Министерство образования Республики Беларусь
Учреждение образования «Белорусский государственный университет информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра информатики

Дисциплина: Объектно-ориентированное программирование

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к курсовой работе
на тему

МОБИЛЬНОЕ ПРИЛОЖЕНИЕ ДЛЯ СЕТИ СПОРТЗАЛОВ

БГУИР 1-40 04 312

Студент: гр. 253503 Котова К.А.

Руководитель: ассистент кафедры информатики, Рогов М.Г.

Минск 2024

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В последние годы наблюдается значительное увеличение числа пользователей мобильных приложений, предназначенных для спорта и фитнеса. Эти приложения помогают пользователям планировать тренировки, отслеживать свои достижения и взаимодействовать с тренерами и администрацией спортзала. Целью данной курсовой работы является создание мобильного приложения для сети спортзалов, которое позволит пользователям удобно управлять своими тренировками и абонементами, а также получать актуальную информацию о тренерах и расписании.

Для достижения этой цели необходимо решить такие задачи, как проведение анализа существующих мобильных приложений для фитнеса и спорта, выявление их сильных и слабых сторон, определение функциональных требований к мобильному приложению на основе анализа потребностей потенциальных пользователей, разработка дизайна пользовательского интерфейса, обеспечивающего удобство и интуитивную понятность использования, реализация следующих функциональных модулей: авторизация и регистрация пользователей, покупка, активация и заморозка абонементов, просмотр списка тренеров и запись на тренировки, отображение расписания тренировок по дням недели, проведение тестирования приложения на различных мобильных устройствах для обеспечения его стабильной работы, разработка рекомендаций по дальнейшему развитию и улучшению функционала мобильного приложения.

Детальная постановка задачи заключается в создании мобильного приложения, включающего следующие основные функции: возможность регистрации и авторизации пользователей для доступа к персонализированным данным, покупка абонемента и его активация через приложение, возможность заморозки абонемента на определенный период, просмотр списка тренеров, их специализаций и доступных временных слотов для записи на тренировки, запись на тренировки к выбранному тренеру через приложение, отображение расписания тренировок пользователя по дням недели с указанием времени и тренера.

Разработка данного приложения позволит значительно упростить и улучшить взаимодействие между клиентами и администрацией спортзала, повысить удобство использования услуг спортзала и увеличить удовлетворенность пользователей. Приложение также обеспечит доступ к необходимой информации в любое время и в любом месте, что сделает тренировки еще более удобными и продуктивными.

1 ПЛАТФОРМА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

1.1 Структура и архитектура платформы

Microsoft Visual Studio представляет собой интегрированную среду разработки (IDE), разработанную компанией Microsoft, для создания приложений, работающих на платформе .NET. Основная особенность этой платформы заключается в обширном наборе сервисов, доступных на различных языках программирования. Эти сервисы реализуются в виде промежуточного кода, что обеспечивает независимость от базовой архитектуры.

Главной целью создания Microsoft Visual Studio было оснащение разработчиков специальными сервисно-ориентированными приложениями, способными работать на любой платформе — от персонального компьютера до мобильного устройства. Платформа объединяет огромное количество функций, позволяющих осуществлять разработку для всех версий Windows, включая Windows 8, а также для интернета, различных мобильных устройств и облачных технологий.

Visual Studio предлагает новую среду разработчика, что значительно упрощает процесс создания приложений. Она представляет собой обновленную и упрощенную программную среду, отличающуюся высокой производительностью, не зависящей от особенностей оборудования.

1.2 История, версии, достоинства

История Visual Studio начинается с ее первой версии в 1997 году. С тех пор она прошла через ряд обновлений и улучшений. В настоящее время существуют различные версии Visual Studio, такие как Community, Professional и Enterprise, каждая из которых предназначена для определенных типов разработчиков и организаций.

Visual Studio обладает множеством достоинств: мощные инструменты разработки обеспечивают широкие возможности для создания приложений различной сложности; интеграция с платформой .NET обеспечивает удобство в работе с этой технологией; кроссплатформенная разработка позволяет создавать приложения для разных операционных систем; интеграция с облачными технологиями обеспечивает легкость работы с облачными сервисами; современный интерфейс делает работу с Visual Studio приятной и удобной; совместная работа позволяет командам разработчиков эффективно взаимодействовать; а возможность расширения функциональности позволяет адаптировать среду разработки под конкретные нужды пользователей.

1.3 Обоснование выбора платформы

Выбор Visual Studio для разработки мобильного приложения для сети спортзалов в сочетании с Maui и ASP.NET Core обоснован рядом факторов:

1. Универсальность и удобство разработки с Maui. Проект Maui позволяет создавать кросс-платформенные приложения с общим кодом для iOS, Android и Windows, что существенно упрощает разработку и поддержку приложения. Использование Visual Studio в сочетании с Maui обеспечивает интегрированную среду разработки с мощными инструментами для создания и отладки приложений.

2. ASP.NET Core для серверной части приложения. Использование ASP.NET Core для разработки серверной части приложения обеспечивает высокую производительность, масштабируемость и безопасность. Интеграция ASP.NET Core с Visual Studio позволяет удобно разрабатывать и отлаживать как клиентскую, так и серверную части приложения в единой среде разработки.

3. Совместимость и поддержка среды разработки. Visual Studio обладает богатым набором инструментов и библиотек для разработки, которые хорошо интегрированы с Maui и ASP.NET Core. Это обеспечивает удобство использования, высокую производительность и надежность при разработке и отладке приложения.

Таким образом, выбор Visual Studio как основной платформы для разработки мобильного приложения с использованием Maui и ASP.NET Core обеспечивает не только удобство и эффективность разработки, но и гарантирует высокое качество и производительность разрабатываемого приложения.

1.4 Анализ существующих аналогов

На сегодняшний день все большую популярность набирают мобильные приложение, которые упрощают взаимодействие между клиентами и администрацией спортзала, позволяя клиентам покупать и отслеживать свои абонементы, а также записываться на тренировки.

В качестве примера рассмотрим, что предлагают такие популярные сети, как GYM24 и TITAN. Меню приложений представлено на рисунке ниже.

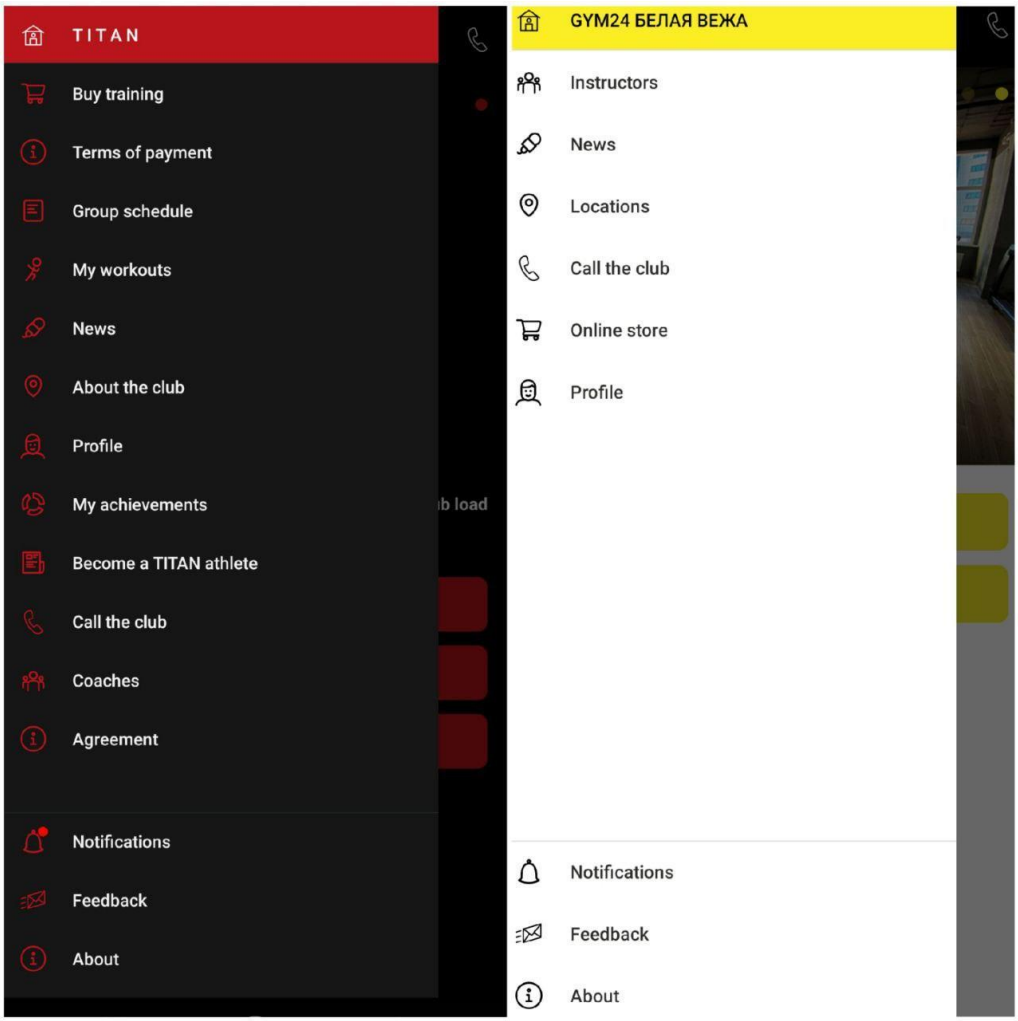

Рисунок 1.1 – Меню приложений

Приложения обладают интуитивно понятным интерфейсом. Каждое из приложений предоставляет возможность купить абонемент как просто для занятий в зале, так и для групповых тренировок с тренером, однако ни одно из приложений не дает возможности записи на персональные тренировки, в то время как такая услуга предоставляется самим залом.

Также в обоих приложениях имеется профиль пользователя, где отображаются купленные абонементы и история посещений (см. рисунок 1.2):

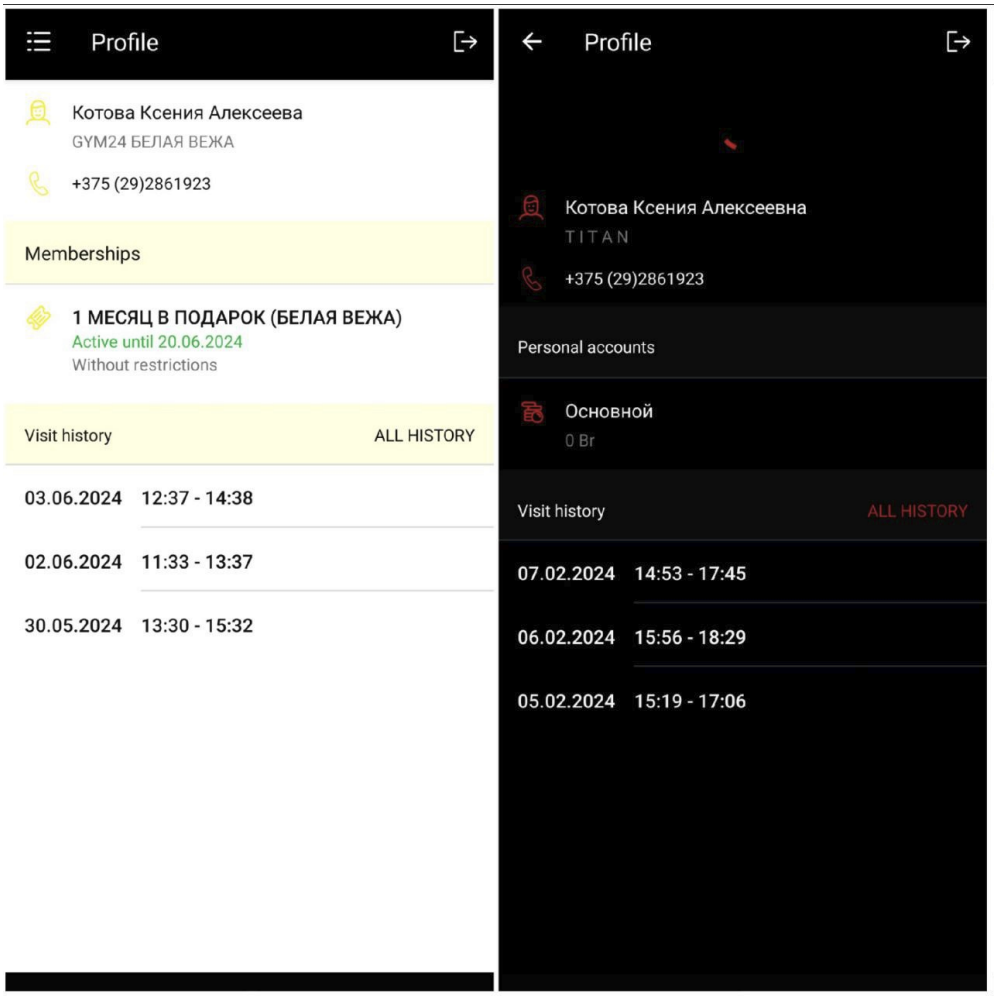

Рисунок 1.2 – Профиль пользователя

В профиле можно отследить срок действия абонемента, а также заморозить или разморозить его.

В каждом приложении есть список тренеров с подробной информацией о них и вкладка с местоположением залов на карте. TITAN также дает возможность отслеживать расписание групповых тренировок, на которые записался клиент по датам, чего нет в приложении GYM24.

Таким образом, стоит учесть возможность записи на персональные тренировки и отслеживания расписания групповых тренировок, так как эти функции могут быть востребованы пользователями.

2 ТЕОРЕТИЧЕСКИЕ ОБОСНОВАНИЯ РАЗРАБОТКИ ПРОГРАММНОГО ПРОДУКТА

2.1 Обоснование необходимости разработки

Современный образ жизни требует удобных и эффективных способов занятий спортом. Мобильные приложения для сети спортзалов представляют собой ответ на этот запрос, обеспечивая клиентам удобство управления своими тренировками и абонементами.

1. Увеличение удобства для клиентов. Мобильное приложение предоставит клиентам спортзала удобный способ управления своими тренировками и абонементами прямо с их смартфонов. Это улучшит их общий опыт взаимодействия с услугами спортзала и повысит их удовлетворенность.

2. Повышение эффективности управления тренировками. Приложение позволит пользователям легко записываться на тренировки, просматривать расписание, выбирать тренеров и многое другое. Это сделает процесс планирования тренировок более эффективным для клиентов и персонала спортзала.

3. Расширение аудитории. Мобильное приложение может привлечь новых клиентов, которые предпочитают использовать мобильные устройства для управления своими занятиями. Это поможет расширить базу клиентов спортзала и увеличить его доходы.

4. Повышение конкурентоспособности. В современном мире мобильные приложения становятся неотъемлемой частью бизнеса. Разработка приложения для спортзала поможет ему оставаться конкурентоспособным на рынке и привлекать больше клиентов.

5. Улучшение взаимодействия с клиентами. Мобильное приложение предоставит спортзалу новые возможности для взаимодействия с клиентами, такие как отправка уведомлений о специальных предложениях и акциях, получение обратной связи от пользователей и т. д. Это поможет укрепить связь между спортзалом и его клиентами.

В целом, разработка мобильного приложения для сети спортзалов обещает принести множество выгод как для клиентов, так и для самого спортзала, делая процесс занятий спортом более удобным, эффективным и привлекательным.

2.2 Технологии программирования, используемые для решения поставленных задач

Для разработки кросс-платформенного приложения на платформе .NET MAUI используются следующие технологии программирования:

1. Язык программирования C#, который является основным языком программирования в .NET и обеспечивает широкие возможности для разработки логики приложения, управления пользовательским интерфейсом и взаимодействия с сервером.

2. ASP.NET Core, мощный и гибкий фреймворк для создания веб-приложений и служб, который обеспечивает высокую производительность и позволяет использовать модель разработки MVC или Razor Pages.

3. Entity Framework Core, ORM-фреймворк, который позволяет работать с базами данных с использованием .NET объектов. Это упрощает операции CRUD и обеспечивает абстракцию от конкретной СУБД.

4. PostgreSQL, мощная объектно-реляционная база данных. Используется для хранения и управления данными приложения, такими как данные пользователей, настройки и другие. PostgreSQL обеспечивает надежное и эффективное управление данными, что важно для крупных и сложных приложений.

5. .NET MAUI (Multi-platform App UI), фреймворк для создания кросс-платформенных приложений. Позволяет разрабатывать приложения для различных платформ, используя единый код и UI.

2.3 Связь архитектуры ООП с разрабатываемым программным обеспечением

Принципы объектно-ориентированного программирования (ООП) являются основой при разработке приложения на платформе .NET MAUI. Ключевые концепции ООП, включая инкапсуляцию, наследование и полиморфизм, способствуют формированию структурированного и адаптивного кода, что упрощает процесс разработки, поддержки и расширения приложения.

Понимание ООП критически важно для эффективной разработки на .NET MAUI, поскольку все элементы пользовательского интерфейса и бизнес-логики должны быть структурированы в виде классов.

В проекте также активно применяются принципы SOLID. Например, принцип единственной ответственности (SRP) обеспечивает, что каждый класс выполняет только свою специфическую функцию. Принцип открытости/закрытости (OCP) позволяет расширять функциональность без изменения существующего кода.

Кроме того, приложение следует принципу подстановки Барбары Лисков (LSP), который гарантирует, что подклассы могут служить заменой для их базовых классов. Принцип разделения интерфейса (ISP) гарантирует, что классы не зависят от интерфейсов, которые они не используют. И наконец, принцип инверсии зависимостей (DIP) помогает уменьшить связанность кода, делая классы независимыми от конкретных реализаций.

Этот подход обеспечивает гибкость и масштабируемость нашего приложения, позволяя нам легко адаптироваться к новым требованиям и вводить новые функции по мере их появления.

3 АРХИТЕКТУРА РАЗРАБОТАННОЙ СИСТЕМЫ

3.1 Общая структура программы

Архитектура клиентской части была разработана с использованием паттерна MVC (Model-View-Controller), который предполагает разделение данных приложения, пользовательского интерфейса и управляющей логики.

MVVM состоит из трех основных компонентов:

1. Model (Модель). Это представление данных и бизнес-логики. Модель включает в себя классы, которые содержат данные, и методы, которые обрабатывают эти данные.

2. View (Представление). Это то, что видит пользователь – пользовательский интерфейс. Представление отображает информацию из модели для пользователя и передает пользовательский ввод (например, нажатия кнопок, ввод текста) в ViewModel.

3. ViewModel (Модель представления). ViewModel является связующим звеном между Model и View. Она получает ввод от представления, обрабатывает его (возможно, с использованием модели) и обновляет представление с помощью механизма привязки данных.

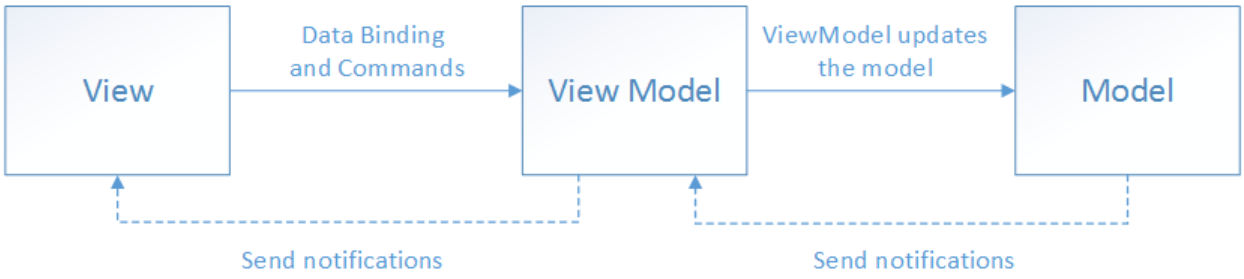

Рисунок 3.1 – Принцип работы паттерна MVVM

Основное преимущество MVVM в том, что он обеспечивает четкое разделение между пользовательским интерфейсом и бизнес-логикой, что упрощает тестирование и поддержку кода.

3.2 Описание функциональных схем программы

При запуске приложения должна отображаться главная страница, предоставляющая возможность регистрации и входа для пользователя. Для входа в систему пользователь должен иметь возможность использовать свою почту и пароль. Вошедший в систему пользователь имеет возможность покупки абонемента, активации абонемента и его последующей заморозки, установление фотографии профиля и просмотра своей личной информации, просмотра списка тренеров и записи к ним на определенную дату и время

разово или на месяц, а также просмотра тренировок на день и отмены тренировки до ее начала.

Техническое изложение предоставляемого функционала представлено на use-case диаграмме(см. рисунок 3.1):

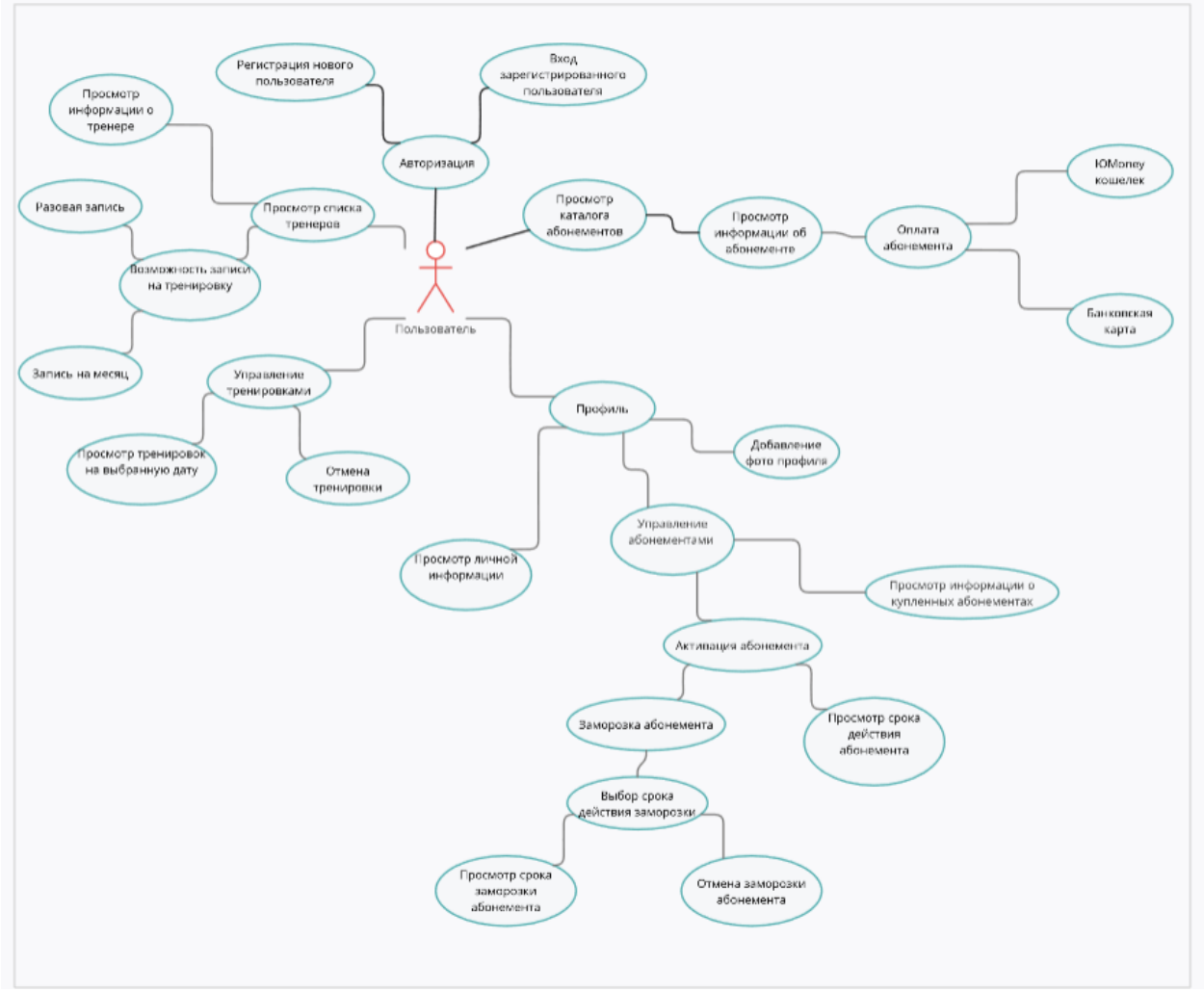

Рисунок 3.1 – Use-case диаграмма приложения для пользователя

При входе в систему администратора, ему доступны такие функции как просмотр списков пользователей, тренеров, заморозок и абонементов, добавление, удаление, поиск и редактирование информации в них.

Техническое изложение предоставляемого функционала представлено на use-case диаграмме(см. рисунок 3.1):

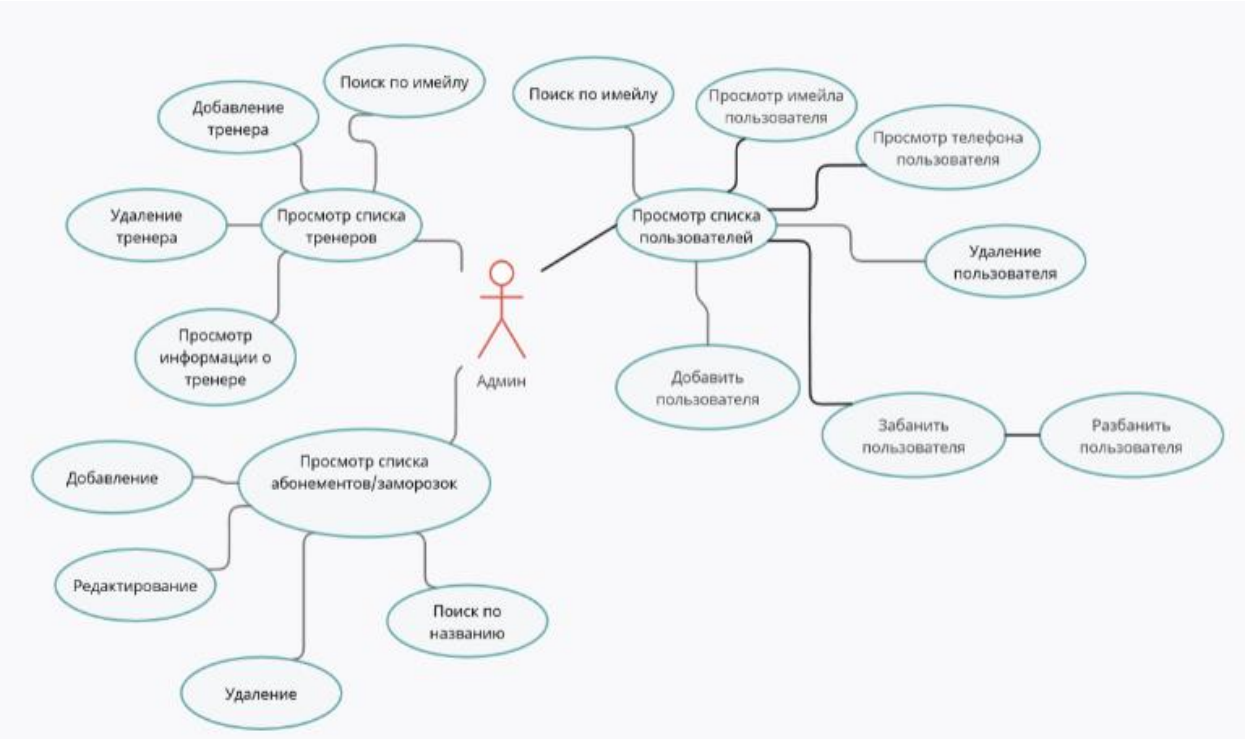

Рисунок 3.2 – Use-case диаграмма приложения для администратора

Тренер может управлять временными слотами для тренировок, а также имеет доступ к информации о записавшихся клиентах.

Рисунок 3.3 – Use-case диаграмма приложения для тренера

3.3 Описание блок-схемы алгоритма программы

Навигация по мобильному приложению представлена на рисунке 3.3

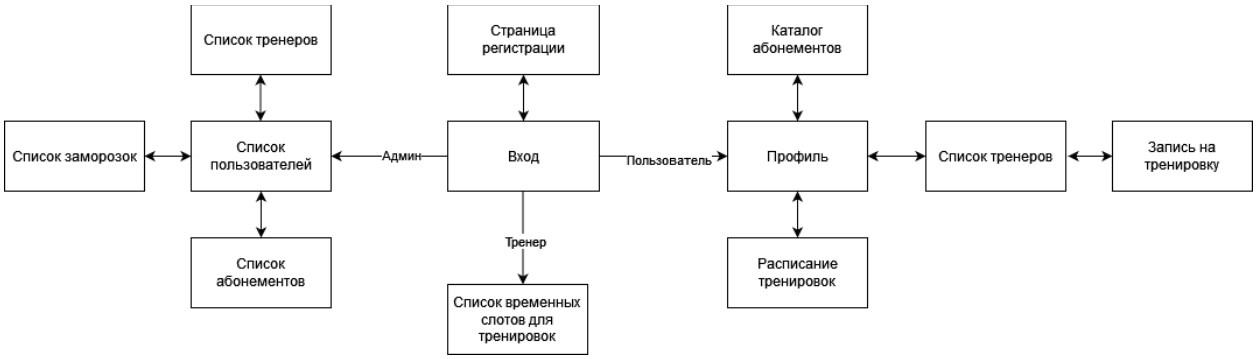

Рисунок 3.3 – Навигация по приложению

Поскольку в процессе разработки программной реализации приложения применялась объектно-ориентированная архитектура, это обеспечило возможность детального описания используемых сущностей и их атрибутов в форме отдельных объектов. Диаграмма классов разработанного мобильного приложения представлена в Приложении Б.

4 ФУНКЦИОНАЛЬНЫЕ ВОЗМОЖНОСТИ ПРОГРАММЫ

Основной функционал программного средства включает в себя возможность аутентификации, регистрации нового пользователя, просмотр и покупка абонементов, а также их заморозка и активация в личном кабинете пользователя. Также основной функционал предоставляет возможность записи на тренировки к тренеру и отслеживания запланированных тренировок.

Начальное окно представляет собой страницу, где пользователь может либо ввести свои данные и авторизироваться, либо пройти регистрацию. На рисунке 4.1 представлено начальное окно. Если пользователь не имеет аккаунта, то он может перейти по ссылке "Sign Up", где он будет перенаправлен на страницу регистрации.

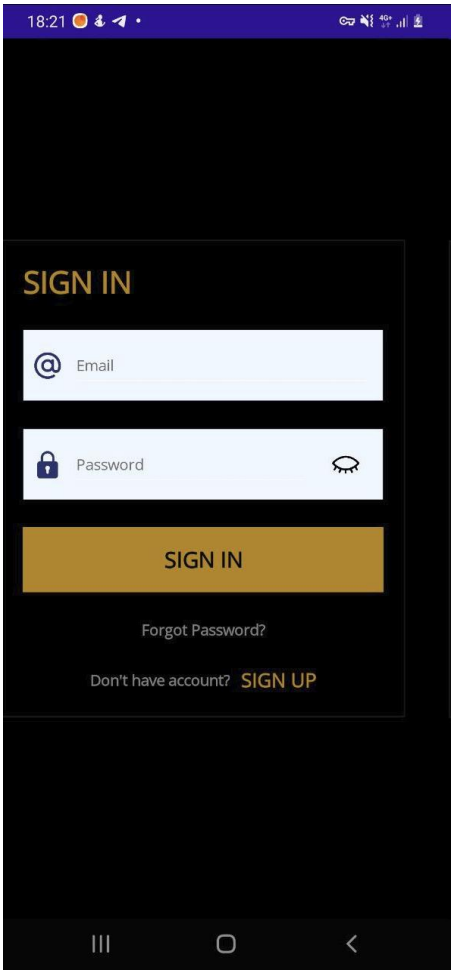

Рисунок 4.1 – Начальная страница

При переходе на страницу регистрации пользователю открывается диалоговое окно (см. рисунок 4.2), в котором пользователь может зарегистрироваться, указав при этом имя пользователя, почту, номер телефона и пароль.

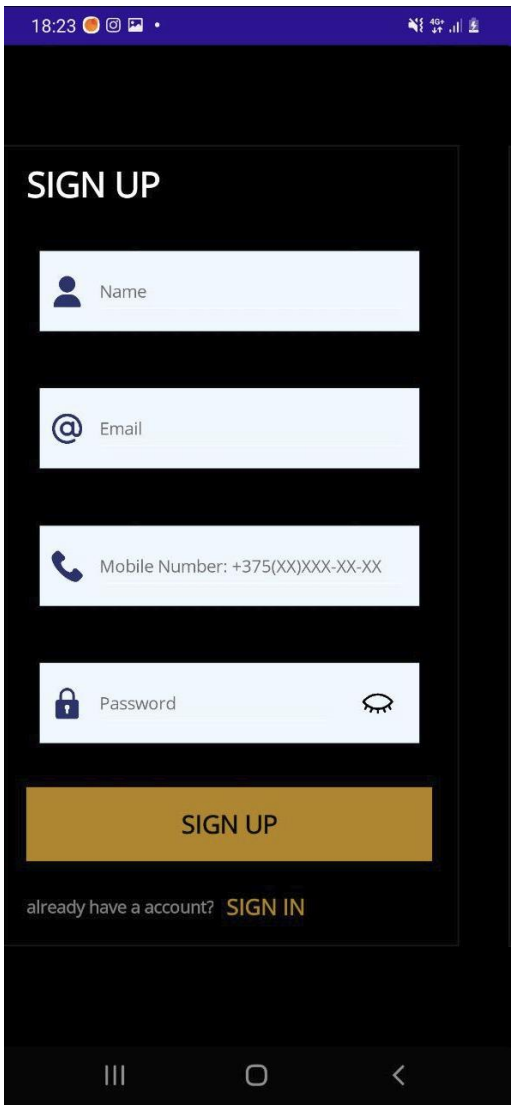

Рисунок 4.2 – Страница регистрации

Если какие-то данные введены неверно или указанная почта занята, выскочит сообщение об ошибке (см. рисунок 4.3).

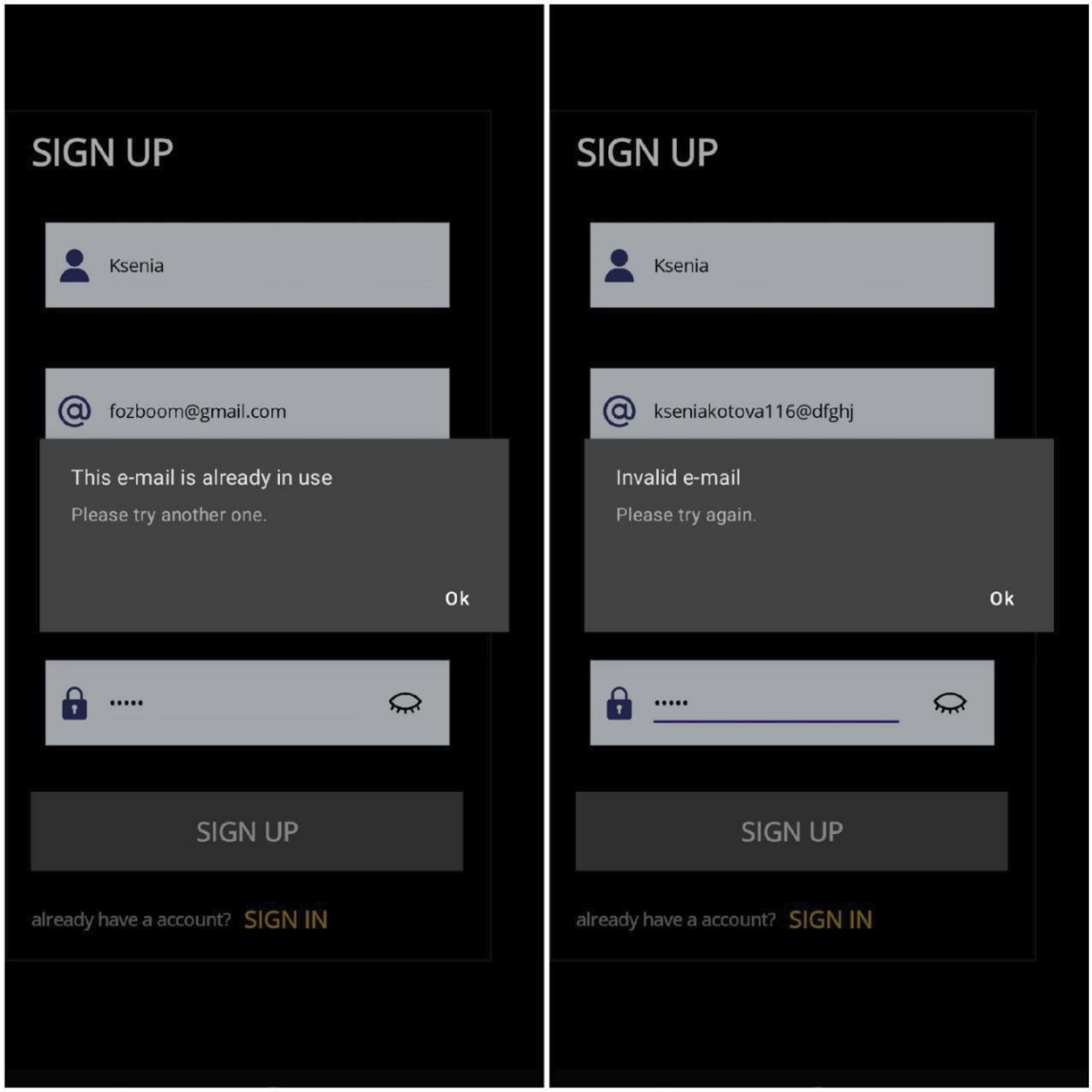

Рисунок 4.3 – Сообщения об ошибках

Если все данные введены корректно, появится сообщение об успешной регистрации и пользователю будет предложено войти в новый аккаунт.

В случае успешной аутентификации, происходит перенаправление на новую страницу (см. рисунок 4.4), где пользователь получает доступ к основным функциям приложения. Начальная страница приложения – профиль.

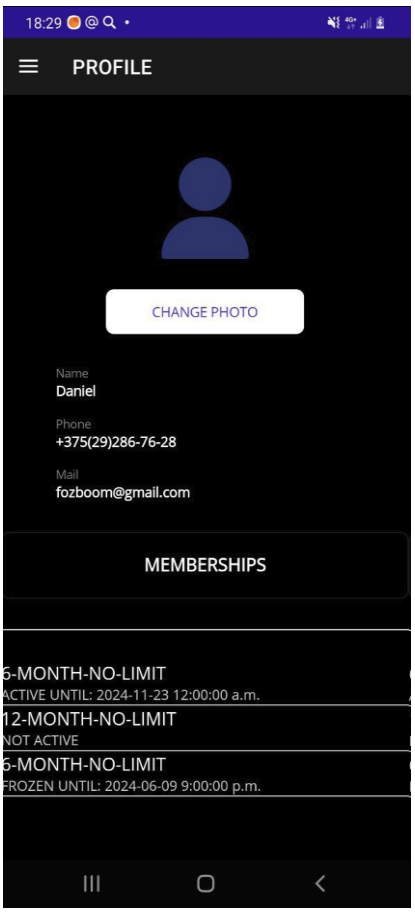

Рисунок 4.4 – Профиль пользователя

Пользователь может добавить, активировать, заморозить и разморозить абонемент(см. рисунок 4.4), а также отслеживать срок действия абонемента или его заморозки.

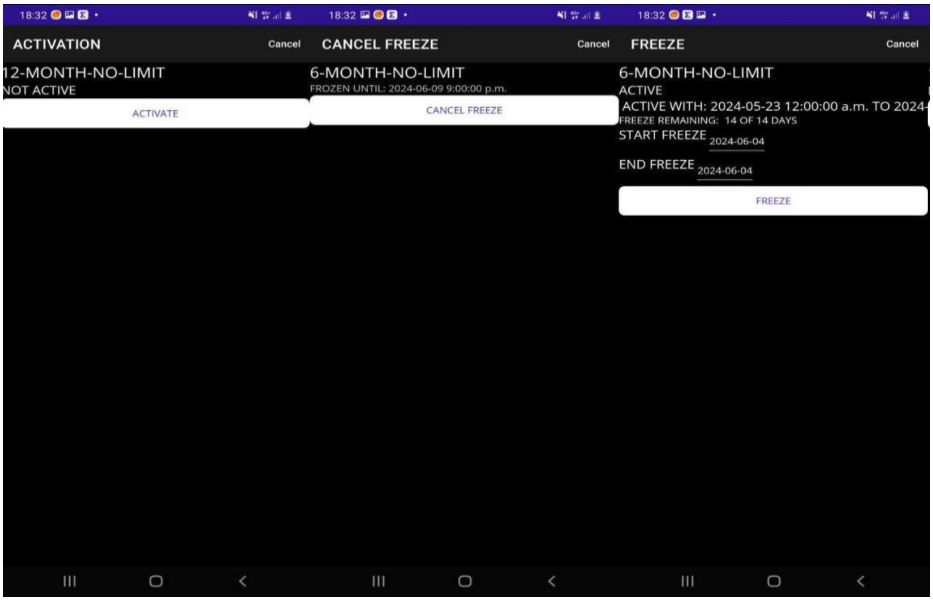

Рисунок 4.4 – Управление абонементами

Для покупки абонемента пользователю необходимо перейти в магазин, выбрать и оплатить подходящий абонемент(см. рисунок 4.5).

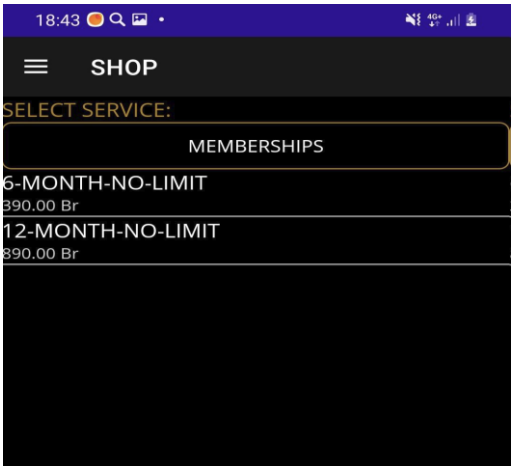

Рисунок 4.5 – Каталог абонемента

Оплата производится через YooMoney с использованием кошелька или банковской карты(см. рисунок 4.6).

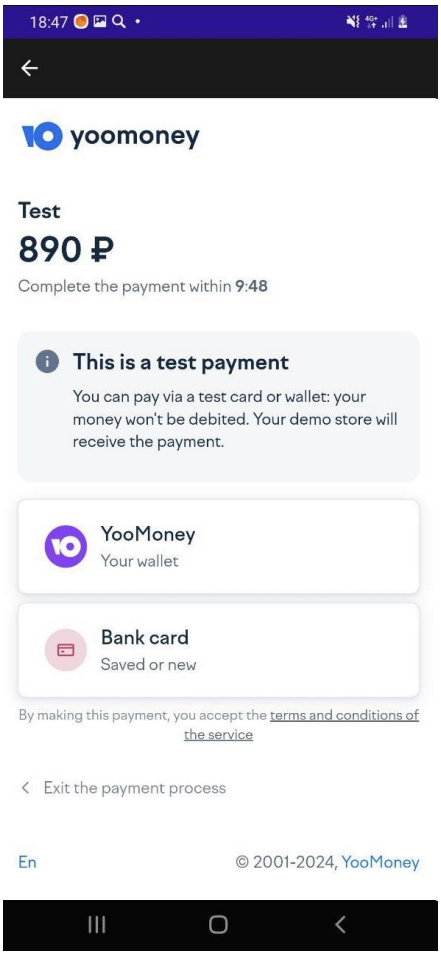

Рисунок 4.6 – Оплата абонемента

После успешной оплаты абонемент появится в личном кабинете.

Также пользователь может выбрать тренера из списка тренеров зала(см. рисунок 4.7) и записаться на тренировку.

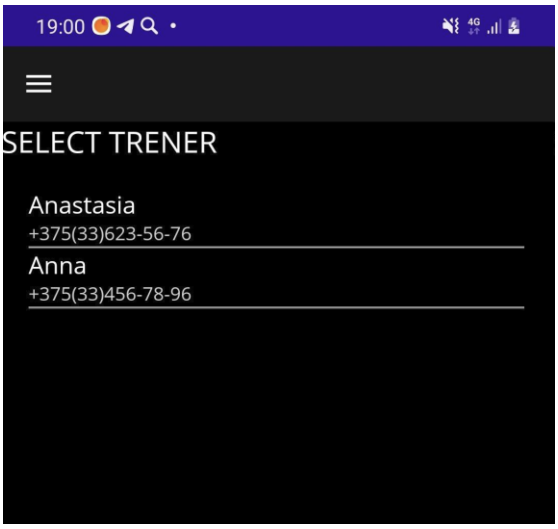

Рисунок 4.7 – Список тренеров

Для записи нужно выбрать дату и время.

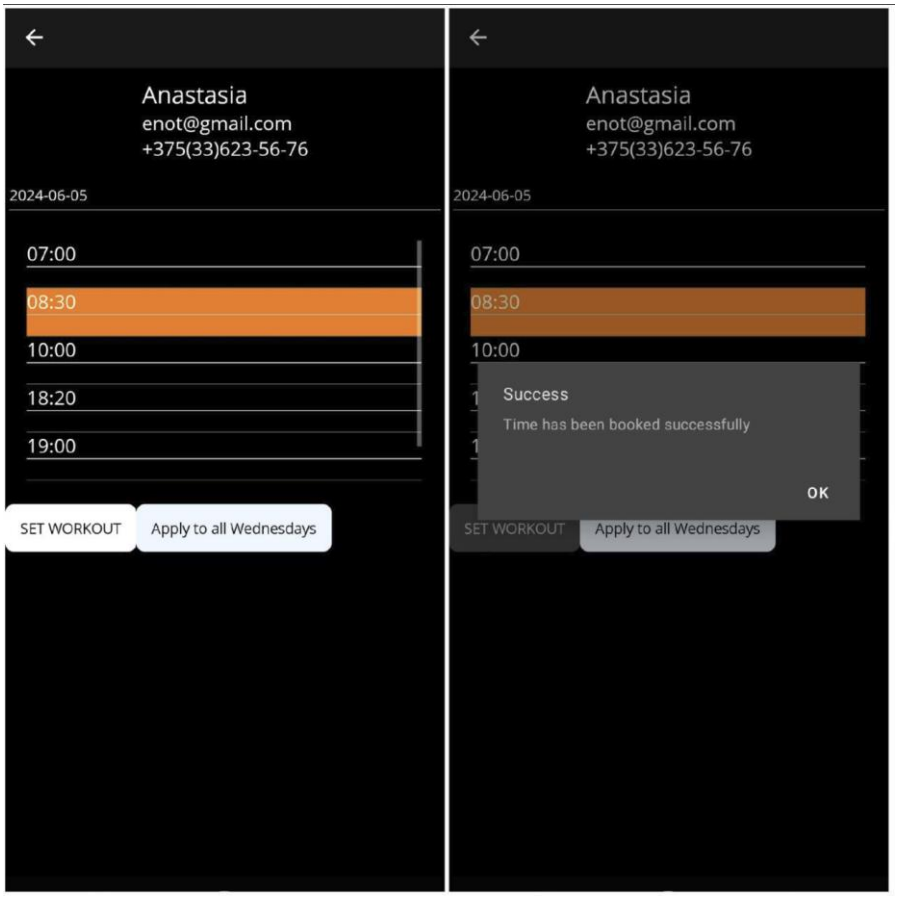

Рисунок 4.8 – Запись на тренировку

Если запись прошла успешно, запланированная тренировка появится в личном кабинете (см. рисунок 4.9).

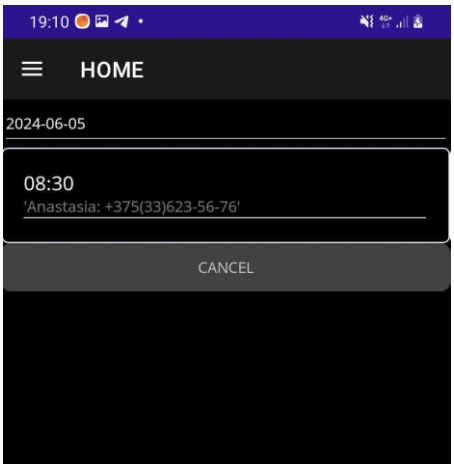

Рисунок 4.9 – Вкладка с запланированными тренировками

В приложение также может войти администратор, который управляет списками тренеров, клиентов, абонементов (см. рисунок 4.10).

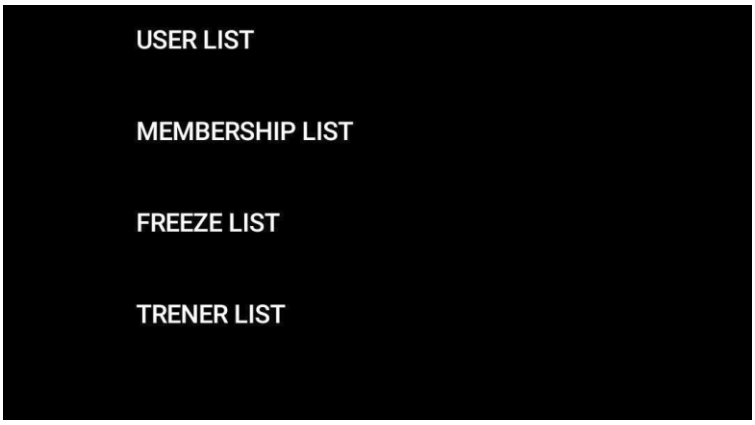

Рисунок 4.10 – Списки для администратора

Администратор может добавлять, удалять, редактировать данные из списков, а также осуществлять поиск по почте для пользователей или названию для абонементов и заморозок.

Например, можно изменить название, стоимость, количество дней и тип заморозки для выбранного абонемента (см. рисунок 4.11).

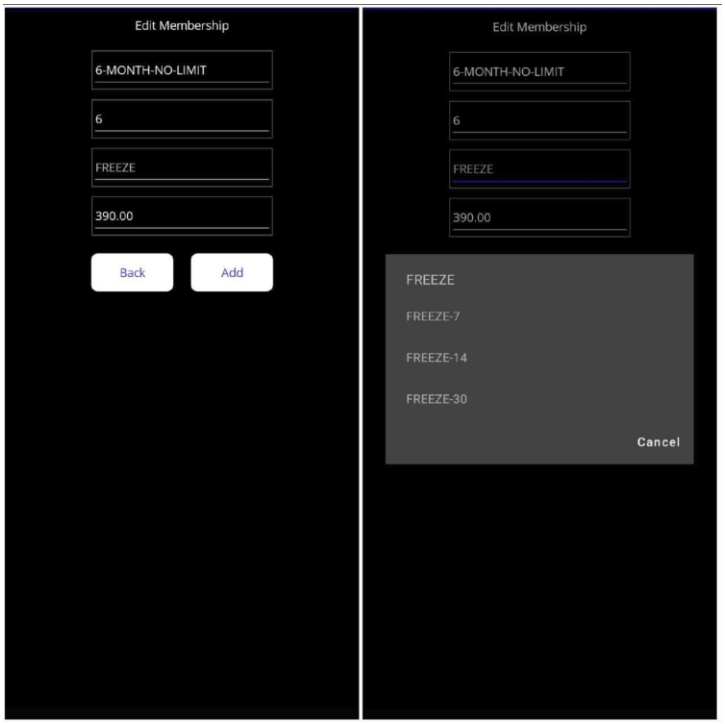

Рисунок 4.11 – Редактирование абонемента

Тренер в приложении может управлять временными слотами для тренировки, добавляя или копируя их на определенную дату, удаляя, просматривая информацию о записанных клиентах (см. рисунок 4.12).

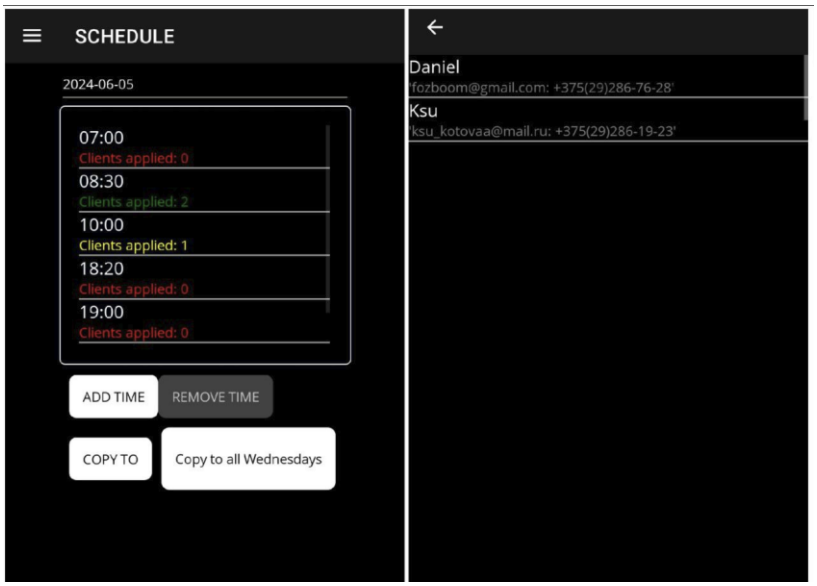

Рисунок 4.12 – Расписание у тренера

В идеале, на одно время должны быть записаны два клиента, такой случай подсвечивается зеленым цветом. Случаи, где записался только один человек или ни одного, выделяются желтым и красным цветами соответственно.

ЗАКЛЮЧЕНИЕ

В данной курсовой работе было разработано мобильное приложение для сети спортзалов на языке программирования C# с использованием концепций объектно-ориентированного программирования. Были применены принципы ООП, а также принципы SOLID и паттерны программирования. Итоговый продукт удовлетворяет не только техническим, но и функциональным требованиям. Благодаря соблюдению принципов ООП и SOLID приложение является легко масштабируемым без необходимости переписывания и усложнения кода.

Был разработан интуитивно понятный интерфейс, который мало зависит от бизнес-логики приложения, что придает продукту гибкость и возможность расширения. Это позволяет легко вносить изменения и адаптировать приложение под различные потребности пользователей.

Также был использован шаблон MVVM, что значительно облегчило процесс разработки и сделал код более структурированным.

Таким образом, данная курсовая работа демонстрирует возможности и преимущества использования принципов ООП и SOLID при разработке мобильного приложения для управления тренировками и абонементами в спортзалах.

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

[1] Документация по языку C# [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/en-us/dotnet/csharp/ – Дата доступа: 30.05.2024

[2] Ссылка на github репозиторий [Электронный ресурс]. – Режим доступа: https://github.com/kseniakot/Gym-app – Дата доступа: 30.05.2024

[3] Паттерн MVVM [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/en-us/dotnet/architecture/maui/mvvm – Дата доступа: 30.05.2024.

[4] Документация ASP .NET Core [Электронный ресурс]. – Режим доступа: https://dotnet.microsoft.com/en-us/apps/aspnet – Дата доступа: 01.06.2024.

[5] Документация Entity Framework Core [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/en-us/ef/ – Дата доступа: 01.06.2024.

[6] Документация .NET MAUI [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/en-us/dotnet/maui/what-is-maui?view=net-maui-8.0 – Дата доступа: 01.06.2024.

ПРИЛОЖЕНИЕ А
(обязательное)
Диаграмма классов

ПРИЛОЖЕНИЕ Б
(обязательное)
Исходный код программы

Файл ActivateMembershipView.xaml.cs:

```
using Gym.Model;
using Gym.ViewModel;
namespace Gym.View;
[QueryProperty(nameof(MembershipId), "MembershipId")]
public partial class ActivateMembershipView : ContentPage {
  ActivateMembershipViewModel _activateMembershipViewModel;
  string _membershipId;
  public                    ActivateMembershipView(ActivateMembershipViewModel
activateMembershipViewModel) {
    _activateMembershipViewModel = activateMembershipViewModel;
            InitializeComponent();
    BindingContext = _activateMembershipViewModel; }
  public string MembershipId {
    set {
      _membershipId = value;
      _activateMembershipViewModel.MembershipId = int.Parse(_membershipId);
      OnPropertyChanged(); }
    get => _membershipId; } }
```

Файл ActivateMembershipViewModel.cs:

```
using CommunityToolkit.Mvvm.ComponentModel;
using CommunityToolkit.Mvvm.Input;
using Gym.Model;
using Gym.Services;
using Gym.Exceptions;
using System.Diagnostics;
namespace Gym.ViewModel;
public partial class ActivateMembershipViewModel : ObservableObject {
  [ObservableProperty]
  private MembershipInstance? _membership;
  private int _membershipId;
```

```
  readonly WebService webService;
  public ActivateMembershipViewModel(WebService webService) {
    this.webService = webService;
    //  InitializeAsync(); }
  public int MembershipId {
    get { return _membershipId; }
    set {
      _membershipId = value;
      OnPropertyChanged(nameof(MembershipId));
      // Load the Membership when the MembershipId is set
      LoadMembership(); } }
  private async Task LoadMembership() {
    try {
      Membership = await webService.GetMembershipInstanceById(MembershipId); }
    catch (SessionExpiredException) {
      await Shell.Current.DisplayAlert("Session Expired", "Your session has
expired. Please sign in again.", "Ok");
      await Shell.Current.GoToAsync("SignInView");
      Application.Current.MainPage = new AppShell(); }
    catch (Exception e) {
      await Shell.Current.DisplayAlert("Error", e.Message, "Ok"); } }
  [RelayCommand]
  private async Task CancelAsync() {
    await Shell.Current.GoToAsync("//ProfileView"); }
  [RelayCommand]
  private async Task ActivateAsync() {
    try {
      if          (!(await          webService.DoesActiveMembershipExist((await
webService.GetUserFromToken()).Id))) {
        await webService.ActivateMembershipInstance(MembershipId);
        await  Shell.Current.DisplayAlert("Success",  "Membership  activated",
"Ok"); }
      else {
        await  Shell.Current.DisplayAlert("Warning",  "You  already  have  one
ACTIVE membership", "Ok");
```

```
        bool answer = await Shell.Current.DisplayAlert("Question", "Activate anyway?", "Yes", "No");
        if (answer) {
          await webService.ActivateMembershipInstance(MembershipId);
          await Shell.Current.DisplayAlert("Success", "Membership activated", "Ok"); } }
    await Shell.Current.GoToAsync("//ProfileView"); }
    catch (SessionExpiredException) {
      await Shell.Current.DisplayAlert("Session Expired", "Your session has expired. Please sign in again.", "Ok");
      await Shell.Current.GoToAsync("SignInView");
      Application.Current.MainPage = new AppShell(); }
    catch (Exception e) {
      await Shell.Current.DisplayAlert("Error", e.Message, "Ok"); } } }
```

Метод для записи на тренировку в файле WebService.cs

```
    public async Task ApplyWorkout(int trenerId, int memberId, DateTime date)
{
      string dateString = date.ToString("yyyy.MM.dd HH:mm"); ;
      HttpResponseMessage response = await client.PostAsync($"{socket}/treners/{trenerId}/workday/{memberId}?dateString={dateString}", null);
      if (response.IsSuccessStatusCode) {
        return; }
      else if (response.StatusCode == System.Net.HttpStatusCode.Unauthorized)
{
        throw new SessionExpiredException(); }
      else if (response.StatusCode == System.Net.HttpStatusCode.Conflict) {
        throw new Exception("You already have a workout for today"); }
      else {
        throw new Exception(response.StatusCode.ToString()); } }
```

Метод для записи на тренировку в файле WebService.cs

```
public async Task<string> MakePayment(int userId, Membership membership) {
      var order = new Order {
        UserId = userId,
        MembershipId = membership.Id,
```

```
        Amount = new Amount {
          Currency = "RUB",
          Value = membership.Price.ToString()
        },
        Confirmation = new Redirection {
          Type = "redirect",
          Return_url = "https://www.google.com"
        },
        Capture = true,
      };
      var options = new JsonSerializerOptions { PropertyNameCaseInsensitive = true };
      HttpContent content = new StringContent(JsonSerializer.Serialize(order, options), Encoding.UTF8, "application/json");
      HttpResponseMessage response = await client.PostAsync($"{socket}/users/payment?userId={userId}&membershipId={membership.Id}", content);
      if (response.IsSuccessStatusCode) {
        var content_with_payment = await response.Content.ReadAsStringAsync();
        Payment payment = JsonSerializer.Deserialize<Payment>(content_with_payment, options);
        Debug.WriteLine(payment.Confirmation.Confirmation_url);
        return payment.Confirmation.Confirmation_url; }
      else {
        Debug.WriteLine(response.StatusCode.ToString());
        throw new Exception(response.StatusCode.ToString()); } }
```

Методы для работы с оплатой в файле Program.cs

```
app.MapPost("/users/payment",
     async (int userId, int membershipId, Order order, DBContext dbContext) => {
       Console.WriteLine();
       Console.WriteLine(order.Amount.Currency);
       // GET USER
       var user = await dbContext.Users.
       Include(u => u.Orders)
       .FirstOrDefaultAsync(u => u.Id == userId);
```

```
        //GET MEMBERSHIP

        var membership = await dbContext.Memberships.FirstOrDefaultAsync(m =>
m.Id == membershipId);

        // CREATE REQUEST

        var client = new HttpClient();

        var                              byteArray                              =
Encoding.ASCII.GetBytes("385708:test_QjpJqSgi_4o7cPSsSDe667iS9sUBdafzKvBLjmGm
dvU");

        client.DefaultRequestHeaders.Authorization            =            new
System.Net.Http.Headers.AuthenticationHeaderValue("Basic",
Convert.ToBase64String(byteArray));

        //ADD PAYMENT REQUEST TO DB

        membership.Orders.Add(order);

        user.Orders.Add(order);

        //dbContext.Orders.Add(order);

        dbContext.Users.Update(user);

        dbContext.Memberships.Update(membership);

        dbContext.SaveChanges();

        var options = new JsonSerializerOptions { PropertyNamingPolicy =
JsonNamingPolicy.CamelCase };

        string    json    =    System.Text.Json.JsonSerializer.Serialize(order,
options);

      Console.WriteLine();

      Console.WriteLine(json);

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

        //CREATE REQUEST

        content.Headers.ContentType                    =                    new
MediaTypeHeaderValue("application/json");

        var request = new HttpRequestMessage {

          Method = HttpMethod.Post,

          RequestUri = new Uri("https://api.yookassa.ru/v3/payments"),

          Headers = {

         { "Idempotence-Key", DateTime.Now.ToString() },

      },

          Content = content

        };

        // SEND REQUEST
```

```
       var response = await client.SendAsync(request);
       if (response.IsSuccessStatusCode) {
         var content_with_payment = await response.Content.ReadAsStringAsync();
         var options_2 = new JsonSerializerOptions { PropertyNameCaseInsensitive = true };
         Console.WriteLine();
         Console.WriteLine(content_with_payment);
         Payment payment = JsonSerializer.Deserialize<Payment>(content_with_payment, options_2);
         order.Payment = payment;
         dbContext.Orders.Update(order);
         dbContext.SaveChanges();
         return Results.Ok(payment); }
       else {
         Console.WriteLine(response.StatusCode.ToString());
         return Results.Problem("Error occurred while making the payment."); }
     });
//CHECK PAYMENT STATUS
app.MapPost("/users/payment/notification",
     async (Notification notification, DBContext dbContext) => {
       Console.WriteLine();
       Console.WriteLine(notification.Object.Status);
       Console.WriteLine("NOTIFICATION!!!!!!!!");
       if (notification.Object.Status == "succeeded") {
         var paymentInDb = await dbContext.Payments
         .Include(p => p.Order)
           .ThenInclude(o => o.User)
         .Include(p => p.Order)
           .ThenInclude(o => o.Membership)
         .FirstOrDefaultAsync(p => p.Id == notification.Object.Id);
         if (paymentInDb == null) return Results.NotFound(new { message = "No such payment" });
         paymentInDb.Status = notification.Object.Status;
         paymentInDb.Paid = true;
         dbContext.Payments.Update(paymentInDb);
```

```
        var user = paymentInDb.Order.User;
        var membership = paymentInDb.Order.Membership;
        var membershipInstance = new MembershipInstance {
          MembershipId = membership.Id,
          UserId = user.Id,
        };
        if (user is Member member) {
          member?.UserMemberships?.Add(membershipInstance);
          dbContext.Members.Update(member); }
        else {
          Member newMember = new Member(user);
          newMember?.UserMemberships?.Add(membershipInstance);
          dbContext.Users.Remove(user);
          dbContext.Members.Add(newMember); }
        dbContext.SaveChanges(); }
      return Results.Ok();
    });
```

Метод для аутентификации пользователя в файле Webservice.cs

```
    public async Task LogIn(User user) {
      HttpContent content = new StringContent(JsonSerializer.Serialize(user),
Encoding.UTF8, "application/json");
       //         HttpResponseMessage         response         =         await
client.PostAsync("https://localhost:7062/login", content);
       HttpResponseMessage            response            =            await
client.PostAsync($"{socket}/login", content);
      if (response.IsSuccessStatusCode) {
        await                                tokenService.SaveTokenAsync((await
response.Content.ReadAsStringAsync()).Trim('"'));
        client.DefaultRequestHeaders.Authorization           =           new
AuthenticationHeaderValue("Bearer", await tokenService.GetTokenAsync()); }
      else if (response.StatusCode == System.Net.HttpStatusCode.Unauthorized)
{
        throw new Exception("Invalid email or password"); }
      else if(response.StatusCode == System.Net.HttpStatusCode.Forbidden) {
        throw new Exception("Banned email"); }
      else {
```

```
        throw new Exception(response.StatusCode.ToString()); } }
    //DECRYPT TOKEN AND GET USER INFO
    public async Task<User> GetUserFromToken() {
      var stream = await tokenService.GetTokenAsync();
      var handler = new JwtSecurityTokenHandler();
      var jsonToken = handler.ReadToken(stream);
      var tokenS = jsonToken as JwtSecurityToken;
      var user = new User() {
        Email = tokenS.Claims.First(claim => claim.Type == ClaimTypes.Email).Value,
        Id = int.Parse(tokenS.Claims.First(claim => claim.Type == ClaimTypes.NameIdentifier).Value),
        //Password = tokenS.Claims.First(claim => claim.Type == "Password").Value,
        Name = tokenS.Claims.First(claim => claim.Type == ClaimTypes.Name).Value,
        PhoneNumber = tokenS.Claims.First(claim => claim.Type == ClaimTypes.MobilePhone).Value,
        IsBanned = bool.Parse(tokenS.Claims.First(claim => claim.Type == "IsBanned").Value),
        Orders = new List<Order>()
      };
      //Debug.WriteLine("User from token: " + user.Email);
      return user; }
    //REMOVE USER
    public async Task RemoveUser(int id) {
      //         HttpResponseMessage response = await client.DeleteAsync($"https://localhost:7062/users/{id}");
      HttpResponseMessage response = await client.DeleteAsync($"{socket}/users/{id}");
      if (response.StatusCode == System.Net.HttpStatusCode.Unauthorized) {
        throw new SessionExpiredException(); }
      else if (!response.IsSuccessStatusCode) {
        throw new Exception("Something went wrong"); } }
```

Файл UserMainPage.xaml.cs:

```
using Gym.ViewModel;
```

```
namespace Gym.View;
public partial class UserMainPage : ContentPage {
  UserMainViewModel _userMainViewModel;
      public UserMainPage(UserMainViewModel userMainViewModel) {
            InitializeComponent();
            BindingContext = userMainViewModel;
    _userMainViewModel = userMainViewModel; }
  protected override void OnAppearing() {
    base.OnAppearing();
    // Assuming you have a ViewModel property in your Page class
    _userMainViewModel.LoadDataCommand.Execute(null); } }
```

Файл UserMainViewModel.cs:

```
using CommunityToolkit.Mvvm.ComponentModel;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Gym.Services;
using Gym.Model;
using CommunityToolkit.Mvvm.Input;
using System.Collections.ObjectModel;
namespace Gym.ViewModel {
  public partial class UserMainViewModel : ObservableObject {
    readonly WebService webService;
    [ObservableProperty]
    private User _user;
    [ObservableProperty]
    private DateTime _selectedDate = DateTime.UtcNow;
    [ObservableProperty]
    bool isButtonEnabled = false;
    //[ObservableProperty]
    //private WorkHour selectedWorkout;
```

```
    private WorkHour _selectedWorkout;
    public WorkHour SelectedWorkout {
      get { return _selectedWorkout; }
      set {
        if (_selectedWorkout != value) {
          _selectedWorkout = value;
          OnPropertyChanged();
          IsButtonEnabled = true; } } }
    [ObservableProperty]
    private ObservableCollection<WorkHour> workouts;
    public UserMainViewModel(WebService webService) {
      this.webService = webService; }
    private async Task InitializeAsync() {
      User = await webService.GetUserFromToken(); }
    [RelayCommand]
    private async Task LoadData() {
      try {
        await InitializeAsync();
        Workouts = new ObservableCollection<WorkHour>(await
webService.GetUserWorkouts(User.Id, SelectedDate)); }
      catch (Exception ex) {
        Workouts = new ObservableCollection<WorkHour>(); } }
    [RelayCommand]
    public async Task DateSelectedAsync() {
      try {
        Workouts = new ObservableCollection<WorkHour>(await
webService.GetUserWorkouts(User.Id, SelectedDate)); }
      catch {
        Workouts = []; } }
    [RelayCommand]
    public async Task CancelWorkoutAsync() {
      if(SelectedWorkout == null) {
        await Shell.Current.DisplayAlert("Error", "Please select a workout to
cancel", "Ok");
        return; }
```

```
      try {
        await webService.RemoveWorkHourClient(SelectedWorkout.Id, User.Id);
        Workouts.Remove(SelectedWorkout);
        SelectedWorkout = null;
        isButtonEnabled = false; }
      catch(Exception e) {
        await Shell.Current.DisplayAlert("Error", e.Message, "Ok"); } } } }
```

Файл UserShellViewModel.cs:

```
using CommunityToolkit.Mvvm.ComponentModel;
using CommunityToolkit.Mvvm.Input;
//using CoreBluetooth;
using Gym.Services;
using System.Diagnostics;
namespace Gym.ViewModel;
public partial class UserShellViewModel : ObservableObject {
      [ObservableProperty]
      private bool _isMember;
  private readonly WebService _webService;
  public UserShellViewModel(WebService webService) {
    _webService = webService;
     SetIsMember(); }
  private async void SetIsMember() {
    if (await _webService.CheckUserOrMember((await
_webService.GetUserFromToken()).Email)) {
      IsMember = true; }
    else {
      IsMember = false; } } }
```

Файл EditMembershipView.xaml.cs:

```
using Gym.ViewModel;
using System.Diagnostics;
namespace Gym.View;
[QueryProperty(nameof(MembershipId), "MembershipId")]
public partial class EditMembershipView : ContentPage {
```

```
  EditMembershipViewModel _editMembershipViewModel;
  string _membershipId;
  public string MembershipId {
    set {
      _membershipId = value;
      _editMembershipViewModel.MembershipId = int.Parse(_membershipId);
      OnPropertyChanged(); }
    get => _membershipId; }
  public EditMembershipView(EditMembershipViewModel editMembershipViewModel) {
    _editMembershipViewModel = editMembershipViewModel;
    InitializeComponent();
    BindingContext = _editMembershipViewModel;
    //Debug.WriteLine("EditMembershipView");
    //Debug.WriteLine(MembershipId); } }
```

Файл EditMembershipViewModel.cs:

```
using Gym.Model;
using CommunityToolkit.Mvvm.ComponentModel;
using CommunityToolkit.Mvvm.Input;
using Gym.Services;
using System.Diagnostics;
using Gym.Exceptions;
using System.Collections.ObjectModel;
namespace Gym.ViewModel;
public partial class EditMembershipViewModel : ObservableObject {
  [ObservableProperty]
  private Membership? _membership;
  [ObservableProperty]
  private ObservableCollection<Freeze> _freezes;
  // [ObservableProperty]
  private Freeze _selectedFreeze;
  public Freeze SelectedFreeze {
    get { return _selectedFreeze; }
    set {
```

```
      _selectedFreeze = value;
      //Membership.Freeze = value;
      Membership.FreezeId = value.Id;
      //Debug.WriteLine(Membership.Freeze.Id); } }
  private int _membershipId;
  readonly WebService webService;
  public EditMembershipViewModel(WebService webService) {
     this.webService = webService;
    InitializeAsync(); }
  private async Task InitializeAsync() {
    try {
      Freezes = new ObservableCollection<Freeze>(await
webService.GetAllFreezes());
      //Freezes.Insert(0, new Freeze { Id = 0, Name = "None" }); }
    catch (SessionExpiredException) {
      await Shell.Current.DisplayAlert("Session Expired", "Your session has
expired. Please sign in again.", "Ok");
      await Shell.Current.GoToAsync("SignInView");
      Application.Current.MainPage = new AppShell(); }
    catch (Exception e) {
      await Shell.Current.DisplayAlert("Error", e.Message, "Ok"); } }
  public int MembershipId {
    get { return _membershipId; }
    set {
      _membershipId = value;
      OnPropertyChanged(nameof(MembershipId));
      // Load the Membership when the MembershipId is set
      LoadMembership(); } }
  private async Task LoadMembership() {
    try {
      Membership = await webService.GetMembershipById(MembershipId); }
    catch (SessionExpiredException) {
      await Shell.Current.DisplayAlert("Session Expired", "Your session has
expired. Please sign in again.", "Ok");
      await Shell.Current.GoToAsync("SignInView");
```

```
      Application.Current.MainPage = new AppShell(); }
    catch (Exception e) {
      await Shell.Current.DisplayAlert("Error", e.Message, "Ok"); } }
  private static bool IsAnyNullOrEmpty(object membership) {
    var propertiesToCheck = new[] { "Name", "Months", "Price" };
    return membership.GetType()
      .GetProperties()
      .Where(pt => propertiesToCheck.Contains(pt.Name) && (pt.PropertyType == typeof(string) || pt.PropertyType == typeof(int?) || pt.PropertyType == typeof(decimal?)))
      .Select(v => v.GetValue(membership))
      .Any(value => value == null || string.IsNullOrWhiteSpace(value.ToString())); }
  private async Task<bool> DoesMembershipExist(Membership membership) {
    try {
      return await webService.DoesMembershipExistAsync(membership); }
    catch (SessionExpiredException) {
      await Shell.Current.DisplayAlert("Session Expired", "Your session has expired. Please sign in again.", "Ok");
      await Shell.Current.GoToAsync("SignInView");
      Application.Current.MainPage = new AppShell();
      return false; }
    catch (Exception) {
      await Shell.Current.DisplayAlert("Something went wrong", "Please try again", "Ok");
      return false; } }
  [RelayCommand]
  private static async Task BackAsync() {
    await Shell.Current.Navigation.PopAsync(); }
  [RelayCommand]
  private async Task EditAsync() {
    if (IsAnyNullOrEmpty(Membership)) {
      Membership = new();
      await Shell.Current.DisplayAlert("There is an empty field", "Please fill it out and try again.", "Ok"); }
    else if (await DoesMembershipExist(Membership)) {
       // Membership = new();
```

```
      await Shell.Current.DisplayAlert("Such membership already exists", "Make
some changes or go back", "Ok"); }

    else {

      try {

        await webService.EditMembership(Membership);

        await  Shell.Current.DisplayAlert("Memebrship  edited  successfully",
"Press Ok and return to the membership list", "Ok");

        BackCommand.Execute(null); }

      catch (SessionExpiredException) {

        await Shell.Current.DisplayAlert("Session Expired", "Your session has
expired. Please sign in again.", "Ok");

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

        Application.Current.MainPage = new AppShell(); }

      catch (Exception e) {

        await Shell.Current.DisplayAlert("Error", e.Message, "Ok"); } } } }
```