МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ

БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ

ГОСУДАРСТВЕННЫЙ

ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных технологий

Кафедра Программной инженерии

Специальность 6-05-0612-01 «Программная инженерия»

Направление специальности 6-05-0612-01 «Программная инженерия» (программирование интернет-приложений)

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**КУРСОВОГО ПРОЕКТА:**

по дисциплине «Объектно-ориентированные технологии программирования и стандарты проектирования»

Тема Программное средство «Фитнес-клуб»

Исполнитель

студент (ка) 2 курса группы 6 Романов Игорь Вячеславович

(Ф.И.О.)

Руководитель работы ассистент Мущук А.Н.

(учен. степень, звание, должность, подпись, Ф.И.О.)

Курсовой проект защищен с оценкой

Председатель Смелов В.В.

(подпись)

Минск 2025

Оглавление

[ВВЕДЕНИЕ 6](#_Toc198682133)

[1 Аналитический обзор литературы 7](#_Toc198682134)

[1.1 Анализ прототипов 7](#_Toc198682135)

[1.1.1 Интернет-ресурс «iron-fitness.by» 7](#_Toc198682136)

[1.1.2 Интернет-ресурс «olympia-gs.by» 8](#_Toc198682137)

[1.1.3 Интернет-ресурс «adrenalin-fitness.by» 9](#_Toc198682138)

[1.2 Постановка задачи 10](#_Toc198682139)

[1.3. Вывод 10](#_Toc198682140)

[2 Анализ требований к программному средству и разработка функциональных требований 11](#_Toc198682141)

[2.1 Описание средств разработки 11](#_Toc198682142)

[2.1.1 Microsoft Visual Studio 2022 11](#_Toc198682143)

[2.1.2 Программная платформа .NET Framework 11](#_Toc198682144)

[2.1.3. Язык программирования C# 11](#_Toc198682145)

[2.1.4 Технология WPF 11](#_Toc198682146)

[2.1.5 Технология Entity Framework 12](#_Toc198682147)

[2.1.6 MS SQL Server 12](#_Toc198682148)

[2.1.7 Архитектурный паттерн MVVM 12](#_Toc198682149)

[2.2 Описание функциональности программного средства 12](#_Toc198682150)

[2.3 Спецификация функциональных требований 13](#_Toc198682151)

[3 Проектирование программного средства 14](#_Toc198682152)

[3.1 Архитектура приложения 14](#_Toc198682153)

[3.2 Структура базы данных 15](#_Toc198682154)

[3.3 Структура проекта 17](#_Toc198682155)

[3.4 Диаграмма классов 18](#_Toc198682156)

[4 Реализация программного средства 19](#_Toc198682157)

[4.1 Основные классы программного средства 19](#_Toc198682158)

[4.1.1 Классы для реализации паттерна MVVM 19](#_Toc198682159)

[4.1.2 Классы для работы с базой данных 21](#_Toc198682160)

[4.2 Реализация функционала приложения 22](#_Toc198682161)

[4.2.1 Метод для удаления плана тренировок и питания тренером 22](#_Toc198682162)

[4.2.2 Метод для регистрации пользователя в приложении 22](#_Toc198682163)

[4.2.3 Метод для добавления нового абонемента 23](#_Toc198682164)

[4.2.4 Метод для удаления отзыва пользователем 23](#_Toc198682165)

[5 Тестирование, проверка работоспособности приложения и анализ результатов 24](#_Toc198682166)

[5.1 Тестирование функций страницы регистрации 24](#_Toc198682167)

[5.2 Тестирование функций страницы авторизации 25](#_Toc198682168)

[5.3 Тестирование функций редактирования профиля 26](#_Toc198682169)

[5.4. Тестирование добавления абонементов 27](#_Toc198682170)

[5.5. Тестирование добавления отзывов 28](#_Toc198682171)

[5.6. Тестирование управление пользователями 29](#_Toc198682172)

[6 Руководство по использованию 30](#_Toc198682173)

[6.1 Регистрация и авторизация 30](#_Toc198682174)

[6.2 Использование приложения в роли клиента 31](#_Toc198682175)

[6.3 Использование приложения в роли тренера 32](#_Toc198682176)

[6.4 Использование приложения в роли администратора 33](#_Toc198682177)

[ЗАКЛЮЧЕНИЕ 35](#_Toc198682178)

[Список использованных источников 36](#_Toc198682179)

[ПРИЛОЖЕНИЕ А 37](#_Toc198682180)

[ПРИЛОЖЕНИЕ Б 38](#_Toc198682181)

[ПРИЛОЖЕНИЕ В 39](#_Toc198682182)

[ПРИЛОЖЕНИЕ Г 43](#_Toc198682183)

[ПРИЛОЖЕНИЕ Д 45](#_Toc198682184)

[ПРИЛОЖЕНИЕ Е 47](#_Toc198682185)

# **ВВЕДЕНИЕ**

В современном мире, с ростом интереса к здоровому образу жизни и физической активности, услуги фитнес-клубов становятся всё более востребованными. Всё больше людей выбирают тренировки в специализированных клубах, где им доступны разнообразные программы, профессиональное сопровождение и современное оборудование. Такой подход позволяет не только повысить эффективность тренировок, но и сделать процесс занятий более комфортным и безопасным.

Цель данного курсового проекта – разработка программного средства, обеспечивающего доступ к информации о фитнес-клубе. С его помощью пользователи смогут ознакомиться с общей информацией о клубе, его услугах и особенностях. Это позволит клиентам получать всю необходимую информацию и планировать свои посещения с максимальным удобством.

Программное средство предоставит возможность легко просматривать расписание тренировок, узнавать дни и время их проведения, а также выбирать подходящий вариант в зависимости от личных предпочтений и уровня подготовки. Кроме того, будет реализована функция онлайн-записи, позволяющая пользователям регистрироваться на тренировки в режиме реального времени без необходимости посещения клуба. Это не только упростит процесс записи, но и сократит время ожидания, повысив удобство взаимодействия с фитнес-центром.

Для успешного выполнения проекта необходимо:

* провести анализ тематической литературы;
* изучить существующие аналоги;
* определить функциональные требования;
* разработать структуру базы данных и проекта;
* реализовать и протестировать программное средство;
* подготовить руководство пользователя.

Главная задача данного курсового проектирования – это разработка программного средства, которое реализует все вышеперечисленные функции и решает поставленные задачи. При разработке будут применяться принципы объектно-ориентированного программирования (ООП), а также использоваться база данных MS SQL Server и технология Windows Presentation Foundation (WPF).

1 Аналитический обзор литературы

1.1 Анализ прототипов

Проведение анализа прототипов фитнес-клуба играет ключевую роль в его развитии и успешном позиционировании на рынке. Оно позволяет понять сильные и слабые стороны других клубов, выявить их ценовую политику, качество услуг и уровень сервиса. Это даёт возможность не только учесть лучшие практики, но и избежать распространённых ошибок. В ходе выполнения курсового проекта были проанализированы цели и задачи, а также рассмотрены аналогичные примеры их решений. На основании анализа всех достоинств и недостатков данных альтернативных решений были сформулированы требования к данному программному средству.

1.1.1 Интернет-ресурс «iron-fitness.by»

Iron Fitness [1]представляет собой фитнес-клуб премиального класса. Их веб-сайт предоставляет разнообразный функционал, который помогает пользователям ознакомиться с услугами фитнес-клубов, а также приобрести абонемент или записаться на тренировку. Интерфейс сайта представлен на рисунке 1.1.



Рисунок 1.1 – Главная страница сайта Iron Fitness

Преимущества сайта:

* удобно оформленные карточки цен на предоставляемые услуги;
* указаны цены на разное количество посещений и на различный период времени;
* присутствует онлайн-запись на пробные занятия, на тренировки;
* расписание тренировок по времени;

Недостатки сайта:

* не указана цена vip-карты – требуется связываться с администраторами для заказа, что неудобно для клиентов;
* в расписании тренировок не указаны ФИО тренера, ответственного за услугу; место проведения и продолжительность тренировки;
* отсутствует фильтрация занятий по типу занятий и по вышеперечисленными недостаткам;
* при заполнении формы на пробную тренировку не хватает данных для регистрации клиента.

1.1.2 Интернет-ресурс «olympia-gs.by»

Olympia Gym & Spa [2]— это премиальный фитнес-клуб, предлагающий широкий спектр услуг для занятий спортом и отдыха. Его сайт создан для любителей фитнеса и тех, кто ценит комфортный отдых. Платформа обладает удобным интерфейсом и богатым функционалом, позволяя пользователям легко найти и выбрать подходящую тренировку, ознакомиться с тренерским составом и приобрести абонемент на посещение клуба. Визуальное представление интерфейса сайта показано на рисунке 1.2.

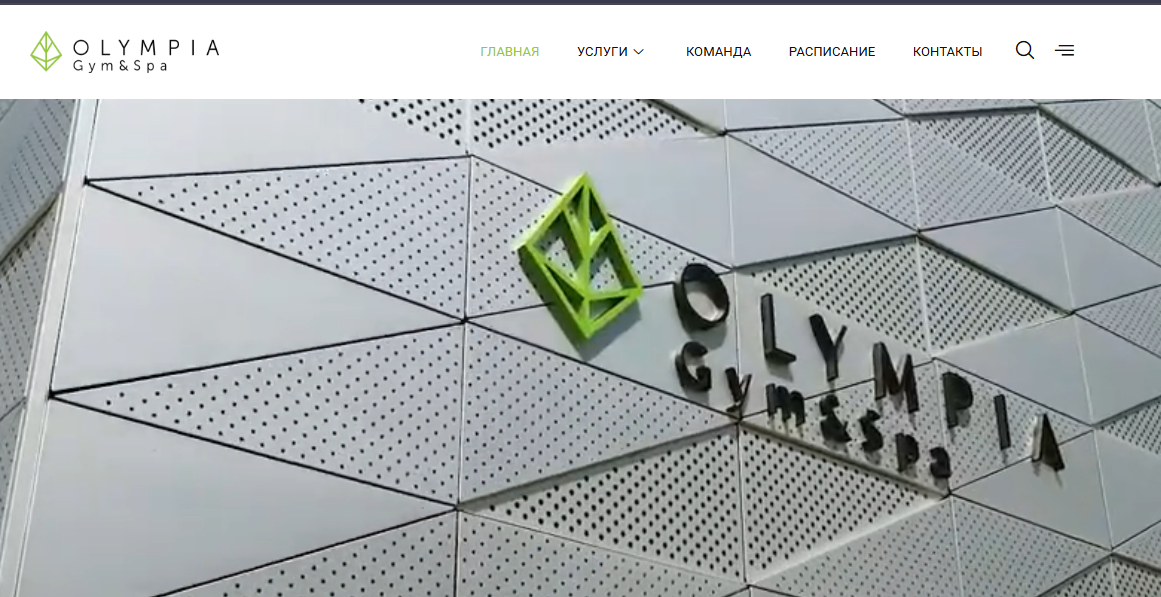


Рисунок 1.2 – Главная страница сайта Olymia Gym&Spa

Преимущества сайта:

* удобно оформленные карточки цен на предоставляемые услуги;
* указаны цены на разное количество посещений и на различный период времени;
* присутствует онлайн-запись на пробные занятия, на тренировки
* расписание тренировок по времени;
* фильтрация занятий по различным критериям.

Недостатки сайта:

* веб-сайт предполагает внутренний поиск по страницам, однако данная функция не работает;
* отсутствуют цены на большинство услуг – необходимо обращаться к администратору;
* при заполнении формы на пробную тренировку не хватает данных для регистрации клиента.

1.1.3 Интернет-ресурс «adrenalin-fitness.by»

«Адреналин» [3] является популярной сетью фитнес-клубов по всей Беларуси. Подобно ранее рассмотренным аналогам, он ориентирован на продажу абонементов и предоставляет пользователям возможность записаться на тренировку. Визуальное отображение интерфейса сайта представлено на рисунке 1.3.

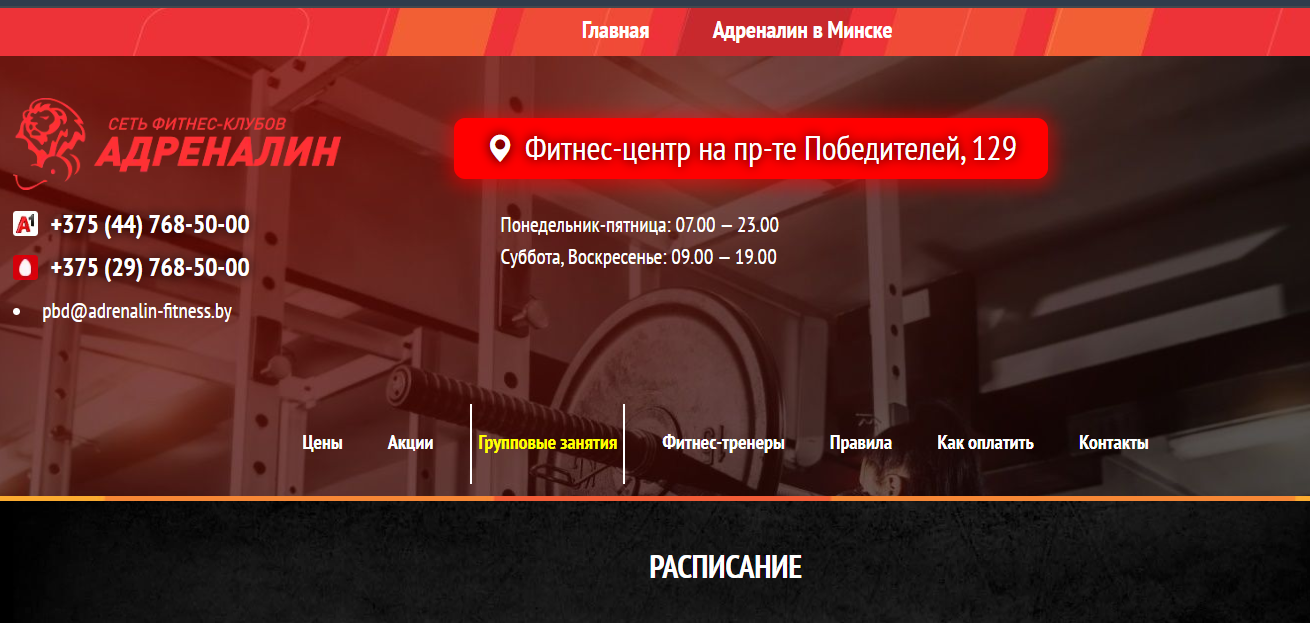


Рисунок 1.3 – Главная страница сайта Адреналин

Преимущества сайта:

* удобно оформленные карточки цен на предоставляемые услуги;
* указаны цены на разное количество посещений и на различный период времени;
* присутствует онлайн-запись на пробные занятия, на тренировки;
* расписание тренировок по времени;
* фильтрация занятий по различным критериям.

Недостатки сайта:

* отсутствуют цены на большинство услуг – необходимо обращаться к администратору;
* при заполнении формы на пробную тренировку не хватает данных для регистрации клиента.

1.2 Постановка задачи

Обзор аналогичных решений позволяет проанализировать все преимущества и недостатки альтернативных возможностей и позволяет сформировать список требований, предъявляемых к программному средству, разрабатываемому в данном курсовом проекте.

Программное средство должно обеспечивать возможность выполнения перечисленных ниже функций:

Функции клиента:

* регистрация и авторизация;
* просмотр расписания тренировок;
* фильтрация и поиск тренировок;
* осуществление записи на выбранную тренировку;
* возможность отмены тренировки в личном кабинете;
* добавление отзывов.

Функция тренера:

* управление тренировками клиентов;
* составление планов тренировок и питания;
* просмотр активных и отмененных записей.

Функции администратора:

* редактирование расписания с добавлением и удалением новых тренировок;
* просмотр данных о зарегистрированных пользователях.

1.3. Вывод

На основе рассмотренного функционала было сформулировано техническое задание для разработки собственного программного средства. Оно обеспечит пользователей удобным доступом к информации о фитнес-клубе, его услугах, расписании тренировок, актуальных акциях и специальных предложениях. Одним из ключевых элементов станет возможность онлайн-записи, что значительно упростит процесс взаимодействия клиентов с клубом и повысит их удовлетворённость сервисом.

Таким образом, разрабатываемое программное средство улучшить пользовательский опыт и повысить эффективность управления фитнес-клубом, делая его более доступным и комфортным для клиентов.

2 Анализ требований к программному средству и разработка функциональных требований

2.1 Описание средств разработки

В ходе реализации приложения были задействованы следующие средства:

* интегрированная среда разработки Microsoft Visual Studio 2022;
* программная платформа .NET Framework версии 4.8;
* язык программирования C#;
* технология WPF;
* технология ORM Entity Framework 6.5.1;
* система управления базами данных Microsoft SQL Server 2022.

2.1.1 Microsoft Visual Studio 2022

Microsoft Visual Studio 2022 — это современная интегрированная среда разработки (IDE), разработанная корпорацией Microsoft. Она предоставляет разработчику весь необходимый набор инструментов для создания приложений на платформе WPF, обеспечивая эффективную работу с кодом, интерфейсом и отладкой.

2.1.2 Программная платформа .NET Framework

.NET Framework — это программная платформа, выпущенная Microsoft в 2002 году. Она предназначена для разработки и выполнения как Windows-приложений, так и веб-служб. Платформа обеспечивает объектно-ориентированную среду выполнения, основанную на CLR (Common Language Runtime), поддерживающую многозадачность и взаимодействие между языками программирования на уровне байт-кода.

2.1.3. Язык программирования C#

В рамках проекта используется язык программирования C# — основной язык разработки в .NET-среде. Этот язык поддерживает объектно-ориентированную парадигму, обладает строгой статической типизацией и широким набором возможностей, таких как обработка событий, свойства, исключения, перегрузка операторов и работа с указателями, что делает его удобным для создания WPF-приложений.

2.1.4 Технология WPF

WPF (Windows Presentation Foundation) применяется для создания графического пользовательского интерфейса. Она позволяет отделить визуальную составляющую приложения от бизнес-логики, используя декларативный подход к проектированию интерфейсов через язык разметки XAML, что делает разработку более гибкой и модульной.

XAML (eXtensible Application Markup Language) — это язык разметки, основанный на XML, применяемый в WPF для описания интерфейса пользователя. С помощью XAML можно задавать элементы управления, макеты, стили, анимации и другие визуальные компоненты. Он обеспечивает удобную связку между представлением и логикой, облегчая процесс проектирования интерфейсов.

2.1.5 Технология Entity Framework

Для работы с базами данных в проекте используется технология Entity Framework Core — современная версия ORM-решения от Microsoft с открытым исходным кодом. Она упрощает доступ к данным за счёт абстракции от конкретной СУБД и позволяет реализовать подходы Database First, Model First и Code First в зависимости от требований проекта.

2.1.6 MS SQL Server

В качестве системы управления базами данных используется Microsoft SQL Server. Эта реляционная СУБД хранит данные в табличной структуре и позволяет выполнять над ними различные операции с помощью языка SQL. Для работы с данными применяются расширения SQL — Transact-SQL, обеспечивающие расширенные возможности по обработке запросов.

2.1.7 Архитектурный паттерн MVVM

Архитектура MVVM (Model-View-ViewModel) применяется для отделения пользовательского интерфейса от логики приложения. Model реализует бизнес-логику и взаимодействует с хранилищем данных, View отвечает за отображение данных, а ViewModel служит связующим звеном, обеспечивая привязку данных и команд. Такой подход упрощает тестирование, повышает читаемость кода и способствует независимой разработке отдельных компонентов приложения.

2.2 Описание функциональности программного средства

Программное средство должно предоставлять следующие функциональные возможности:

Для администратора:

* управление пользователями;
* управление отзывами.

Для тренера:

* управление тренировками;
* составление планов тренировок и питания.

Для клиента:

* просмотр и фильтр тренировок;
* запись на тренировки;
* просмотр планов тренировок и питания;
* написание отзывов.

2.3 Спецификация функциональных требований

В программном обеспечении WPF\_FitnessClub реализован комплекс функций, обеспечивающих автоматизацию работы фитнес-клуба.

Приложение позволяет пользователям регистрироваться, указывая свои персональные данные. При регистрации осуществляется проверка уникальности email, а также валидация формата введённых данных. Если пользователи могут авторизироваться, вводя логин и пароль.

В системе предусмотрено разграничение прав доступа: администратор может управлять пользователями и отзывами, а также управлять абонементами. Тренеры имеют возможность добавлять, редактировать и удалять абонементы, указывая их название, описание, цену и срок действия; составлять планы тренировок и питания для своих клиенток. При этом осуществляется валидация введённых данных, чтобы избежать ошибок. Клиенты могут самостоятельно оформлять абонементы, выбирая подходящий вариант и подтверждая оплату. Система проверяет наличие активного абонемента у клиента, чтобы избежать дублирования.

Тренеры имеют возможность создавать и редактировать индивидуальные тренировочные и пищевые планы для клиентов, указывая необходимые упражнения, рацион и длительность. Каждый план привязывается к конкретному клиенту, а доступ к этим функциям ограничен только для тренеров. Клиенты, в свою очередь, могут просматривать только свои назначенные планы, что обеспечивает конфиденциальность информации.

Клиенты могут оставлять отзывы об абонементах, указывая оценку и текст комментария. Оставлять отзывы могут только авторизованные пользователи, а система следит за корректностью и длиной введённого текста.

Use Case диаграмма для приложения «CoreX» представлена в приложении А.

3 Проектирование программного средства

3.1 Архитектура приложения

Архитектура программного обеспечения представляет собой совокупность ключевых решений, связанных с построением и организацией программной системы. Она охватывает выбор компонентов, их интерфейсных соглашений, способы взаимодействия между ними, а также их интеграцию в общее решение. Основу архитектуры составляет архитектурный стиль, который задаёт общую концепцию построения системы, включая определение элементов, их связей и взаимодействий.

Для достижения необходимых качественных характеристик проектируемого программного средства применяются проверенные архитектурные шаблоны. В рамках данного проекта выбран паттерн Model-View-ViewModel (MVVM), включающий три ключевых компонента: модель (Model), представление (View) и модель представления (ViewModel).

Использование паттерна MVVM способствует повышению модульности приложения, улучшает читаемость и повторное использование кода, а также упрощает процессы тестирования и сопровождения. За счёт разделения бизнес-логики и пользовательского интерфейса достигается высокая гибкость архитектуры, что облегчает адаптацию к изменяющимся требованиям и упрощает масштабирование программной системы.

На рисунке 3.1 представлена схема архитектурной структуры MVVM.

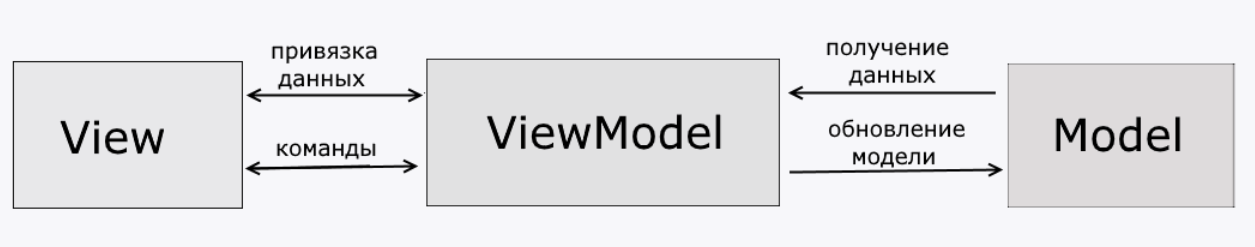


Рисунок 3.1 – Структура шаблона MVVM

Model отвечает за представление и хранение данных, а также может содержать вспомогательную логику, например, проверку корректности данных. Однако данная часть приложения не содержит кода, связанного с отображением или взаимодействием с интерфейсом. Модель реализует интерфейс INotifyPropertyChanged, что позволяет эффективно использовать механизм привязки данных без прямой зависимости от интерфейса.

View реализует визуальную часть приложения и обеспечивает взаимодействие с пользователем. Основное назначение View — отображать данные и визуально реагировать на команды пользователя. При этом логика обработки пользовательских событий реализуется не во View, а делегируется во ViewModel посредством команд.

ViewModel играет роль посредника между View и Model. Она содержит необходимую логику для обработки пользовательских действий, получения и обновления данных, а также реализует свойства и команды, доступные для привязки к элементам интерфейса. Благодаря INotifyPropertyChanged, изменения в модели автоматически отображаются в интерфейсе, обеспечивая двустороннюю привязку данных.

Кроме MVVM, в проекте реализованы дополнительные архитектурные решения – паттерны Repository и Unit of Work, применяемые для работы с базой данных.

Паттерн Repository служит абстракцией над источником данных, отделяя бизнес-логику от механизма доступа к данным. Он позволяет централизовать операции по извлечению и сохранению данных, делая архитектуру более гибкой и удобной для сопровождения. Также применение репозиториев способствует упрощению тестирования — за счёт возможности подмены реальных хранилищ на тестовые реализации.

Паттерн Unit of Work обеспечивает координацию операций с несколькими репозиториями, управляя общим контекстом данных. Он гарантирует целостность транзакций, позволяя применять или отменять все изменения одновременно. Это особенно важно при работе с базами данных, где требуется контроль над целостностью и согласованностью данных.

3.2 Структура базы данных

Для реализации поставленной задачи была создана база данных FitnessClubDB. Для ее создания использовалась система управления реляционными базами данных MS SQL Server. На рисунке 3.2 представлена логическая модель базы данных FitessClubDB.

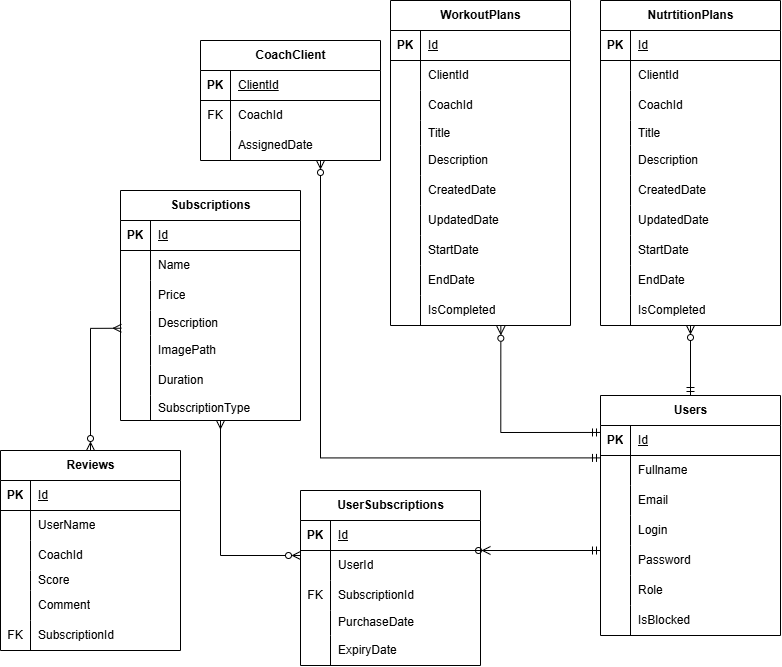


Рисунок 3.2 — Логическая схема базы данных FitessClubDB

Между таблицами существуют связи. Существуют два основных типа связей: один ко многим, а также многие ко многим.

Примером связи один ко многим является связь между таблицами Users и WorkoutPlans по полю ClientId: один пользователь (клиент или тренер) может иметь множество планов тренировок, но каждый план принадлежит только одному пользователю. Аналогично, существует связь между Users и UserSubscriptions — один пользователь может иметь несколько подписок, но каждая подписка привязана к одному пользователю.

Также существует тип связи многие ко многим, но в данной базе данных он реализован через вспомогательные таблицы, такие как CoachClients, которая устанавливает связь между тренерами и клиентами. То есть один тренер может иметь несколько клиентов, и один клиент может одновременно работать с несколькими тренерами.

База данных для приложения «Фитнес-клуб» состоит из 7 таблиц: CoachClients, NutritionPlans, Reviews, Subscriptions, Users, UserSubscriptions, WorkoutPlans. Назначение каждой из таблиц представлено в таблице 3.1.

Таблица 3.1 – Назначение таблиц в базе данных FitnessClubDB

|  |  |
| --- | --- |
| Таблица | Назначение |
| CoachClients | Связь между тренером и его клиентами |
| NutritionPlans | Хранение планов питания |
| Reviews | Хранение отзывов пользователей |
| Subscriptions | Содержание информации об абонементах клуба |
| Users | Хранение информации о пользователях |
| UserSubscriptions | Хранение информации о приобретенных абонементах клиента |
| WorkoutPlans | Словари ресурсов стилей |

В данном проекте для взаимодействия с базой данных используется библиотека Entity Framework Core версии 9.0.4. Эта библиотека предоставляет разработчику удобный способ работы с данными, представляя таблицы базы данных в виде привычных C#-классов. В отличие от традиционного подхода с использованием ADO.NET, при котором необходимо вручную писать SQL-запросы, Entity Framework Core позволяет использовать язык запросов LINQ, что значительно упрощает доступ и манипуляции с данными. Все преобразования запросов и их выполнение берет на себя сама библиотека, освобождая разработчика от необходимости взаимодействовать с базой данных напрямую.

Существует несколько подходов к проектированию базы данных в контексте Entity Framework: Model-First, Database-First и Code-First. В рамках данного проекта был выбран подход Code-First, как наиболее подходящий для гибкой и быстрой разработки.

Подход Code-First позволяет начать создание базы данных с написания моделей в коде, описывающих сущности предметной области. Эти модели формируют структуру базы данных, и на их основе Entity Framework автоматически создает необходимые таблицы и связи между ними. Такой подход особенно удобен на ранних этапах разработки, когда структура данных может часто меняться, и требуется высокая адаптивность архитектуры.

Преимуществом Code-First является его тесная интеграция с Agile-подходом: изменения в модели можно быстро вносить прямо в код, не редактируя базу данных вручную. Это облегчает поддержку проекта, ускоряет разработку и улучшает качество тестирования, поскольку можно легко создавать и использовать фиктивные данные в изолированной среде.

Использование Entity Framework Core в сочетании с подходом Code-First обеспечивает более удобную и контролируемую работу с базой данных, позволяет сфокусироваться на бизнес-логике и структуре приложения, повышая производительность и гибкость процесса разработки.

В целом, структура базы данных FitnessClubDB предоставляет необходимую основу для хранения и управления информацией о пользователях, тренировках, планах питания и тренировок и отзывах в приложении «Фитнес-клуб».

3.3 Структура проекта

Программное средство «**CoreX**» имеет следующею структуру, представленную на рисунке 3.3.

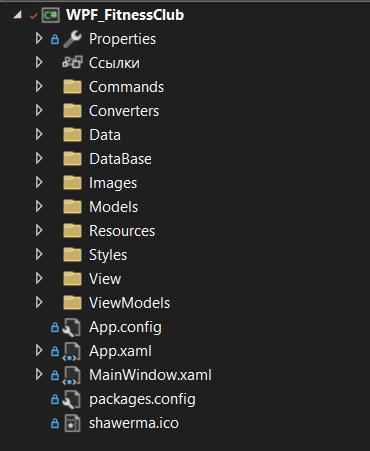


Рисунок 3.3 – Структура проекта

Описание структуры основных папок и файлов проекта и библиотеки классов представлено в таблице 3.2.

Таблица 3.2 – Описание структуры папок и файлов проекта

|  |  |
| --- | --- |
| Имя файла | Содержание |
| Папка Commands | Реализация команд |
| Папка Converters | Конвертеры для корректного отображения данных |
| Папка Data | Управление данными базы данных |
| Папка DataBase | Миграции базы данных |
| Папка Images | Все изображения |
| Папка Models | Данные и состояние приложения |
| Папка Resources | Словари ресурсов языков |
| Папка Styles | Словари ресурсов стилей |
| Папка ViewModel | Классы модели представления для окон пользователя |
| Папка View | XAML разметка окон приложения для пользователя |
| App.config | Файл с параметрами проекта |
| App.xaml | Определение глобальных ресурсов |
| MainWindow.xaml | Главное окно приложения, на котором отображаются все страницы |

Таким образом, сформированная таблица помогает понять общую структура проекта проектируемого программного средства.

3.4 Диаграмма классов

При создании программного продукта для планирования задач и ведения проектов, диаграмма классов выступает важным средством для моделирования и проектирования системы. Она визуализирует организацию системы, отображая классы, их характеристики и связи между ними.

В ходе разработки программного обеспечения для организации задач и управления проектами, диаграмма классов способствует выявлению ключевых классов, их свойств и методов. Она предоставляет возможность увидеть структуру данных, применяемых в приложении, а также взаимосвязи между классами. Помимо этого, диаграмма классов способствует более глубокому пониманию концептуальной модели системы и создает фундамент для дальнейшей разработки.

Диаграмма классов представлена в приложении Б.

4 Реализация программного средства

4.1 Основные классы программного средства

В приложении реализован широкий набор классов, каждый из которых выполняет специфическую функцию, способствуя созданию модульной и масштабируемой архитектуры. Однако в данном разделе акцент будет сделан на тех классах, которые задействованы в реализации архитектурного паттерна MVVM и организации взаимодействия с базой данных.

Паттерн MVVM (Model-View-ViewModel) лежит в основе архитектурной схемы приложения, обеспечивая четкое разграничение обязанностей между отображением пользовательского интерфейса и обработкой бизнес-логики. Это повышает читаемость кода, упрощает сопровождение и тестирование системы.

Классы, обеспечивающие взаимодействие с базой данных, отвечают за выполнение запросов, управление транзакциями и поддержание соединения с источником данных. Благодаря этому приложение может в реальном времени загружать и сохранять информацию, обеспечивая её актуальность и целостность данных.

Данные классы играют важнейшую роль в архитектуре системы, выполняя функцию связующего звена между визуальной частью и логикой обработки данных.

4.1.1 Классы для реализации паттерна MVVM

Как уже отмечалось ранее, приложение «CoreX» реализует архитектурный паттерн MVVM. Данный паттерн предполагает разделение приложения на три основных компонента, каждый из которых выполняет строго определённую роль:

* Model — представляет данные и бизнес-логику приложения, описывает структуру таблиц базы данных и отвечает за доступ к информации;
* View — визуальный интерфейс пользователя, отображающий данные и обеспечивающий взаимодействие с пользователем;
* ViewModel — промежуточное звено между моделью и представлением, которое обрабатывает команды от пользователя и управляет отображаемыми данными.

Принцип работы паттерна MVVM на практике будет рассмотрен на примере страницы авторизации. В листинге 4.1 представлен фрагмент XAML-разметки, относящейся к представлению (View). На данном примере реализовано создание кнопки для закрытие с использованием механизма привязки (binding) к команде LoginCommand. Это демонстрирует связь между представлением и моделью представления (ViewModel), где интерфейсный элемент (в данном случае — кнопка) инициирует выполнение команды, определённой в слое ViewModel. Такая привязка позволяет достичь чёткого разделения логики от пользовательского интерфейса.

Полный код страницы авторизации для более детального ознакомления приведён в приложении В.

Код модели представления (ViewModel), реализующий логику страницы авторизации, представлен в приложении Г. ViewModel играет ключевую роль в передаче данных между моделью и представлением. Этот класс реализует интерфейс INotifyPropertyChanged, который позволяет обновлять пользовательский интерфейс при изменении данных в реальном времени. ViewModel получает данные из модели, преобразует их при необходимости и предоставляет для отображения во View. Он не изменяет модель напрямую, а взаимодействует с ней через вызов соответствующих методов. Обработка действий пользователя осуществляется посредством команд (Commands), что повышает читаемость и масштабируемость кода.

В приложение В будет вынесен полный код данной страницы для подробного ознакомления.

|  |
| --- |
| <Button x:Name="EnterButton"  Content="{DynamicResource LoginButton}"  Width="54"  MaxWidth="200"  Margin="0,15,0,0"  Style="{DynamicResource EnterButton}"  Command="{Binding EnterCommand}"  CommandParameter="{Binding ElementName=PasswordInput}"/> |

Листинг 4.1 – Фрагмент кода xaml, демонстрирующий кнопку с привязкой к команде

Код модели представления (viewmodel) для данной страницы будет вынесен в приложение Г. ViewModel выступает посредником между View и Model, предоставляя данные для отображения и обрабатывая действия пользователя. Этот класс реализует интерфейс INotifyPropertyChanged для динамического обновления UI, вызывая событие PropertyChanged при изменении свойств. Он получает данные из Model, преобразует их для View и выполняет логику, но не модифицирует Model напрямую, а вызывает её методы. В данном классе действия пользователя обрабатываются через команды (Commands).

Перейдем к последней составляющей MVVM, которая отвечает за данные на нашей странице. В данном случае в коде команды EnterCommand, происходит обращение к данным о пользователях через модель User, которая содержат информацию о пользователях. По сути своей модель данных представляет из себя обычный класс, который представляет данные приложения. Она предоставляет интерфейс для доступа и изменения данных, может содержать логику для валидации и преобразования данных, а также для сохранения и извлечения данных из хранилища. Преимущества включают в себя повторное использование в разных частях приложения или даже в разных приложениях, лёгкую тестируемость, разделение ответственности между компонентами MVVM и гибкость при изменениях или расширениях без влияния на UI или ViewModel. Этот класс не зависит от конкретной реализации пользовательского интерфейса. Это позволяет использовать его в разных приложениях, а также быть легко тестируемым независимо от UI-логики, что делает разработку более эффективной и поддерживаемой.

В листинге 4.2 представлена модель User.

|  |
| --- |
| public class User : BaseEntity  {  private string fullName;  private string email;  private string login;  private string password;  private UserRole role;  private bool isBlocked;  }  public enum UserRole  {  Client = 1,  Coach = 2,  Admin = 3  } |

Листинг 4.2 – Модель User

В результате анализа реализации паттерна MVVM на примере страницы авторизации приложения становится очевидным, как эффективно организованы компоненты системы. Классы, отвечающие за модель, представление и модель представления, обеспечивают четкое разделение логики и интерфейса, что способствует упрощению разработки и тестирования.

4.1.2 Классы для работы с базой данных

Для работы с базой данных, взаимодействие с которой происходит при помощи специальной библиотеки Entity Framework, также требует наличия специальных классов. Данные классы представляют из себя класс сущности (по сути, это и есть модель) и класс контекста данных, который используется для взаимодействия непосредственно с самой базой данных. Класс контекста должен наследоваться от класса DbContext. В реализации данного класса с помощью DbSet указываются какие таблицы нужно создать в базе данных. Также следует реализовать в этом классе два перегруженных метода OnModelCreating и OnConfiguring. OnModelCreating используется для конфигурации модели данных при создании базы данных. Здесь можно задать правила отображения (mapping) между классами модели и таблицами базы данных, а также определить связи между сущностями, ограничения и индексы. OnConfiguring отвечает за настройку параметров подключения к базе данных. В нем можно задать строку подключения, а также конфигурации, такие как тип провайдера базы данных. Этот метод позволяет задать дополнительные параметры, такие как уровень ведения журнала или настройки кэширования.

Стоит сказать также, что AppDBContext реализует паттерны Unit of Work и Repository, что значительно упрощает управление доступом к данным и взаимодействие с базой данных. Паттерн Unit of Work обеспечивает согласованное выполнение операций с данными, позволяя отслеживать изменения в объектах и координировать их сохранение, что минимизирует риски нарушения целостности данных и упрощает управление транзакциями. В свою очередь, паттерн Repository служит абстракцией для доступа к данным, скрывая детали реализации хранения и запроса информации. Он предоставляет удобный интерфейс для выполнения операций CRUD, что облегчает тестирование бизнес-логики независимо от конкретной реализации базы данных. Таким образом, класс AppDBContext не только структурирует взаимодействие с данными, но и способствует чистоте архитектуры приложения, делая его более удобным для сопровождения и расширения.

4.2 Реализация функционала приложения

В данном подразделе будут представлены некоторые методы, обеспечивающие функционал приложения: метод для бронирования абонемента, метод для добавления и удаления товара из корзины, метод для регистрации пользователя в приложении, а также метод для формирования заказа и отправки его на рассмотрение администратору.

**4.2.1 Метод для удаления плана тренировок и питания тренером**

Код данного метода представлен в приложении Е. Метод доступен только пользователям с ролью "Тренер". После инициализации подключения к базе данных через AppDBContext, осуществляется поиск плана по заданному идентификатору. В зависимости от типа удаляемого плана (тренировочный или по питанию), выполняется соответствующий запрос к таблице WorkoutPlans или NutritionPlans. Если нужный план найден, он удаляется из базы данных. После удаления вызываются метод LoadClientPlans();, чтобы изменения отобразились в пользовательском интерфейсе и уведомили всех подписчиков через OnPropertyChanged().

**4.2.2 Метод для регистрации пользователя в приложении**

Код данного метода представлен в приложении Е. В методе создаётся объект пользователя, после чего проверяется наличие в базе данных пользователя с таким же логином. Если совпадение найдено, пользователю выводится сообщение об ошибке, и регистрация прерывается. В случае отсутствия дубликата осуществляется валидация всех обязательных полей — имени, фамилии, логина, номера телефона и email. Если валидация проходит успешно, информация о пользователе добавляется в таблицу Users. Затем пользователь перенаправляется на главную страницу приложения. При ошибке валидации выводится соответствующее сообщение, и регистрация не продолжается.

**4.2.3 Метод для добавления нового абонемента**

Код данного метода размещён в приложении Е. Данный метод используется администраторами или менеджерами клуба. После подключения к базе данных создаётся объект Subscription, включающий наименование, описание, цену, длительность и условия. Перед добавлением проводится проверка на уникальность названия. Если абонемент с таким названием уже существует, добавление не выполняется, и пользователю выводится сообщение. В противном случае новая запись добавляется в таблицу Subscriptions. Далее выполняется сохранение через db.SaveChanges() и обновление интерфейса с помощью метода UpdateSubscriptionListUI(), а также уведомление всех подписчиков.

**4.2.4 Метод для удаления отзыва пользователем**

Код метода приведён в приложении Е. Данный метод позволяет пользователю удалить ранее оставленный отзыв. После идентификации текущего пользователя и подключения к базе данных осуществляется поиск отзыва по идентификатору пользователя и объекта, к которому он относится. Если отзыв найден, он удаляется из таблицы Reviews. Также вызывается метод OnPropertyChanged() для уведомления подписчиков об изменениях.

5 Тестирование, проверка работоспособности приложения и анализ результатов

В данном разделе рассматривается процесс тестирования разработанной программной системы, направленный на проверку её функциональной пригодности. Также проводится анализ поведения системы в условиях возникновения исключительных ситуаций, с целью выявления устойчивости и корректности обработки возможных ошибок. Это позволяет убедиться в надежности программного продукта при нестандартных сценариях.

Основной задачей данного этапа является демонстрация того, что программная система функционирует стабильно, выполняет поставленные задачи и может быть рекомендована к внедрению и дальнейшей эксплуатации.

5.1 Тестирование функций страницы регистрации

Первым тестом будет попытка оставить одно поле пустым, например, поле имени. На рисунке 5.1. показан результат.

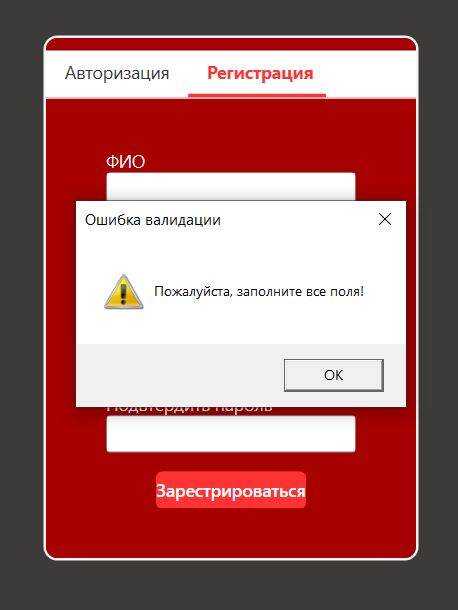


Рисунок 5.1 – Тест на пустое поле имени

Далее вторым тестом будет попытка ввести пользовательское имя, которое уже занято. Результат отображен на рисунке 5.2.

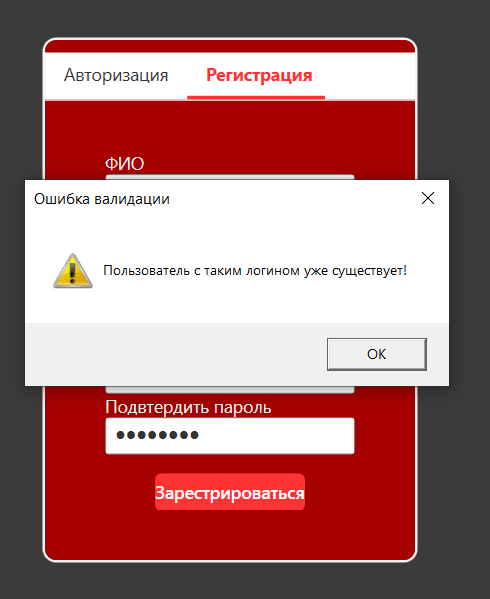


Рисунок 5.2 – Тест на уже существующий логин

Можно сделать вывод, что в окне авторизации валидация также работает корректно и информирует пользователя об совершенных ошибках.

5.2 Тестирование функций страницы авторизации

Протестируем авторизацию с пустыми полями. Результат представлен на рисунке 5.3.

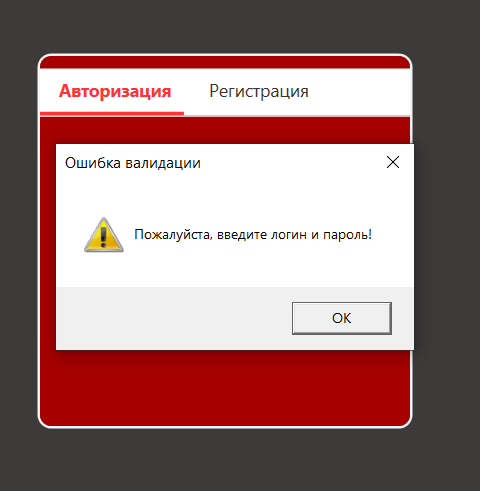


Рисунок 5.3 – Тест на пустое поле с паролем

Также возможна ситуация, когда пользователь пытается войти в несуществующий аккаунт. Результат показан на рисунке 5.4.

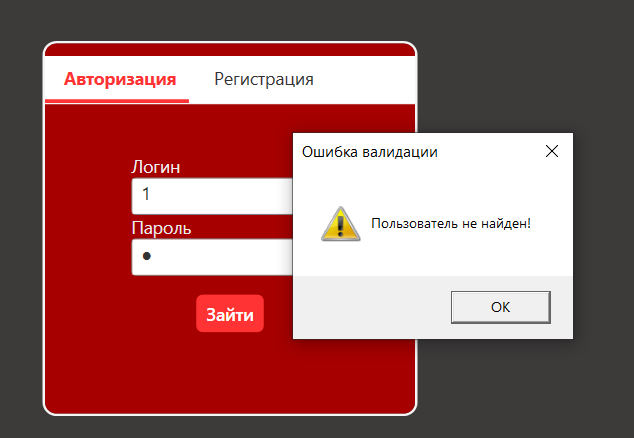


Рисунок 5.4 – Тест на некорректные данные аккаунта

Реализация валидации для введенных в форме данных играет важную роль в обеспечении корректного и надежного функционирования приложения. Это позволяет предотвратить ввод некорректных или недопустимых значений, что может привести к ошибкам или некорректной работе приложения.

5.3 Тестирование функций редактирования профиля

Пользовательское приложение предусматривает возможность редактирования данных профиля, включая логин, имя, фамилию, номер телефона и адрес электронной почты. При корректном вводе и прохождении всех этапов валидации информация успешно сохраняется в базе данных, и изменения вступают в силу.

В рамках тестирования была рассмотрена ситуация, при которой введённые данные не соответствуют требованиям валидации. В таком случае система корректно реагирует, отображая пользователю предупреждение о наличии ошибки или конфликта данных. Результаты тестирования данной функциональности представлены на рисунке 5.5.

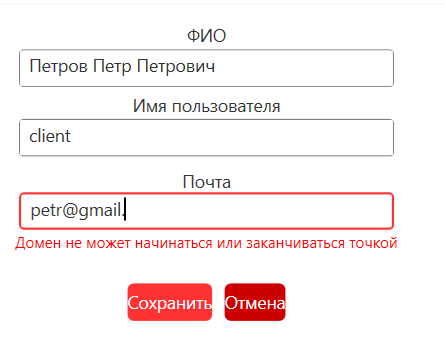


Рисунок 5.5 – Результат тестирования редактирования профиля

Также была проведена проверка валидации всех полей формы при изменении данных пользователя, а также при подтверждении нового пароля. В случае некорректного заполнения любого из полей пользователю выводились информативные сообщения об ошибках, указывающие на причину отказа валидации и рекомендации по исправлению.

Результаты проверки корректности отображены на рисунке 5.6.

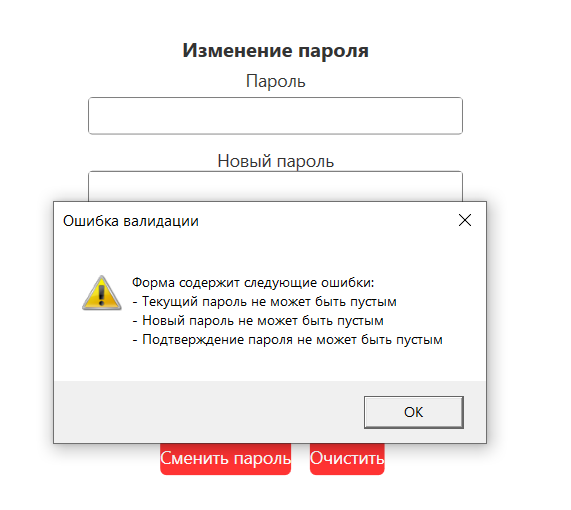


Рисунок 5.6 – Результат тестирования изменения пароля

Тестирование подтвердило надёжность и удобство функций редактирования профиля и изменения пароля, обеспечивая корректную валидацию и обработку ошибок.

5.4. Тестирование добавления абонементов

Одной из ключевых функций тренера является возможность добавления абонементов на тренировки. Для выполнения данной операции тренер или администратор заполняет специальную форму добавления абонемента, где требуется указать все необходимые данные.

Каждое поле должно быть корректно заполнено и соответствовать установленным требованиям валидации. В случае успешного прохождения всех проверок, абонемент добавляется в систему и становится доступным для просмотра и бронирования. В противном случае система информирует администратора об ошибках ввода.

Результаты тестирования процесса добавления акций представлены на рисунке 5.7.

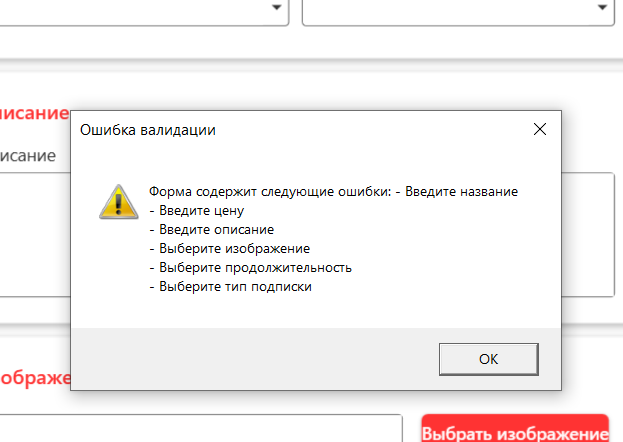


Рисунок 5.7 – Результат тестирования добавления абонементов

Тестирование подтвердило корректную работу механизма добавления абонементов с учётом всех требований валидации и удобством для пользователя.

5.5. Тестирование добавления отзывов

В приложении реализована возможность оставлять комментарии, что позволяет пользователям делиться своими впечатлениями о тренировках или работе сервиса. Стоит упомянуть, что клиент может оставить отзыв, только если он приобрел данный абонемент. Также пользователь может оставить только один отзыв на каждый абонемент. Для этого необходимо ввести текст отзыва в специальное поле ввода. Если пользователь попытается отправить комментарий, не заполнив это поле, система выполнит проверку на корректность ввода и отобразит сообщение об ошибке, информируя о необходимости ввести текст перед отправкой.

Результаты тестирования данной ситуации представлены на рисунке 5.8.

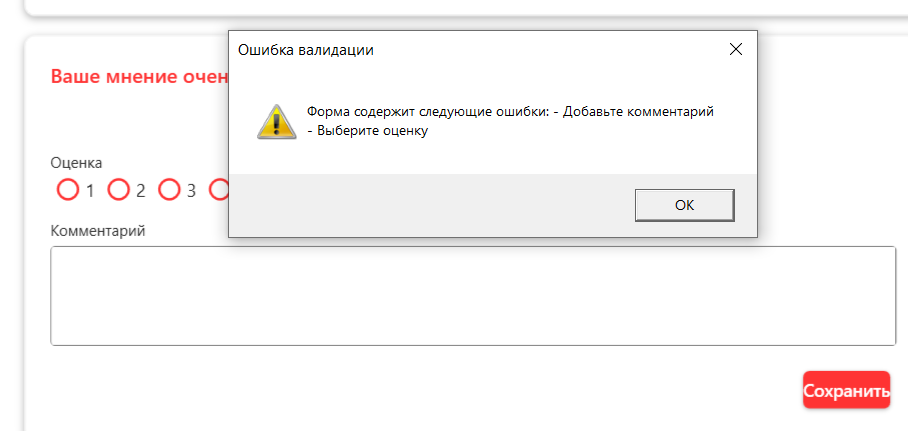


Рисунок 5.8 – Результат тестирования добавления отзывов

Тестирование показало, что система корректно обрабатывает добавление отзывов, обеспечивая соблюдение ограничений и информируя пользователя о необходимости заполнения обязательных полей.

5.6. Тестирование управление пользователями

В перечень функционала администратора входить управление пользователями. В целях предотвращения непредвиденных ситуаций предусмотрен запрет на блокировку или удаления аккаунта текущего администратора. Результат представлен на рисунке 5.9.

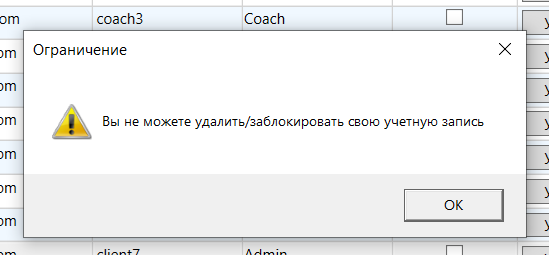


Рисунок 5.9 – Результат тестирования удаления текущего аккаунта

Тестирование управления пользователями подтвердило, что система надёжно предотвращает блокировку и удаление аккаунта текущего администратора, а также обеспечивает корректное добавление новых пользователей при соблюдении всех требований валидации.

6 Руководство по использованию

6.1 Регистрация и авторизация

После запуска приложения пользователь перенаправляется на стартовую страницу авторизации, где необходимо ввести логин пароль для доступа к системе. Только после успешной аутентификации пользователь получает возможность работать с функционалом приложения в соответствии со своей ролью. Визуальное представление страницы показано на рисунке 6.1.

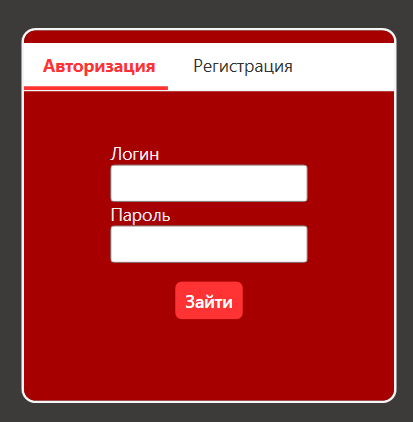


Рисунок 6.1 – Страница авторизации

Если у пользователя еще нет аккаунта, то он может перейти во вкладку с регистрацией, которая показана на рисунке 6.2.

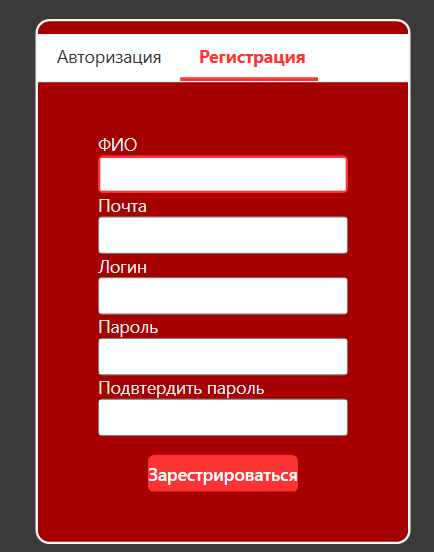


Рисунок 6.2 – Страница регистрации

После заполнения всех обязательных полей в форме регистрации и подтверждения введённых данных, пользователь получает возможность создать учётную запись и получить доступ к функционалу приложения.

6.2 Использование приложения в роли клиента

Когда пользователь успешно выполнит вход в систему, он будет автоматически перенаправлен на главную страницу приложения. На стартовой странице расположены все доступные абонементы тренировок фитнес-клуба

Слева на экране приложения размещено навигационное меню, которое обеспечивает удобный и быстрый доступ к основным разделам сервиса. Меню выполнено в виде вертикальной панели с яркими иконками, что облегчает ориентирование даже новым пользователям. Через навигационное меню можно перейти к просмотру расписания тренировок, планов питания, разделу отзывов, а также личному кабинету.

Главная страница приложения, представленная на рисунке 6.3, выполнена в современном и интуитивно понятном дизайне, который обеспечивает комфортную работу пользователя.

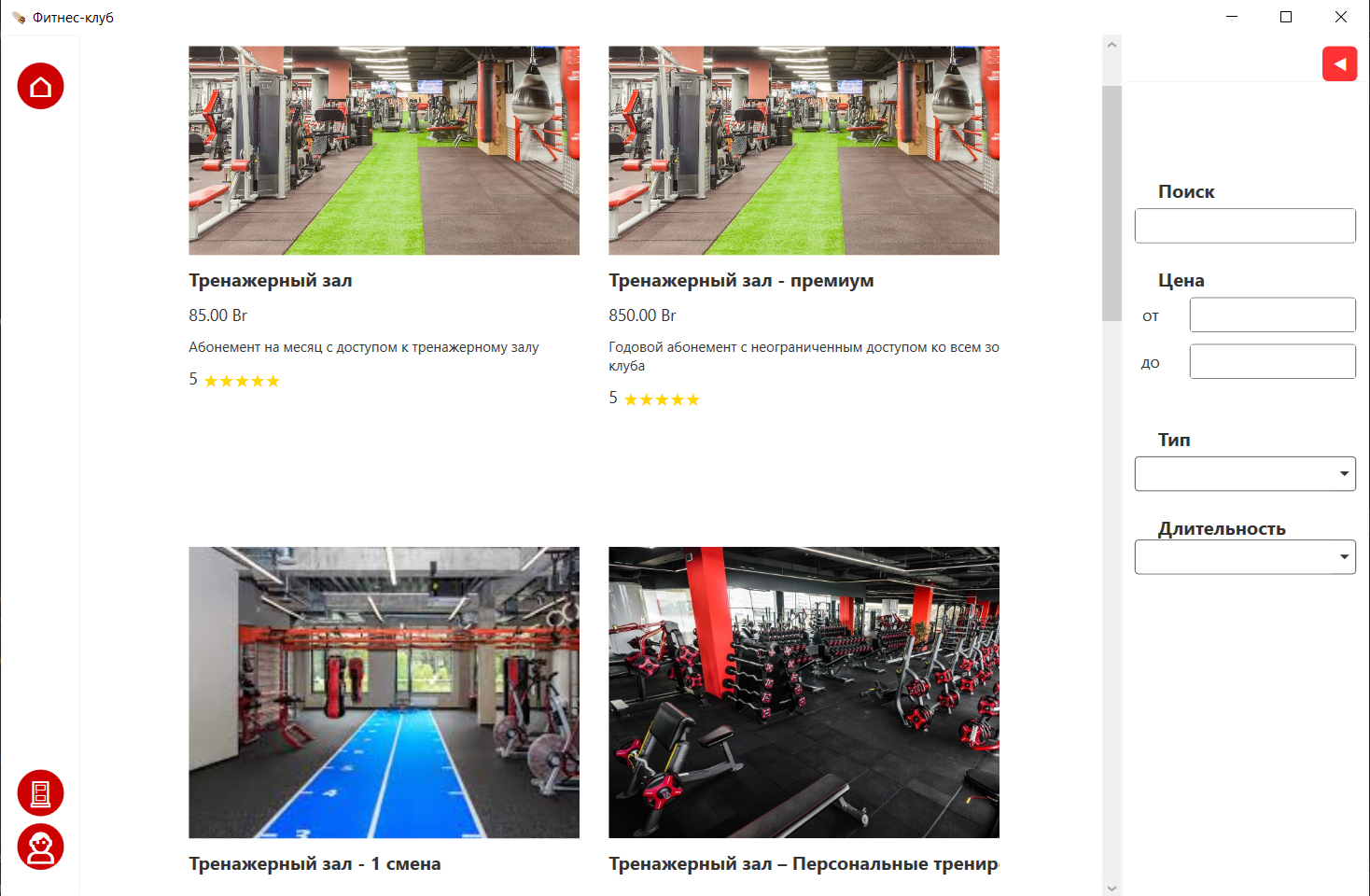


Рисунок 6.3 – Главная страница приложения

Таким образом, после входа пользователь получает доступ к основным функциям сервиса, а удобная структура и оформление главной страницы способствуют положительному опыту взаимодействия с приложением.

Внизу навигационного меню располагается кнопка перехода в личный кабинет. На данной странице пользователь может просмотреть и изменить свои личные данные, а также настройки приложения, включающие в себя язык и цветовую тему. Также можно просмотреть свои активные записи на тренировки, которые еще не прошли, а также отменить запись на тренировку. Также можно посмотреть свои планы питания и тренировок. Страница «Личный кабинет» продемонстрирована на рисунке 6.4.

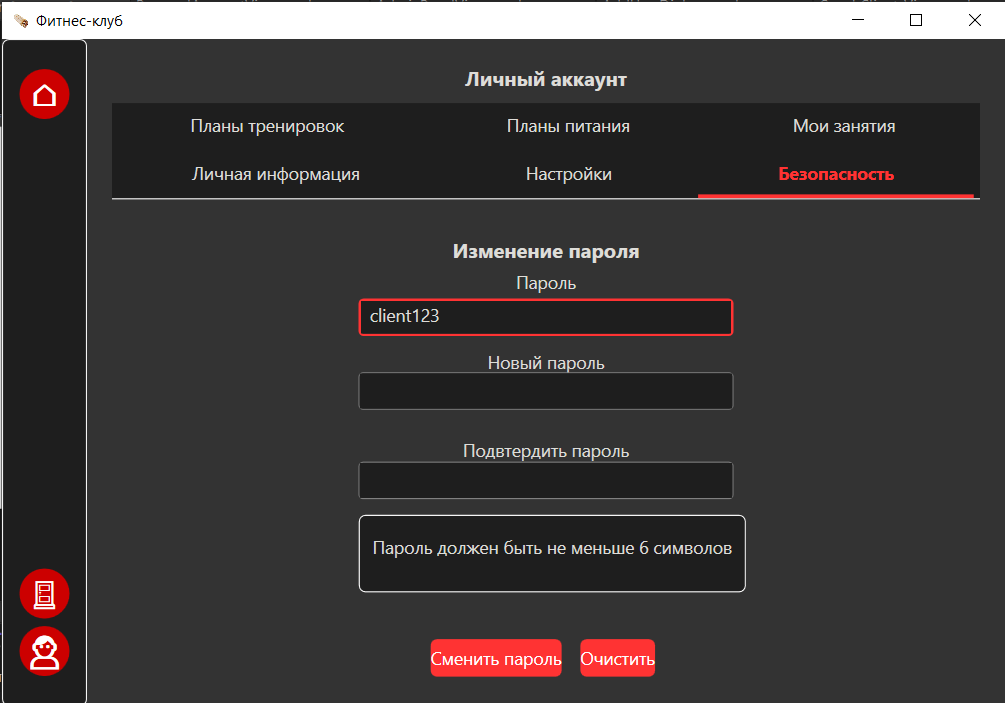


Рисунок 6.4 — Личный кабинет

Приложение «CoreX» обеспечивает удобный пользовательский интерфейс, для изучения абонементов фитнес-клуба, осуществление записи на тренировку и их отслеживание.

6.3 Использование приложения в роли тренера

После входа в приложение тренер попадает на главную страницу. Слева находится меню, с помощью которого можно переходить в разные разделы. Для работы с планами тренировок и планами питания достаточно выбрать нужный пункт меню.

На главной странице тренер видит список всех доступных абонементов. Если нужно добавить новый абонемент, достаточно нажать на кнопку добавления, ввести нужную информацию и сохранить. Если нужно изменить уже существующий абонемент, его можно выбрать из списка, внести изменения и сохранить их. А если какой-то абонемент больше не нужен, его можно просто удалить, выбрав его и нажав соответствующую кнопку. На рисунке 6.5 изображено окно добавления абонемента.

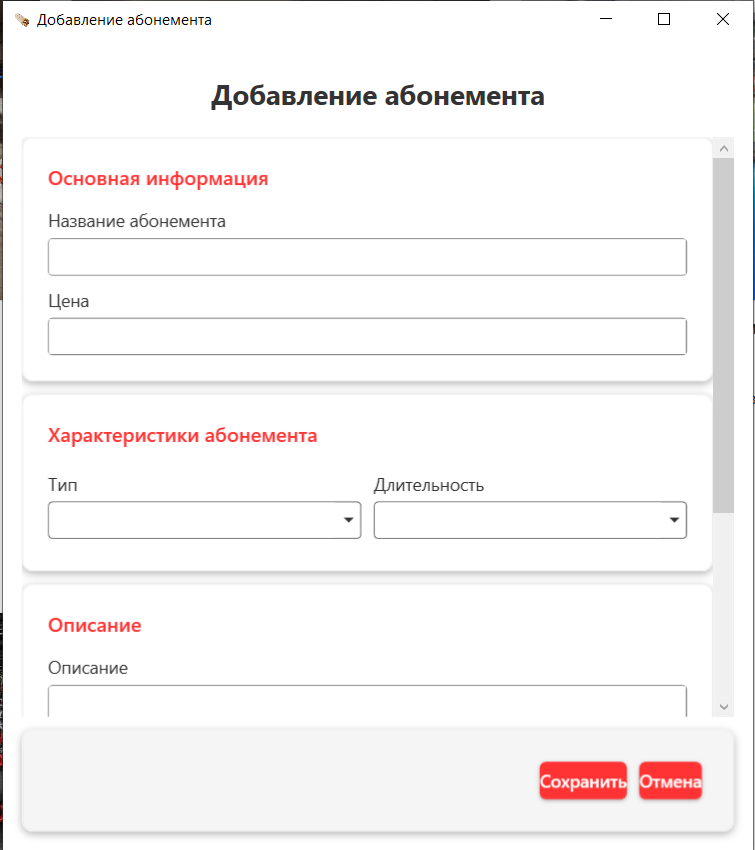


Рисунок 6.5 — Личный кабинет

Аналогично устроена работа с планами тренировок. В разделе «Планы тренировок» тренер может создавать новые планы, указывая, для кого они предназначены, какие упражнения в них входят и как часто их нужно выполнять. Уже созданные планы можно открыть, изменить или удалить, если они больше не нужны.

Для планов питания действует такой же принцип. В разделе «Планы питания» тренер может добавлять новые рекомендации, расписывать рацион по дням, редактировать уже созданные планы или удалять устаревшие.

Приложение сделано так, чтобы тренер мог легко и быстро вносить нужную информацию, обновлять планы и следить за актуальностью абонементов, помогая пользователям достигать своих целей.

6.4 Использование приложения в роли администратора

После входа в систему администратор попадает на главную страницу приложения. Слева расположено удобное меню, с помощью которого можно быстро переходить в нужные разделы. Администратору доступны функции по управлению пользователями, отзывами и абонементами.

Чтобы управлять пользователями, нужно открыть раздел «Пользователи». Здесь отображается список всех зарегистрированных пользователей системы. Администратор может выбрать любого пользователя, чтобы просмотреть его данные. Если нужно, можно отредактировать информацию — например, изменить имя, роль или другую информацию. Также здесь можно удалить пользователя, если он больше не должен иметь доступ к приложению.

На рисунке 6.6 изображена панель администратора.

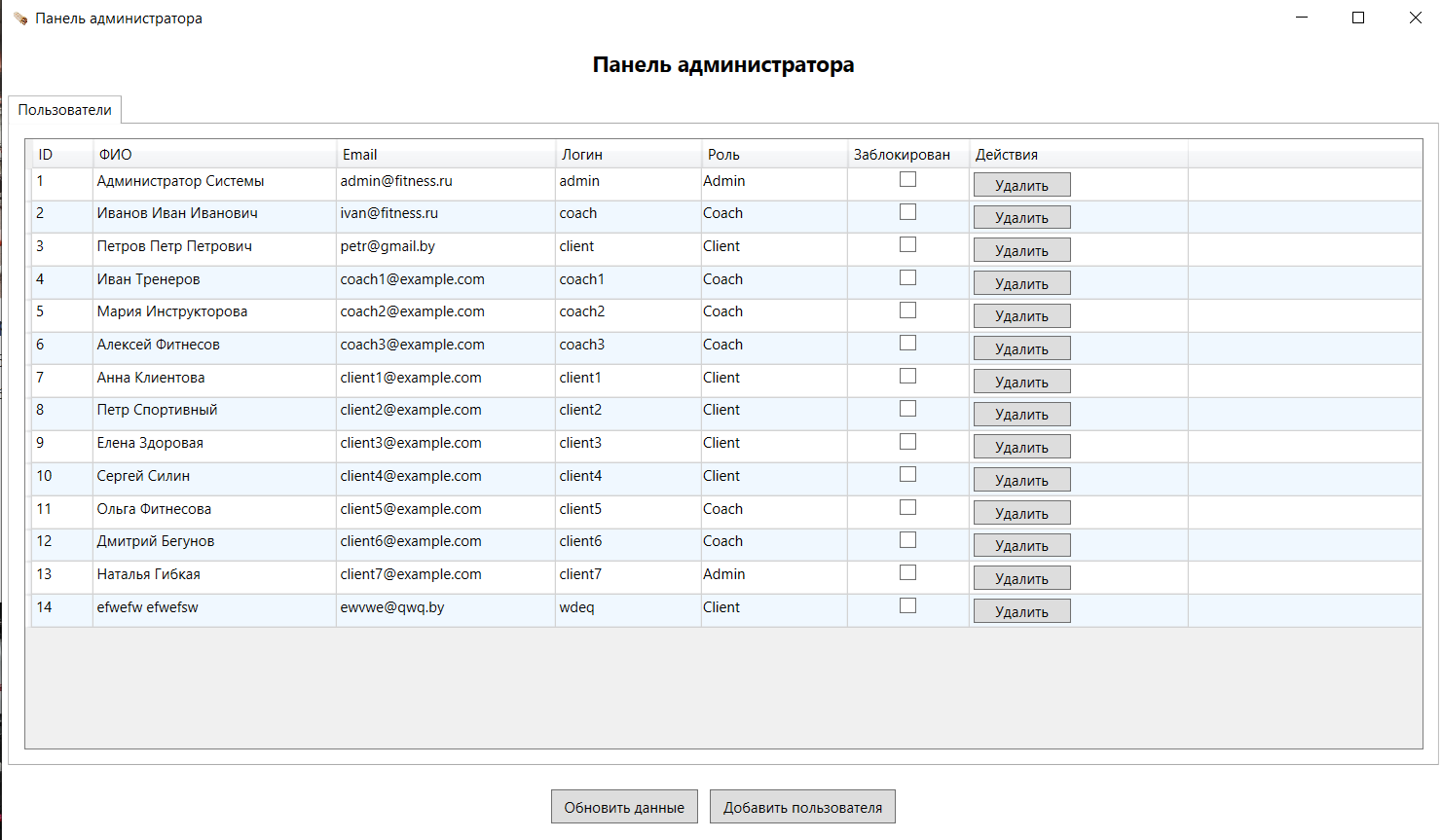


Рисунок 6.6 — Панель администратора

В окне с деталями к абонементу может просматривать отзывы. Если они содержат некорректную информацию или нарушает правила, их можно удалить. Это помогает поддерживать порядок и корректность информации в приложении. Управление отзывами представлено на рисунке 6.7.

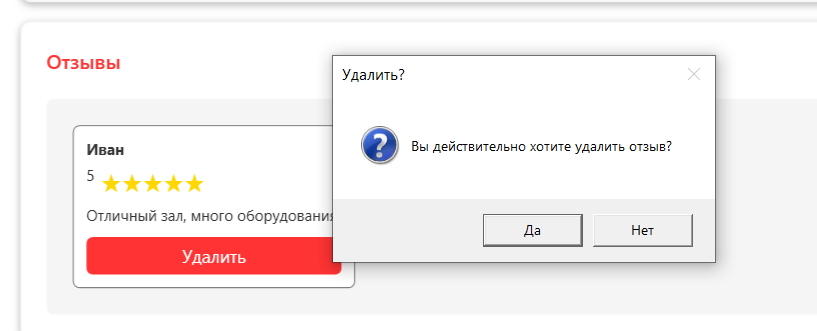


Рисунок 6.7 — Управление отзывами

Все действия выполняются через понятный интерфейс, где каждая кнопка подписана и находится на видном месте. Администратор может быстро находить нужную информацию, вносить изменения и следить за порядком в системе. Таким образом, приложение предоставляет администратору все необходимые инструменты для удобного управления пользователями, отзывами и абонементами — легко, понятно и без лишней сложности.

# **ЗАКЛЮЧЕНИЕ**

Данное курсовое проектное задание было выполнено с целью разработки программного средства "CoreX", предназначенного для предоставления пользователям удобного способа выбора и записи на тренировки, тренерам добавления планов тренировок и питания своим клиентам, а также для обеспечения администратора возможностью управления базой данных и технической поддержки приложения.

В процессе выполнения проекта были достигнуты следующие результаты:

* разработано интуитивно понятное клиентское приложение с современным и удобным интерфейсом, обеспечивающим эффективный просмотр, фильтрацию и выбор тренировок;
* реализована функциональность записи на тренировку с возможностью последующей отмены;
* создана тренерская часть приложения, включающая инструменты для управления расписанием тренировок, мониторинга данных пользователей, обработки отзывов и управления пользовательскими записями;
* создана администраторская часть приложения, включающая инструменты для управления абонементами, планами питания и тренировок;
* создана администраторская часть приложения, включающая мониторинга данных пользователей, обработки отзывов;
* проведено комплексное тестирование программного обеспечения с целью проверки корректности выполнения всех функций, устойчивости к ошибкам и соответствия требованиям функциональности.

Для реализации приложения был выбран язык программирования C# в связке с технологией WPF, что позволило создать гибкий и отзывчивый интерфейс. В качестве системы управления базами данных использовалась Microsoft SQL Server, которая обеспечивает надежное хранение и обработку информации о пользователях, тренировках и расписаниях.

В архитектуре программного средства активно использовался паттерн проектирования MVVM. Благодаря этому структура проекта стала более понятной, масштабируемой и удобной для поддержки.

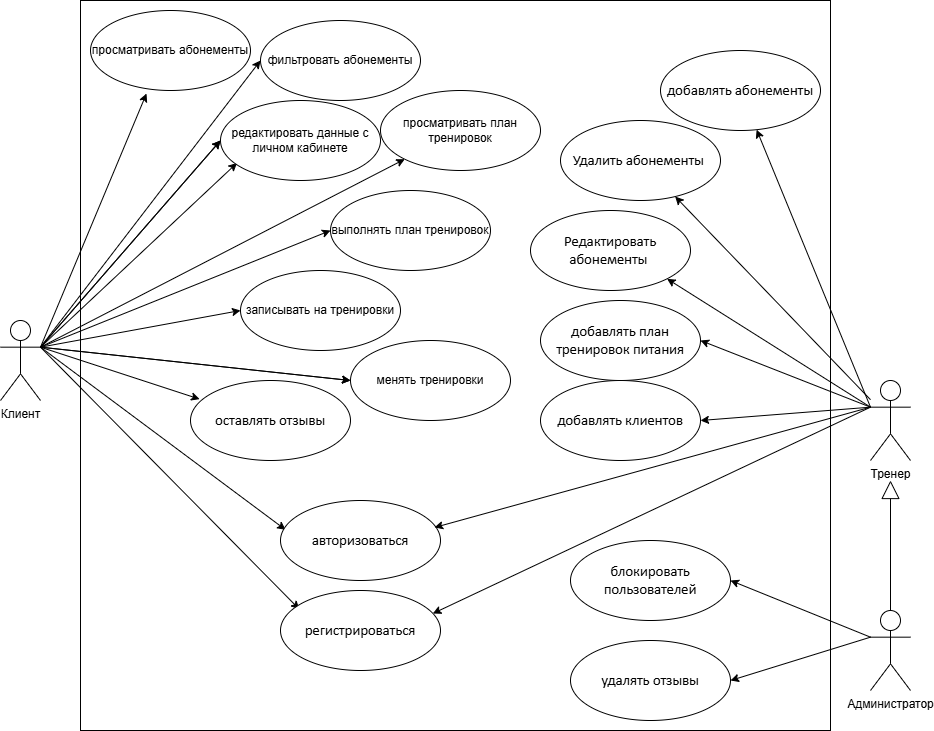
В результате выполнения курсового проекта было создано функциональное, стабильное и удобное в использовании программное средство "CoreX", обладающее всеми необходимыми возможностями для организации тренировочного процесса.

# **Список использованных источников**

1. Онлайн-ресурс «iron-fitness.by» [Электронный ресурс] – Режим доступа: iron-fitness.by – Дата доступа: 11.03.2025
2. Онлайн-ресурс «olympia-gs.by» [Электронный ресурс] – Режим доступа: olympia-gs.by – Дата доступа: 11.03.2025
3. Онлайн-ресурс «adrenalin-fitness.by» [Электронный ресурс] – Режим доступа: adrenalin-fitness.by – Дата доступа: 11.03.2025
4. Руководство по WPF // [Электронный ресурс]. – Режим доступа: https://metanit.com/sharp/wpf/ – Дата доступа: 20.03.2025
5. Работа с Entity Framework Core [Электронный ресурс] – Режим доступа: https://professorweb.ru/my/entity-framework/6/level1/ – Дата доступа 28.04.2025

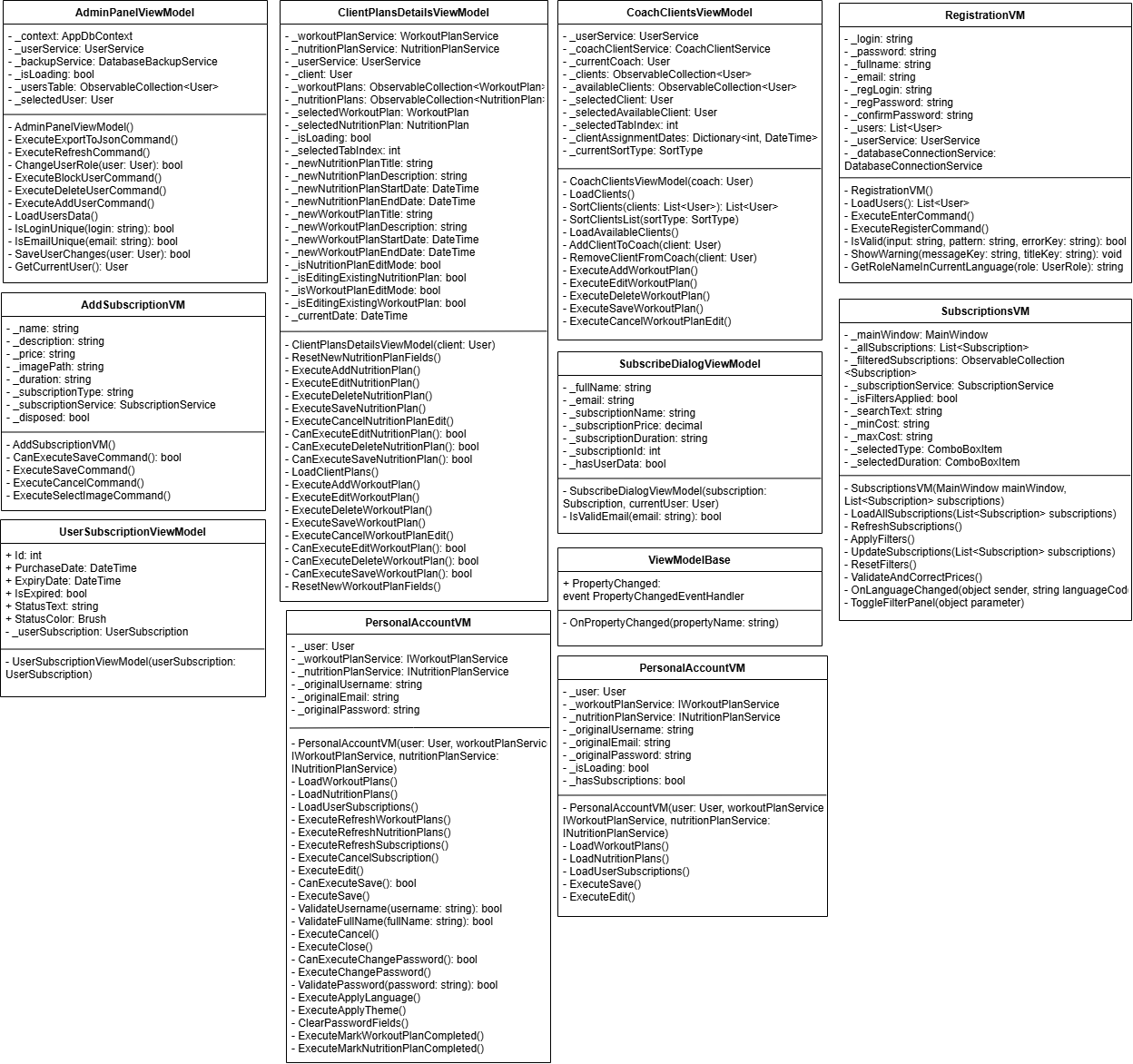
# **ПРИЛОЖЕНИЕ А**

Диаграмма вариантов использования

****

# **ПРИЛОЖЕНИЕ Б**

Диаграмма классов ViewModel



# **ПРИЛОЖЕНИЕ В**

Код представления страницы авторизации и регистрации

|  |
| --- |
| <Window x:Class="WPF\_FitnessClub.View.RegistrationView"  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"  xmlns:d="http://schemas.microsoft.com/expression/blend/2008"  xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"  xmlns:local="clr-namespace:WPF\_FitnessClub"  xmlns:windows="clr-namespace:WPF\_FitnessClub.View" xmlns:viewmodels="clr-namespace:WPF\_FitnessClub.ViewModels"  mc:Ignorable="d"  Title="{DynamicResource RegistrationTitle}"  Height="500" Width="800">  <Grid  Background="#3d3a3a">  <Grid.RowDefinitions>  <RowDefinition Height="Auto"/>  <RowDefinition Height="\*"/>  <RowDefinition Height="Auto"/>  </Grid.RowDefinitions>  <Grid.ColumnDefinitions>  <ColumnDefinition Width="\*"/>  <ColumnDefinition Width="Auto"/>  <ColumnDefinition Width="\*"/>  </Grid.ColumnDefinitions>  <StackPanel  Grid.Row="0"  Grid.ColumnSpan="3"  Orientation="Horizontal"  HorizontalAlignment="Right">  <Button  x:Name="RussianButton"  Content="RU"  Click="RussianButton\_Click" HorizontalAlignment="Right" Style="{DynamicResource LanguageButton}"/>  <Button  x:Name="EnglishButton"  Content="EN"  Click="EnglishButton\_Click" HorizontalAlignment="Right" Style="{DynamicResource LanguageButton}"/>  </StackPanel>  <Border x:Name="MainBorder"  Grid.Row="1"  Grid.Column="1"  BorderThickness="2"  CornerRadius="10"  Width="300"  Height="270"    Background="#A60000"  HorizontalAlignment="Center">  <TabControl x:Name="MainTabControl" Style="{StaticResource CustomTabControlStyle}" SelectionChanged="MainTabControl\_SelectionChanged" >  <TabItem Header="{DynamicResource LoginTitle}"  Style="{StaticResource CustomTabItemStyle}">    <StackPanel    Margin="30"  HorizontalAlignment="Center">  <TextBlock    Text="{DynamicResource Login}" Foreground="#FFF1F1F1"  />  <TextBox x:Name="LoginInput"  TextAlignment="Left"  Width="158"  MaxWidth="200"  MaxLength="30"  Text="{Binding Login, UpdateSourceTrigger=PropertyChanged}"  Style="{DynamicResource CustomTextBoxStyle}" />  <TextBlock  Text="{DynamicResource Password}" Foreground="#FFEDEDED"  />  <PasswordBox x:Name="PasswordInput"  Width="158"  MaxWidth="200"  Foreground="{DynamicResource TextBrush}"  Style="{DynamicResource CustomPasswordBoxStyle}"  PreviewKeyDown="PasswordBox\_PreviewKeyDown" />  <Button x:Name="EnterButton"  Content="{DynamicResource LoginButton}"  Width="54"  MaxWidth="200"  Margin="0,15,0,0"  Style="{DynamicResource EnterButton}"  Command="{Binding EnterCommand}"  CommandParameter="{Binding ElementName=PasswordInput}"/>  </StackPanel>  </TabItem>  <TabItem Header="{DynamicResource RegistrationTitle}"  Style="{StaticResource CustomTabItemStyle}">  <StackPanel Margin="30" HorizontalAlignment="Center">  <TextBlock Foreground="#FFF1F1F1" Text="{DynamicResource FullName}" />  <TextBox x:Name="FullNameInput"  TextAlignment="Left"  Width="200"  MaxLength="30"  Text="{Binding FullName, UpdateSourceTrigger=PropertyChanged}"  Style="{DynamicResource CustomTextBoxStyle}" />  <TextBlock Text="{DynamicResource Email}" Foreground="#FFF1F1F1" />  <TextBox x:Name="EmailInput"  TextAlignment="Left"  Width="200"  MaxLength="30"  Text="{Binding Email, UpdateSourceTrigger=PropertyChanged}"  Style="{DynamicResource CustomTextBoxStyle}" />  <TextBlock Text="{DynamicResource Login}" Foreground="#FFF1F1F1" />  <TextBox x:Name="RegLoginInput"  TextAlignment="Left"  Width="200"  MaxLength="30"  Text="{Binding RegLogin, UpdateSourceTrigger=PropertyChanged}"  Style="{DynamicResource CustomTextBoxStyle}" />  <TextBlock Text="{DynamicResource Password}" Foreground="#FFF1F1F1" />  <PasswordBox x:Name="RegPasswordInput"  Width="200"  Foreground="{DynamicResource TextBrush}"  Style="{DynamicResource CustomPasswordBoxStyle}"  PreviewKeyDown="PasswordBox\_PreviewKeyDown" />  <TextBlock Text="{DynamicResource ConfirmPassword}" Foreground="#FFF1F1F1" />  <PasswordBox x:Name="ConfirmPasswordInput"  Width="200"  Foreground="{DynamicResource TextBrush}"  Style="{DynamicResource CustomPasswordBoxStyle}"  Tag="{Binding ElementName=RegPasswordInput}"  PreviewKeyDown="PasswordBox\_PreviewKeyDown" />  <Button x:Name="RegisterButton"  Content="{DynamicResource RegisterButton}"  Width="120"  Margin="0,15,0,0"  Style="{DynamicResource EnterButton}"  Command="{Binding RegisterCommand}"  CommandParameter="{Binding ElementName=ConfirmPasswordInput}" />  </StackPanel>  </TabItem>  </TabControl>  </Border>  </Grid>  </Window> |

# **ПРИЛОЖЕНИЕ Г**

Код модели представления страницы авторизации

|  |
| --- |
| public class RegistrationVM : ViewModelBase  {  string \_login;  string \_password;  string \_fullname;  string \_email;  string \_regLogin;  string \_regPassword;  string \_confirmPassword;  string roleName;  string welcomeMessage;  private List<User> \_users;  private readonly UserService \_userService;  private readonly DatabaseConnectionService \_databaseConnectionService;  public string Login  {  get => \_login;  set  {  if (\_login != value)  {  \_login = value;  OnPropertyChanged(nameof(Login));  }  }  }  public string Password  {  get => \_password;  set  {  if (\_password != value)  {  \_password = value;  OnPropertyChanged(nameof(Password));  }  }  }  public string FullName  {  get => \_fullname;  set  {  if (\_fullname != value)  {  \_fullname = value;  OnPropertyChanged(nameof(FullName));  }  }  }  public string Email  {  get => \_email;  set  {  if (\_email != value)  {  \_email = value;  OnPropertyChanged(nameof(Email));  }  }  }  public string RegLogin  {  get => \_regLogin;  set  {  if (\_regLogin != value)  {  \_regLogin = value;  OnPropertyChanged(nameof(RegLogin));  }  }  }  public string RegPassword  {  get => \_regPassword;  set  {  if (\_regPassword != value)  {  \_regPassword = value;  OnPropertyChanged(nameof(RegPassword));  }  }  }  public string ConfirmPassword  {  get => \_confirmPassword;  set  {  if (\_confirmPassword != value)  {  \_confirmPassword = value;  OnPropertyChanged(nameof(ConfirmPassword));  }  }  }  private readonly DatabaseConnectionService \_databaseConnectionService; |

# **ПРИЛОЖЕНИЕ Д**

Код класса AppDbContext

|  |
| --- |
| using System;  using System.Data.Entity;  using System.Data.Entity.ModelConfiguration.Conventions;  using WPF\_FitnessClub.Models;  using WPF\_FitnessClub.DataBase;  namespace WPF\_FitnessClub.Data  {  public class AppDbContext : DbContext  {  public AppDbContext() : base("FitnessClubConnectionString")  {  Configuration.LazyLoadingEnabled = false;    Configuration.AutoDetectChangesEnabled = true;    Database.SetInitializer<AppDbContext>(null);  }  public DbSet<User> Users { get; set; }  public DbSet<Subscription> Subscriptions { get; set; }  public DbSet<Review> Reviews { get; set; }  public DbSet<NutritionPlan> NutritionPlans { get; set; }  public DbSet<WorkoutPlan> WorkoutPlans { get; set; }  public DbSet<CoachClient> CoachClients { get; set; }  public DbSet<UserSubscription> UserSubscriptions { get; set; }  protected override void OnModelCreating(DbModelBuilder modelBuilder)  {  modelBuilder.Conventions.Remove<OneToManyCascadeDeleteConvention>();  modelBuilder.Entity<User>()  .Property(u => u.Role)  .HasColumnName("Role")  .IsRequired()  .HasColumnType("int");  modelBuilder.Entity<Subscription>()  .HasMany(s => s.Reviews)  .WithRequired(r => r.Subscription)  .HasForeignKey(r => r.SubscriptionId);  modelBuilder.Entity<UserSubscription>()  .HasRequired(us => us.User)  .WithMany()  .HasForeignKey(us => us.UserId)  .WillCascadeOnDelete(false);    modelBuilder.Entity<UserSubscription>()  .HasRequired(us => us.Subscription)  .WithMany()  .HasForeignKey(us => us.SubscriptionId)  .WillCascadeOnDelete(false);  modelBuilder.Entity<NutritionPlan>()  .HasRequired(np => np.Client)  .WithMany()  .HasForeignKey(np => np.ClientId)  .WillCascadeOnDelete(false);  modelBuilder.Entity<NutritionPlan>()  .HasRequired(np => np.Coach)  .WithMany()  .HasForeignKey(np => np.CoachId)  .WillCascadeOnDelete(false);  modelBuilder.Entity<WorkoutPlan>()  .HasRequired(wp => wp.Client)  .WithMany()  .HasForeignKey(wp => wp.ClientId)  .WillCascadeOnDelete(false);  modelBuilder.Entity<WorkoutPlan>()  .HasRequired(wp => wp.Coach)  .WithMany()  .HasForeignKey(wp => wp.CoachId)  .WillCascadeOnDelete(false);  modelBuilder.Entity<CoachClient>()  .HasRequired(cc => cc.Coach)  .WithMany()  .HasForeignKey(cc => cc.CoachId)  .WillCascadeOnDelete(false);  modelBuilder.Entity<CoachClient>()  .HasRequired(cc => cc.Client)  .WithMany()  .HasForeignKey(cc => cc.ClientId)  .WillCascadeOnDelete(false);  base.OnModelCreating(modelBuilder);  }  }  } |

# **ПРИЛОЖЕНИЕ Е**

Код функции для удаления плана тренировок и питания тренером

|  |
| --- |
| private void ExecuteDeleteNutritionPlan(object parameter)  {  if (SelectedNutritionPlan != null)  {  var result = MessageBox.Show(  string.Format(Application.Current.FindResource("ConfirmDeleteNutritionPlan") as string, SelectedNutritionPlan.Title),  Application.Current.FindResource("ConfirmationDelete") as string,  MessageBoxButton.YesNo,  MessageBoxImage.Question);    if (result == MessageBoxResult.Yes)  {  try  {    int planId = SelectedNutritionPlan.Id;  var planToRemove = SelectedNutritionPlan;    SelectedNutritionPlan = null;    \_nutritionPlanService.Delete(planId);    NutritionPlans.Remove(planToRemove);    MessageBox.Show(  Application.Current.FindResource("NutritionPlanDeleteSuccess") as string,  Application.Current.FindResource("Success") as string,  MessageBoxButton.OK,  MessageBoxImage.Information);  }  catch (Exception ex)  {  MessageBox.Show(  string.Format(Application.Current.FindResource("NutritionPlanDeleteError") as string,  ex.Message, ex.InnerException?.Message),  Application.Current.FindResource("Error") as string,  MessageBoxButton.OK,  MessageBoxImage.Error);    LoadClientPlans();  }  }  }  } |

Код функции для регистрации пользователя

|  |
| --- |
| private void ExecuteRegisterCommand(object parameter)  {  var passwordBox = parameter as PasswordBox;  var confirmPasswordBox = passwordBox?.Tag as PasswordBox;    if (passwordBox == null || confirmPasswordBox == null)  {  return;  }    RegPassword = passwordBox.Password;  ConfirmPassword = confirmPasswordBox.Password;    if (string.IsNullOrEmpty(RegLogin) || string.IsNullOrEmpty(RegPassword) ||  string.IsNullOrEmpty(ConfirmPassword) || string.IsNullOrEmpty(Email) ||  string.IsNullOrEmpty(FullName))  {  ShowWarning("FillAllFields");  return;  }    if (RegPassword != ConfirmPassword)  {  ShowWarning("PasswordsMismatch");  return;  }    if (!IsValid(RegLogin, @"^[a-zA-Z0-9]{3,15}$", "InvalidUsername"))  return;    if (!IsValid(Email, @"^[a-zA-Z0-9.\_%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$", "InvalidEmail"))  return;    if (!IsValid(RegPassword, @"^.{4,}$", "InvalidPassword"))  return;    if (!Regex.IsMatch(FullName, @"^[а-яА-Яa-zA-ZёЁ\s]+$"))  {  ShowWarning("FullNameOnlyLetters");  return;  }    string[] nameParts = FullName.Trim().Split(new[] { ' ' }, StringSplitOptions.RemoveEmptyEntries);  if (nameParts.Length != 3)  {  ShowWarning("FullNameRequireThreeWords");  return;  }    bool isLoginUnique = true;  bool isEmailUnique = true;    if (\_databaseConnectionService.IsDatabaseExists())  {  try  {  isLoginUnique = \_userService.IsLoginUnique(RegLogin);  isEmailUnique = \_userService.IsEmailUnique(Email);  }  catch (Exception ex)  {  System.Diagnostics.Debug.WriteLine($"Ошибка при проверке уникальности в БД: {ex.Message}");  }  }    if (!isLoginUnique || \_users.Any(u => u.Login == RegLogin))  {  ShowWarning("LoginAlreadyExists");  return;  }    if (!isEmailUnique || \_users.Any(u => u.Email == Email))  {  ShowWarning("EmailAlreadyExists");  return;  }    var newUser = new User(FullName, Email, RegLogin, RegPassword, UserRole.Client);    if (\_databaseConnectionService.IsDatabaseExists())  {  try  {  int userId = \_userService.Add(newUser);  if (userId > 0)  {    System.Diagnostics.Debug.WriteLine($"Пользователь успешно сохранен в БД с ID: {userId}");  MessageBox.Show((string)Application.Current.Resources["RegistrationSuccess"],  (string)Application.Current.Resources["SuccessTitle"],  MessageBoxButton.OK, MessageBoxImage.Information);    roleName = GetRoleNameInCurrentLanguage(newUser.Role);  welcomeMessage = string.Format(  (string)Application.Current.Resources["WelcomeUser"],  newUser.Login,  roleName);    MessageBox.Show(  welcomeMessage,  (string)Application.Current.Resources["SuccessTitle"],  MessageBoxButton.OK,  MessageBoxImage.Information);    OpenMainWindow(newUser);  return;  }  }  catch (Exception ex)  {  System.Diagnostics.Debug.WriteLine($"Ошибка при сохранении пользователя в БД: {ex.Message}");  }  }    \_users.Add(newUser);    MessageBox.Show((string)Application.Current.Resources["RegistrationSuccess"],  (string)Application.Current.Resources["SuccessTitle"],  MessageBoxButton.OK, MessageBoxImage.Information);    roleName = GetRoleNameInCurrentLanguage(newUser.Role);  welcomeMessage = string.Format(  (string)Application.Current.Resources["WelcomeUser"],  newUser.Login,  roleName);    MessageBox.Show(  welcomeMessage,  (string)Application.Current.Resources["SuccessTitle"],  MessageBoxButton.OK,  MessageBoxImage.Information);    OpenMainWindow(newUser);  } |

Код функции для добавления абонемента

|  |
| --- |
| public void ExecuteSaveCommand(object parameter)  {  try  {  List<string> validationErrors = new List<string>();    string namePattern = @"^[a-zA-Zа-яА-Я\s]+$";  if (string.IsNullOrEmpty(Name?.Trim()))  {  validationErrors.Add((string)Application.Current.Resources["NameRequired"]);  }  else if (!Regex.IsMatch(Name, namePattern))  {  validationErrors.Add((string)Application.Current.Resources["InvalidName"]);  }  decimal priceValue = 0;  if (string.IsNullOrEmpty(Price?.Trim()))  {  validationErrors.Add((string)Application.Current.Resources["PriceRequired"]);  }  else if (!decimal.TryParse(Price, out priceValue) || priceValue < 0)  {  validationErrors.Add((string)Application.Current.Resources["InvalidPrice"]);  }  if (string.IsNullOrEmpty(Description?.Trim()))  {  validationErrors.Add((string)Application.Current.Resources["EnterDescription"]);  }    if (string.IsNullOrEmpty(ImagePath?.Trim()))  {  validationErrors.Add((string)Application.Current.Resources["EmptyImagePath"]);  }  if (string.IsNullOrEmpty(Duration?.Trim()))  {  validationErrors.Add((string)Application.Current.Resources["EmptyDuration"]);  }    if (string.IsNullOrEmpty(SubscriptionType?.Trim()))  {  validationErrors.Add((string)Application.Current.Resources["EmptySubscriptionType"]);  }    if (validationErrors.Count > 0)  {  string errorList = string.Join("\n- ", validationErrors);  errorList = "- " + errorList;    string message = string.Format(  (string)Application.Current.Resources["ValidationSummary"],  errorList);    MessageBox.Show(  message,  (string)Application.Current.Resources["ValidationErrorTitle"],  MessageBoxButton.OK,  MessageBoxImage.Warning);  return;  }  NewSubscription = new Subscription  {  Name = Name.Trim(),  Price = priceValue,  Description = Description.Trim(),  ImagePath = ImagePath,  Duration = Duration,  SubscriptionType = SubscriptionType,  Reviews = new List<Review>()  };  int subscriptionId = \_subscriptionService.Add(NewSubscription);    if (subscriptionId > 0)  {  NewSubscription.Id = subscriptionId;    MessageBox.Show(  (string)Application.Current.Resources["SubscriptionAddedSuccess"],  (string)Application.Current.Resources["SuccessTitle"],  MessageBoxButton.OK,  MessageBoxImage.Information);  CloseRequested?.Invoke(true, NewSubscription);  }  else  {  MessageBox.Show(  (string)Application.Current.Resources["ErrorSavingSubscription"],  (string)Application.Current.Resources["ErrorTitle"],  MessageBoxButton.OK,  MessageBoxImage.Error);  }  }  catch (Exception ex)  {  MessageBox.Show(  $"{(string)Application.Current.Resources["ErrorSavingSubscription"]}: {ex.Message}",  (string)Application.Current.Resources["ErrorTitle"],  MessageBoxButton.OK,  MessageBoxImage.Error);  }  } |

Код функции для удаления отзыва

|  |
| --- |
| private void ExecuteDeleteReview(object parameter)  {  if (parameter is int reviewId)  {  var result = MessageBox.Show((string)Application.Current.Resources["DeleteReviewConfirm"],  (string)Application.Current.Resources["DeleteConfirmTitle"],  MessageBoxButton.YesNo, MessageBoxImage.Question);  if (result == MessageBoxResult.Yes)  {  try  {    string currentUserName = \_mainWindow?.\_user?.Login;  bool isCurrentUserReview = false;    var reviewToRemove = Reviews.FirstOrDefault(r => r.Id == reviewId);  if (reviewToRemove != null)  {  if (!string.IsNullOrEmpty(currentUserName))  {  isCurrentUserReview = string.Equals(reviewToRemove.User, currentUserName, StringComparison.OrdinalIgnoreCase);  }    \_reviewService.Delete(reviewId);    Reviews.Remove(reviewToRemove);    if (isCurrentUserReview)  {  HasUserReviewed = false;  }    if (\_currentSubscription.Reviews != null)  {  var subReview = \_currentSubscription.Reviews.FirstOrDefault(r => r.Id == reviewId);  if (subReview != null)  {  \_currentSubscription.Reviews.Remove(subReview);  }  }    RecalculateRating();    var subscriptionInCollection = \_subscriptions.FirstOrDefault(s => s.Id == \_currentSubscription.Id);  if (subscriptionInCollection != null)  {  subscriptionInCollection.Rating = Rating;  }    \_currentSubscription.Rating = Rating;  bool updateSuccess = \_subscriptionService.Update(\_currentSubscription);    \_justDeletedReview = true;  \_lastDeletedReviewId = reviewId;    ReviewDeleted?.Invoke(this, reviewToRemove);    var updatedSubscriptions = \_subscriptionService.GetAll().ToList();  \_subscriptions = new ObservableCollection<Subscription>(updatedSubscriptions);    \_mainWindow.UpdateUIWithSubscriptions(\_subscriptions.ToList());    MessageBox.Show((string)Application.Current.Resources["ReviewDeletedSuccessfully"],  (string)Application.Current.Resources["SuccessTitle"],  MessageBoxButton.OK, MessageBoxImage.Information);    LoadReviews();  }  else  {  \_reviewService.Delete(reviewId);    var freshSubscription = \_subscriptionService.GetById(\_currentSubscription.Id);  if (freshSubscription != null)  {  \_currentSubscription = freshSubscription;  RecalculateRating();    \_currentSubscription.Rating = Rating;  \_subscriptionService.Update(\_currentSubscription);    var subscriptionInCollection = \_subscriptions.FirstOrDefault(s => s.Id == \_currentSubscription.Id);  if (subscriptionInCollection != null)  {  subscriptionInCollection.Rating = Rating;  }    var updatedSubscriptions = \_subscriptionService.GetAll().ToList();  \_subscriptions = new ObservableCollection<Subscription>(updatedSubscriptions);  \_mainWindow.UpdateUIWithSubscriptions(\_subscriptions.ToList());  }    LoadReviews();  }  }  catch (Exception ex)  {  MessageBox.Show($"{(string)Application.Current.Resources["ErrorDeletingReview"]}: {ex.Message}",  (string)Application.Current.Resources["ErrorTitle"],  MessageBoxButton.OK, MessageBoxImage.Error);  }  finally  {  \_justDeletedReview = false;  }  }  }  } |