

[

BoxShadow(

color: Colors.black12,

blurRadius: 5.0,

spreadRadius: 0.5,

offset: Offset(0.7, 0.7),

),

],

),

child: Padding(

padding: const EdgeInsets.only(left: 24, top: 48, right: 24, bottom: 20),

child: Column(

children: [

const SizedBox(height: 6,),

//icon button - title

Stack(

children: [

GestureDetector(

onTap: ()

{

Navigator.pop(context);

},

child: const Icon(Icons.arrow\_back, color: Colors.white,),

),

const Center(

child: Text(

"Set Dropoff Location",

style: TextStyle(

fontSize: 18,

fontWeight: FontWeight.bold,

),

),

),

],

),

const SizedBox(height: 18,),

//pickup text field

Row(

children: [

Image.asset(

"assets/images/initial.png",

height: 16,

width: 16,

),

const SizedBox(width: 18,),

Expanded(

child: Container(

decoration: BoxDecoration(

color: Colors.grey,

borderRadius: BorderRadius.circular(5),

),

child: Padding(

padding: const EdgeInsets.all(3),

child: TextField(

controller: pickUpTextEditingController,

decoration: const InputDecoration(

hintText: "Pickup Address",

fillColor: Colors.white12,

filled: true,

border: InputBorder.none,

isDense: true,

contentPadding: EdgeInsets.only(left: 11, top: 9, bottom: 9)

),

),

),

),

),

],

),

const SizedBox(height: 11,),

//destination text field

Row(

children: [

Image.asset(

"assets/images/final.png",

height: 16,

width: 16,

),

const SizedBox(width: 18,),

Expanded(

child: Container(

decoration: BoxDecoration(

color: Colors.grey,

borderRadius: BorderRadius.circular(5),

),

child: Padding(

padding: const EdgeInsets.all(3),

child: TextField(

controller: destinationTextEditingController,

onChanged: (inputText)

{

searchLocation(inputText);

},

decoration: const InputDecoration(

hintText: "Destination Address",

fillColor: Colors.white12,

filled: true,

border: InputBorder.none,

isDense: true,

contentPadding: EdgeInsets.only(left: 11, top: 9, bottom: 9)

),

),

),

),

),

],

),

const SizedBox(height: 11,),

Row(

children: [

Checkbox(

value: isElectricCar,

onChanged: (value) {

setState(() {

isElectricCar = value!;

if (isElectricCar) {

isCombustionCar = false;

}

});

},

),

Text('Electric Car'),

],

),

Row(

children: [

Checkbox(

value: isCombustionCar,

onChanged: (value) {

setState(() {

isCombustionCar = value!;

if (isCombustionCar) {

isElectricCar = false;

}

});

},

),

Text('Combustion Car'),

],

),

],

),

),

),

),

//display prediction results for destination place

(dropOffPredictionsPlacesList.length > 0)

? Padding(

padding: const EdgeInsets.symmetric(vertical: 8, horizontal: 16),

child: ListView.separated(

padding: const EdgeInsets.all(0),

itemBuilder: (context, index)

{

return Card(

elevation: 3,

child: PredictionPlaceUI(

predictedPlaceData: dropOffPredictionsPlacesList[index],

),

);

},

separatorBuilder: (BuildContext context, int index) => const SizedBox(height: 2,),

itemCount: dropOffPredictionsPlacesList.length,

shrinkWrap: true,

physics: const ClampingScrollPhysics(),

),

)

: Container(),

],

),

),

);

}

}

class TripsHistoryPage extends StatefulWidget

{

const TripsHistoryPage({super.key});

@override

State<TripsHistoryPage> createState() => \_TripsHistoryPageState();

}

class \_TripsHistoryPageState extends State<TripsHistoryPage>

{

final completedTripRequestsOfCurrentUser = FirebaseDatabase.instance.ref().child("tripRequests");

@override

Widget build(BuildContext context)

{

return Scaffold(

appBar: AppBar(

title: const Text(

'My Trips History',

style: TextStyle(

color: Colors.white,

),

),

leading: IconButton(

onPressed: ()

{

Navigator.pop(context);

},

icon: const Icon(Icons.arrow\_back, color: Colors.white,),

),

),

body: StreamBuilder(

stream: completedTripRequestsOfCurrentUser.onValue,

builder: (BuildContext context, snapshotData)

{

if(snapshotData.hasError)

{

return const Center(

child: Text(

"Error Occurred.",

style: TextStyle(color: Colors.white),

),

);

}

if(!(snapshotData.hasData))

{

return const Center(

child: Text(

"No record found.",

style: TextStyle(color: Colors.white),

),

);

}

Map dataTrips = snapshotData.data!.snapshot.value as Map;

List tripsList = [];

dataTrips.forEach((key, value) => tripsList.add({"key": key, ...value}));

return ListView.builder(

shrinkWrap: true,

itemCount: tripsList.length,

itemBuilder: ((context, index)

{

if(tripsList[index]["status"] != null

&& tripsList[index]["status"] == "ended"

&& tripsList[index]["userID"] == FirebaseAuth.instance.currentUser!.uid)

{

return Card(

color: Colors.white12,

elevation: 10,

child: Padding(

padding: const EdgeInsets.symmetric(horizontal: 16, vertical: 16),

child: Column(

crossAxisAlignment: CrossAxisAlignment.start,

children: [

//pickup - fare amount

Row(

children: [

Image.asset('assets/images/initial.png', height: 16, width: 16,),

const SizedBox(width: 18,),

Expanded(

child: Text(

tripsList[index]["pickUpAddress"].toString(),

overflow: TextOverflow.ellipsis,

style: const TextStyle(

fontSize: 18,

color: Colors.white38,

),

),

),

const SizedBox(width: 5,),

Text(

"\$ " + tripsList[index]["fareAmount"].toString(),

style: const TextStyle(

fontSize: 16,

color: Colors.white,

),

),

],

),

const SizedBox(height: 8,),

//dropoff

Row(

children: [

Image.asset('assets/images/final.png', height: 16, width: 16,),

const SizedBox(width: 18,),

Expanded(

child: Text(

tripsList[index]["dropOffAddress"].toString(),

overflow: TextOverflow.ellipsis,

style: const TextStyle(

fontSize: 18,

color: Colors.white38,

),

),

),

],

),

],

),

),

);

}

else

{

return Container();

}

}),

);

},

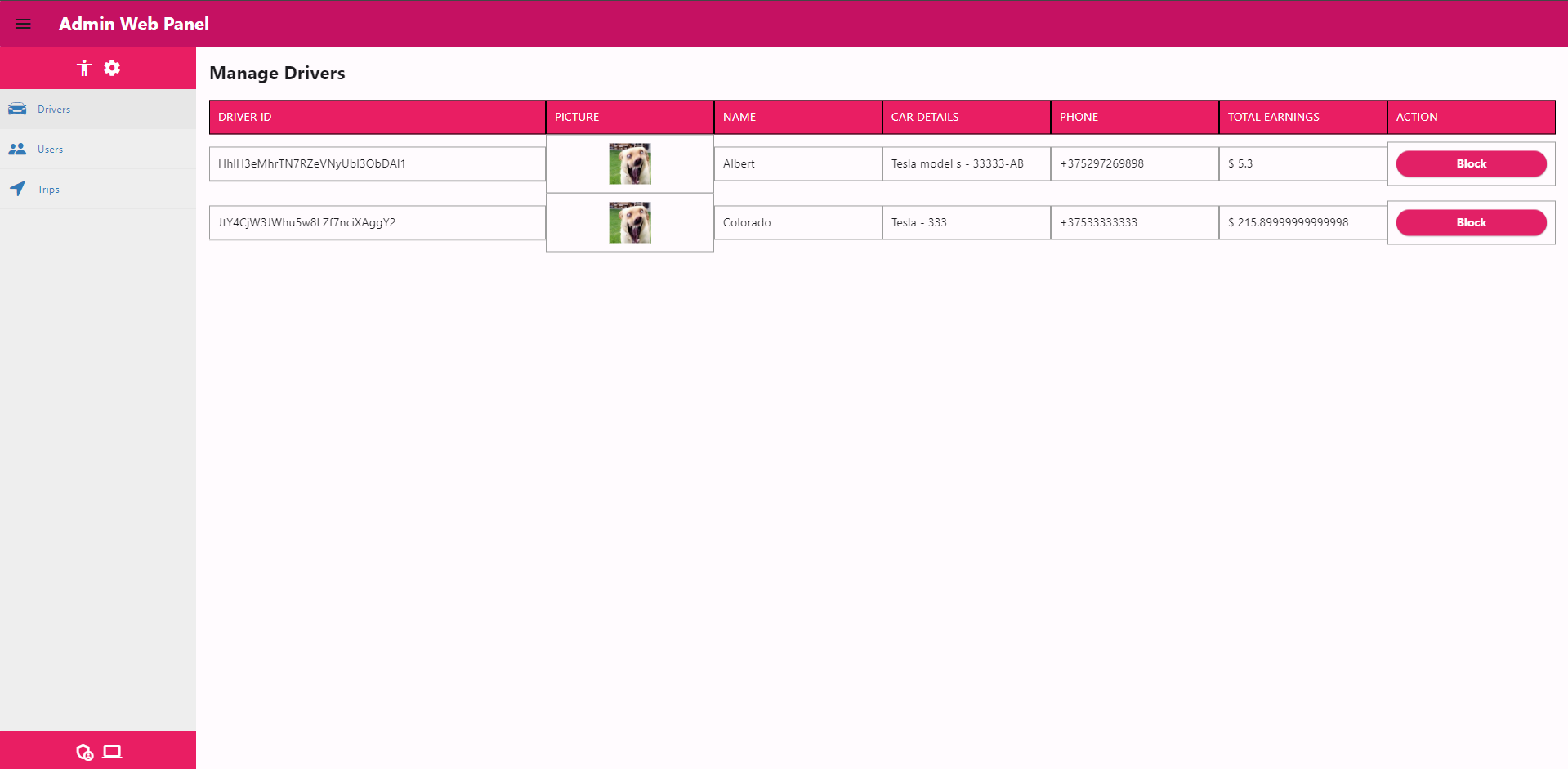
),

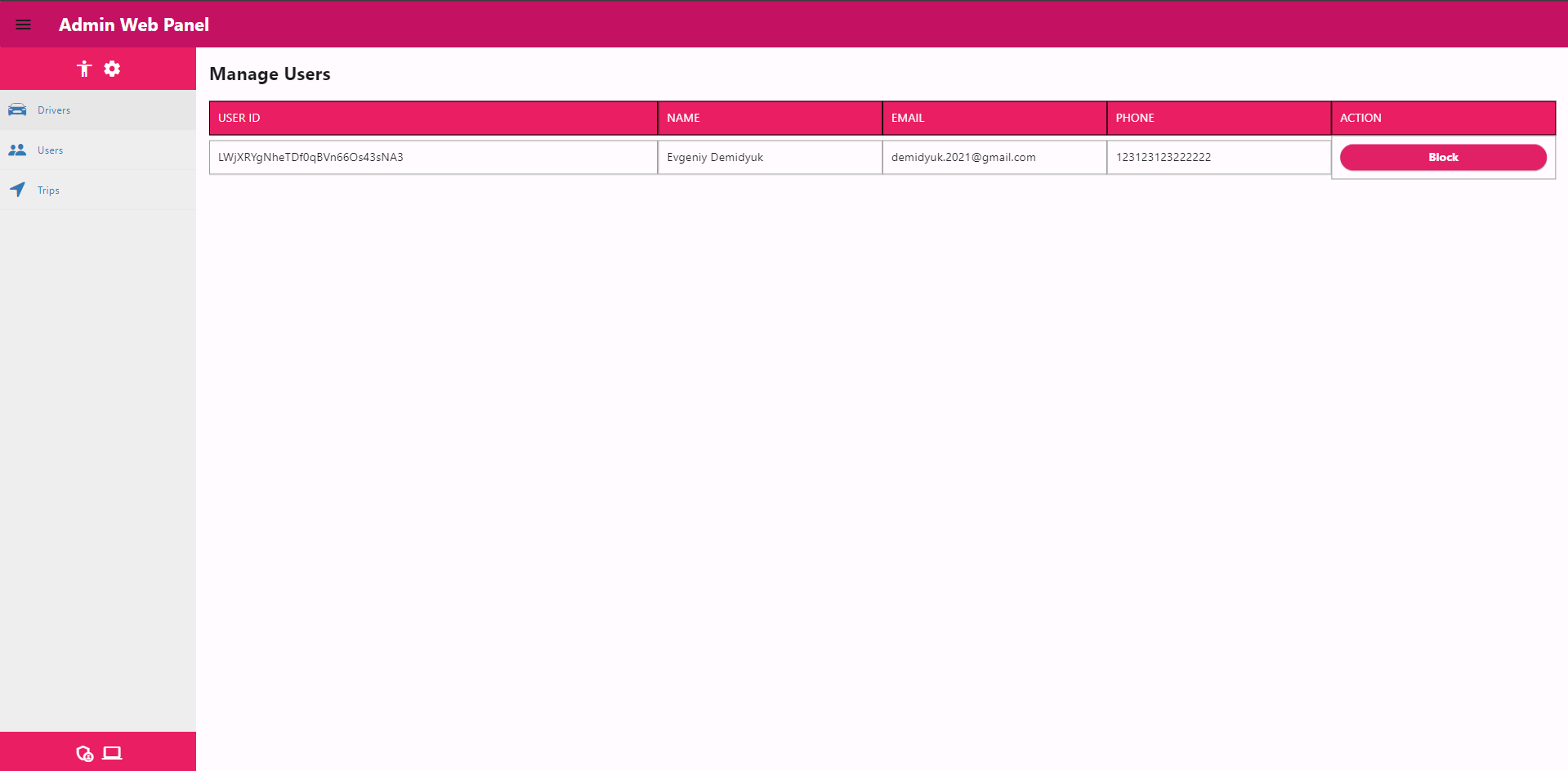
);

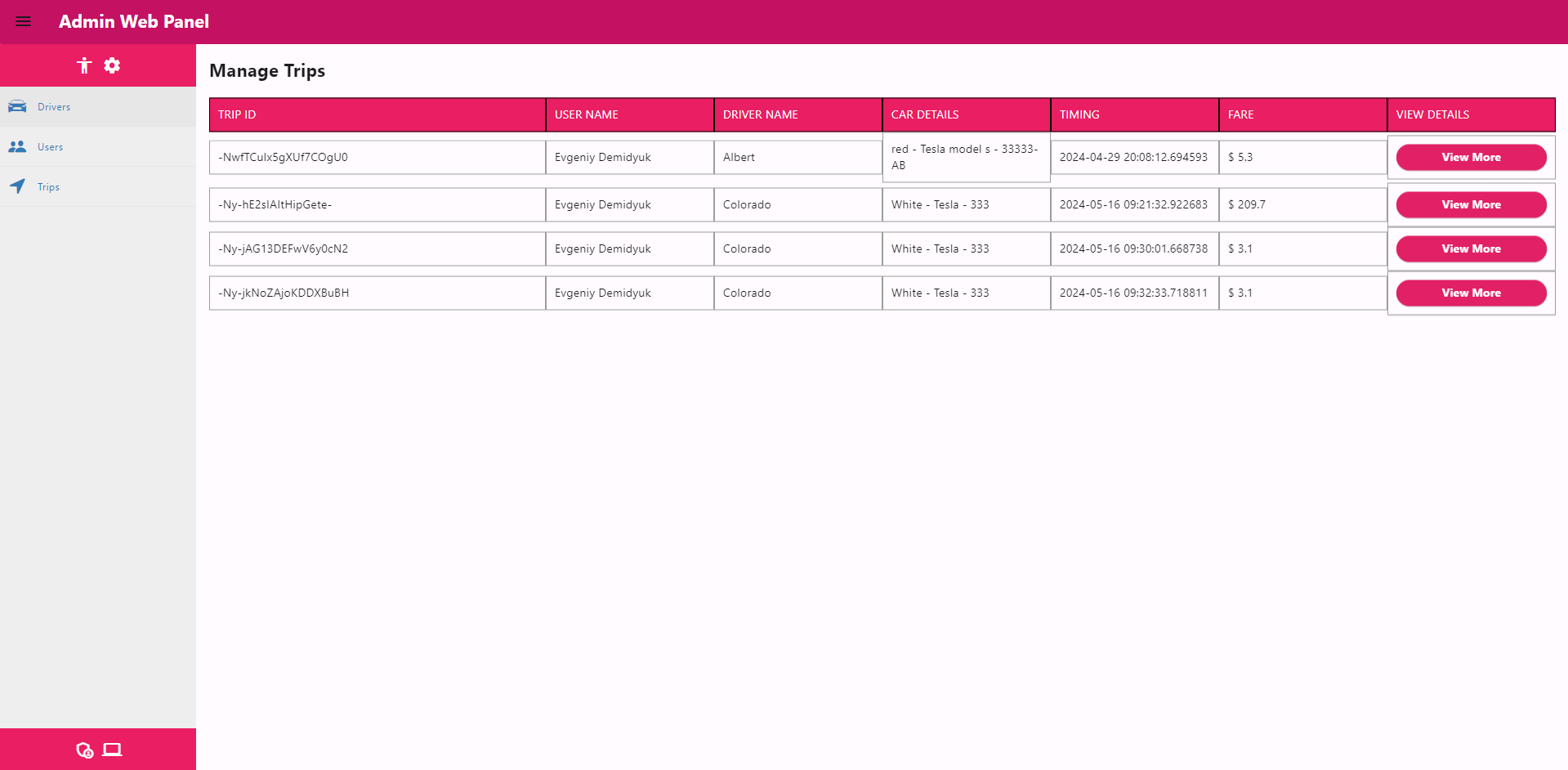
}

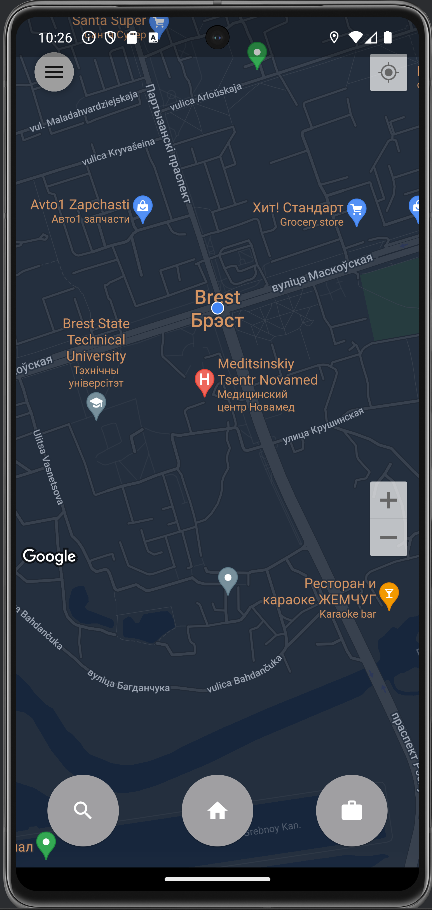
}

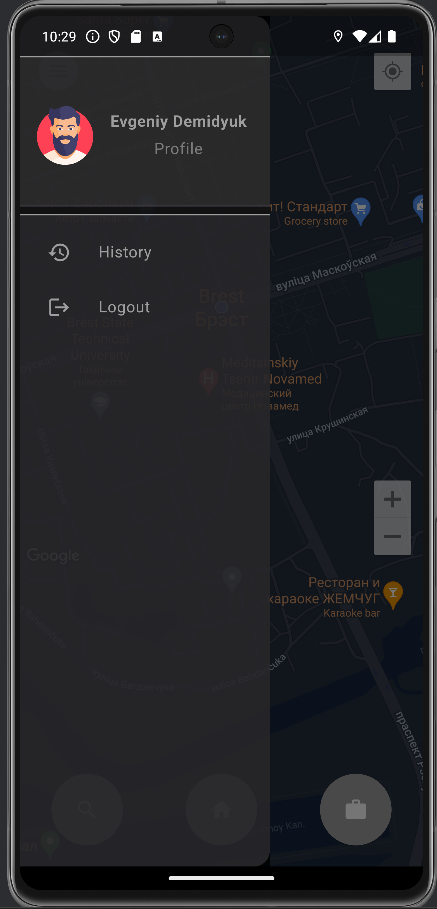
**Приложение Б**

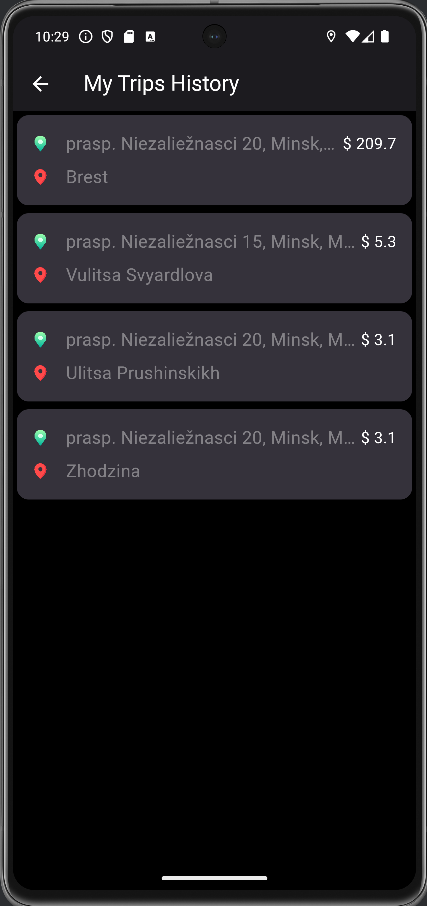
****

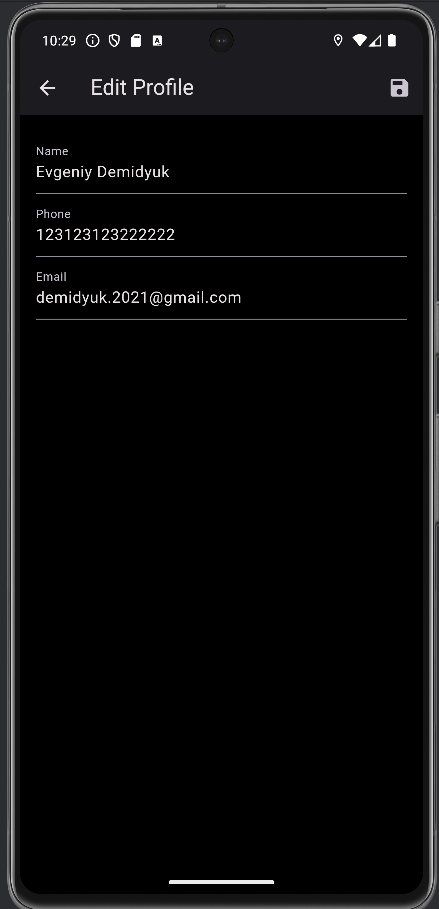
****

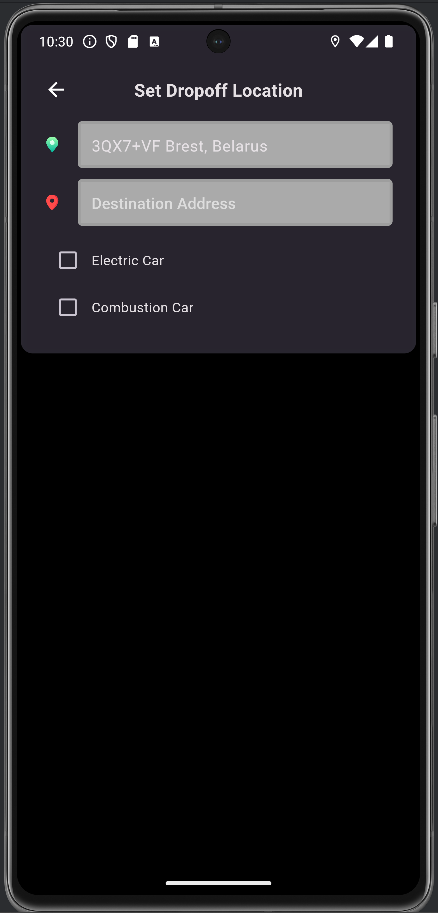
****

****

****

****

****

****