**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ**

**БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ**

**МЕХАНИКО-МАТЕМАТИЧЕСКИЙ ФАКУЛЬТЕТ**

[**Кафедра веб-технологий и компьютерного моделирования**](http://www.bsu.by/ru/main.aspx?guid=4361)

Жегздрин

Алексей Андреевич

**WEB-API И МОБИЛЬНОЕ ПРИЛОЖЕНИЕ НА ПЛАТФОРМЕ .NET ДЛЯ ПОИСКА ПОПУТЧИКОВ «DriveYou»**

Курсовая работа

Жегздрин Алексея Андреевича

студента 3 курса,

специальность  
«Математика и информационные технологии»

Научный руководитель:

старший преподаватель

Политаев Дмитрий Николаевич

Минск, 2022

СОДЕРЖАНИЕ

[Введение 3](#_Toc107436735)

[ГЛАВА 1. ПОСТАНОВКА ЗАДАЧИ 5](#_Toc107436736)

[1.1 Описание предметной области 5](#_Toc107436737)

[1.2 Модель предметной области 5](#_Toc107436738)

[1.3 Функциональная модель игры 6](#_Toc107436739)

[ГЛАВА 2. ОБЗОР СРЕДСТВ РАЗРАБОТКИ CЕРВИСА 8](#_Toc107436740)

[2.1 Задачи, решаемые при разработке сервиса «DriveYou» 8](#_Toc107436741)

[2.2 Средства разработки и фреймворки 8](#_Toc107436742)

[2.3 Выбор и средств разработки 10](#_Toc107436743)

[ГЛАВА 3. ПРОЦЕСС СОЗДАНИЯ СЕРВИСА 12](#_Toc107436744)

[3.1 Установка и описание средств для разработки 12](#_Toc107436745)

[3.2 Разработка сервиса 16](#_Toc107436746)

[3.2.1 Разработка Web-API 16](#_Toc107436747)

[3.2.2 Разработка мобильного приложения 17](#_Toc107436748)

[ГЛАВА 4. ОПИСАНИЕ СЕРВИСА 21](#_Toc107436749)

[4.1 Программно-аппаратные ресурсы 21](#_Toc107436750)

[4.2 Демонстративный пример работы сервиса 21](#_Toc107436751)

[4.3 Тестирование 28](#_Toc107436752)

[Заключение 29](#_Toc107436753)

[Список использованных источников 30](#_Toc107436754)

[Приложения 31](#_Toc107436755)

## Введение

Тема данной курсовой работы – Web-API и мобильное приложение для смартфонов под управлением Android.

Данная курсовая работа была выполнена с целью практического освоения основных приемов и правил создания Web-сервисов и мобильных приложений. В качестве основных инструментов была выбрана платформа .NET. Для Web-API использовался .NET 5.0, мобильно приложение создавалось с помощью .NET Standard 2.0 и фреймворка Xamarin.

В наше время пассажирские перевозки пользуются большой популярностью. Однако цены не всегда способны порадовать пользователей, да и комфорт в некоторых случаях оставляет желать лучшего, ведь никому не хочется ехать в доверху наполненном людьми плацкарте с температурой +30 за бортом. Да и нужно подстраиваться под время отправления транспорта, что-то же не всегда удобно. Раньше многие довольно часто пользовались «автостопом», ловили машины на трассе которые едут в тот-же город или хотя-бы в ту же сторону. Кому-то везло, и они сразу находили машину едущую, к примеру до Минска, кому-то везло меньше и ему приходилось делать несколько «пересадок». Сейчас, с текущим развитием информационных технологий поиск попутчиков и попутных авто, стал намного проще и удобнее. Плюс для пассажиров заключается в большем комфорте, гибком времени и стоимости, которая зачастую будет меньше, чем на общественном транспорте. Плюс для водителей в том, что их поездки частично компенсируются за счет пассажиров, а также приятная компания на время поездки.

Данная курсовая работа посвящена разработке сервиса «DriveYou» с использованием .NET 5.0, .NET Standard 2.0 и Xamarin.

Главной задачей исследования входит обучение, изучение любых материалов для реализации своей работы. Необходимо изучить всевозможные функции платформы .NET и фреймворка Xamarin, которые помогут в процессе разработки моего приложения. Также для исследования входит изучение языка программирования C# и XAML.

Для выполнения дипломной работы решатся следующие задачи:

* проанализировать предметную область;
* составить необходимые схемы, графики для понятия логики и структуры работы разработки;
* поставить цели и задачи;
* проанализировать аналоги сервисов в зоне интернета;
* произвести обзор инструментальных средств, которые будут использованы при разработке;
* разработать сервис «DriveYou»;
* произвести тесты на выявление неисправностей и багов.

Практическая значимость исследовательской работы заключается в том, что результаты исследования могут быть использованы в реализации аналогичных приложений на различных платформах.

## ГЛАВА 1. ПОСТАНОВКА ЗАДАЧИ

## 1.1 Описание предметной области

Интерес к исследованию данной работы представляющую часть реального мира и называется предметной областью.

Интерес к разработке данного сервиса вызван поглощением сервисом «BlaBlaCar», другого сервиса «BeeCar» и интересом к разработке мобильных приложений и веб-сервисов на платформе .NET. Поскольку на данный момент эта платформа пользуется довольно большой популярностью среди разработчиков, благодаря активной поддержке со стороны сообщества и компании Microsoft.

Основным назначением подобных сервисов является упрощение и улучшение жизни человека. Мой сервис представляет с собой, интуитивно понятное и простое в использовании мобильное приложение. Где водитель может создать свою поездку, а попутчики найти авто, идущее до нужного им пункта.

Проанализировав предметную область, можно сделать вывод, что подобные сервисы довольно актуальны, хотя и существуют их аналоги, имеющие большую популярность.

Целью разработки этого сервиса является возможность стать хорошим разработчиком, заинтересовать людей, и в будущем монетизировать приложение в случае успеха проекта.

## 1.2 Модель предметной области

Сервис «DriveYou» включает в себя следующий функционал:

* + Web-API:
    - Работа с БД:
      * Добавление
      * Удаление
      * Изменение
      * Обновление данных в БД
    - Авторизация
    - Аутентификация
* Мобильное приложение:
  + Авторизация;
    - Регистрация;
    - Поиск поездки;
    - Добавление поездки.
    - Просмотр информации о поездках
    - Просмотр и редактирование профиля
    - Просмотр профилей пользователей

Модель данных, взаимодействие пользователя на логическом уровне к данным определения объектов. С помощью логической модели можно проследить структуру приложения, то, как пользователь будет взаимодействовать с ним. Понять приложение как единое целое.

Логическая функциональная модель графически опишет весь перечисленный функционал приложения, покажет состояние всех объектов сущностей предметной области и взаимосвязей между ними, который будет воспроизведен в мобильном приложении «DriveYou».

Исходя из полного определенного функционала программы, была разработана логическая модель мобильного приложения «DriveYou»   
(рисунок 1.1.).

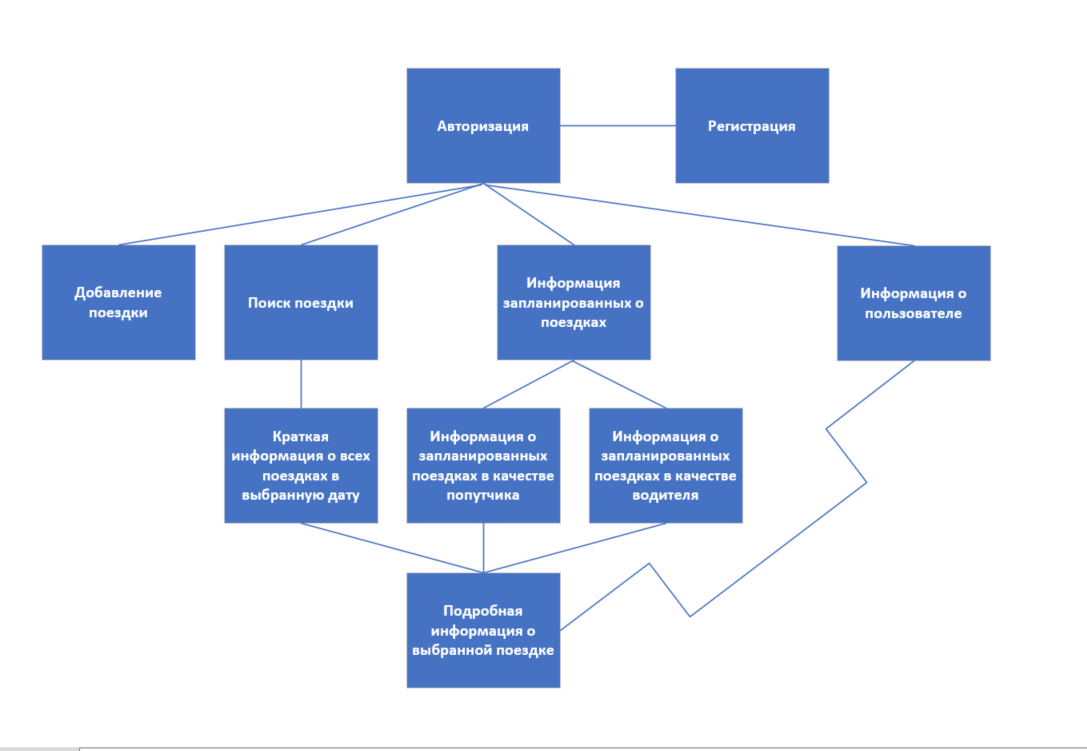


Рисунок 1.1 –Логическая модель приложения

## 1.3 Функциональная модель игры

На диаграмме прецедентов (рисунок 1.4) показаны отношения между пользователем и приложением, что позволяет описать приложение на концептуальном уровне, показать его общую структуру.



Рисунок 1.4 – Диаграмма прецедентов приложения категории «Пользователь»

В этом разделе была приведена и описана диаграмма прецедентов мобильного приложения «DriveYou». Теперь можно явно увидеть все отношения между пользователем и функционалом приложения.

## ГЛАВА 2. ОБЗОР СРЕДСТВ РАЗРАБОТКИ CЕРВИСА

## 2.1 Задачи, решаемые при разработке сервиса «DriveYou»

При подготовке к разработке любого проекта необходимо построить план его реализации, определить задачи, которые необходимо решить. Безусловно, для реализации больших проектов используется команда разработчиков, дизайнеров, проектировщиков, тестировщиков, но эта работа будет производиться в учебных целях и поэтому можно поставить для себя следующие задачи:

1. Определить, какая будет структура мобильного приложения и Web-API.
2. Создать прототип приложения на «бумаге», построить логические схемы и определить интересность проекта.
3. Построить диаграмму БД.
4. Выбрать наиболее удобную среду и язык для разработки приложения.
5. Проработать дизайн приложения.
6. Подготовить визуальное оформление (иконки, картинки, векторная графика).
7. Разработать приложение и Web-API
8. Создать необходимое количество окон(страниц) и объединить их между собой.
9. Протестировать приложение для выявления ошибок и их исправления.

Выполнив поставленные задачи, можно получить мобильное приложение готовую к использованию в повседневной жизни.

## 2.2 Средства разработки и фреймворки

Прежде чем начинать разрабатывать сервис «DriveYou», рассмотрим наиболее подходящие средства. Мой выбор пал на разработку мобильного приложения с помощью .NET Standard 2.0 и фреймворка Xamarin. Для разработки Web-API был использован .NET 5.0. Средства такого рода значительно упрощают разработку.

Есть узкоспециализированные средства для разработки мобильных приложений, такие как Android Studio (Java) и XCode (Swift). Но также есть и средства для кроссплатформенной разработки, такие как Xamarin и .NET MAUI.

Итак, перейдем к анализу фреймворков для разработки мобильного приложения. Для этого зайдем на популярную «Свободную энциклопедию ВикипедиЯ», в поиск введем «Mobile development framework». Википедия нам предлагает ряд фреймворков. Для себя я определил двух лидеров: React Native и Xamarin сравнение которых будет приведено в таблице 1.1.

Таблица 1.1 – Сравнение различных мобильных фреймворков

|  |  |  |  |
| --- | --- | --- | --- |
| Название | Описание | Язык | Стоимость |
| React Native | React Native прост и масштабируем, предоставляет высокую скорость разработки, так же имеет возможность использовать пользовательские библиотеки для разработки пользовательских интерфейсов, поддерживает интеграцию в существующие приложения, а так-же позволяет использовать хуки без написания классов. Построение пользовательских хуков позволяет помещать логику компонента в повторно используемые функции. | JS с поддержкой синтаксиса ES6 | бесплатный |
| Xamarin | Xamarin имеет общую кодовую базу, так называемый «Shared code» в котором разработчик пишет один код для всех платформ. Таким образом благодаря Xamarin в среднем 90% кода приложения может использоваться без изменений на разных платформах. С помощью этого шаблона разработчик может написать всю бизнес-логику на одном языке, но при этом получить характеристики производительности, оформление и поведение, характерные для каждой соответствующей платформы. | C# | бесплатный |

Необходимо долго выбирать и тестировать, чтобы найти для себя лучший. Но сформулировав все свои потребности в разработке, можно легко выбрать из описания, опираясь на предоставленный бюджет.

Что касательно средств для разработки Web-API для проекта, была выбрана платформа .NET 5.0. Выбор на нее пал поскольку это бесплатная кроссплатформенная платформа с большим сообществом разработчиков и поддержкой компании Microsoft. Она позволяет быстро и легко развернуть приложение под управлением ОС Windows/Windows Server/Linux.

## 2.3 Выбор и средств разработки

Платформ и фреймворков в современном мире огромное множество.

Не зная языков программирования создать приложения не сильно сложно, достаточно установить бесплатную IDE, найти готовые уроки, выделить немного времени и в очень короткие сроки создать свой первый проект. Так же опытные программисты могут использовать эти средства для написания своих независимых модулей и библиотек, которые подключаются к основной программе.

Для разработки мобильного приложения «DriveYou» будет использоваться мощный фреймворк Xamarin вместе с Visual Studio 2022.

Почему не использовать Android Studio? Android Studio используется для нативной разработки приложений исключительно для устройств под управлением Android. IDE со всеми своими SDK и эмуляторами занимает довольно много места, так-же требует много аппаратных ресурсов. С экономической точки зрения, если развивать сервис дальше и выпустить приложение не только под Android, но и под iOS и Mac, то выгоднее будет нанять 1 специалиста, который разрабатывает под Xamarin, чем искать несколько специалистов для каждой платформы. Ведь в данном случае программист пишет 1 код под все платформы.

Для разработки Web-API будет использоваться платформа .NET 5.0, поскольку она кроссплатформенная, проста в понимании и позволяет быстро развернуть приложение на разных ОС.

Xamarin имеет очень много функций:

* Hot Reload - Горячая перезагрузка позволяет вносить изменения в исходный код приложения во время его выполнения без необходимости приостанавливать его вручную или создавать точку останова. Другими словами, разработчику не обязательно компилировать приложение каждый раз когда он вносит в него изменения, что очень упрощает и ускоряет разработку;
* Создание CustomControl с использованием XAML и C# - кастомные контролы позволяют создавать уникальные элементы управления необходимые для работы с приложением. Плюсом создания таких контролов является то, что визуальная чать (дизайн) создается на языке XAML, который прост и интуитивно понятен, а также основан на XML, а функционал контрола пишется на C#
* Единая кодовая база, позволяющая программисту не переписывать код для каждой отдельной платформы, а написать90% процентов логики и визуальной части приложения которые будут прекрасно работать на каждой платформе.
* Xamarin Essentials – предоставляет для разработчиков кроссплатформенные API-интерфейсы для мобильных приложений.

## ГЛАВА 3. ПРОЦЕСС СОЗДАНИЯ СЕРВИСА

## 3.1 Установка и описание средств для разработки

Unity легкодоступная программа, которую без проблем можно скачать в интернете. Она частично бесплатная и имеет много обучающих материалов.

Для разработки нам потребуется IDE Visual Studio 2022, установленными средствами для разработки мобильных и веб-приложений. Эта среда бесплатная, а также имеет отличную документацию и кучу примеров с уроками по разработке.

Скачиваем инсталлятор с официального сайта. Размер файла примерно 1.6 МБ. После загрузки запускаем файл, появится мастер установки Visual Studio, с помощью которого можно загрузить и установить все необходимые пакеты для разработки. (рисунок 3.1).

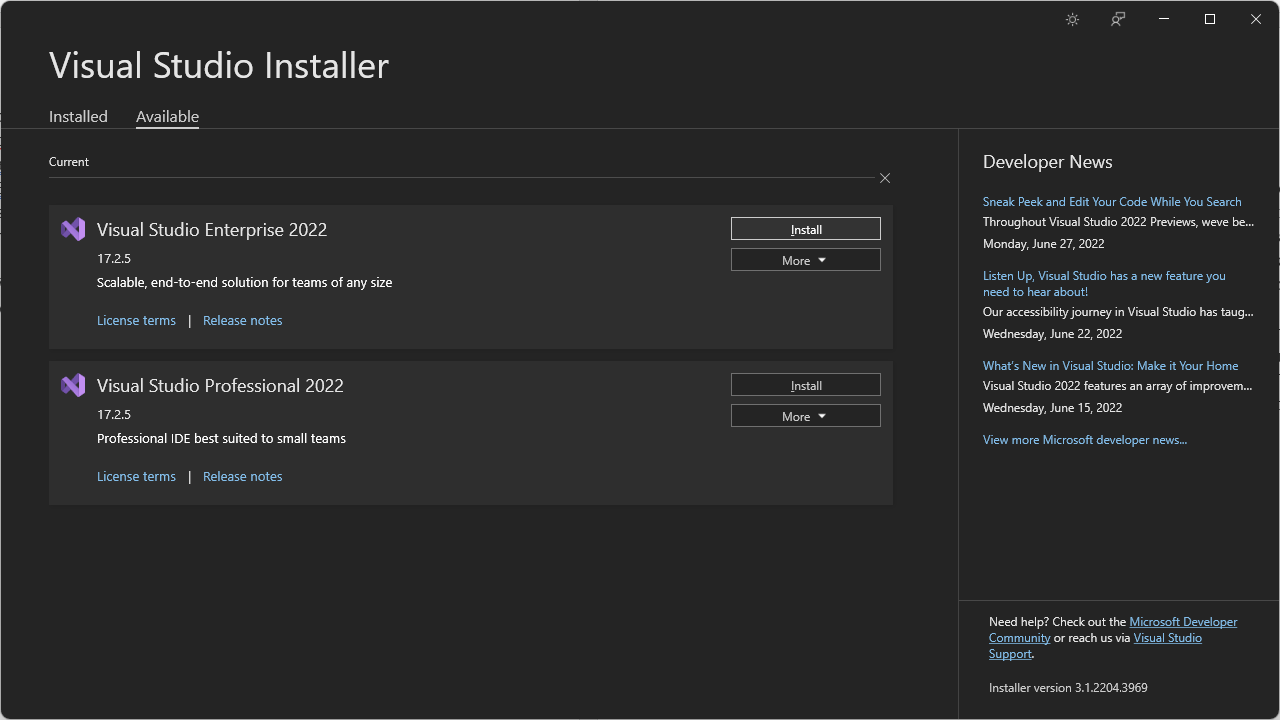


Рисунок 3.1 – Мастер установки приложений

Соглашаемся с «Лицензионным соглашением» и нажимаем кнопку «Install». Далее выбираем необходимые компоненты. Нас интересуют компоненты «ASP.NET and web development» и «Mobile development with .NET» (рисунок 3.2),

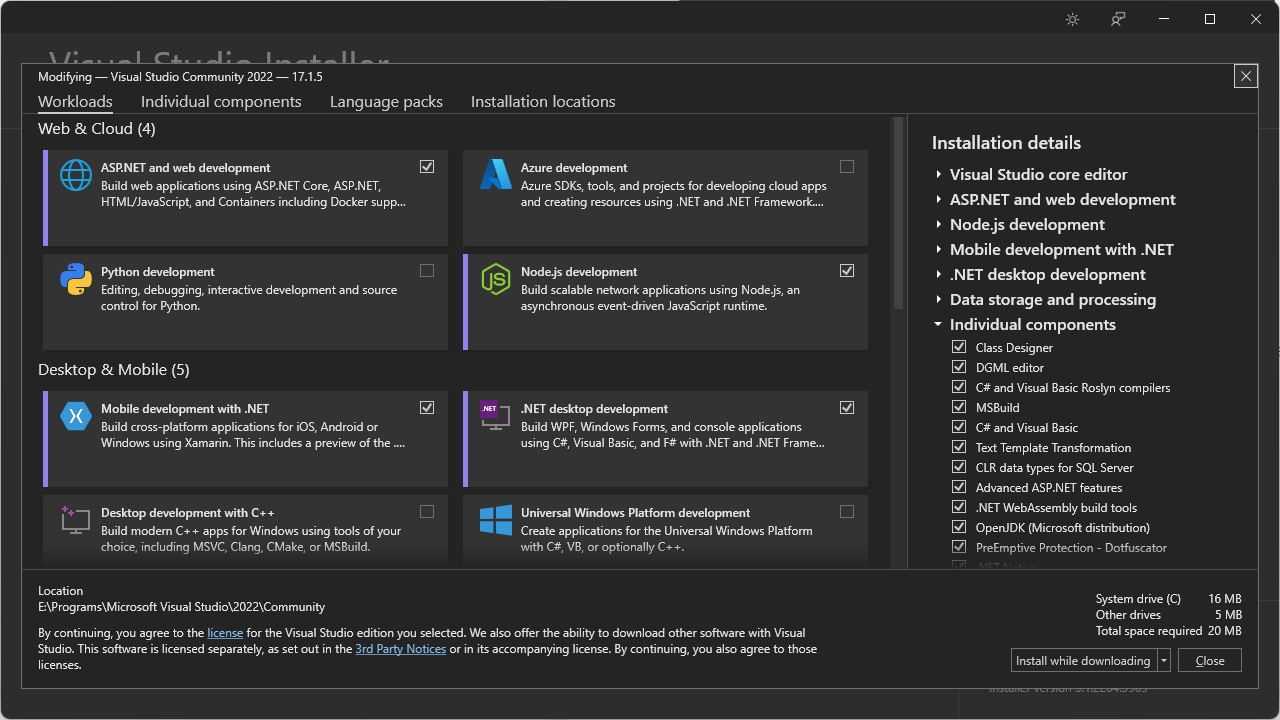


Рисунок 3.2 – Выбор компонентов

Далее нажимаем кнопку «Install», и мастер установки загрузит и установит необходимые нам компоненты. После завершения будет предложено перезагрузить компьютер. Соглашаемся и перезагружаем ПК.

После того, как программа была установлена, на рабочем столе появится ярлык программы (рисунок 3.3).

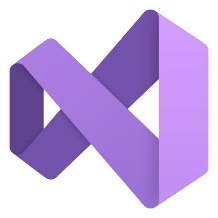


Рисунок 3.3 – Ярлык Visual Studio

Запускаем IDE с помощью двойного щелчка левой кнопки мыши по нему. Откроется главное окно программы, где можно выбрать существующий проект или создать новый (рисунок 3.4).

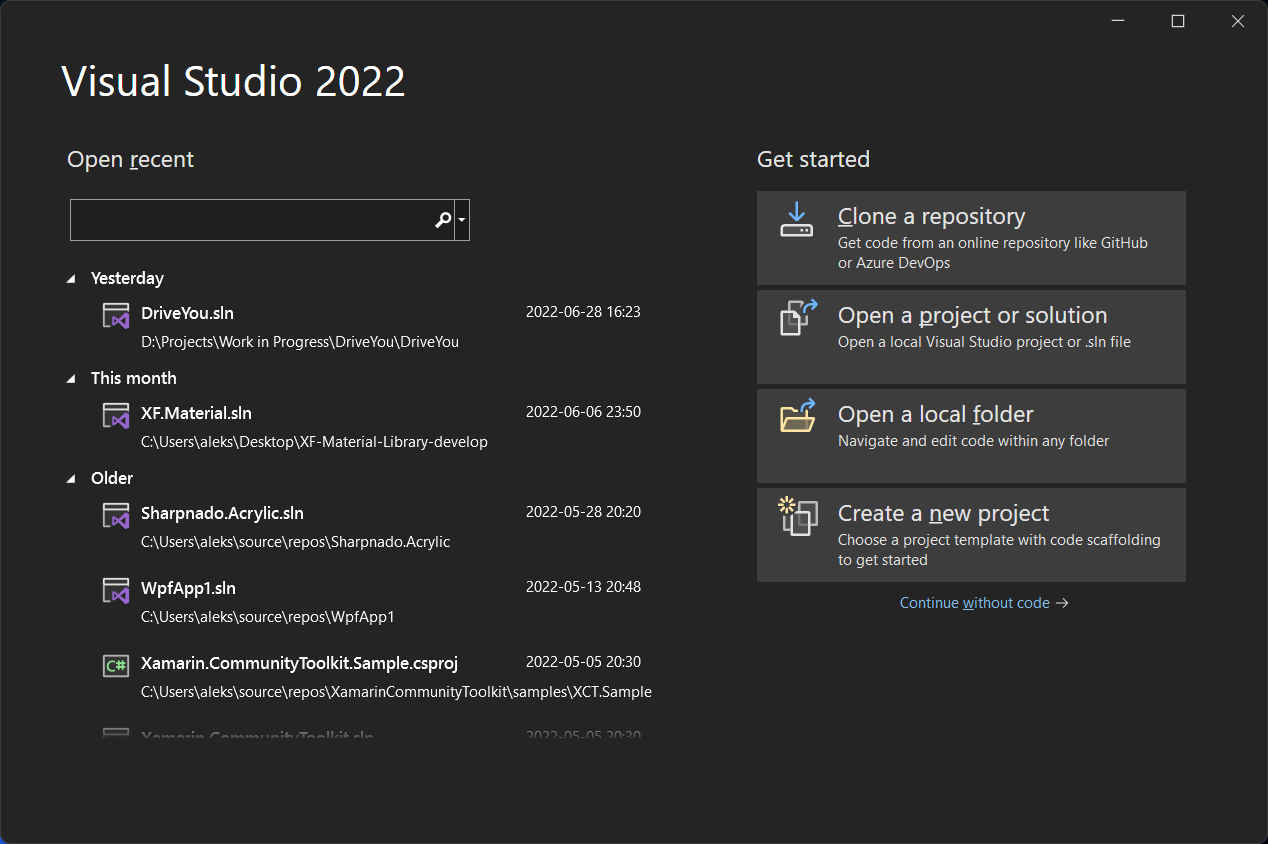


Рисунок 3.4 – Главное окно Visual Studio

Создадим новый проект для обзора программы. В правой части окна выбираем «Create a new project» Открывается окно, в котором необходимо выбрать тип проекта. (рисунок 3.5)

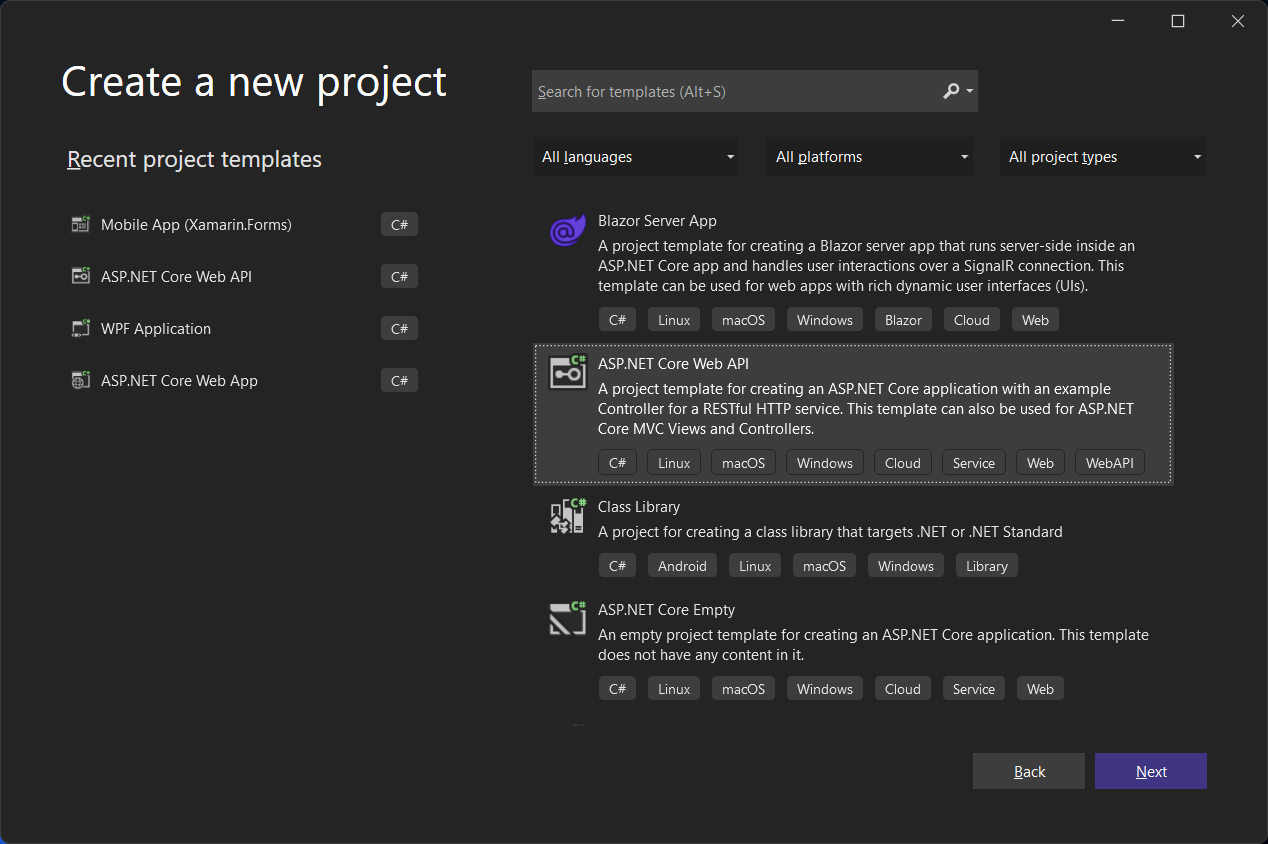


Рисунок 3.5 – Выбор типа проекта

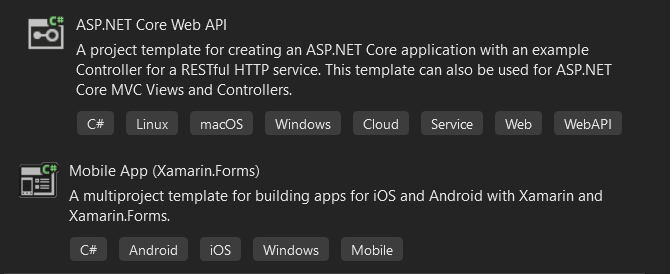
Нас интересуют только 2 проекта, Web-API и Mobile App (рисунок 3.6)

Рисунок 3.6 – Типы проектов

Слева находится окно «Solution Explorer», которое необходимо для навигации по проекту. (рисунок 3.7)

Снизу окно «Output» и «Error List», туда выводится информация о процессе компиляции, приложения и ошибках в коде. (рисунок 3.7)

По центру находится редактор кода. (рисунок 3.7)

Вверху расположен «Toolbar», который включает в себя базовые функции (запуск проекта, приостановка выполнения, переключение между проектами). (рисунок 3.7)

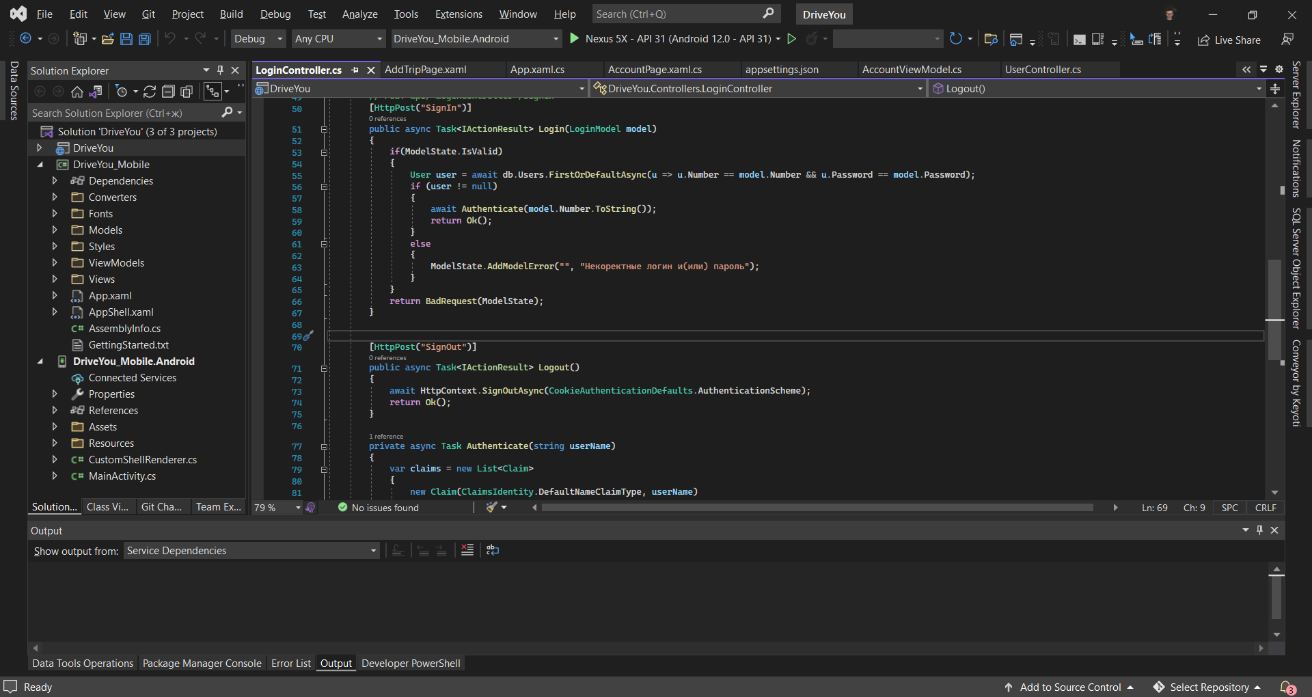
Используя различные объекты, классы и NuGet пакеты создать потрясающий функционал приложения, создать все с нуля, придумать свою идею, реализовать похожие проекты. Функционал IDE очень разнообразен и позволяет реализовать большинство задумок. (рисунок 3.7)

Рисунок 3.7 – Окно IDE.

## 3.2 Разработка сервиса

## 3.2.1 Разработка Web-API

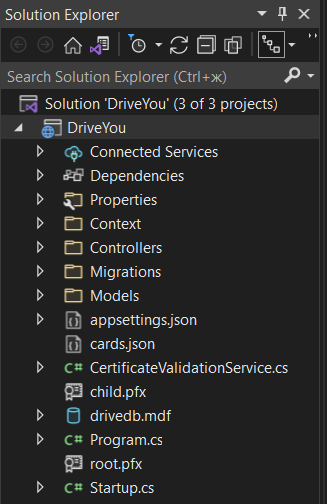
Разработка приложений с помощью Visual Studio очень увлекательный процесс, где можно сразу видеть результат. На рисунке 3.8 представлена структура Web-API

Рисунок 3.8 – Структура проекта Web-API.

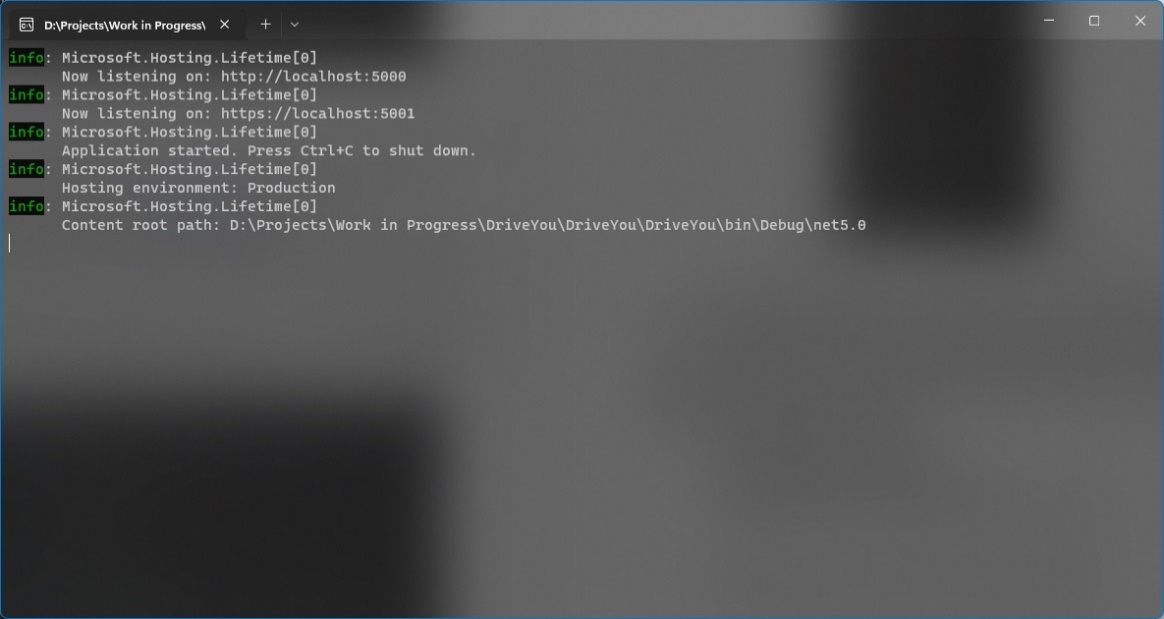
Web-API представляет из себя консольное приложение которое можно запустить на Windows или опубликовать на сервере под управлением IIS. (рисунок 3.9)

Рисунок 3.8 – Вид консольного приложения Web-API.

Это приложение является серверным и контролирует процесс работы с данными. Мобильное приложение посредством HTTP запросов к API, получает данные из базы данных. Которые сервер конвертирует в формат JSON и отправляет клиенту. Web-API имеет функционал для:

1. Авторизации пользователя
2. Регистрации пользователя
3. Получение данных пользователем
4. Добавление и изменение данных пользователем

За авторизацию пользователя отвечает контроллер «LoginController»

За регистрацию пользователя отвечает контроллер «RegisterController»

За работу с данными отвечают контроллеры «TripsController» и «UserController». Они позволяют получить, добавить или изменить информацию о пользователе, или поездке.

## 3.2.2 Разработка мобильного приложения

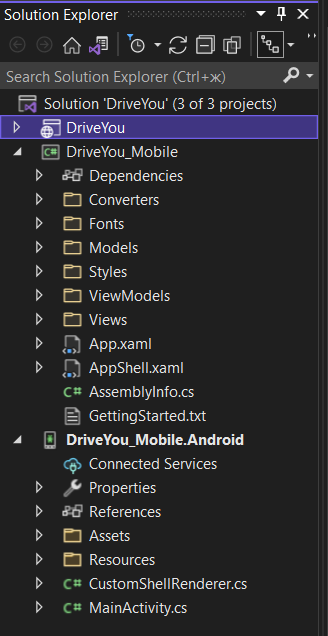
 На Рисунке 3.10 показана структура мобильного приложения.

Рисунок 3.10 – Структура мобильного приложения.

Как можно видеть приложение создано с использованием паттерна MVVM (Model-View-ViewModel). Этот паттерн помогает облегчить разработку больших проектов.

Главное окно программы – это страница аутентификации пользователя.

Если пользователь не зарегистрирован, то он нажимает на кнопку «Register» и переходит на страницу регистрации. Если же у пользователя есть аккаунт, то он вводит номер телефона и пароль, после чего жмет кнопку «Login» и если пароль верен приложение перенаправляет его на домашнюю страницу.

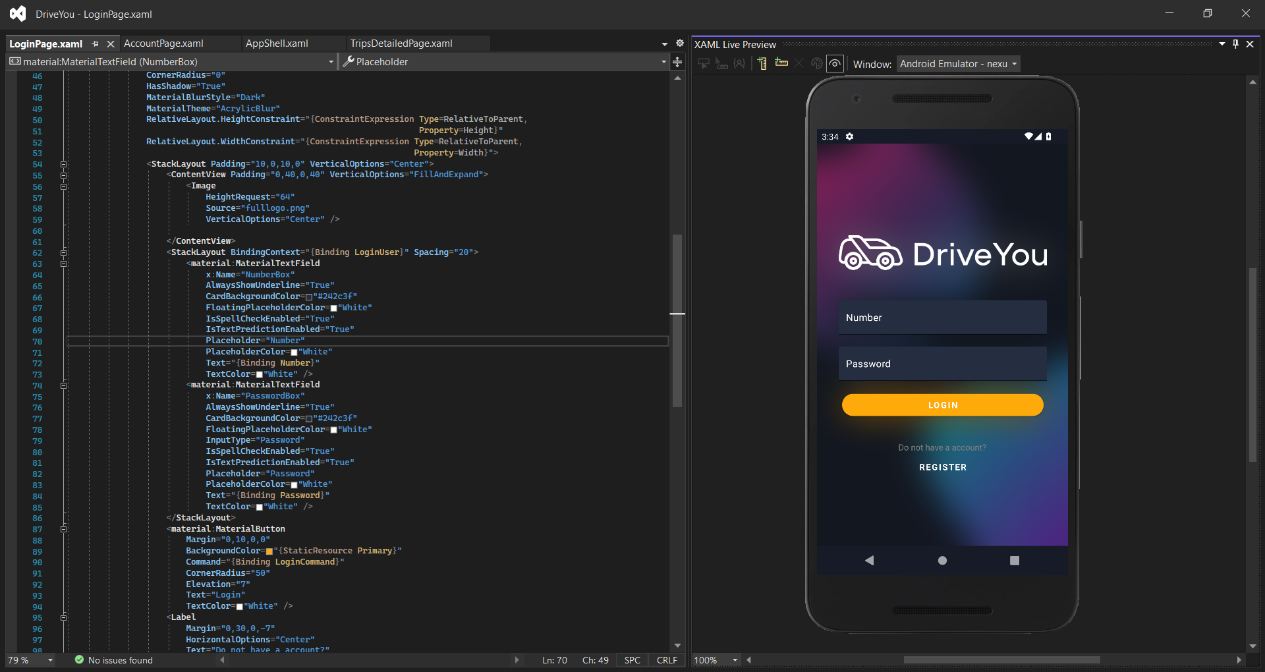
На рисунке 3.11 представлена визуальная форма страницы входа.

Рисунок 3.11 – страница входа

Для страницы входа используется XAML код – «LoginPage» который отвечает за визуальное представление страницы, С# код «LoginModel», который отвечает за определение модели данных для этой страницы и «LoginViewModel» который реализует логику работы с данными.

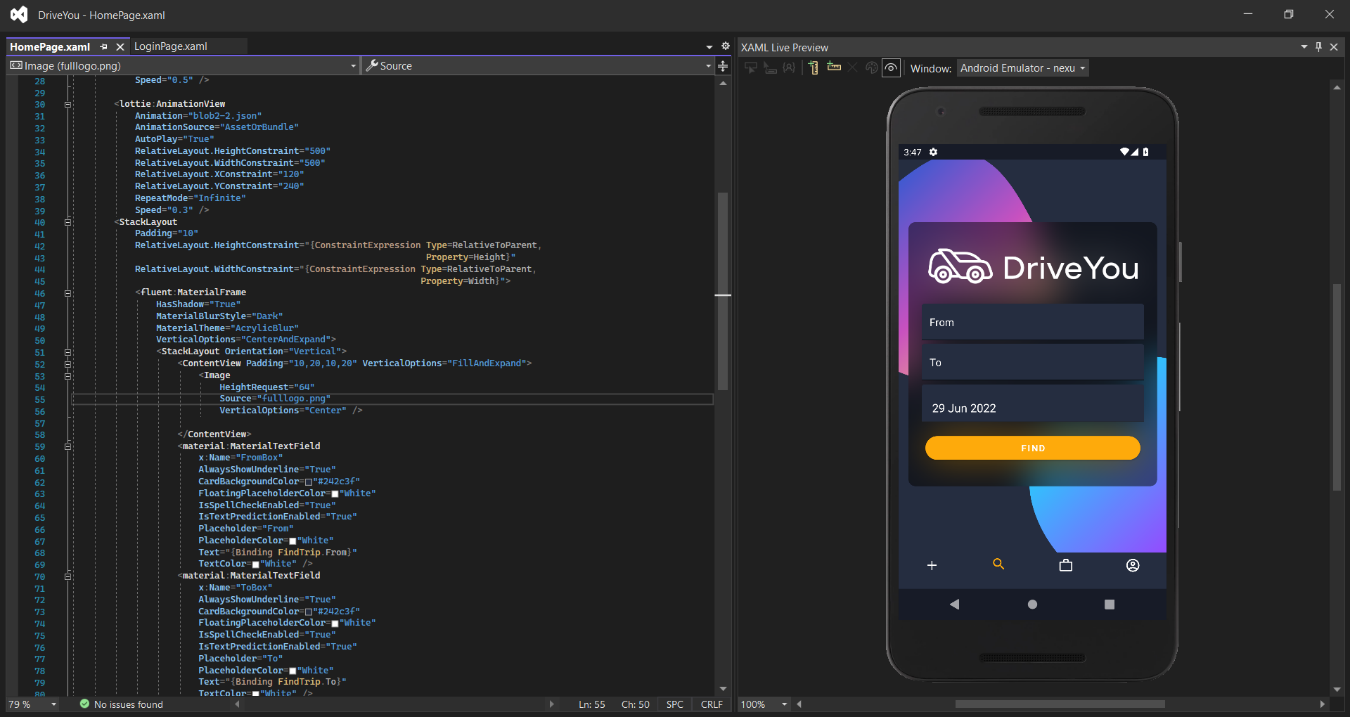
Домашней страницей считается страница поиска поездки. Пользователь вводит город, из которого он едет, город в который хочет попасть и дату поездки. После нажимает на кнопку «Find» и его перенаправляет на страницу с найденными поездками, запланированными на этот день. (рисунок 3.12)

Рисунок 3.12 – Страница поиска

Для приложения был написан следующий код:

* LoginPage – страница авторизации пользователя.
  + LoginModel – модель данных для авторизации.
  + LoginViewModel – логика работы с данными для авторизации пользователя.
* RegisterPage – страница регистрации пользователя.
  + RegisterModel – модель данных для регистрации.
  + RegisterViewModel – логика работы с данными для регистрации пользователя.
* HomePage – домашняя(поиска) страница.
  + FindTripModel – модель данных для поиска поездок.
  + HomeViewModel – логика работы с данными для поиска поездок.
* HomeTrips – страница с информацией искомых поездках.
  + ScheduledTripsWithUserModel – модель данных для запланированных поездок.
  + HomeTripsViewModel – логика работы с данными для запланированных поездок.
* TripsDetailedPage – страница с подробной информацией о поездке.
  + TripsDetailedPageViewModel – логика работы с детальными данными запланированных поездок.
* AddTripsPage – страница добавления новой поездки.
  + ScheduledTripsModel – модель данных для запланированных поездок.
  + AddTripViewModel – логика работы с данными для новой поездки.
* AccountPage – страница с информацией о пользователе.
  + UserModel – модель данных с информацией о пользователе.
  + AccountViewModel – логика работы с данными пользователя.
* EditCarPage – модальное окно для добавления или изменения данных о автомобиле пользователя.
  + EditCarPageViewModel – логика работы с данными для автомобиля пользователя.
* MyTripsPage – страница с вкладками (TabbedPage) содержащая другие страницы.
* MyTripsDriverPage – страница с информацией о запланированных поездках в качестве водителя.
  + MyTripsDriverViewModel – логика работы с данными у запланированных поездок в качестве водителя.
* MyTripsCompanionPage – страница с информацией о запланированных поездках в качестве попутчика.
  + MyTripsCompanionViewModel – логика работы с данными у запланированных поездок в качестве водителя.

Пользователь может как находить и «подписываться» на поездки, так и добавлять их самостоятельно, предварительно добавив информацию о своем автомобиле.

## ГЛАВА 4. ОПИСАНИЕ СЕРВИСА

## 4.1 Программно-аппаратные ресурсы

В процессе работы использовались следующие технические средства:

* ноутбук Lenovo Legion 5;

Параметры ноутбука Lenovo Legion 5:

* + - процессор Intel® Core™ i7-10750H (2,60 ГГц);
    - оперативная память 16Gb;
    - графический адаптер NVIDIA GeForce GTX 1650 Ti;
    - видеоадаптер Intel(R) UHD Graphics;
    - локальная видеопамять 4Gb;
    - жесткий диск WDC PC SN730 SDBPNTY-256G-1101;
    - монитор ноутбука - 15,6 дюймов;
    - дополнительный монитор – 27 дюймов;
    - Windows 11 Pro (64 Бит).

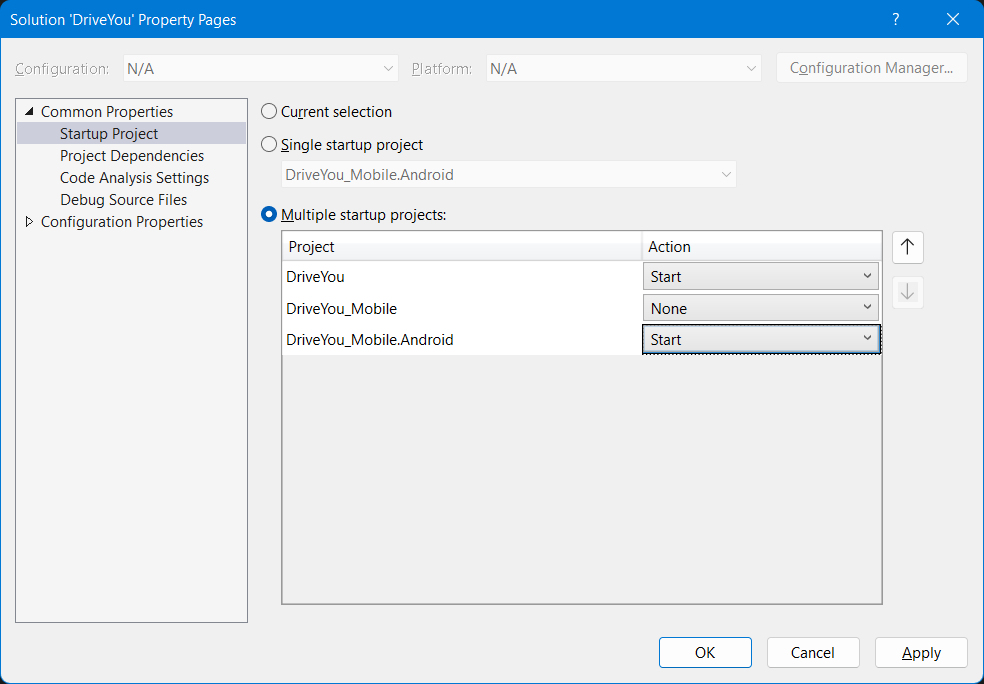
Программные средства:

* + - Visual Studio Community 2022.
    - Adobe Illustrator
    - Android Emulator

В этой главе были указаны параметры всех используемых устройств для разработки сервиса и перечислены все программные средства, которые понадобились в ходе работы.

## 4.2 Демонстративный пример работы сервиса

Для работы сервиса «DriveYou», необходимо собрать и запустить проект Web-API и мобильное приложение. Нажать правой кнопкой мыши на решение, выбрать пункт «Set Startup Projects…», выбрать пункт Multiple startup projects и выбрать проекты для запуска. (рисунок 4.1).

Рисунок 4.1 – Сборка проекта

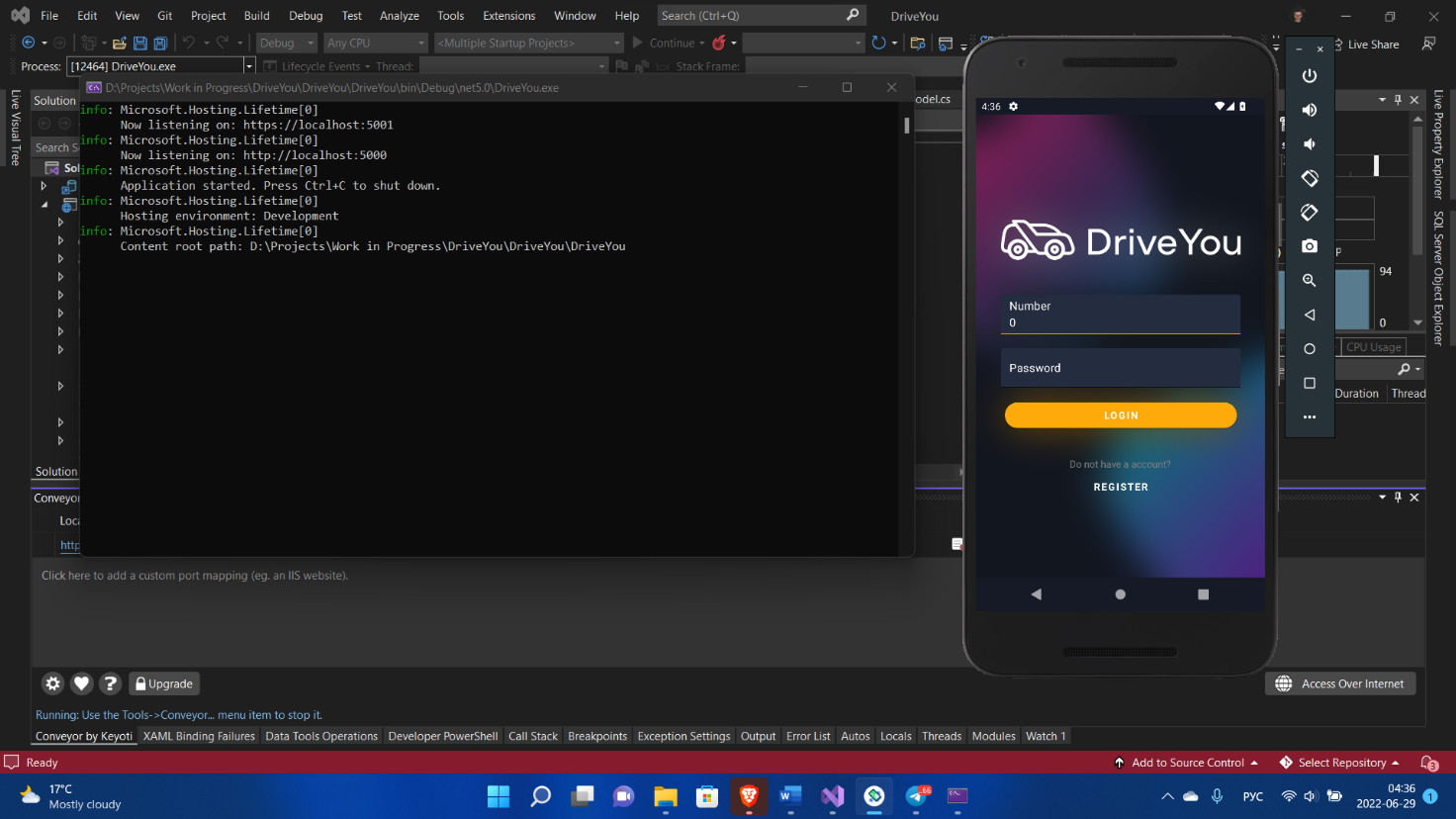
После компиляции проекта, запустится консольное приложение (Web-API) и мобильное приложение (рисунок 4.2).

Рисунок 4.2 – Главное окно мобильного приложения и запущенный Web-API

В главном окне имеется 2 кликабельных кнопки: «Login» и «Register», а так же 2 поля для ввода номера телефона и пароля.

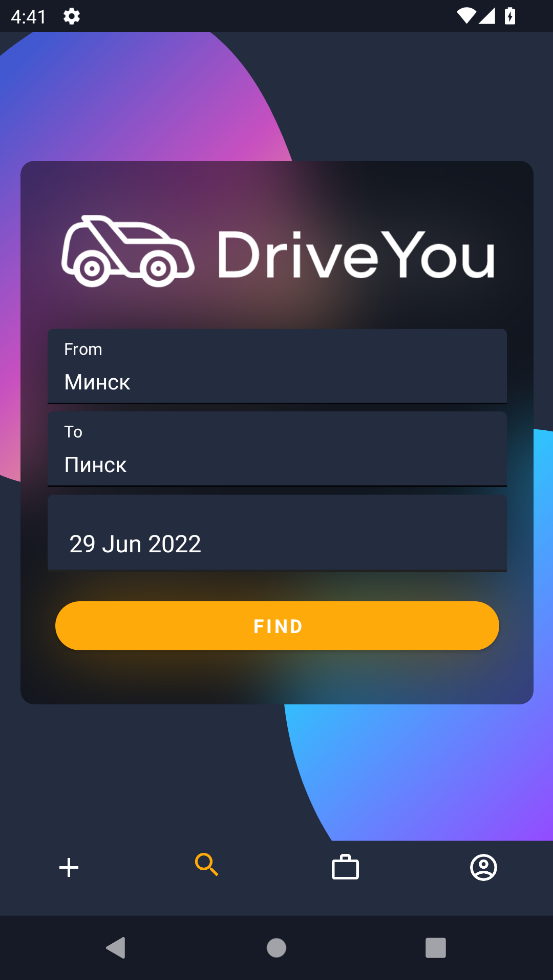
Дальше переходим на страницу поиска и вводим данные для поиска поездки и нажимаем кнопку найти (рисунок 4.3)

Рисунок 4.3 – Страница поиска

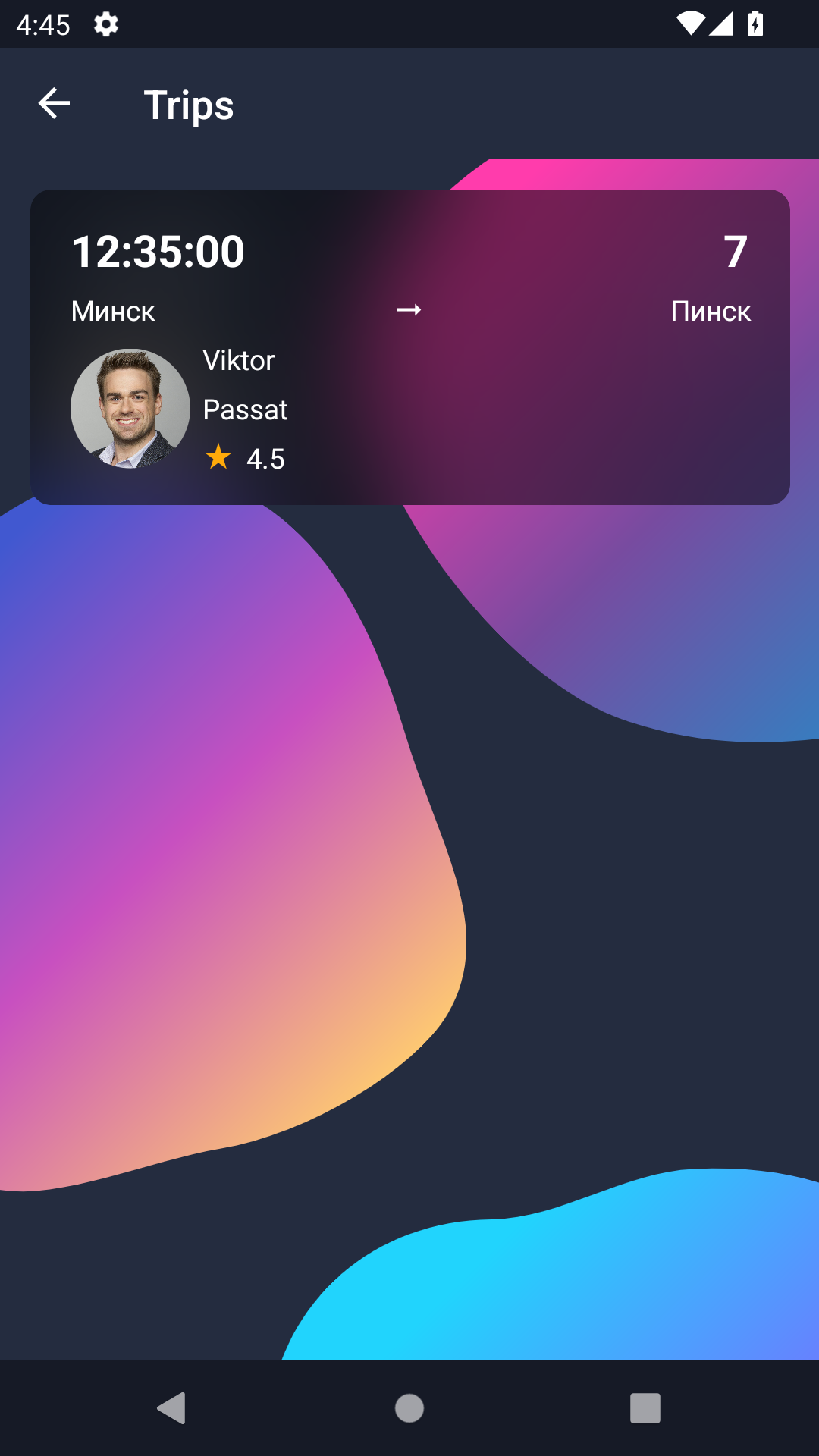
Если такие поездки запланированны то пользователю выведется информация с краткой информацией об этих поездках. (рисунок 4.4)

Рисунок 4.4 – Найденные поездки

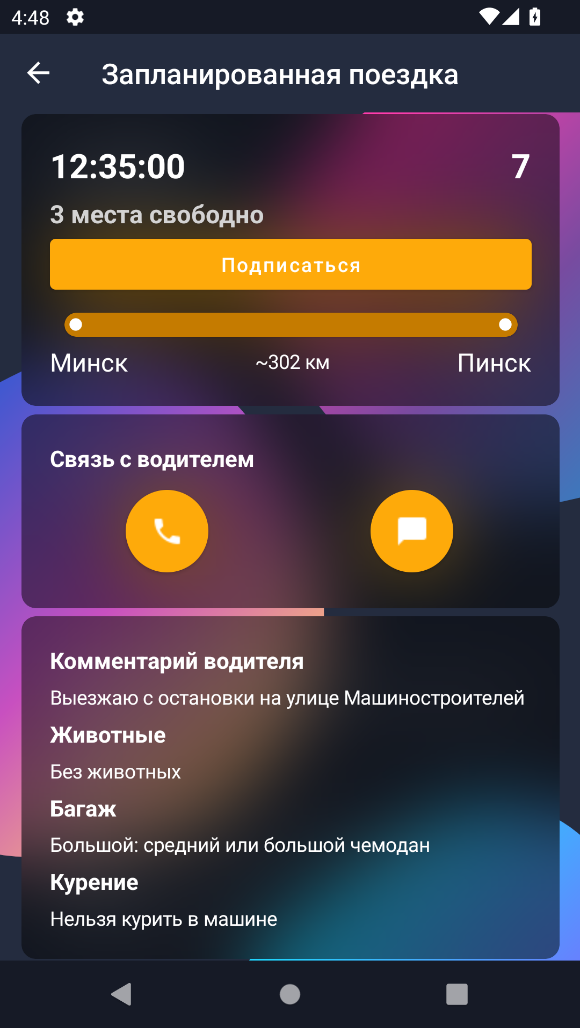
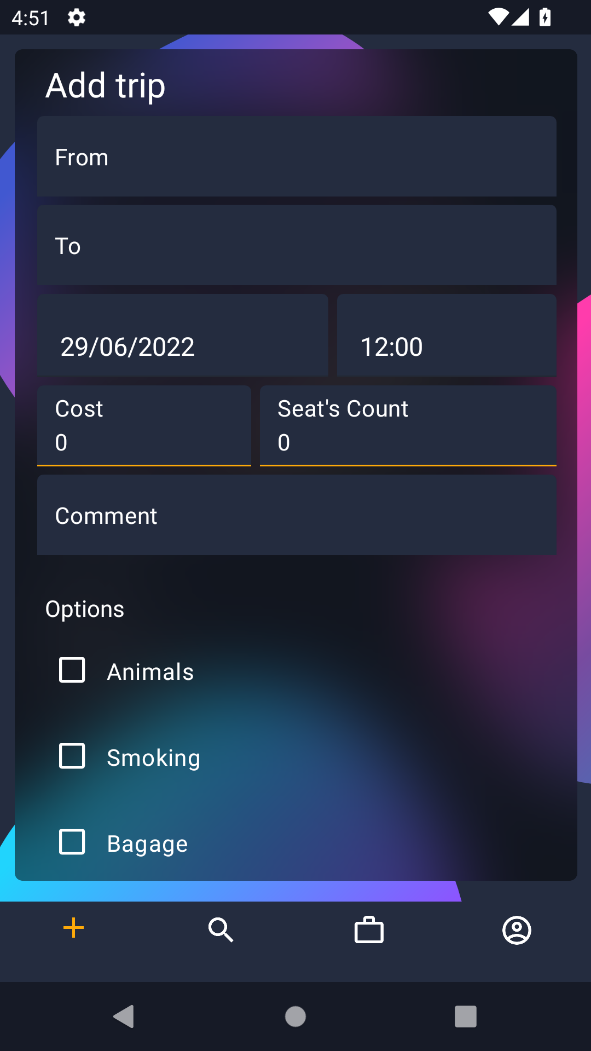
Если нажать на эту «карточку» откроется детальная ифнормация об поездке. (рисунок 4.5)

Рисунок 4.5 – Детальная информация о поездке

Вернемся назад и перейдем на страницу добавления поездки. Здесь пользователь должен ввести данные о поездке которую он запланировал. Откуда и куда, дату и время отправления, стоимость, кол-во мест, комментарий и опции: можно ли с животными, можно ли курить, большой ли багаж. (рисунок 4.6)

Рисунок 4.6 – Страница добавления поездки

Так же есть страница с запланированными поездками пользователя в качестве водителя (рисунок 4.7) и с поездками в качестве попутчика (рисунок 4.8). Так же как и на странице найденных поездок, если нажать на на «карточку» поездки выведется детальная информация о ней.

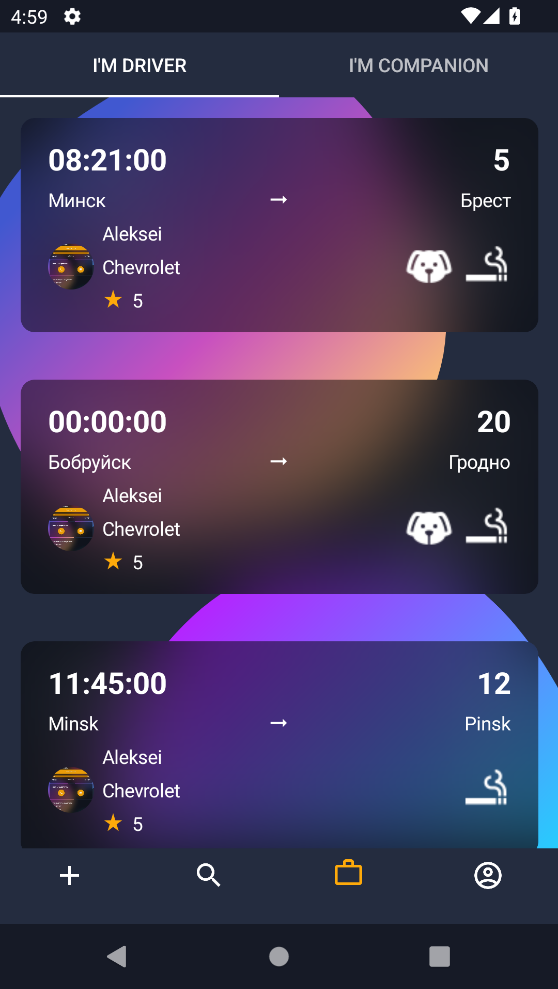
Рисунок 4.7 – Страница запланированных поездок в качестве водителя

Рисунок 4.8 – Страница запланированных поездок в качестве попутчика

Следующая страница, это страница с информацией о пользователе которая содержит в себе всю краткую информацию: Имя и фамилия, рейтинг, кол-во поездок, кол-во отзывов, сами отзывы и информацию об автомобиле.

Пользователь может редактировать свое фото, путем загрузки нового из галереи, а так же редактировать информацию об автомобиле. (рисунок 4.9 и рисунок 4.10)

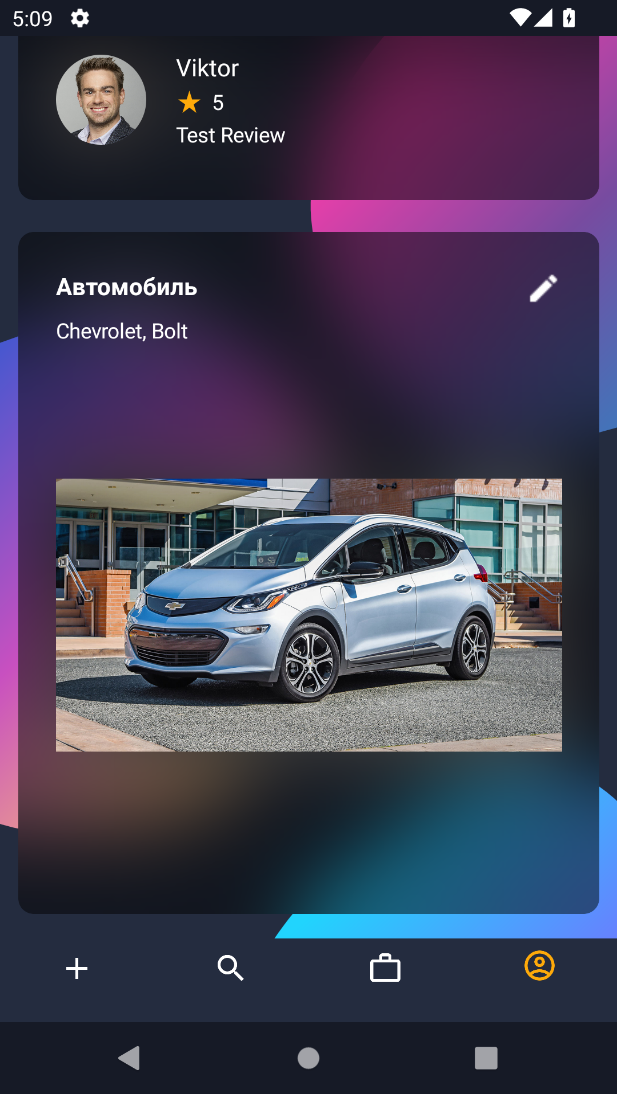
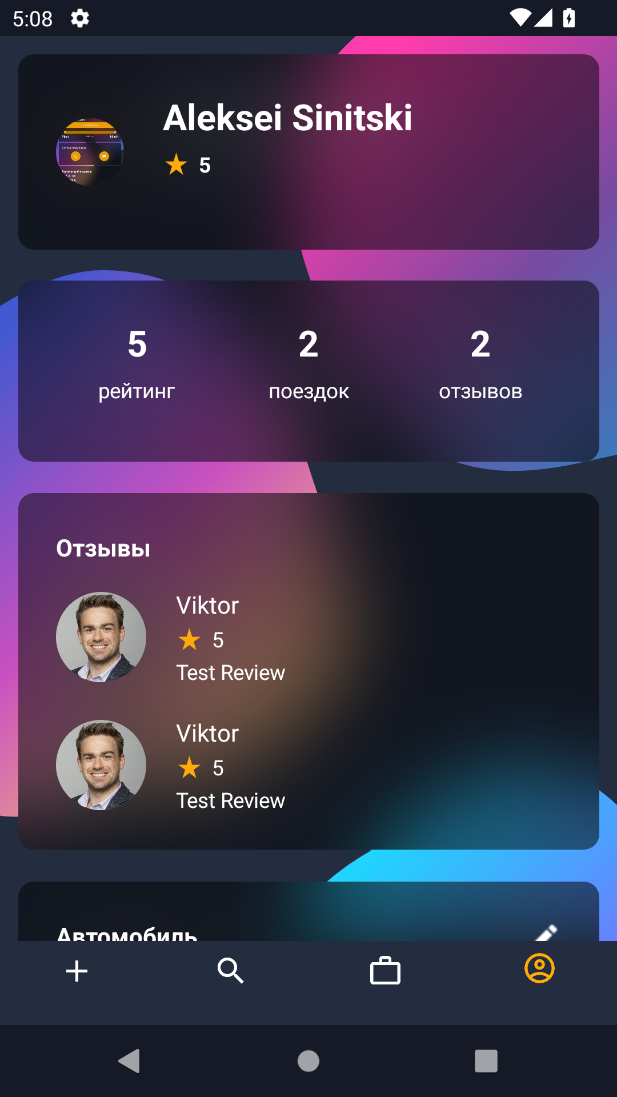


Рисунок 4.9 и 4.10 – Страница с информацией о пользователе

В этом разделе была приведена полная демонстрация работы сервиса «DriveYou».

## 4.3 Тестирование

По окончанию разработки необходимо проверить сервис на полное соответствие всем требованиям, выявить самые уязвимые места и устранить их.

Существует достаточно много видов тестирования приложений. Для тестирования сервиса «DriveYou» использовался в основном вид – «Нагрузочное тестирование». Основной целью такого вида теста является выявление всех несоответствий с необходимым функционалом. Для этого пользователь, либо тестировщик должен пройтись по всем страницам приложения, использовать все вызовы API и загрузить разное кол-во данных для проверки правильности их отображения и нагрузки на систему.

Функциональное тестирование проводилось на всех этапах разработки компьютерной игры, так как её можно разделить на некоторые модули. Когда был обнаружен неверный функционал, то он сразу устранялся.

Существует позитивное и негативное функциональное тестирование. К позитивному можно отнести проверку соответствуя определенным стандартам использования приложения, где пользователь выполняет свои правильные действия и вводит верные данные. А негативное, когда пользователь пытается сделать то, чего обычно не делают, но ему захотелось пойти против правил.

Тестирование мобильного приложения происходило на ноутбуке в эмуляторе, а также на устройстве Xiaomi Redmi Note 9 Pro. Тестирование Web-API проходило на ноутбуке в основной системе, а также на виртуальной машине под управлением Windows Server.

Как пример, на одном из этапов тестирования была обнаружена ошибка загрузки запланированных поездок, не связанных с текущим пользователем.

## Заключение

В ходе выполнения курсовой работы были изучены основные принципы разработки сервиса средствами Visual Studio, Xamarin и .NET. Был приобретен бесценный опыт в разработке мобильных и веб-приложений.

В ходе исследования были решены следующие задачи:

* определена структура проекта, те результаты, которые должны быть по окончанию разработки;
* построено два вида UML-моделей, для визуализации сервиса;
* изучены технологии разработки мобильных и веб-приложений;
* проведен анализ фреймворков и платформ, их сопоставление и выбор наиболее подходящего для разработки мобильного приложения и веб-сервиса;
* рассмотрен фреймворк Xamarin и платформа .NET, где были определены их достоинства для реализации проекта;
* был разработан сервис «DriveYou».

В первой главе курсовой работы рассматривалась теоретическая часть, ставились цели и задачи для получения желаемого результата разработки.

Во второй главе рассматривались различные средства для разработки и был проведен анализ на выбор наиболее подходящего. В итоге для Web-API была выбрана платформа .NET, а для мобильного приложения фреймворк Xamarin.

При написании третьей главы – практической части, была произведена установка и настройка среды разработки а, так же установка всех необходимых библиотек, фреймворков и SDK. Был разработан полноценный программный продукт.

Четвертая глава показала демонстрацию и полное описание всего функционала проекта.

Сервис представляет собой законченный программный продукт и может быть использована в реальной жизни.

## Список использованных источников

1. Документация по Xamarin. <https://docs.microsoft.com/ru-ru/xamarin/>
2. Документация по .NET <https://docs.microsoft.com/ru-RU/dotnet/>
3. Руководство по программированию Xamarin Forms <https://metanit.com/sharp/xamarin/>
4. Интернет-сообщество для разработчиков ПО <https://www.c-sharpcorner.com/>
5. Сайт вопросов и ответов для программистов https://stackoverflow.com/

## Приложения

Приложение A

LoginController.cs

namespace DriveYou.Controllers

{

[Route("api/[controller]")]

[ApiController]

[AllowAnonymous]

[RequireHttps]

public class LoginController : ControllerBase

{

private readonly ILogger<LoginController> \_logger;

private ApplicationDbContext db;

public LoginController(ILogger<LoginController> logger, ApplicationDbContext context)

{

\_logger = logger;

db = context;

}

// GET api/<LoginController>

[HttpGet]

public IActionResult Get()

{

if (User.Identity.Name != null && User.Identity.IsAuthenticated)

{

return Ok();

}

else

{

return Unauthorized();

}

}

// POST api/<LoginController>/signin

[HttpPost("SignIn")]

public async Task<IActionResult> Login(LoginModel model)

{

if(ModelState.IsValid)

{

User user = await db.Users.FirstOrDefaultAsync(u => u.Number == model.Number && u.Password == model.Password);

if (user != null)

{

await Authenticate(model.Number.ToString());

return Ok();

}

else

{

ModelState.AddModelError("", "Некоректные логин и(или) пароль");

}

}

return BadRequest(ModelState);

}

[HttpPost("SignOut")]

public async Task<IActionResult> Logout()

{

await HttpContext.SignOutAsync(CookieAuthenticationDefaults.AuthenticationScheme);

return Ok();

}

private async Task Authenticate(string userName)

{

var claims = new List<Claim>

{

new Claim(ClaimsIdentity.DefaultNameClaimType, userName)

};

ClaimsIdentity identity = new ClaimsIdentity(claims, "ApplicationCookie", ClaimsIdentity.DefaultNameClaimType, ClaimsIdentity.DefaultRoleClaimType);

await HttpContext.SignInAsync(CookieAuthenticationDefaults.AuthenticationScheme, new ClaimsPrincipal(identity));

}

}

}

RegisterController.cs

namespace DriveYou.Controllers

{

[Route("api/[controller]")]

[ApiController]

[AllowAnonymous]

[RequireHttps]

public class RegisterController : ControllerBase

{

private readonly ILogger<RegisterController> \_logger;

private ApplicationDbContext db;

public RegisterController(ILogger<RegisterController> logger, ApplicationDbContext context)

{

\_logger = logger;

db = context;

}

// POST api/<RegisterController>

[HttpPost]

public async Task<IActionResult> Login(RegisterModel model)

{

if (ModelState.IsValid)

{

User user = await db.Users.FirstOrDefaultAsync(u => u.Number == model.Number || u.Email == model.Email);

if (user != null)

{

ModelState.AddModelError("", "Аккаунт с таким номером или email-ом уже существует");

return BadRequest(ModelState);

}

else

{

db.Add(new User()

{

Number = model.Number,

Email = model.Email,

Password = model.Password,

Name = model.Name,

Surname = model.Surname

});

await db.SaveChangesAsync();

return Ok();

}

}

return BadRequest(ModelState);

}

}

}

TripsController.cs

namespace DriveYou.Controllers

{

[Route("api/[controller]")]

[ApiController]

[Authorize(AuthenticationSchemes = CookieAuthenticationDefaults.AuthenticationScheme)]

public class TripsController : ControllerBase

{

private ApplicationDbContext db;

private readonly ILogger<TripsController> \_logger;

public TripsController(ILogger<TripsController> logger, ApplicationDbContext context)

{

\_logger = logger;

db = context;

}

[HttpGet("scheduled/my")]

public async Task<IActionResult> GetMyScheduledTrips()

{

if (ModelState.IsValid)

{

if (User.Identity.IsAuthenticated)

{

var user = await db.Users.Where(u => u.Number == long.Parse(User.Identity.Name))

.Select(i => i.ID)

.FirstOrDefaultAsync();

var tripsId = await db.ScheduledTrips.Where(u=>u.UserID == user).ToListAsync();

List<SubscribedOnTripsModel> subs = new List<SubscribedOnTripsModel>();

for (int i = 0; i < tripsId.Count; i++)

{

var subUser = await db.SubscribedOnTrips.Where(u => u.ScheduledTripsModelID == tripsId[i].ID).FirstOrDefaultAsync();

if (subUser != null)

{

subs.Add(subUser);

}

}

for (int i = 0; i < subs.Count; i++)

{

subs[i].User = await db.Users.Where(u=>u.ID == subs[i].UserID).FirstOrDefaultAsync();

}

var result = await db.ScheduledTrips.Where(u => u.UserID == user).Join(db.Users,

s => s.UserID,

u => u.ID,

(s, u) => new ScheduledTripsWithUserModel()

{

ID = u.ID,

Name = u.Name,

Date = u.Date,

Photo = u.Photo,

Rating = u.Rating,

CarMark = u.CarMark,

CarModel = u.CarModel,

CarImage = u.CarImage,

ScheduledTrips = s,

SubscribedOnTrips = subs

}).ToListAsync();

return new JsonResult(result);

}

}

return BadRequest();

}

[HttpPost("scheduled")]

public async Task<IActionResult> GetScheduledTrips(FindTripModel model)

{

if (ModelState.IsValid)

{

if (User.Identity.IsAuthenticated)

{

var trips = await db.ScheduledTrips.Where(t=>t.From == model.From && t.To == model.To && EF.Functions.Like(t.Date.ToString(), $"%{model.Date.ToShortDateString()}%")).Join(db.Users,

s => s.UserID,

u => u.ID,

(s, u) => new ScheduledTripsWithUserModel()

{

ID = u.ID,

Number = u.Number,

Name = u.Name,

Date = u.Date,

Photo = u.Photo,

Rating = u.Rating,

CarMark = u.CarMark,

CarModel = u.CarModel,

CarImage = u.CarImage,

ScheduledTrips = s

}).ToListAsync();

return new JsonResult(trips);

}

}

return BadRequest();

}

[HttpPost("scheduled/new")]

public async Task<IActionResult> PostNewScheduledTrip(ScheduledTripsModel model)

{

if (ModelState.IsValid)

{

if (User.Identity.IsAuthenticated)

{

var user = await db.Users.Where(u=>u.Number == long.Parse(User.Identity.Name) && u.CarModel != null && u.CarMark != null).FirstOrDefaultAsync();

if (user == null)

{

ModelState.AddModelError("", "Please, add car info first");

return NotFound(ModelState);

}

model.UserID = user.ID;

db.ScheduledTrips.Add(model);

await db.SaveChangesAsync();

return Ok();

}

return Unauthorized();

}

return BadRequest(ModelState);

}

[HttpDelete("scheduled/delete/{id}")]

public async Task<IActionResult> DeleteScheduledTrip(int id)

{

if (ModelState.IsValid)

{

if (User.Identity.IsAuthenticated)

{

var user = await db.Users.FirstOrDefaultAsync(u => u.Number == long.Parse(User.Identity.Name));

var trip = await db.ScheduledTrips.Where(i => i.ID == id && i.UserID == user.ID).FirstOrDefaultAsync();

if (trip == null)

{

ModelState.AddModelError("", "Can't find trip for current user");

return NotFound(ModelState);

}

db.ScheduledTrips.Remove(trip);

await db.SaveChangesAsync();

return Ok();

}

}

return BadRequest();

}

[HttpGet("scheduled/subscribed")]

public async Task<IActionResult> Subscribed()

{

if (ModelState.IsValid)

{

if (User.Identity.IsAuthenticated)

{

var user = await db.Users.Where(u => u.Number == long.Parse(User.Identity.Name))

.Select(i => i.ID)

.FirstOrDefaultAsync();

var subs = await db.SubscribedOnTrips.Where(u=>u.UserID == user).Join(db.Users,

s => s.UserID,

u => u.ID,

(s, u) => new SubscribedOnTripsModel()

{

ID = s.ID,

User = u,

UserID = s.UserID,

ScheduledTripsModelID = s.ID

}).ToListAsync();

var result = await db.ScheduledTrips.Where(u => u.ID == u.SubscribedOnTrips

.Where(s=>s.UserID == user)

.Select(t=>t.ScheduledTripsModelID)

.FirstOrDefault() && u.SubscribedOnTrips

.Select(s=>s.UserID)

.FirstOrDefault() == user)

.Join(db.Users,

s => s.UserID,

u => u.ID,

(s, u) => new ScheduledTripsWithUserModel()

{

ID = u.ID,

Number = u.Number,

Name = u.Name,

Date = u.Date,

Photo = u.Photo,

Rating = u.Rating,

CarMark = u.CarMark,

CarModel = u.CarModel,

CarImage = u.CarImage,

ScheduledTrips = s,

SubscribedOnTrips = subs

}).ToListAsync();

return new JsonResult(result);

}

else

{

return Unauthorized();

}

}

else

{

ModelState.AddModelError("", "Model state is not valid");

}

return BadRequest(ModelState);

}

[HttpPost("scheduled/subscribe/{id}")]

public async Task<IActionResult> SubscribeOnScheduledTrip(int id)

{

if (ModelState.IsValid)

{

if (User.Identity.IsAuthenticated)

{

var user = await db.Users.FirstOrDefaultAsync(u => u.Number == long.Parse(User.Identity.Name));

int countSubs = db.SubscribedOnTrips.Where(t => t.ScheduledTripsModelID == id).ToList().Count();

var countSeats = await db.ScheduledTrips.FirstOrDefaultAsync(s => s.ID == id);

if (countSubs >= countSeats.Seats)

{

ModelState.AddModelError("", "Not enought seats");

return BadRequest(ModelState);

}

SubscribedOnTripsModel model = new SubscribedOnTripsModel()

{

ScheduledTripsModelID = id,

UserID = user.ID

};

db.SubscribedOnTrips.Add(model);

await db.SaveChangesAsync();

return Ok();

}

}

return BadRequest();

}

[HttpPost("scheduled/unsubscribe/{id}")]

public async Task<IActionResult> UnSubscribeOnScheduledTrip(int id)

{

if (ModelState.IsValid)

{

if (User.Identity.IsAuthenticated)

{

var user = await db.Users.FirstOrDefaultAsync(u => u.Number == long.Parse(User.Identity.Name));

SubscribedOnTripsModel model = new SubscribedOnTripsModel()

{

ScheduledTripsModelID = id,

UserID = user.ID

};

db.SubscribedOnTrips.Remove(model);

await db.SaveChangesAsync();

return Ok();

}

return Unauthorized();

}

return BadRequest();

}

[HttpDelete("scheduled/end/{id}")]

public async Task<IActionResult> EndScheduledTrip(int id)

{

if (ModelState.IsValid)

{

if (User.Identity.IsAuthenticated)

{

var trip = await db.ScheduledTrips.FirstOrDefaultAsync(t => t.ID == id);

EndedTripsModel model = new EndedTripsModel()

{

TripID = id,

UserID = trip.UserID,

Date = trip.Date,

From = trip.From,

To = trip.To,

Cost = trip.Cost,

Comment = trip.Comment,

};

await db.AddAsync(model);

db.ScheduledTrips.Remove(trip);

await db.SaveChangesAsync();

return Ok();

}

return Unauthorized();

}

return BadRequest();

}

[HttpPost("scheduled/end/rate/{id}")]

public async Task<IActionResult> RateEndedTrip(int id)

{

if (ModelState.IsValid)

{

if (User.Identity.IsAuthenticated)

{

var user = await db.Users.FirstOrDefaultAsync(u => u.Number == long.Parse(User.Identity.Name));

var driver = await db.EndedTrips.FirstOrDefaultAsync(d => d.TripID == id);

UserReviewModel model = new UserReviewModel()

{

EndedTripsID = id,

UserID = user.ID,

ToID = driver.UserID,

Date = DateTime.Now,

Assessment = 4.5,

Comment = "Test"

};

await db.UserReviews.AddAsync(model);

await db.SaveChangesAsync();

return Ok();

}

return Unauthorized();

}

return BadRequest();

}

}

}

UserController.cs

namespace DriveYou.Controllers

{

[Route("api/[controller]")]

[ApiController]

[RequireHttps]

[Authorize(AuthenticationSchemes = CookieAuthenticationDefaults.AuthenticationScheme)]

public class UserController : ControllerBase

{

private ApplicationDbContext db;

private readonly ILogger<UserController> \_logger;

public UserController(ILogger<UserController> logger, ApplicationDbContext context)

{

\_logger = logger;

db = context;

}

// GET: api/<UserController>

[HttpGet]

[AllowAnonymous]

public IActionResult Get()

{

if (ModelState.IsValid)

{

if (User.Identity.IsAuthenticated)

{

return Ok();

}

}

return BadRequest();

}

[HttpGet("my")]

public async Task<IActionResult> GetMy()

{

if (ModelState.IsValid)

{

if (User.Identity.IsAuthenticated)

{

var user = await db.Users.Where(u => u.Number == long.Parse(User.Identity.Name)).Select(u => u.ID).FirstOrDefaultAsync();

var reviews = await db.UserReviews.Where(u => u.ToID == user).Include(d => d.User).ToListAsync();

var result = await db.Users.Where(u => u.ID == user).Select(ui => new User

{

ID = ui.ID,

Number = ui.Number,

Password = ui.Password,

Email = ui.Email,

Name = ui.Name,

Surname = ui.Surname,

Date = ui.Date,

Photo = ui.Photo,

Rating = ui.Rating,

CarMark = ui.CarMark,

CarModel = ui.CarModel,

CarImage = ui.CarImage,

ScheduledTrips = ui.ScheduledTrips,

EndedTrips = ui.EndedTrips,

UserReviews = reviews,

SubscribedOnTrips = ui.SubscribedOnTrips,

}).FirstOrDefaultAsync();

return new JsonResult(result)

{

ContentType = "application/json;charset=utf-8",

StatusCode = 200,

};

}

}

return BadRequest(ModelState);

}

[HttpPost("userphoto/set")]

public async Task<IActionResult> SetUserImage(User model)

{

if (ModelState.IsValid)

{

if (User.Identity.IsAuthenticated)

{

User user = await db.Users.Where(u => u.Number == long.Parse(User.Identity.Name)).FirstOrDefaultAsync();

user.Photo = model.Photo;

db.Users.Update(user);

await db.SaveChangesAsync();

return Ok();

}

}

return BadRequest();

}

[HttpPost("carphoto/set")]

public async Task<IActionResult> SetCarImage(User model)

{

if (ModelState.IsValid)

{

if (User.Identity.IsAuthenticated)

{

User user = await db.Users.Where(u => u.Number == long.Parse(User.Identity.Name)).FirstOrDefaultAsync();

user.CarImage = model.CarImage;

if (string.IsNullOrWhiteSpace(model.CarMark))

{

user.CarMark = model.CarMark;

}

if (string.IsNullOrWhiteSpace(model.CarModel))

{

user.CarModel = model.CarModel;

}

db.Users.Update(user);

await db.SaveChangesAsync();

return Ok();

}

}

return BadRequest();

}

// GET api/<UserController>/5

//[Authorize]

[HttpGet("{id}")]

public async Task<IActionResult> Get(int id)

{

if (ModelState.IsValid)

{

if (User.Identity.IsAuthenticated)

{

var reviews = await db.UserReviews.Where(u => u.ToID == id).ToListAsync();

var result = await db.Users.Where(u => u.ID == id).Select(ui => new User

{

ID = ui.ID,

Number = ui.Number,

Password = ui.Password,

Email = ui.Email,

Name = ui.Name,

Surname = ui.Surname,

Date = ui.Date,

Photo = ui.Photo,

Rating = ui.Rating,

CarMark = ui.CarMark,

CarModel = ui.CarModel,

CarImage = ui.CarImage,

ScheduledTrips = ui.ScheduledTrips,

EndedTrips = ui.EndedTrips,

UserReviews = reviews,

SubscribedOnTrips = ui.SubscribedOnTrips,

}).FirstOrDefaultAsync();

return new JsonResult(result)

{

ContentType = "application/json;charset=utf-8",

StatusCode = 200,

};

}

else

{

return Unauthorized();

}

}

return BadRequest(ModelState);

}

// PUT api/<UserController>/5

[HttpPut("{id}")]

public IActionResult PutUpdateUserInfo(User user)

{

if (ModelState.IsValid)

{

if (User.Identity.IsAuthenticated)

{

if (user != null)

{

db.Users.Update(user);

return Ok();

}

else

{

ModelState.AddModelError("Error\_NotFound", "User not found!");

return NotFound(ModelState);

}

}

else

{

ModelState.AddModelError("Error\_Auth", "User is not authenticated!");

return BadRequest(ModelState);

}

}

return BadRequest(ModelState);

}

// DELETE api/<UserController>/5

[HttpDelete("{id}")]

public async Task<IActionResult> Delete(int id)

{

if (ModelState.IsValid)

{

if (User.Identity.IsAuthenticated)

{

User delUser = db.Users.Find(id);

if (delUser != null)

{

ModelState.AddModelError("Error\_NotFound", "User not found");

return NotFound(ModelState);

}

//TODO: Relocate user to "Deleted User's" Table

await HttpContext.SignOutAsync(CookieAuthenticationDefaults.AuthenticationScheme);

db.Users.Remove(delUser);

await db.SaveChangesAsync();

return Ok();

}

}

return BadRequest(ModelState);

}

}

}

HomeTripsViewModel.cs

namespace DriveYou\_Mobile.ViewModels

{

public class HomeTripsViewModel : BaseViewModel

{

private FindTripModel \_findTrip;

private ObservableCollection<ScheduledTripsWithUserModel> trips;

public Command UpdateCommand { get; }

public Command BackToHomeCommand { get; }

public Command OpenTripDetails { get; }

public HomeTripsViewModel()

{

}

public HomeTripsViewModel(FindTripModel model)

{

\_findTrip = model;

UpdateCommand = new Command(LoadTrips);

BackToHomeCommand = new Command(BackToHomeCommandClicked);

LoadTrips();

}

public ObservableCollection<ScheduledTripsWithUserModel> Trips

{

get { return trips; }

set

{

trips = value;

OnPropertyChanged("Trips");

}

}

private ScheduledTripsWithUserModel \_selectedTrip;

public ScheduledTripsWithUserModel SelectedTrip

{

get { return \_selectedTrip; }

set

{

\_selectedTrip = value;

OnPropertyChanged("SelectedTrip");

if (value != null)

{

AppShell.Current.Navigation.PushAsync(new TripsDetailedPage(SelectedTrip, true));

SelectedTrip = null;

}

}

}

private bool isRefreshing;

public bool IsRefreshing

{

get { return isRefreshing; }

set {

isRefreshing = value;

OnPropertyChanged("IsRefreshing");

}

}

private async void BackToHomeCommandClicked()

{

await AppShell.Current.Navigation.PopAsync();

}

public async void LoadTrips()

{

string jsonContent = System.Text.Json.JsonSerializer.Serialize(\_findTrip, typeof(FindTripModel));

HttpContent postContent = new StringContent(jsonContent, Encoding.UTF8, "application/json");

HttpResponseMessage response = await App.client.PostAsync("trips/scheduled", postContent);

if (response.IsSuccessStatusCode)

{

using (StreamReader reader = new StreamReader(await response.Content.ReadAsStreamAsync(), Encoding.UTF8))

{

string jsonStr = reader.ReadToEnd();

Trips = JsonConvert.DeserializeObject<ObservableCollection<ScheduledTripsWithUserModel>>(jsonStr);

};

IsRefreshing = false;

}

}

}

}

LoginVewModel.cs

namespace DriveYou\_Mobile.ViewModels

{

public class LoginViewModel : BaseViewModel

{

public Command LoginCommand { get; }

public Command GoToRegisterCommand { get; }

public LoginViewModel()

{

LoginCommand = new Command(OnLoginClicked);

GoToRegisterCommand = new Command(OnToRegisterClicked);

LoginUser = new LoginModel();

}

private LoginModel \_loginUser;

public LoginModel LoginUser

{

get { return \_loginUser; }

set

{

\_loginUser = value;

OnPropertyChanged("LoginUser");

}

}

private async void OnToRegisterClicked()

{

await Shell.Current.GoToAsync("RegisterPage");

}

private async void OnLoginClicked()

{

string jsonContent = JsonSerializer.Serialize(LoginUser, typeof(LoginModel));

HttpContent postContent = new StringContent(jsonContent, Encoding.UTF8, "application/json");

HttpResponseMessage response = await App.client.PostAsync("login/signin", postContent);

if (response.IsSuccessStatusCode)

{

await Shell.Current.GoToAsync("HomePage", true);

(AppShell.Current as AppShell).DisableLogin();

}

}

}

}

AccountViewModel.cs

namespace DriveYou\_Mobile.ViewModels

{

public class AccountViewModel : BaseViewModel

{

public Command EditCarInfoCommand { get; }

public Command LoadUserPhotoCommand { get; }

public Command RefreshCommand { get; }

public Command HideBusyIndicator { get; }

private int userID = -1;

public AccountViewModel()

{

IsOwner = true;

IsBusyLoading = true;

LoadUserInfo(-1);

EditCarInfoCommand = new Command(EditCarInfoCommand\_Click);

LoadUserPhotoCommand = new Command(LoadUserPhotoCommand\_Click);

RefreshCommand = new Command(RefreshCommand\_Action);

HideBusyIndicator = new Command(HideBusyIndicator\_Event);

RepCounter = 20;

}

public AccountViewModel(bool isOwner, int id = -1)

{

LoadUserInfo(id);

IsBusyLoading = true;

userID = id;

IsOwner = isOwner;

EditCarInfoCommand = new Command(EditCarInfoCommand\_Click);

LoadUserPhotoCommand = new Command(LoadUserPhotoCommand\_Click);

HideBusyIndicator = new Command(HideBusyIndicator\_Event);

RepCounter = 20;

}

private bool isBusyLoading;

public bool IsBusyLoading

{

get { return isBusyLoading; }

set { isBusyLoading = value;

OnPropertyChanged("IsBusyLoading");

}

}

private bool isRefreshing;

public bool IsRefreshing

{

get { return isRefreshing; }

set { isRefreshing = value;

OnPropertyChanged("IsRefreshing");

}

}

private string animationUrl;

public string AnimationUrl

{

get { return animationUrl; }

set

{

animationUrl = value;

if (value == "https://assets7.lottiefiles.com/private\_files/lf30\_nrnx3s.json")

{

RepCounter = 0;

}

OnPropertyChanged("AnimationUrl");

}

}

private string carInfo;

public string CarInfo

{

get { return carInfo; }

set { carInfo = value;

OnPropertyChanged("CarInfo");

}

}

private double listHeight;

public double ListHeight

{

get { return listHeight; }

set { listHeight = value;

OnPropertyChanged("ListHeight");

}

}

public bool IsOwner { get; set; }

private UserModel \_selectedUser;

public UserModel SelectedUser

{

get { return \_selectedUser; }

set { \_selectedUser = value;

if (value != null)

{

AppShell.Current.Navigation.PushAsync(new AccountPage(false, SelectedUser.ID));

}

SelectedUser = null;

OnPropertyChanged("SelectedUser");

}

}

private UserModel \_currentUser;

public UserModel CurrentUser

{

get { return \_currentUser; }

set

{

\_currentUser = value;

if (CurrentUser.UserReviews != null)

{

ListHeight = 80 \* CurrentUser.UserReviews.Count;

}

OnPropertyChanged("CurrentUser");

}

}

private async void EditCarInfoCommand\_Click()

{

await Shell.Current.Navigation.PushModalAsync(new EditCarPage(CurrentUser));

}

private void RefreshCommand\_Action()

{

IsRefreshing = true;

LoadUserInfo(userID);

IsRefreshing = false;

}

private async void LoadUserPhotoCommand\_Click()

{

await CrossMedia.Current.Initialize();

if (!CrossMedia.Current.IsPickPhotoSupported)

{

await Shell.Current.DisplayAlert("Not supported", "Your device does not support this functionality", "Ok");

return;

}

else

{

var mediaOptions = new PickMediaOptions()

{

PhotoSize = PhotoSize.Medium

};

var selectedImage = await CrossMedia.Current.PickPhotoAsync(mediaOptions);

if (selectedImage == null)

{

await Shell.Current.DisplayAlert("Error", "Could not get image, please try again", "Ok");

return;

}

else

{

using (MemoryStream ms = new MemoryStream())

{

selectedImage.GetStream().CopyTo(ms);

byte[] byteImage = ms.ToArray();

CurrentUser.Photo = Convert.ToBase64String(byteImage);

}

}

}

CurrentUser.ScheduledTrips = null;

CurrentUser.SubscribedOnTrips = null;

string jsonContent = System.Text.Json.JsonSerializer.Serialize(CurrentUser, typeof(UserModel));

HttpContent postContent = new StringContent(jsonContent, Encoding.UTF8, "application/json");

HttpResponseMessage response = await App.client.PostAsync("user/userphoto/set", postContent);

if (response.IsSuccessStatusCode)

{

await Shell.Current.DisplayAlert("Done", "Completed post metod", "Ok");

}

}

private int repCounter;

public int RepCounter

{

get { return repCounter; }

set { repCounter = value;

OnPropertyChanged("RepCounter");

}

}

private void HideBusyIndicator\_Event()

{

if (AnimationUrl == "https://assets7.lottiefiles.com/private\_files/lf30\_nrnx3s.json")

{

IsBusyLoading = false;

}

}

private async void LoadUserInfo(int id)

{

IsBusyLoading = true;

AnimationUrl = "https://assets2.lottiefiles.com/packages/lf20\_tsxbtrcu.json";

HttpResponseMessage response;

if (id < 0)

{

response = await App.client.GetAsync("user/my");

}

else

{

response = await App.client.GetAsync($"user/{id}");

}

if (response.IsSuccessStatusCode)

{

using (StreamReader reader = new StreamReader(await response.Content.ReadAsStreamAsync(), Encoding.UTF8))

{

string jsonStr = reader.ReadToEnd();

CurrentUser = JsonConvert.DeserializeObject<UserModel>(jsonStr);

CarInfo = string.Format("{0}, {1}", CurrentUser.CarMark, CurrentUser.CarModel);

}

}

AnimationUrl = "https://assets7.lottiefiles.com/private\_files/lf30\_nrnx3s.json";

}

}

}

TripsDetailsPageViewModel.cs

namespace DriveYou\_Mobile.ViewModels

{

public class TripsDetailedPageViewModel : BaseViewModel

{

public Command ActionCommand { get; }

public Command CallCommand { get; }

public Command MessageCommand { get; }

public Command BackToHomeTripsCommand { get; }

public Command OpenDriverInfo { get; }

private bool IsSubscribed = false;

public TripsDetailedPageViewModel()

{

}

public TripsDetailedPageViewModel(ScheduledTripsWithUserModel model, bool isCompanion)

{

IsTripCompanion = isCompanion;

if (IsTripCompanion == true)

{

IsSubscribed\_Check();

}

Trip = model;

LoadDestination();

ActionCommand = new Command(SubscribeUnsubscribeOnTrip);

CallCommand = new Command(CallCommand\_Click);

OpenDriverInfo = new Command(OpenDriverInfo\_Click);

MessageCommand = new Command(MessageCommand\_Click);

BackToHomeTripsCommand = new Command(BackToHomeTripsNavigationClicked);

}

private string distance;

public string Distance

{

get { return distance; }

set { distance = value;

OnPropertyChanged("Distance");

}

}

private double listHeight;

public double ListHeight

{

get { return listHeight; }

set { listHeight = value;

OnPropertyChanged("ListHeight");

}

}

private string carInfo;

public string CarInfo

{

get { return carInfo; }

set { carInfo = value;

OnPropertyChanged("CarInfo");

}

}

private string seatInfo;

public string SeatInfo

{

get { return seatInfo; }

set { seatInfo = value;

OnPropertyChanged("SeatInfo");

}

}

private bool istripCompanion;

public bool IsTripCompanion

{

get { return istripCompanion; }

set { istripCompanion = value;

OnPropertyChanged("IsTripCompanion");

}

}

private string actionCaption;

public string ActionCaption

{

get { return actionCaption; }

set { actionCaption = value;

OnPropertyChanged("ActionCaption");

}

}

private ScheduledTripsWithUserModel trip;

public ScheduledTripsWithUserModel Trip

{

get { return trip; }

set { trip = value;

if (Trip.SubscribedOnTrips != null)

{

ListHeight = value.SubscribedOnTrips.Count \* 80;

}

OnPropertyChanged("Trip");

}

}

private SubscribedOnTripsModel subscribedUser;

public SubscribedOnTripsModel SubscribedUser

{

get { return subscribedUser; }

set { subscribedUser = value;

if (value != null)

{

AppShell.Current.Navigation.PushAsync(new AccountPage(false, SubscribedUser.UserID));

SubscribedUser = null;

}

OnPropertyChanged("SubscribedUser");

}

}

private async void OpenDriverInfo\_Click()

{

await AppShell.Current.Navigation.PushAsync(new AccountPage(false, Trip.ScheduledTrips.UserID));

}

private async void IsSubscribed\_Check()

{

HttpResponseMessage response = await App.client.GetAsync("trips/scheduled/subscribed");

if (response.IsSuccessStatusCode)

{

using (StreamReader reader = new StreamReader(await response.Content.ReadAsStreamAsync(), Encoding.UTF8))

{

string jsonStr = reader.ReadToEnd();

List<SubscribedOnTripsModel> model = JsonConvert.DeserializeObject<List<SubscribedOnTripsModel>>(jsonStr);

if (model.Count > 0)

{

foreach (var item in model)

{

IsSubscribed = item.ID == Trip.ID ? true : false;

ActionCaption = item.ID == Trip.ID ? "Отписаться" : "Подписаться";

}

}

else

{

IsSubscribed = false;

ActionCaption = "Подписаться";

}

}

}

}

private async void BackToHomeTripsNavigationClicked()

{

await AppShell.Current.Navigation.PopAsync();

}

private async void SubscribeUnsubscribeOnTrip()

{

string command = IsSubscribed == true ? "unsubscribe" : "subscribe";

HttpResponseMessage response = await App.client.PostAsync($"trips/scheduled/{command}/{Trip.ScheduledTrips.ID}", null);

if (response.IsSuccessStatusCode)

{

IsSubscribed = !IsSubscribed;

ActionCaption = IsSubscribed == true ? "Отписаться" : "Подписаться";

}

}

private void CallCommand\_Click()

{

PhoneDialer.Open($"+{Trip.Number.ToString()}");

}

private async void MessageCommand\_Click()

{

var message = new SmsMessage("Здравствуйте !", $"+{Trip.Number.ToString()}");

await Sms.ComposeAsync(message);

}

private async void LoadDestination()

{

if (Trip.ScheduledTrips != null)

{

if (!string.IsNullOrEmpty(Trip.CarMark) && !string.IsNullOrEmpty(Trip.CarModel))

{

CarInfo = $"{Trip.CarMark}, {Trip.CarModel}";

}

if (Trip.SubscribedOnTrips != null)

{

SeatInfo = $"{Trip.ScheduledTrips.Seats - Trip.SubscribedOnTrips.Count} из {Trip.ScheduledTrips.Seats} свободно";

}

else

{

SeatInfo = $"{Trip.ScheduledTrips.Seats} места свободно";

}

}

var sourceLocation = await Geocoding.GetLocationsAsync(Trip.ScheduledTrips.From);

var destinationLocation = await Geocoding.GetLocationsAsync(Trip.ScheduledTrips.To);

var source = sourceLocation.FirstOrDefault();

var destination = destinationLocation.FirstOrDefault();

HttpClient geolocationApi = new HttpClient()

{

BaseAddress = new Uri("https://api.mapbox.com/directions/v5/mapbox/driving/", UriKind.RelativeOrAbsolute)

};

string geoStr = $"{source.Longitude}%2C{source.Latitude}%3B{destination.Longitude}%2C{destination.Latitude}" +

$"?alternatives=false&continue\_straight=false&geometries=geojson&overview=simplified&steps=false&" +

$"access\_token=pk.eyJ1Ijoic3BhY2Vjb2RlciIsImEiOiJjbDRnZnM2MHUwOGl1M2NuendscHRhM2IxIn0.AyPSVzxpvOxT3q0DrNpbdg";

HttpResponseMessage response = await geolocationApi.GetAsync(geoStr);

if (response.IsSuccessStatusCode)

{

using (StreamReader reader = new StreamReader(await response.Content.ReadAsStreamAsync(), Encoding.UTF8))

{

string jsonStr = reader.ReadToEnd();

dynamic data = JsonConvert.DeserializeObject(jsonStr);

var values = data["routes"];

for (int i = 0; i < values.Count; i++)

{

double distance = (double)values[i]["distance"];

distance = distance / 1000;

Distance = string.Format("~{0} км", Convert.ToInt32(distance));

}

}

}

}

}

}