**КОМИТЕТ ПО НАУКЕ И ВЫСШЕЙ ШКОЛЕ СПБ**

Санкт-Петербургское государственное бюджетное профессиональное образовательное учреждение

«Санкт-Петербургский технический колледж управления и коммерции»

**КУРСОВОЙ ПРОЕКТ**

|  |
| --- |
| по МДК 01.01 Разработка программных модулей |
| на тему: Разработка информационной системы по для автоматизации |
| работы фитнесс-клуба |
|  |

Специальность: 09.02.07 Информационные системы и программирование

|  |  |
| --- | --- |
| Студент: | Ефимов Е.Е / / |
| Группа: | 9ПО-32 |
| Преподаватель: | Смирнова А.В. / / |
| Оценка: | / / |

Санкт-Петербург

2025 г.

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ **Ошибка! Закладка не определена.**](#_Toc197303179)

[1 ПРОЕКТИРОВАНИЕ ИНФОРМАЦИОННОЙ СИСТЕМЫ 5](#_Toc197303180)

[1.1 Постановка задачи 5](#_Toc197303181)

[1.2 Проектирование функциональных возможностей системы 7](#_Toc197303182)

[1.3 Разработка концептуальной и логической модели данных 8](#_Toc197303183)

[2 РАЗРАБОТКА ИНФОРМАЦИОННОЙ СИСТЕМЫ 10](#_Toc197303184)

[2.1 Выбор средств и среды разработки 10](#_Toc197303185)

[2.2 Реализация базы данных 11](#_Toc197303186)

[2.3 Разработка пользовательского приложения 12](#_Toc197303187)

[2.3.1 Авторизация и регистрация 12](#_Toc197303188)

[2.3.2 Функционал приложения с ролью «Клиент» 15](#_Toc197303189)

[2.3.3 Функционал приложения с ролью «Тренер» 34](#_Toc197303190)

[2.3.4 Функционал приложения с ролью «Администратор» 43](#_Toc197303191)

[2.4 Тестирование разработанной автоматизированной системы 55](#_Toc197303192)

[ЗАКЛЮЧЕНИЕ 64](#_Toc197303193)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ И ЛИТЕРАТУРЫ 65](#_Toc197303194)

[ПРИЛОЖЕНИЕ 66](#_Toc197303195)

# ВВЕДЕНИЕ

В современном мире эффективное управление фитнес-центром требует не только высокого качества предоставляемых услуг, но и автоматизации внутренних процессов, таких как: учет клиентов, управление расписанием занятий, контроль абонементов, услуг и финансовых транзакций. Внедрение информационной системы позволяет организовать все процессы структурированно и без лишней нагрузки на сотрудников.

Ранее многие процессы, включая ведение клиентской базы, составление расписания и учет абонементов, осуществлялись вручную или с помощью разрозненных таблиц и бумажных журналов, что нередко приводило к ошибкам, потере данных и замедлению работы персонала. Сегодня современные технологии позволяют реализовать эти процессы в электронном виде, что обеспечивает надежность хранения информации, повышает скорость доступа к ней и минимизирует влияние человеческого фактора.

С развитием программного обеспечения появилась возможность создавать специализированные приложения для администрирования всех аспектов фитнес-центра. Это позволяет оптимизировать нагрузку на персонал, своевременно обрабатывать запросы клиентов, эффективно управлять тренерским составом и контролировать предоставление услуг.

Автоматизация процессов в фитнес-центре положительно сказывается на уровне сервиса, упрощает работу администратора, тренеров и клиентов, способствует росту клиентской базы и повышает конкурентоспособность организации.

Для достижения поставленной цели необходимо решить следующие задачи:

1. Провести анализ предметной области;
2. Спроектировать структуру и взаимодействие компонентов информационной системы;
3. Разработать программное обеспечение;
4. Провести тестирование приложения.

# ПРОЕКТИРОВАНИЕ ИНФОРМАЦИОННОЙ СИСТЕМЫ

## Постановка задачи

Разрабатываемая информационная система предназначена для автоматизации работы фитнес-центра.

Деятельность организации включает в себя:

* Предоставление групповых и индивидуальных тренировок;
* Продажу и учёт абонементов;
* Предоставление дополнительных услуг (массаж, аренда шкафчиков и др.);
* Управление персоналом (тренерами);
* Ведение расписания занятий и графиков работы;
* Финансовые операции: оплата услуг, возвраты, пополнение баланса.

Фитнес-центр обслуживает большое количество клиентов, каждый из которых может записываться на различные занятия, арендовать шкафчики, приобретать абонементы и заказывать индивидуальные услуги. Для этого необходимо поддерживать в актуальном состоянии всю информацию о клиентах, занятиях, оплатах, услугах и персонале.

В ручном режиме ведение всех этих процессов становится неэффективным и подверженным ошибкам. Возникают трудности с контролем расписаний, учётом посещаемости, своевременной оплатой, возвратами и доступностью услуг. Поэтому необходима автоматизация этих процессов с использованием информационной системы.

Входными данными системы являются:

* Информация о клиентах, тренерах и администраторах;
* Графики работы тренеров;
* Занятия (их типы, стоимость, расписание);
* Абонементы и услуги;
* Финансовые операции и транзакции.

Выходными данными являются:

* Расписания занятий для клиентов и тренеров;
* История посещений;
* Счета на оплату;
* Отчёты о транзакциях;
* Отзывы клиентов;
* Индивидуальные планы тренировок.

Таким образом, в курсовом проекте необходимо разработать автоматизированную систему управления фитнес-центром, включающую учёт клиентов, занятий, тренеров, расписаний и оплат.

Назначение и цели создания информационной системы:

Автоматизированная система должна:

* Повысить качество обслуживания клиентов;
* Минимизировать ошибки при ведении расписаний и оплат;
* Обеспечить прозрачность финансовых операций;
* Снизить нагрузку на персонал;
* Упростить доступ к информации о занятиях и услугах.

Функциональные возможности системы:

* Авторизация и регистрация пользователей по ролям (клиент, тренер, администратор);
* Просмотр и редактирование расписаний;
* Запись на индивидуальные и групповые занятия;
* Ведение истории посещений;
* Работа с абонементами и услугами;
* Финансовые операции (оплата, возвраты, история транзакций);
* Управление пользователями, оборудованием и занятиями (для администратора);
* Отправка индивидуальных планов и просмотр отзывов (для тренеров);
* Модерация отзывов (администратором).

## Проектирование функциональных возможностей системы

В соответствии с постановкой задачи были определены три типа пользователей: клиент, тренер и администратор. Функциональные возможности информационной системы представлены в виде use-case диаграммы (см. Рисунок 1)

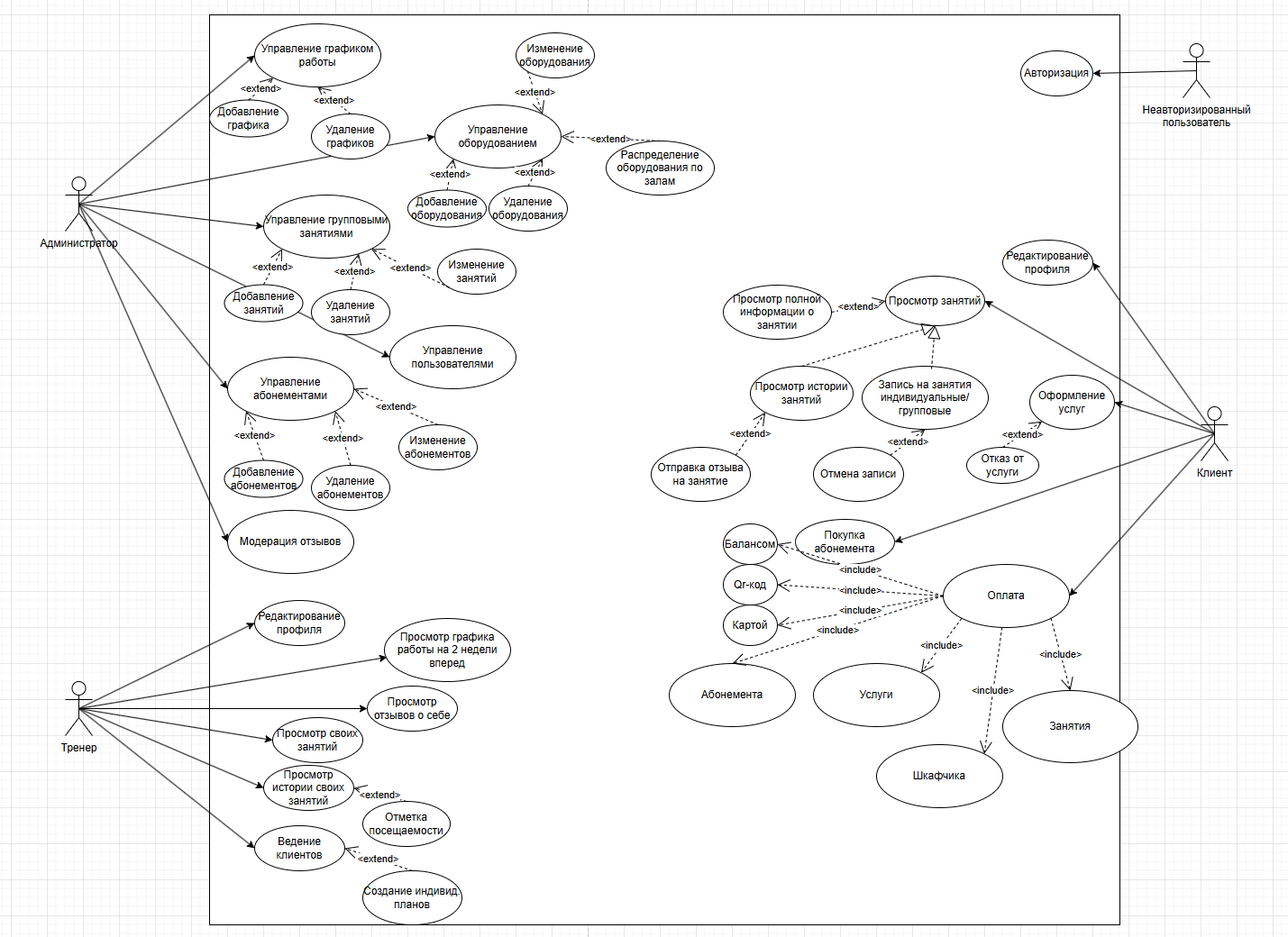


Рисунок 1 – Функциональная схема приложения

## Разработка концептуальной и логической модели данных

Концептуальное (инфологическое) проектирование – построение сематической модели предметной области, то есть информационной модели наиболее высоко уровня абстракции. Такая модель создается без ориентации на какую-либо конкретную СУБД и модель данных. Термины «сематическая модель», «концептуальная модель» и «инфологическая модель» являются синонимами. Кроме того, в этом контекст равноправно могут использоваться слова «модель базы данных» и «модель предметной области», поскольку такая модель является как образом реальности, так и образом проектируемой базы данных для этой реальности.

Конкретный вид и содержание концептуальной модели базы данных определяется выбранным для этого формальным аппаратом. Обычно используются графические нотации, подобные ER-диаграммам.

На основе анализа предметной области была разработана концептуальная модель данных (см. Рисунок 2)

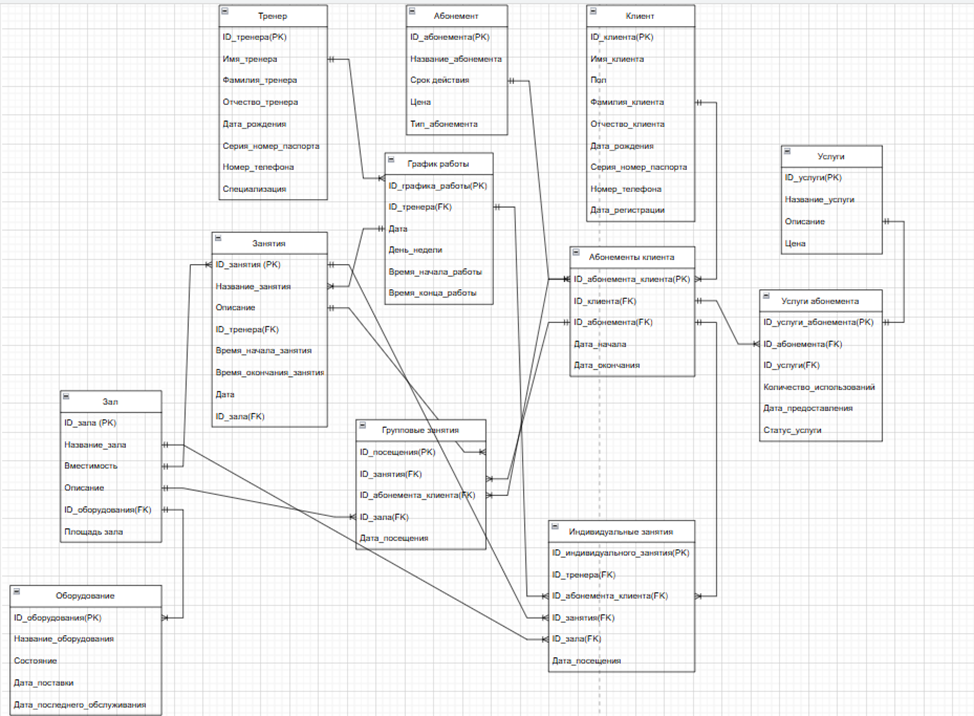


Рисунок 2 – Инфологическая схема

Полный скрипт на создание базы данных представлен в Приложении А.

# РАЗРАБОТКА ИНФОРМАЦИОННОЙ СИСТЕМЫ

## Выбор средств и среды разработки

Хранение данных в информационной системе осуществляется с использованием **системы управления базами данных PostgreSQL**, которая является одной из наиболее надежных, гибких и активно развивающихся СУБД с открытым исходным кодом. Для взаимодействия с базой данных используется графическая среда **pgAdmin 4**, предоставляющая удобные средства администрирования, построения запросов, анализа и визуализации структуры БД.

Создание и взаимодействие с базой данных на стороне приложения реализовано через технологию **Entity Framework (Code First)** — объектно-реляционный маппер (ORM) для .NET. Он позволяет описывать структуру базы данных с помощью C#-классов и аннотаций (Data Annotations), обеспечивая автоматическую генерацию и миграции базы данных без необходимости написания SQL-скриптов вручную. Такой подход ускоряет разработку и уменьшает количество дублирующего кода.

Программная логика и пользовательский интерфейс реализованы на языке **C#** с использованием технологии **Windows Presentation Foundation (WPF)**. WPF предоставляет современные возможности построения гибких и адаптивных интерфейсов с использованием XAML-разметки. Она позволяет легко отделять визуальную составляющую от логики, реализовывать шаблоны (MVVM) и работать с привязками данных (data binding).

Средой разработки выступает **Microsoft Visual Studio 2022**, предоставляющая все необходимые инструменты: редактор кода, отладчик, визуальные дизайнеры, управление пакетами NuGet и поддержку интеграции с системами контроля версий. Эта среда обеспечивает стабильную и продуктивную разработку приложений под платформу .NET.

## Реализация базы данных

В рамках проекта была разработана база данных **FitnessClub** на платформе PostgreSQL. База данных содержит таблицы, отражающие основные сущности предметной области: клиенты, тренеры, занятия, абонементы, оплаты, отзывы, услуги, расписания и др.

Каждая таблица содержит поля различных типов данных: строки (varchar, text), числа (integer, numeric), даты (date), логические значения (boolean) и др. В модели предусмотрены связи один-к-одному, один-ко-многим и многие-ко-многим через промежуточные таблицы.

База данных формируется автоматически на основе C#-моделей с аннотациями, используемыми в Entity Framework. Это упрощает процесс внесения изменений в структуру и синхронизацию модели данных с приложением.

**На рисунке 3 представлена ER-диаграмма базы данных**, отражающая связи между основными таблицами и ключевыми атрибутами (см. Рисунок 3)

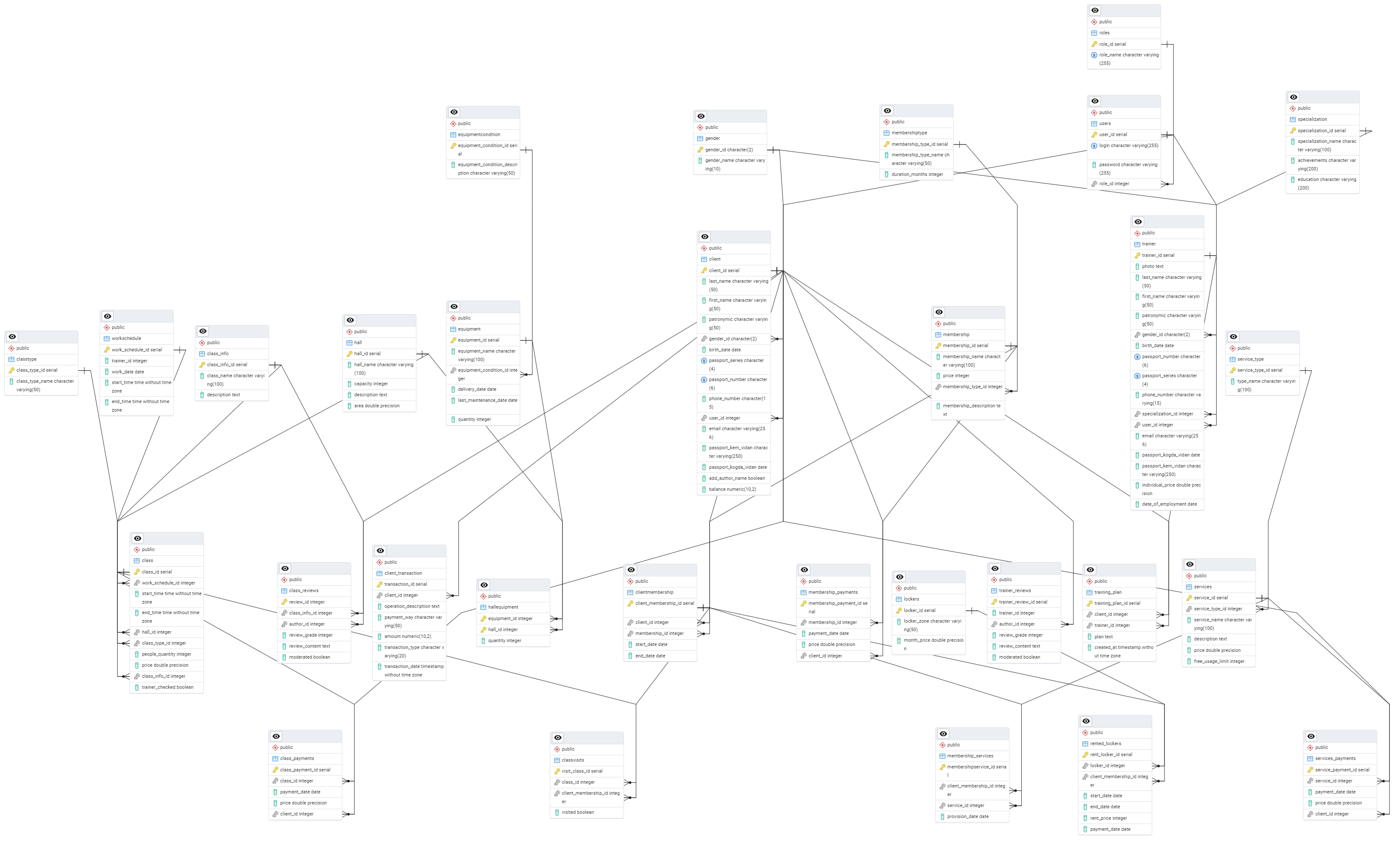


Рисунок 3 – Диаграмма базы данных

Полный скрипт на создание базы данных и таблиц представлен в приложении А.

## Разработка пользовательского приложения

### Авторизация и регистрация

При запуске программы пользователю отображается **форма авторизации** (см. Рисунок 4), предназначенная для входа в систему по учетным данным. Для авторизации необходимо ввести логин и пароль, которые были автоматически сгенерированы системой при регистрации и отправлены на электронную почту пользователя, либо заданные вручную в форме редактирования профиля.

Если пользователь не зарегистрирован в системе, он может перейти на форму регистрации (см. Рисунок 5) с помощью соответствующей кнопки. В процессе регистрации пользователь указывает свои контактные данные: имя, фамилию, отчество, электронную почту, номер телефона, пол, дата рождения, серия паспорта, номер паспорта, кем выдан паспорт, когда выдан паспорт и выбор роли. Кроме того, при попытке регистрации как тренер, пользователь должен ввести кодовое слово — при его некорректном вводе отображается сообщение об ошибке, и регистрация отклоняется. Клиент регистрируется без кодового слова.

После регистрации учетная запись автоматически сохраняется в базе данных:

* Основные данные (логин, хешированный пароль, роль) — в таблице users;
* Контактная информация клиента — в таблице client;
* Контактная информация и специализация тренера — в таблице trainer (если выбрана соответствующая роль).

Логин и пароль, сгенерированные системой, отправляются на электронную почту, указанную при регистрации. Пароль шифруется с использованием SHA512 до сохранения в базу данных.

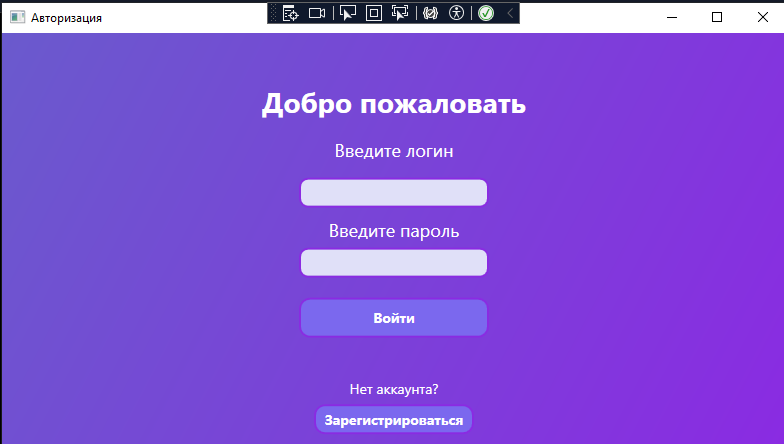


Рисунок 4 – Окно авторизации

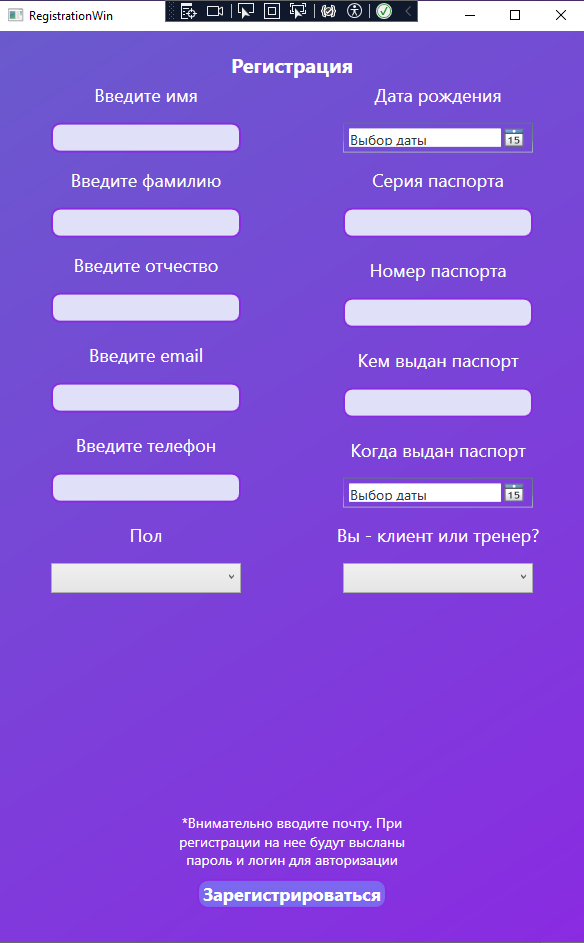


Рисунок 5 – Окно регистрации

Идентификация пользователя в системе осуществляется при выполнении следующих условий:

* регистрационные данные, включая электронную почту, имя, фамилию и номер телефона, введены корректно, пароль подтвержден без ошибок и успешно добавлен в базу данных. После завершения регистрации пользователь автоматически перенаправляется на основную форму, соответствующую его роли (ClientPage, TrainerPage или AdminPage). В случае ошибок отображается окно с предупреждением о некорректном вводе и предоставляется повторная попытка;
* авторизация осуществляется по ранее сгенерированному логину и паролю. Если введённые данные совпадают с записями в таблице users, система предоставляет доступ и открывает соответствующую форму. При несовпадении данных появляется уведомление об ошибке и возможность повторного ввода данных;
* система не допускает использование одного и того же логина для разных пользователей. Проверка уникальности логина осуществляется до добавления записи в базу данных.

В данной системе реализована авторизация по ролям для разграничения доступа к функциям. Это предотвращает возможность несанкционированного изменения или удаления данных. Всего предусмотрено три роли: клиент, тренер и администратор. В зависимости от типа пользователя система открывает соответствующий интерфейс с нужным набором возможностей.

После успешной авторизации клиенту отображается расписание занятий и доступ к функциям личного кабинета, тренеру — расписание занятий, история посещений и профиль, администратору — интерфейс управления пользователями, оборудованием и расписаниями. (приложение Б, приложение В, приложение Г)

### Функционал приложения с ролью «Клиент»

Рассмотрим процесс работы при авторизации с ролью «Клиент». (см. Рисунок 6)

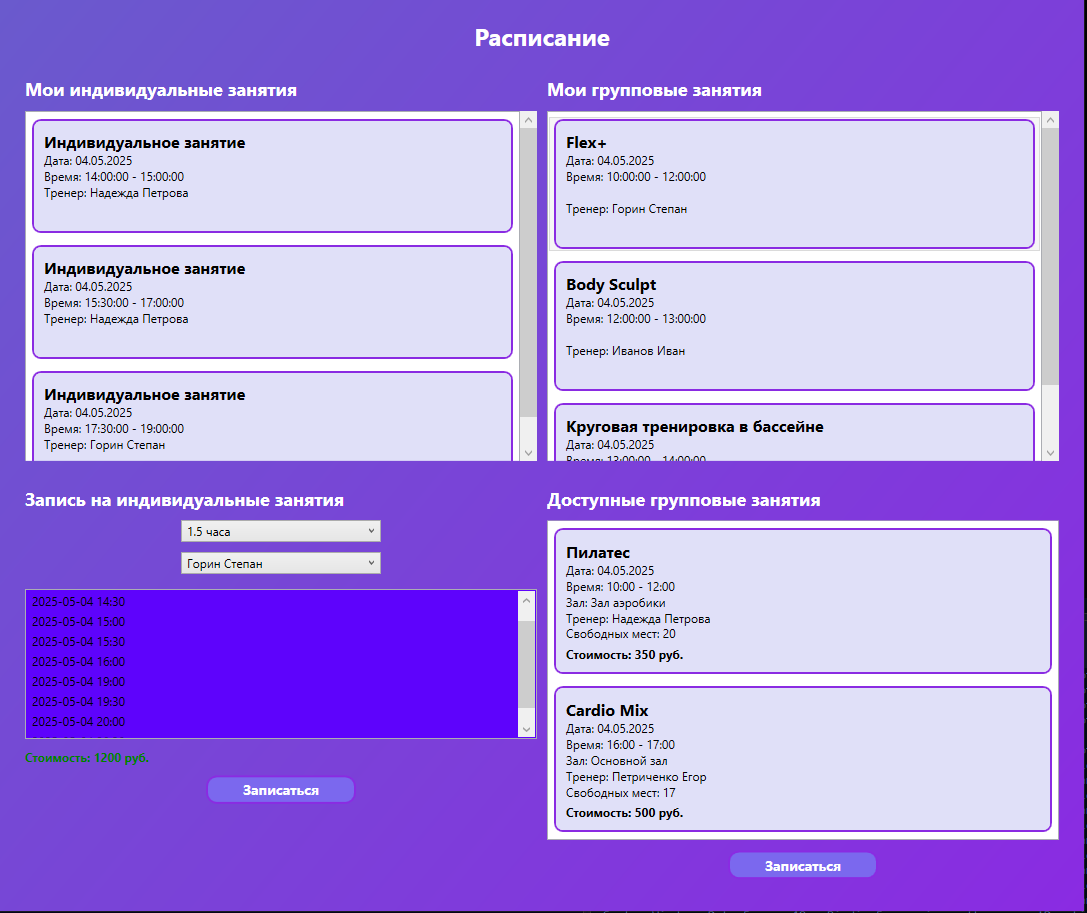


Рисунок 6 – Страница «Расписание занятий»

* На данной форме располагаются все индивидуальные и групповые занятия, на которые записан пользователь, а также доступные для записи тренировки. В верхней части формы представлены два блока:
* список «Мои индивидуальные занятия», где отображаются уже записанные индивидуальные тренировки с указанием даты, времени и имени тренера;
* список «Мои групповые занятия», где выводятся групповые тренировки, на которые пользователь записан.

Ниже расположены два раздела для записи:

* в блоке «Запись на индивидуальные занятия» пользователь может выбрать длительность занятия, тренера и доступный слот. Слоты, пересекающиеся по времени с уже существующими занятиями клиента или тренера, не отображаются. В случае конфликта времени выводится сообщение: «У вас уже есть занятие на это время»;
* в блоке «Доступные групповые занятия» отображаются доступные для записи тренировки с указанием названия, даты, времени, зала, тренера, количества свободных мест и стоимости.

Также на форме присутствуют кнопки для управления занятиями:

* Записаться на индивидуальное занятие;
* Записаться на групповое занятие.

При нажатии правой кнопкой мыши по записи в списке уже запланированных занятий появляется окно с возможностью:

* Просмотреть подробную информацию о занятии (см. Рисунок 7);
* Отменить занятие. В этом случае оно автоматически удаляется из страницы «Оплата» и не подлежит оплате клиентом. (приложение Д)

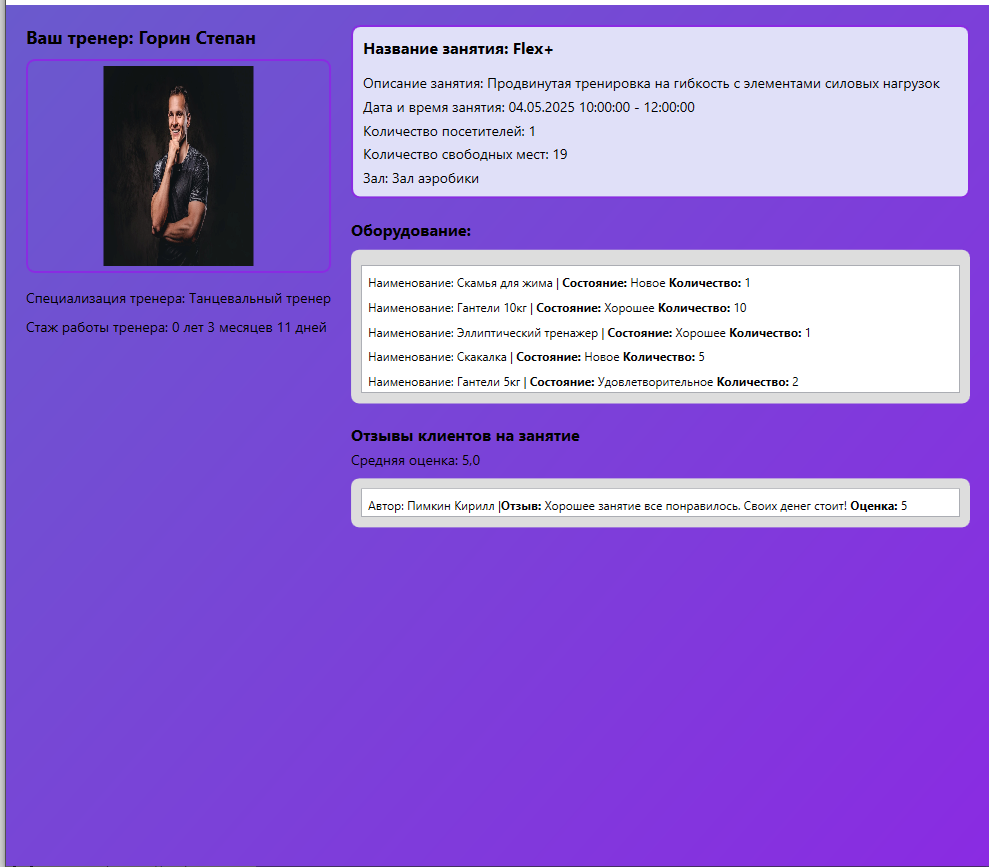


Рисунок 7 – Окно «Подробнее о занятии»

* В данном окне располагается следующая информация о занятии:  
  название занятия;
* описание занятия;
* дата и время занятия;
* количество посетителей;
* количество свободных мест;
* зал, в котором проходит занятие;
* оборудование в этом зале;
* отзывы клиентов на занятие;
* имя и фамилия тренера;
* фото тренера;
* специализация тренера;
* стаж работы тренера. Высчитывается по дате трудоустройства внутри логики окна;

На данной форме представлена страница «Профиль клиента» (см. Рисунок 8), где пользователь может просмотреть и при необходимости отредактировать определённые персональные данные.

В верхней части формы автоматически отображаются следующие поля, доступные только для чтения:

* фамилия, имя и отчество;
* пол;
* дата рождения;
* серия и номер паспорта;
* дата и место выдачи паспорта;
* роль пользователя.

Эти данные защищены от изменений и заполняются из базы данных при первичной регистрации клиента.

Ниже находятся редактируемые поля, доступные для изменения клиентом:

* номер телефона;
* адрес электронной почты;
* логин;
* новый пароль и подтверждение пароля;
* чекбокс для отображения имени в отзывах.

Для обеспечения безопасности данных реализована система валидации:

* телефон проверяется на корректный формат (+7XXXXXXXXXX или 8XXXXXXXXXX);
* email должен соответствовать стандартному формату электронной почты;
* логин должен содержать минимум 4 символа;
* при вводе пароля требуется повторное подтверждение, система проверяет совпадение двух полей;
* если введён новый пароль, он также проходит проверку на безопасность.

После заполнения полей пользователь нажимает кнопку «Сохранить изменения», при этом происходит проверка всех данных. В случае успеха информация сохраняется в базе данных и пользователю отображается соответствующее уведомление. Если введённые данные не соответствуют требованиям, программа выводит сообщение об ошибке с указанием причины и предоставляет возможность корректировки данных.

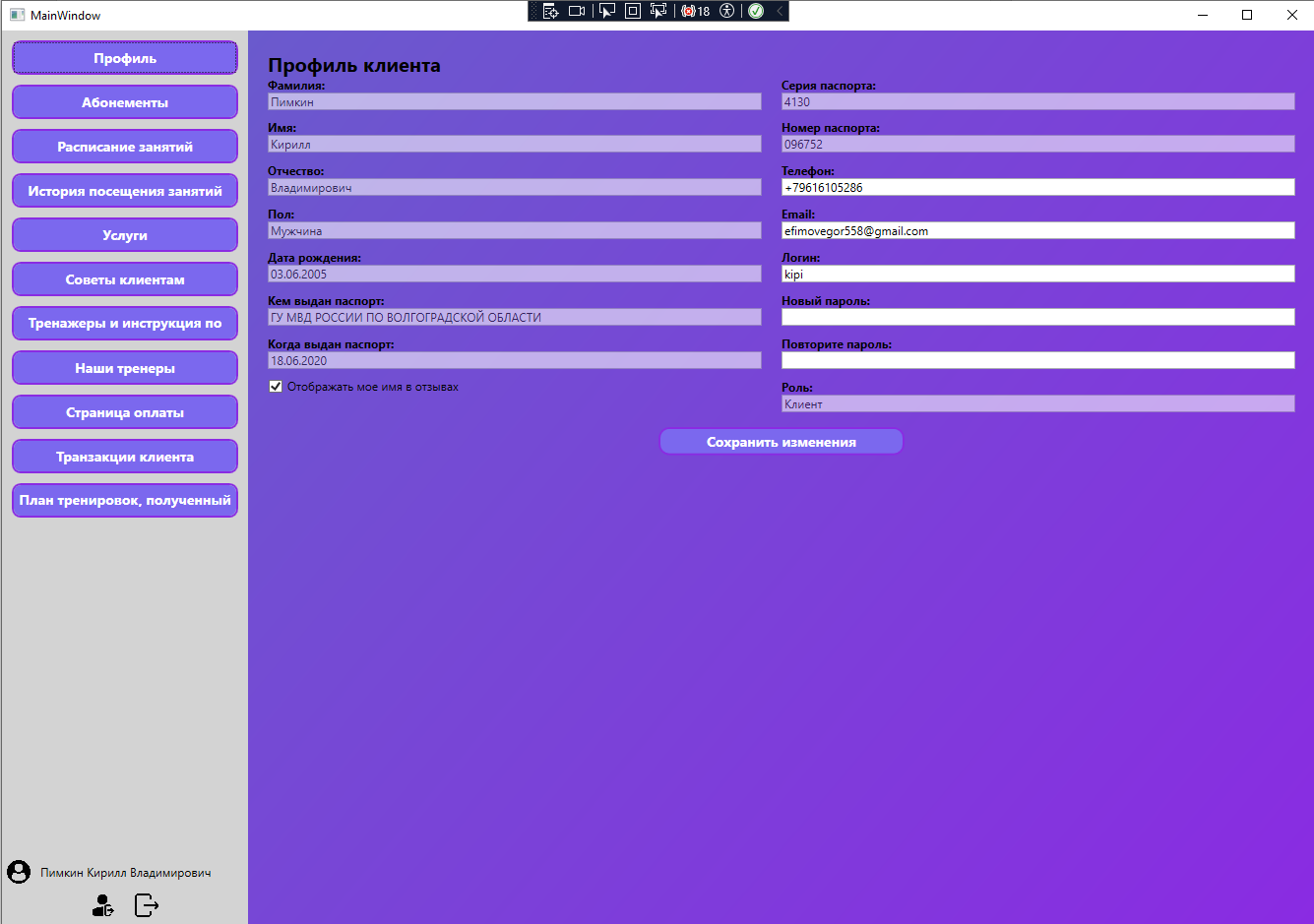


Рисунок 8 – Страница «Профиль клиента»

Листинг кода представлен в приложении И.

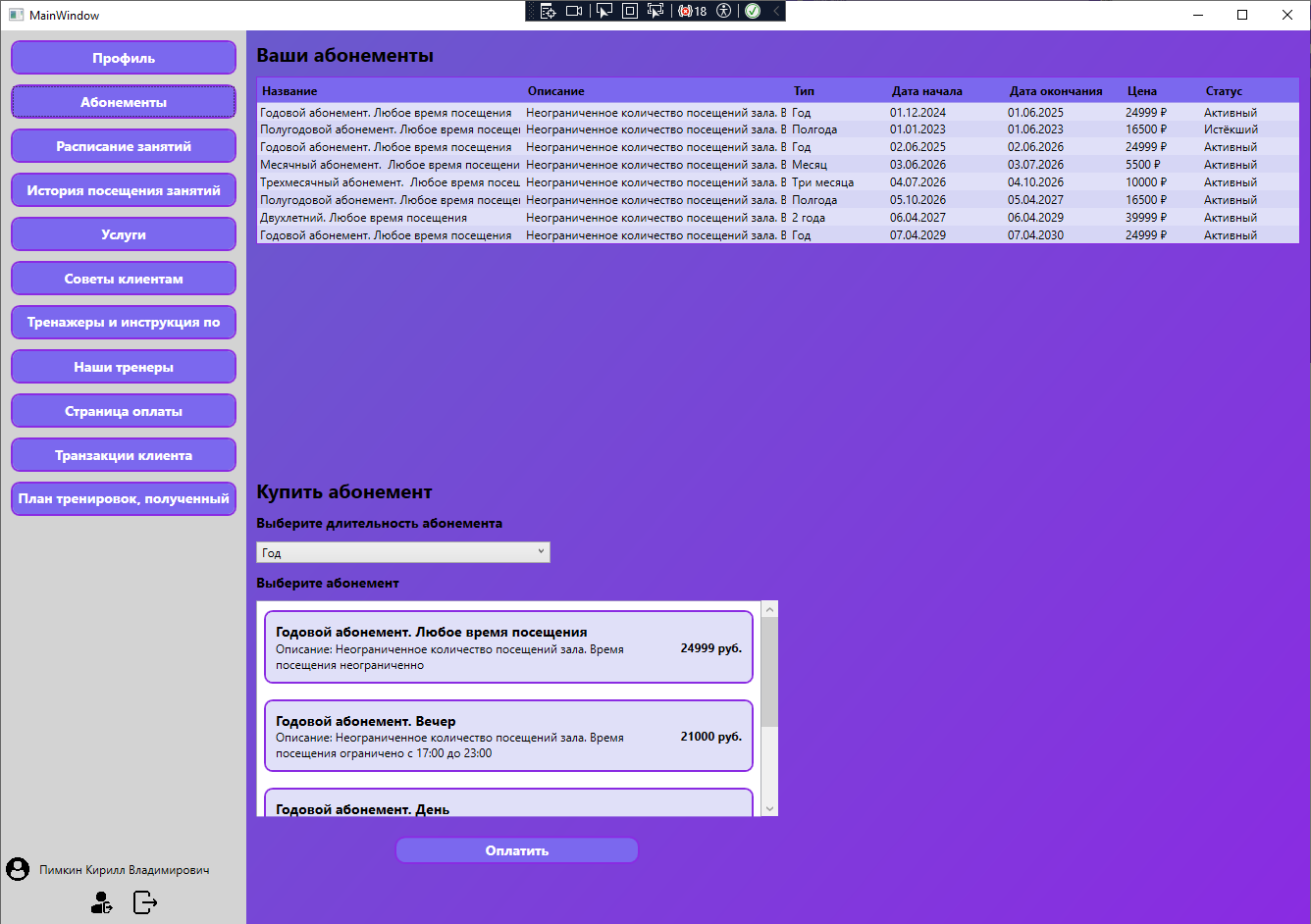


Рисунок 9 – Страница «Абонементы»

На данной форме представлена страница «Мои абонементы» (см. Рисунок 9), где клиент может просматривать все свои действующие, завершённые и приобретённые абонементы, а также оформить покупку нового.

Верхняя часть формы содержит таблицу с данными обо всех ранее приобретённых пользователем абонементах. Для каждого абонемента отображаются:

* название;
* краткое описание;
* тип (например, «Год», «Месяц»);
* дата начала и окончания действия;
* стоимость;
* текущий статус (например, «Активный», «Истёкший»).

Нижняя часть формы предназначена для оформления нового абонемента и включает следующие элементы:

* выпадающий список с возможностью выбора длительности абонемента (формируется на основе данных из таблицы типов абонементов, таких как: Месяц, Полгода, Год, и т.д.);
* список доступных к покупке абонементов по выбранной длительности с указанием названия, описания и стоимости;
* кнопка «Оплатить».

При нажатии на кнопку «Оплатить» выполняется следующая логика:

* если у клиента уже есть абонемент, то дата начала нового устанавливается на следующий день после даты окончания последнего абонемента;
* если активных или завершённых абонементов нет, новый абонемент начинает действовать с текущего дня;
* создаётся запись в таблице оплаты абонементов с соответствующими данными.

После этого пользователь может завершить оплату на отдельной странице «Оплата». Таким образом, обеспечивается последовательное действие абонементов без наложения и автоматическое планирование их срока (листинг кода в приложение Е)

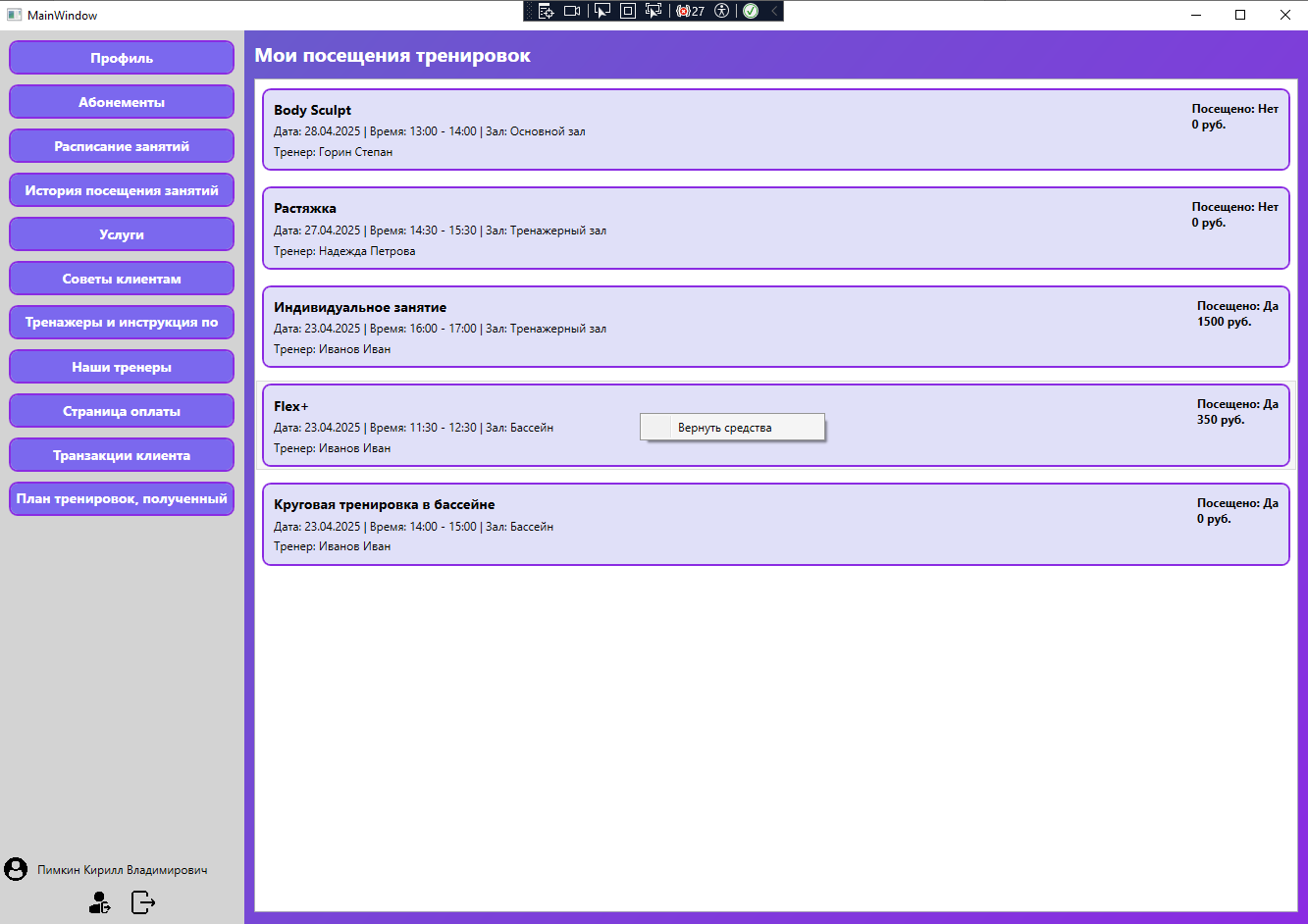


Рисунок 10 – Страница «История посещения занятий»

На данной форме (см. Рисунок 10) «История посещения занятий» отображаются все индивидуальные и групповые тренировки, на которые пользователь был записан, с указанием даты, времени, зала, имени тренера и информации об оплате. В правой части каждого блока занятия указано, было ли оно посещено и оплачено (например: «Посещено: Да / Нет»).

Дополнительные функции:

– При двойном щелчке мыши по занятию открывается окно подробной информации (см. Рисунок 7), в котором дополнительно становится доступна возможность оставить отзыв на занятие;

– Отзыв добавляется в базу и направляется на модерацию администратору. Только после прохождения модерации он может быть отображён другим пользователям;

– При нажатии правой кнопкой мыши по занятию вызывается контекстное меню, в котором доступна опция оформить возврат средств. Данная функция активна только в случае, если занятие не было посещено (то есть отсутствует отметка от тренера), иначе выдается ошибка возврата средств с уточнением причины. (листинг кода приложение Х)

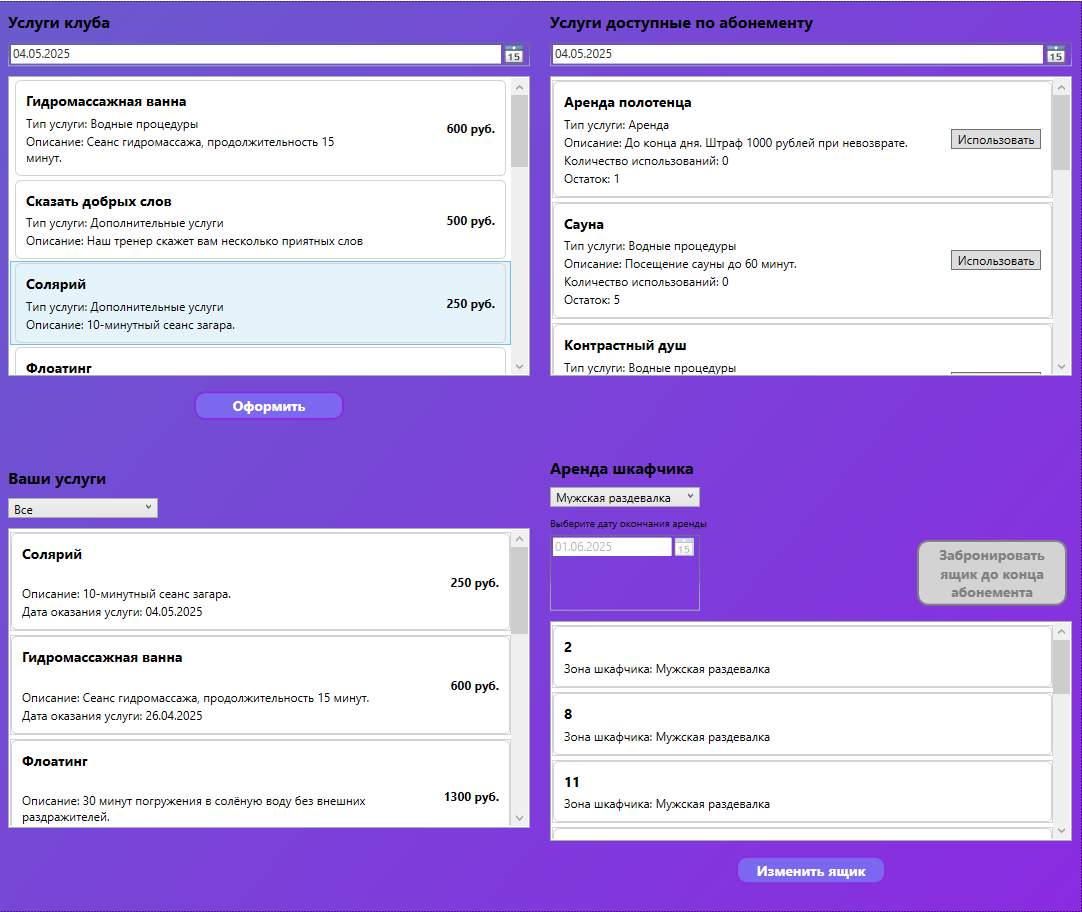


Рисунок 11 – Страница «Услуги»

На данной форме представлена функциональность взаимодействия клиента с клубными услугами и арендой шкафчиков. (см. Рисунок 11) Вся функциональность условно разбита на четыре секции:

В верхней части формы слева отображается список всех платных услуг клуба, доступных для оформления на выбранную дату. Каждая услуга включает тип, краткое описание и стоимость. После выделения нужной услуги пользователь может нажать кнопку «Оформить» — будет создана запись о предоставлении услуги. Если на выбранную дату клиент уже оформил аналогичную услугу, соответствующий элемент будет недоступен.

В правой части формы представлены услуги, доступные по абонементу. Пользователю показываются только те услуги, которые входят в его текущий активный абонемент и по которым ещё не исчерпан лимит использования. Для каждой услуги отображается краткое описание, оставшееся количество использований и кнопка «Использовать». После исчерпания лимита данная услуга перестаёт быть бесплатной и перемещается в список платных услуг, отображаемых в левой части.

В нижней левой части страницы расположен список «Ваши услуги». Он отображает все оформленные пользователем услуги с указанием их описания, стоимости и даты оказания. Над списком расположен выпадающий фильтр, позволяющий отфильтровать записи по признаку: все, использованные, не использованные. Для не использованных услуг возможен возврат средств — опция доступна через контекстное меню (ПКМ).

В нижней правой части формы реализована функциональность аренды шкафчиков. Пользователь может выбрать зону (мужская, женская, семейная), после чего отображается список доступных ячеек для аренды. Для каждой ячейки указана зона и номер.

Аренду можно оформить:

* либо на произвольную дату, не ранее текущей и не позднее даты окончания действующего абонемента;
* либо до конца срока действия последнего приобретённого абонемента при помощи кнопки «Забронировать ящик до конца абонемента».

Если шкафчик уже арендован клиентом, он может изменить только номер ячейки, дата окончания аренды остаётся неизменной. При этом кнопка аренды будет переименована в «Изменить ящик». Все данные обновляются в соответствующей таблице аренды. (листинг кода в приложение К)

Также у клиента есть 2 информационные страницы, в которых нет логики, лишь верстка, они представлены на рисунках 12 и 13(см. Рисунок 12, см. Рисунок 13)

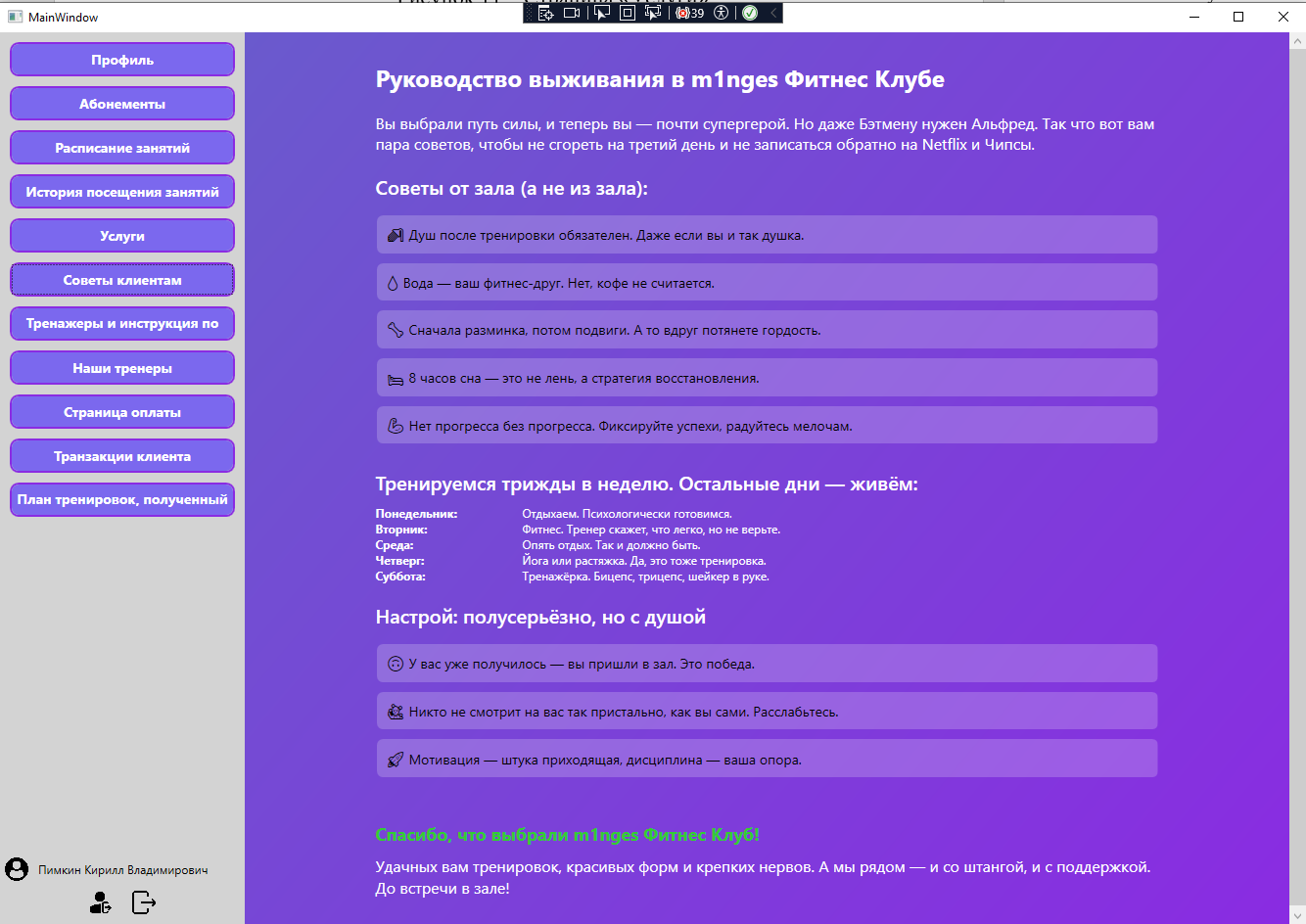


Рисунок 12 – Страница «Советы клиентам»

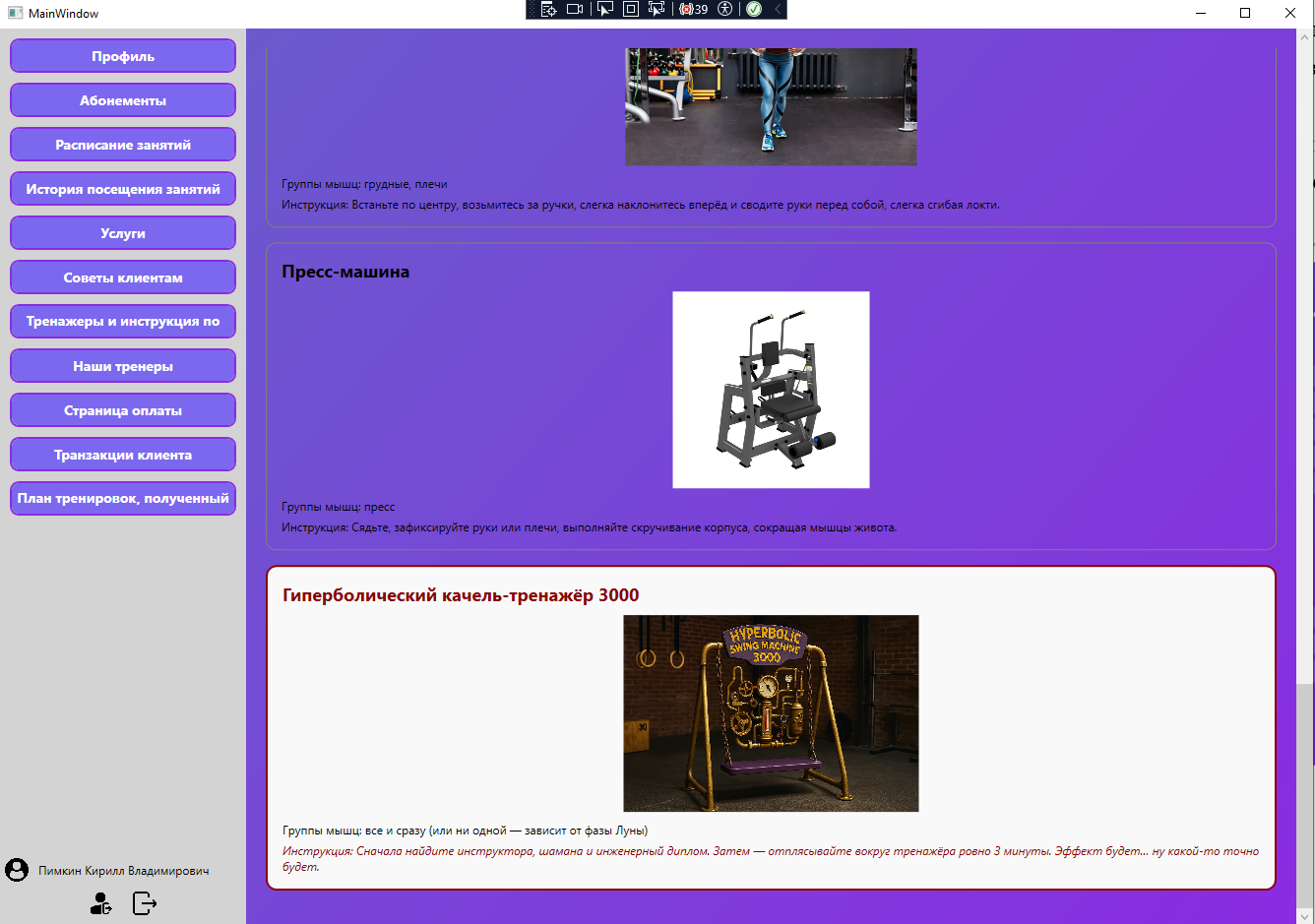


Рисунок 13 – Страница «Тренажеры и инструкция по   
эксплуатации»

Рассмотрим процесс взаимодействия со страницей «Наши тренеры». (см. Рисунок 14)

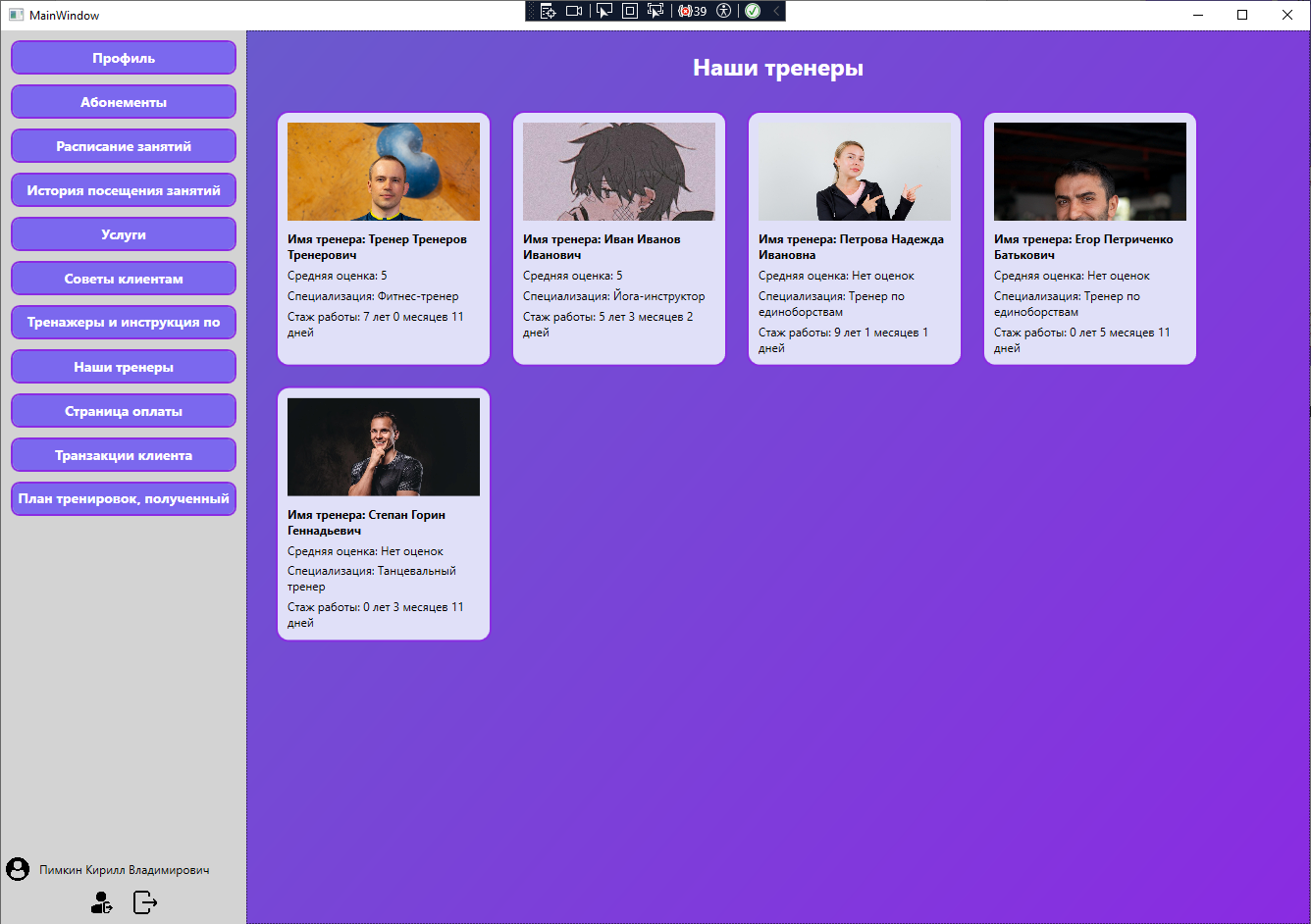


Рисунок 14 – Страница «Наши тренеры»

На данной форме отображаются все тренеры, зарегистрированные в системе. Для каждого тренера отображается следующая информация:

* полное имя тренера;
* средняя оценка по отзывам клиентов;
* специализация тренера;
* стаж работы, автоматически рассчитанный на основе даты трудоустройства. (листинг кода представлен в приложение Л)

Каждый тренер представлен в виде карточки с фотографией. При нажатии на карточку открывается окно с более подробной информацией о выбранном тренере (см. Рисунок 15).

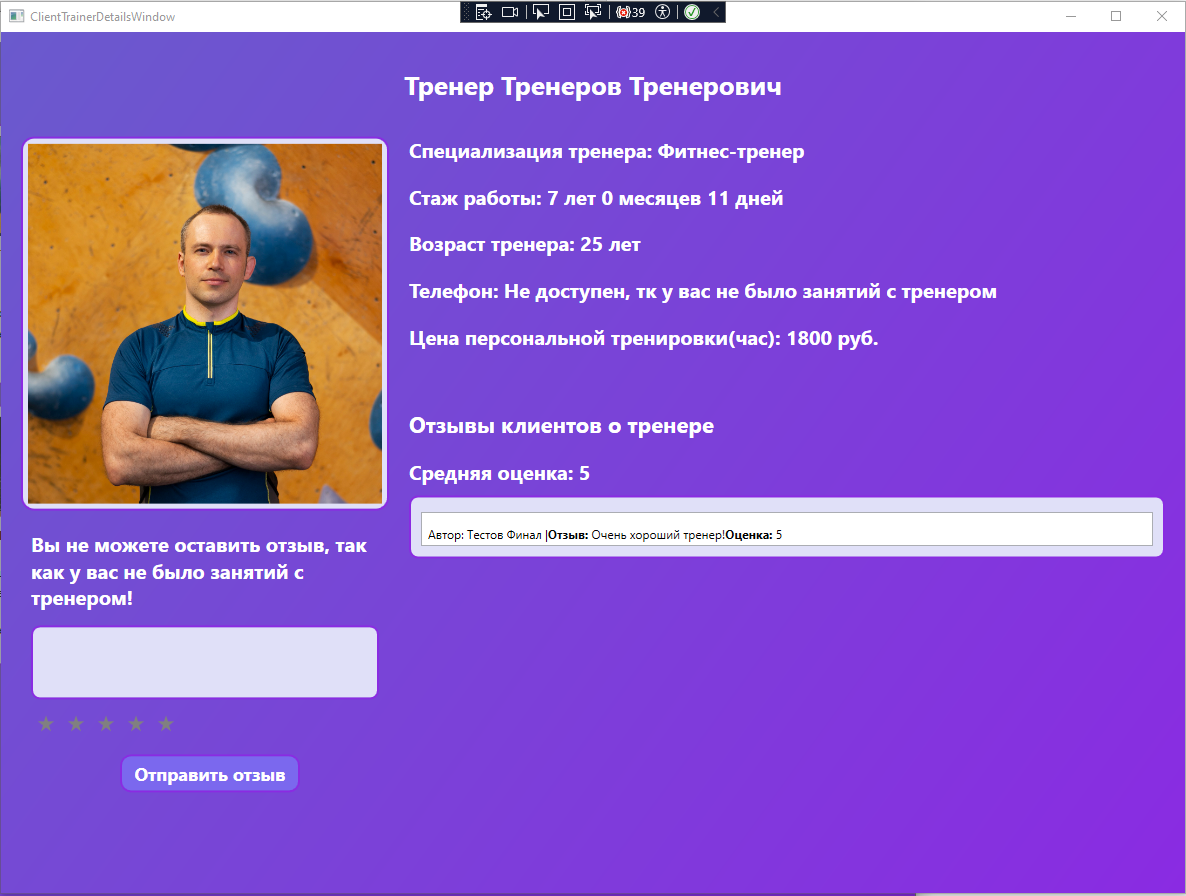


Рисунок 15 – Окно «Подробнее о тренере»

В данном окне представлены:

* фотография тренера;
* специализация;
* стаж работы в годах, месяцах и днях;
* возраст тренера, рассчитанный автоматически на основе даты рождения;
* стоимость персональной тренировки в час;
* отзывы клиентов, оставленные ранее.
* Если пользователь посещал хотя бы одну тренировку с выбранным тренером, ему становится доступна дополнительная информация:
* номер телефона тренера;
* возможность оставить отзыв с оценкой и комментарием. Оценка выставляется с помощью звёзд. Оставленный отзыв отправляется на модерацию администратору.

Если же клиент не посещал занятий с тренером, то информация о номере телефона скрыта, а поле ввода отзыва заблокировано с уведомлением: «Вы не можете оставить отзыв, так как у вас не было занятий с тренером».



Рисунок 16 – Страницы «Оплата задолжностей»

На представленной странице (см. Рисунок 16) отображаются все приобретённые, но ещё не оплаченные услуги, тренировки, абонементы и аренда шкафчиков. Данные подтягиваются из соответствующих таблиц по идентификатору клиента. Если по конкретной записи отсутствует дата оплаты, она считается подлежащей оплате. Все элементы сгруппированы по категориям: услуги к оплате, занятия к оплате, абонементы к оплате, задолженности по шкафчику.

В каждом блоке представлены карточки с подробной информацией: название, описание, стоимость и дата оказания или начала действия услуги. Рядом с каждой карточкой размещена кнопка «К оплате», позволяющая перейти к процессу оплаты отдельно взятой позиции. Также присутствует кнопка «Оплатить все», которая производит оплату сразу всех элементов данной категории.( листинг кода в приложение Ж)

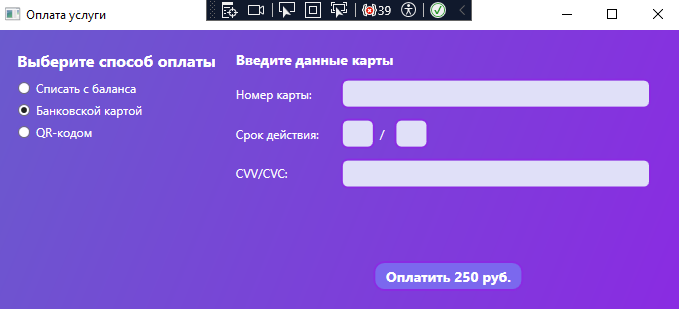


Рисунок 17 – Окно «Оплата»

После нажатия на любую кнопку оплаты открывается универсальное окно оплаты (см. Рисунок 17), куда передаётся сумма и название объекта оплаты (например, «Солярий» или «Индивидуальное занятие»). Пользователь может выбрать один из следующих способов оплаты:

* списание с баланса;
* банковская карта;
* QR-код.

При выборе оплаты банковской картой требуется ввести:

* номер карты;
* срок действия;
* CVV/CVC-код.

Валидация данных осуществляется перед подтверждением оплаты. Оплата с баланса допускает как полную, так и частичную оплату. Если клиент возвращает услугу или занятие, то соответствующая запись удаляется из базы неоплаченных элементов, и она более не отображается в списке.

Таким образом, страница оплаты предоставляет гибкий и интуитивно понятный интерфейс для обработки всех видов финансовых транзакций, связанных с услугами клуба. (листинг кода представлен в приложение П)

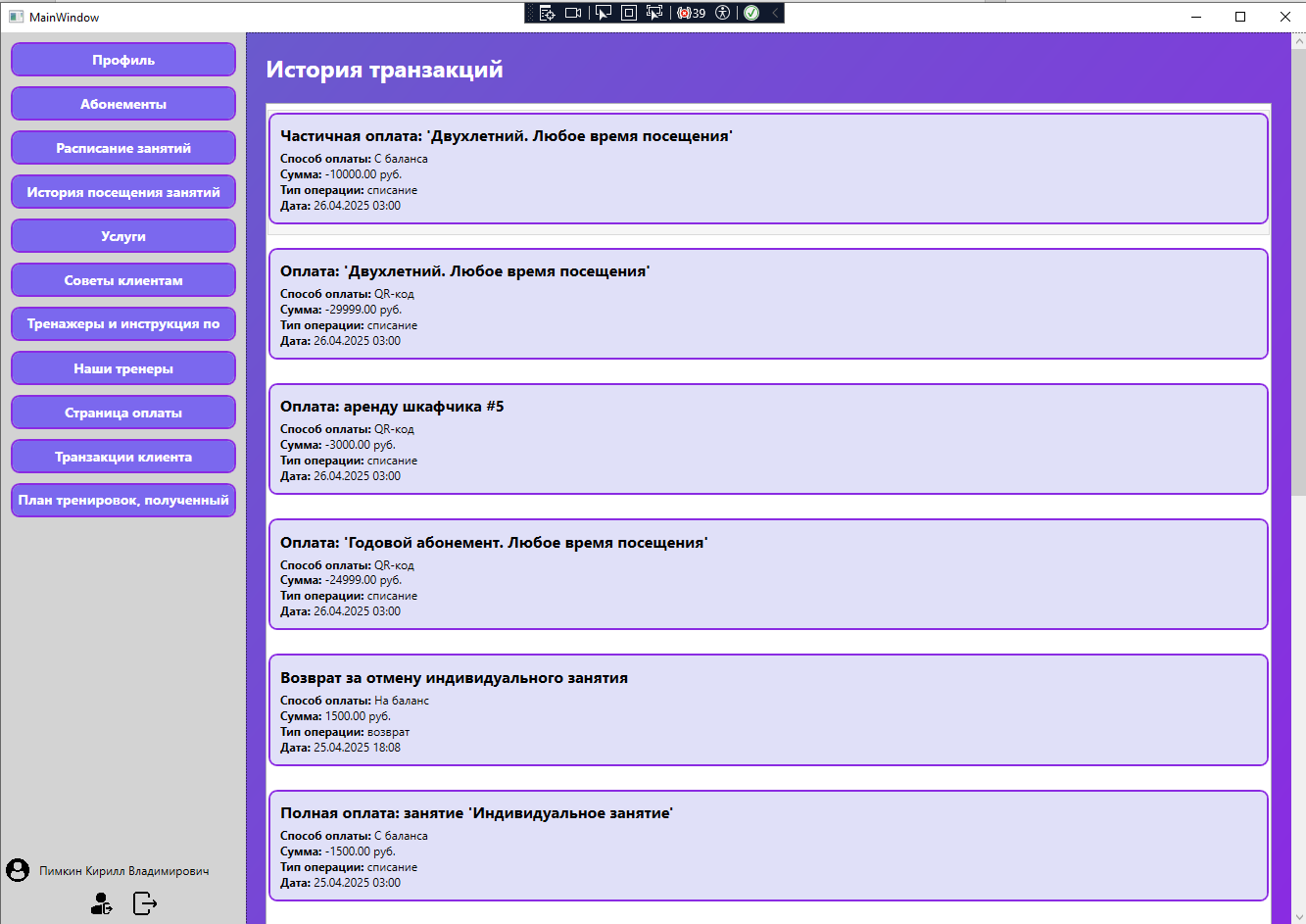


Рисунок 18 – Страница «Транзакции клиента»

На странице «Транзакции клиента» (см. Рисунок 18) отображается история всех финансовых транзакций клиента. Данные берутся из таблицы client\_transactions и отображаются в виде карточек, содержащих следующую информацию:

* название оплачиваемого объекта (например, название абонемента или услуги);
* способ оплаты (например, «С баланса», «QR-код», «Банковская карта»);
* сумма транзакции;
* тип операции (например, списание или возврат);
* дата и время выполнения операции.

Карточки оформлены в хронологическом порядке, начиная с самых свежих записей. В истории отображаются как полные, так и частичные оплаты, а также возвраты средств за отменённые услуги или тренировки. Такой подход позволяет клиенту всегда иметь актуальное представление о своих расходах, отслеживать списания и убедиться в наличии возвратов при отменах. (листинг кода представлен в приложение Н)

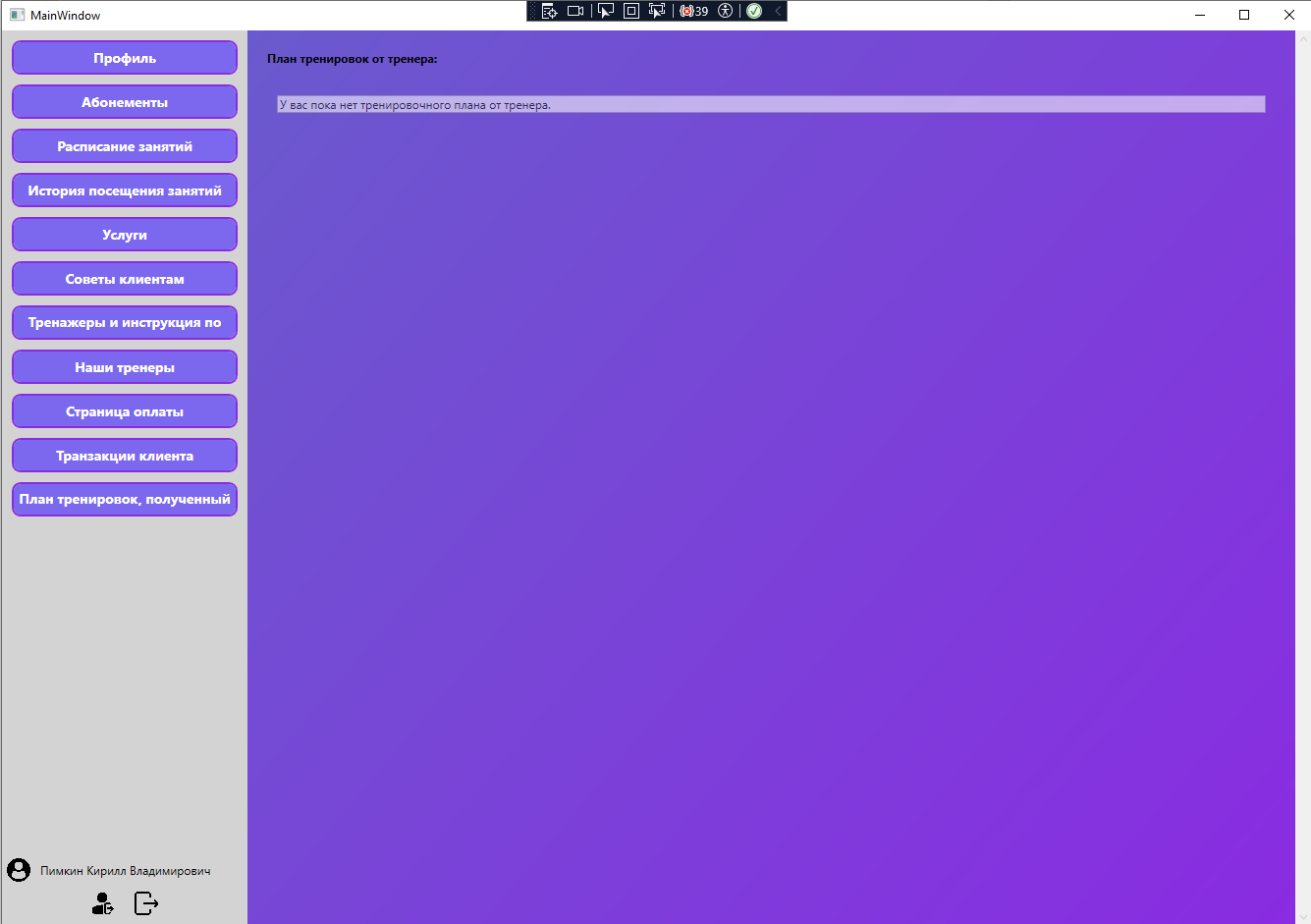


Рисунок 19 – Страница План тренировок от тренера

На данной форме (см. Рисунок 19) клиент может ознакомиться с персональным тренировочным планом, предоставленным тренером. Если план не был выдан, отображается сообщение: «У вас пока нет тренировочного плана от тренера». План тренировок загружается автоматически из базы данных по client\_id текущего пользователя, если он существует.

Тренировочный план включает текстовое описание и рекомендации по выполнению упражнений, длительности, количеству подходов, дням тренировок и иным индивидуальным параметрам. Это упрощает клиенту самостоятельную подготовку и повышает качество взаимодействия между ним и тренером. (листинг кода представлен в приложение М)

Главное меню (навигационная панель)

Слева на всех страницах приложения отображается главное меню, содержащее навигационные кнопки:

* Профиль
* Абонементы
* Расписание занятий
* История посещения занятий
* Услуги
* Советы клиентам
* Тренажёры и инструкция по ним
* Наши тренеры
* Страница оплаты
* Транзакции клиента
* План тренировок, полученный от тренера

В нижней части меню отображается имя и фамилия авторизованного клиента, а также кнопки:

* Выход из профиля (возврат к форме авторизации)
* Выход из приложения (закрытие программы)

Такой подход к построению интерфейса обеспечивает удобную навигацию, постоянный доступ к ключевым разделам и позволяет пользователю в любой момент выйти из системы или завершить работу.

### Функционал приложения с ролью «Тренер»

Рассмотрим функционал при авторизации с ролью тренер. Сразу после входа у пользователя отображается страница «Профиль тренера» (см. Рисунок 20).

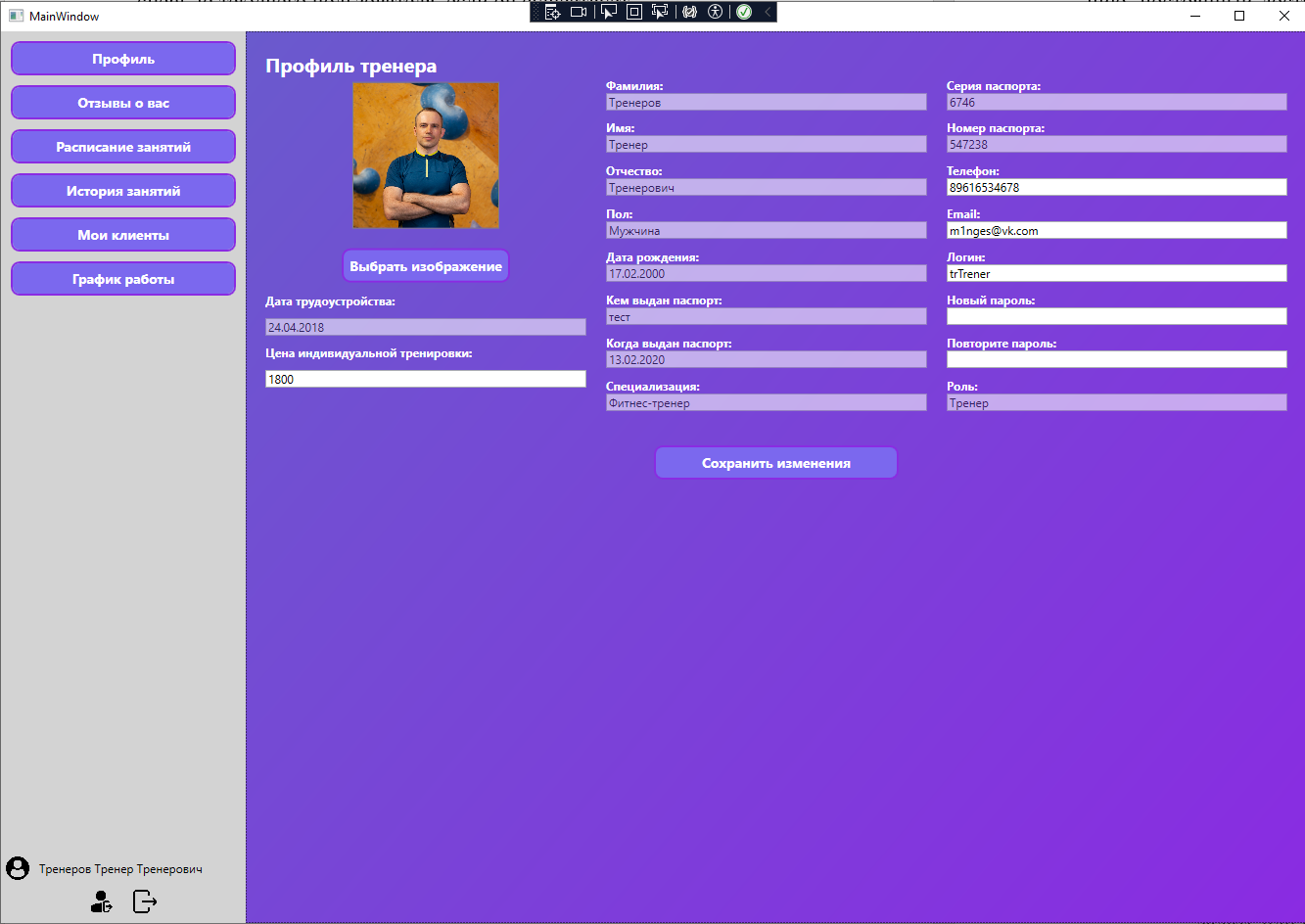


Рисунок 20 – Страница «Профиль тренера»

На данной странице отображается персональная информация тренера, полученная из базы данных, включая:

* фамилию, имя и отчество;
* пол, дату рождения;
* данные паспорта: серия, номер, кем и когда выдан;
* контактные данные: номер телефона и email;
* логин и роль пользователя;
* дата трудоустройства;
* специализация тренера;
* стоимость индивидуальной тренировки (в рублях);
* загруженная фотография тренера.

На форме предусмотрен функционал для изменения части персональной информации. Тренер может:

* сменить логин;
* задать новый пароль (с повторным вводом для подтверждения);
* изменить контактный номер телефона и email;
* задать или изменить стоимость индивидуальной тренировки;
* загрузить новое изображение.

После внесения изменений необходимо нажать кнопку «Сохранить изменения», чтобы данные были обновлены в базе. Для защиты данных осуществляется проверка введённого email и номера телефона на корректность. Также предусмотрена валидация нового пароля и проверка на совпадение с полем повтора пароля. Остальные поля заблокированы для редактирования, так как относятся к статической информации, установленной при создании учетной записи администрацией.(листинг кода представлен в приложение Ф)

На данной странице отображаются отзывы клиентов, оставленные в адрес конкретного тренера, авторизованного в системе (см. Рисунок 21).



Рисунок 21 – Страница «Отзывы о вас»

В верхней части формы указана средняя оценка тренера, рассчитанная на основе всех одобренных отзывов. Ниже расположен список отзывов, каждый из которых содержит:

* имя автора (в соответствии с настройками отображения имени в отзывах);
* текст самого отзыва;
* числовую оценку (по пятибалльной шкале).

Все отзывы, представленные на данной странице, проходят обязательную модерацию администратором перед публикацией. Отзывы добавляются пользователями, посетившими хотя бы одно занятие с тренером. Это гарантирует актуальность и достоверность представленной информации. Страница служит важным инструментом для оценки качества работы тренера и обратной связи с клиентами. (листинг кода представлен в приложение Ц)

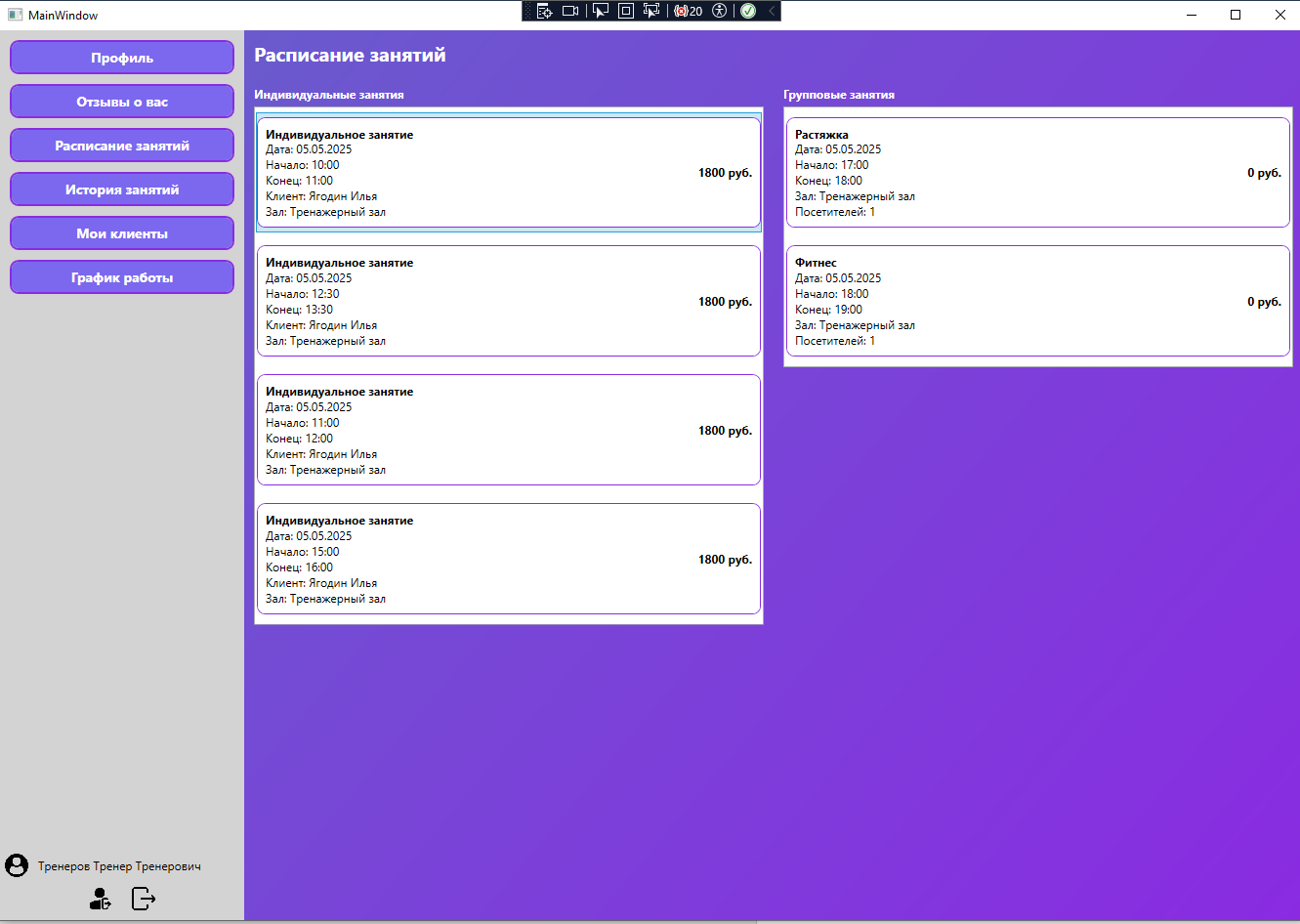


Рисунок 22 – Страница «Расписание занятий»

На данной странице отображаются все предстоящие занятия, назначенные тренеру, авторизованному в системе (см. Рисунок 22). Учитываются только занятия, дата проведения которых больше или равна текущей.

Форма разделена на два блока:

* Индивидуальные занятия. Здесь отображается список персональных тренировок, с указанием даты, времени начала и окончания, имени клиента, зала и стоимости;
* Групповые занятия. Отображаются групповые тренировки, на которых тренер является ведущим. Указана информация о дате, времени, зале проведения и количестве записавшихся клиентов. Стоимость для тренера также указывается.

Форма служит инструментом планирования и позволяет тренеру отслеживать все текущие и будущие тренировки, а также контролировать свою загруженность. (листинг кода представлен в приложение С)

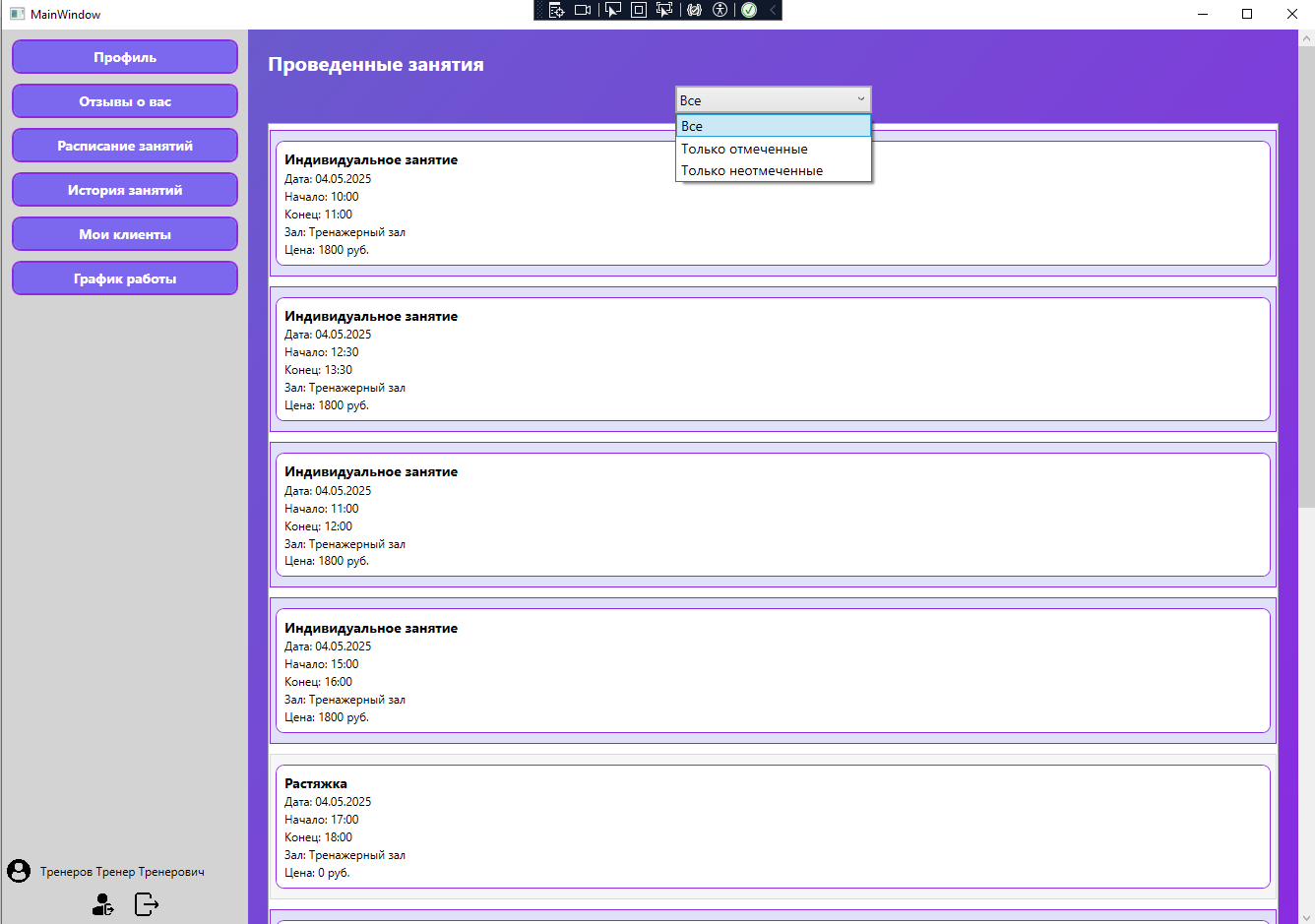


Рисунок 23 – страница «История занятий»

На странице «История занятий» (см. Рисунок 23) отображаются **все прошедшие занятия тренера**, включая как индивидуальные, так и групповые тренировки. Занятия выводятся в виде карточек с указанием:

* Названия занятия;
* Даты и времени начала и окончания;
* Зала проведения;
* Стоимости.

В верхней части формы реализован **выпадающий список**, позволяющий фильтровать занятия по статусу посещаемости:

* **Все** — отображаются все прошедшие занятия;
* **Только отмеченные** — занятия, для которых уже указана посещаемость;
* **Только неотмеченные** — занятия, по которым посещаемость еще не была проставлена.

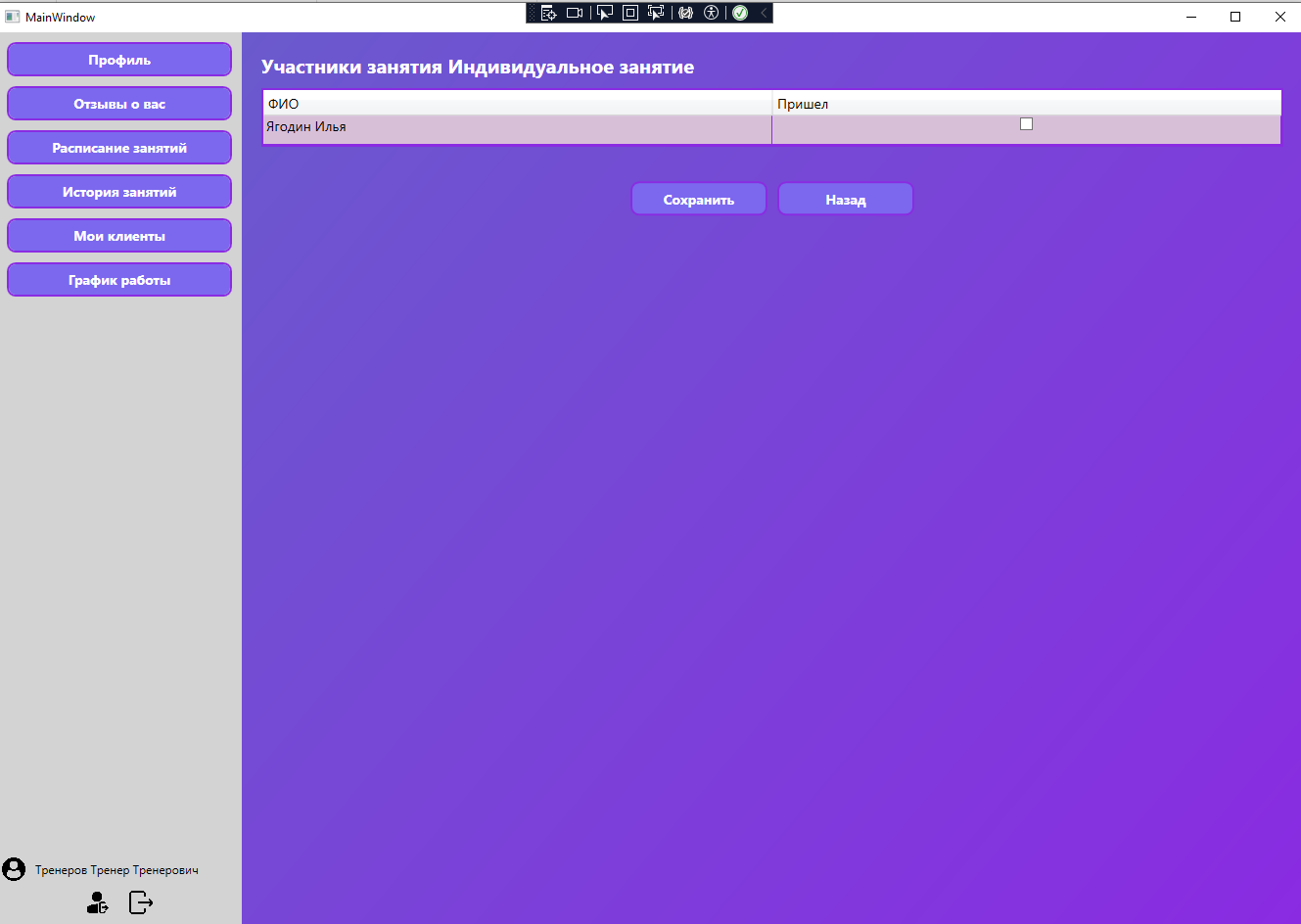


Рисунок 24 – страница «Отметка посещаемости»

Отметка посещаемости

При **двойном клике** на карточку занятия открывается окно **участников занятия** (см. Рисунок 24). В этом окне тренер может отметить, **посетил ли клиент тренировку**, установив флажок в соответствующей строке.

Отметка позволяет зафиксировать факт присутствия, от чего зависят возможность клиента оставить отзыв и необходимость оплаты.

**Если с момента окончания занятия прошло более 3 дней**, то внести изменения уже нельзя — выводится предупреждение:  
«Прошло более 3 дней с момента проведения занятия. Отметка недоступна.»

Это ограничение реализовано для обеспечения достоверности данных и исключения возможности редактирования задним числом. После отметки данные сохраняются нажатием кнопки **«Сохранить»**.

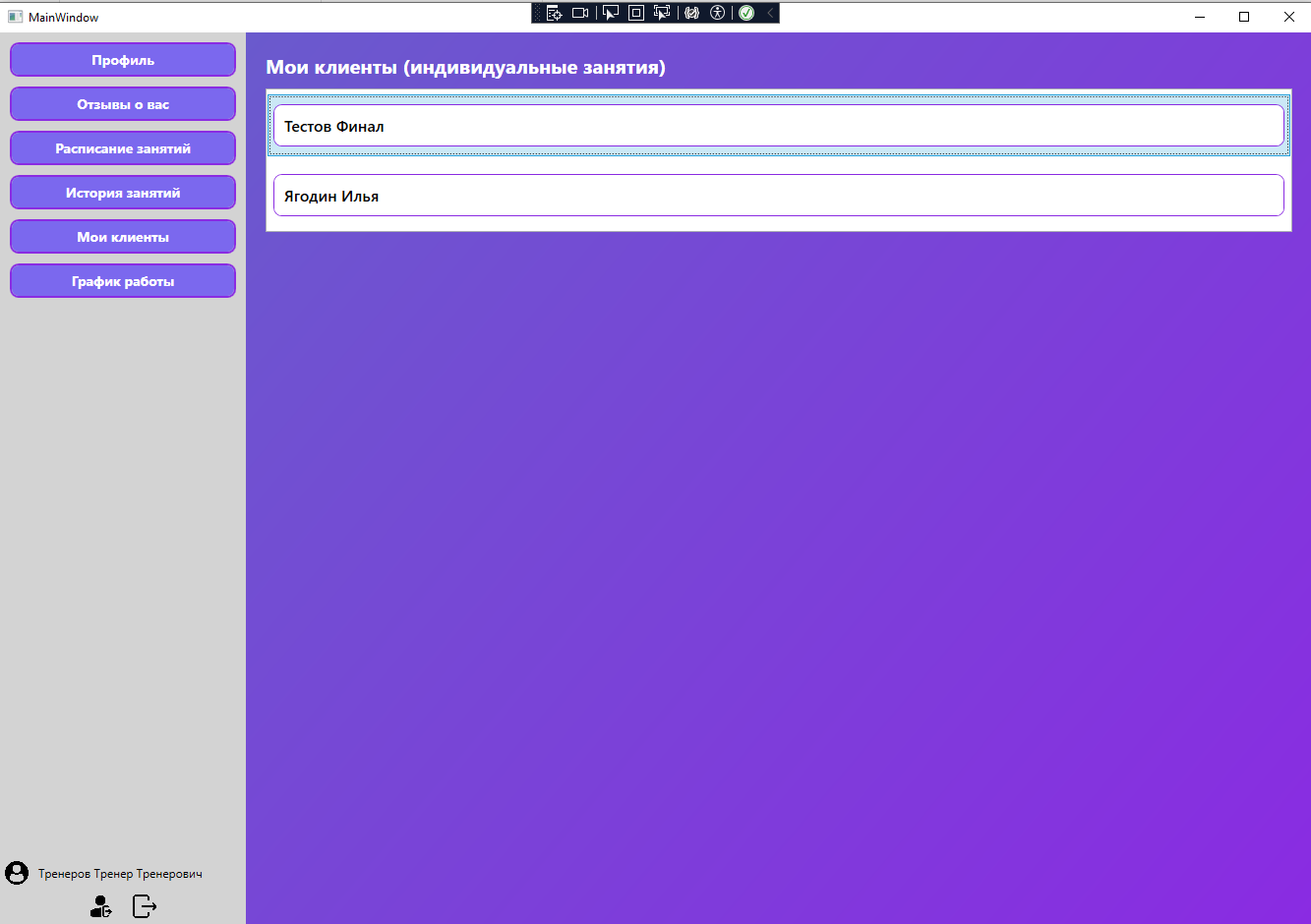


Рисунок 25 – страница «Мои клиенты»

Рассмотрим функциональность страницы «Мои клиенты» (см. Рисунок 25).

На данной странице отображаются все клиенты, с которыми у тренера ранее проводились индивидуальные тренировки. Представление списка реализовано в виде карточек с именами клиентов. Данные подтягиваются на основании записей в базе данных, фиксирующих проведённые индивидуальные занятия.

Для удобства идентификации каждый клиент отображается отдельной строкой. При двойном клике по имени клиента открывается окно с возможностью создать персональный план тренировок для выбранного клиента (см. Рисунок 26). В открывшемся окне присутствует текстовое поле, где тренер может в свободной форме ввести тренировочные рекомендации или инструкции. После заполнения необходимо нажать кнопку «Сохранить», чтобы информация была записана в базу данных. (листинг кода представлен в приложение Т)

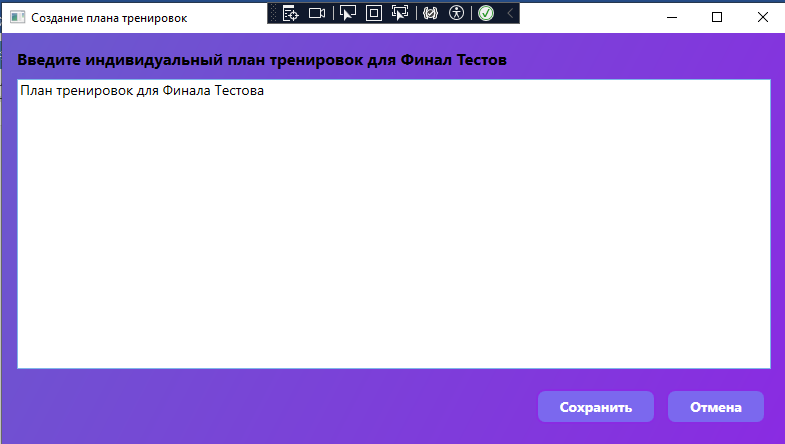


Рисунок 26 – Окно для создания персонального плана тренировок для выбранного клиента

Созданный индивидуальный план автоматически становится доступен клиенту в его личном кабинете, на соответствующей вкладке. Это позволяет наладить персонализированное взаимодействие между клиентом и тренером, а также обеспечить более продуктивный и результативный тренировочный процесс.

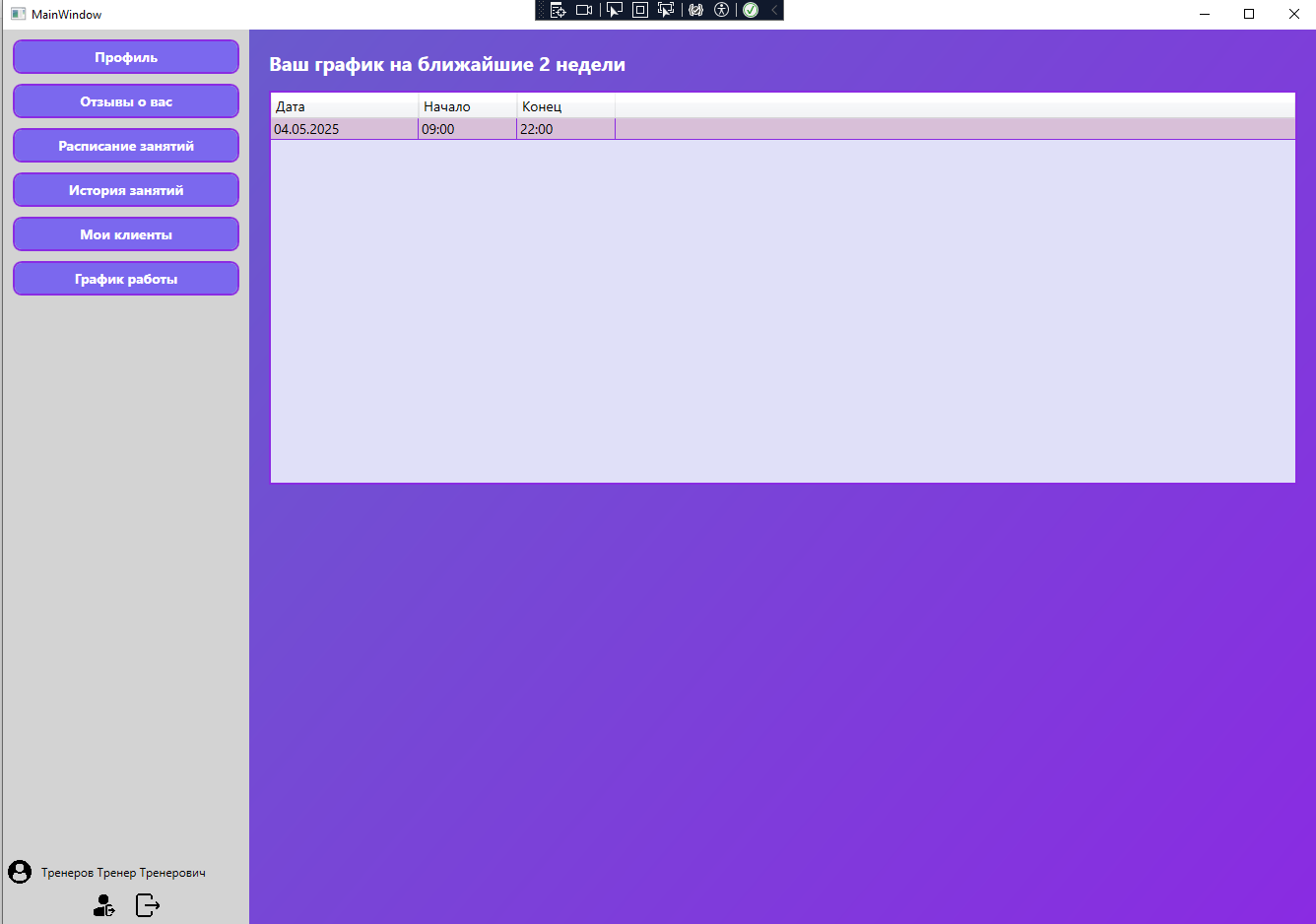


Рисунок 27 – страница «График работы»

Рассмотрим функциональность страницы «График работы» (см. Рисунок 27).

На данной странице тренер может ознакомиться со своим рабочим графиком на ближайшие две недели. Отображение реализовано в виде таблицы, где каждая строка соответствует определенной дате, а столбцы содержат информацию о времени начала и окончания рабочего дня.

Интерфейс страницы позволяет быстро оценить загрузку тренера на ближайший период, что удобно как для планирования индивидуальных тренировок, так и для корректировки расписания групповых занятий. График формируется на основе данных, заранее введенных администратором системы или самим тренером (если предусмотрено соответствующее разрешение).

Таким образом, страница «График работы» обеспечивает наглядное представление и контроль расписания работы тренера в течение двух следующих недель. (листинг кода представлен в приложение Ш)

### Функционал приложения с ролью «Администратор»

Перейдем к функционалу при авторизации с ролью администратор. Первым делом рассмотрим страницу управление графиком работы (см. Рисунок 28)

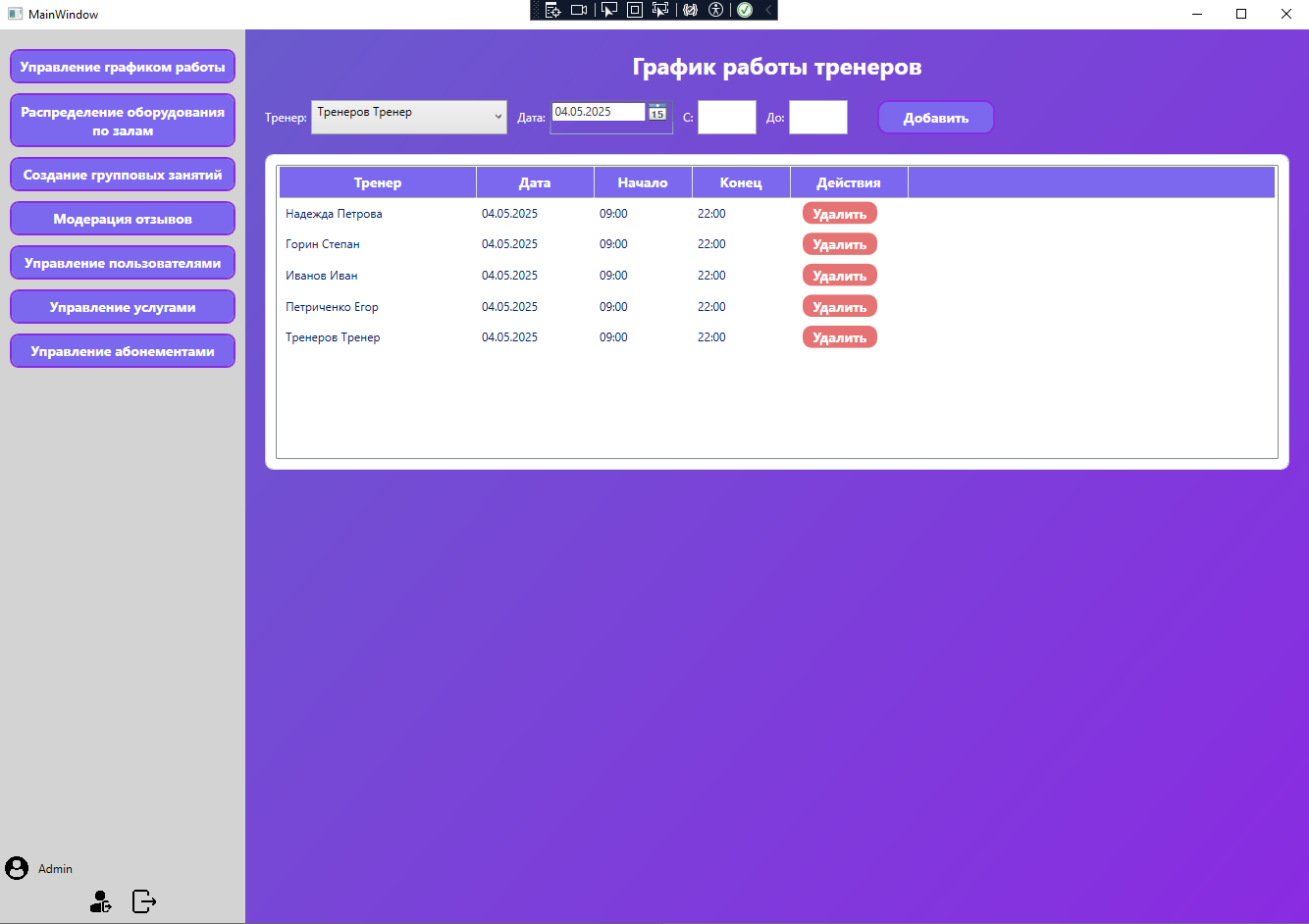


Рисунок 28 – страница «Управление графиком работы»

На данной странице администратор может управлять рабочим графиком всех тренеров, добавляя и удаляя рабочие смены. В верхней части формы предусмотрены элементы управления:

* выпадающий список для выбора тренера;
* выбор даты смены;
* поля для ввода времени начала и окончания рабочего дня;
* кнопка «Добавить» для сохранения указанного графика.

Ниже отображается таблица с уже установленными сменами тренеров, включающая такие столбцы, как: ФИО тренера, дата, время начала и окончания смены, а также кнопка «Удалить» для удаления графика.

Таким образом, страница позволяет централизованно и гибко управлять загрузкой всех тренеров фитнес-клуба на определенные даты, обеспечивая удобное редактирование и актуализацию данных.

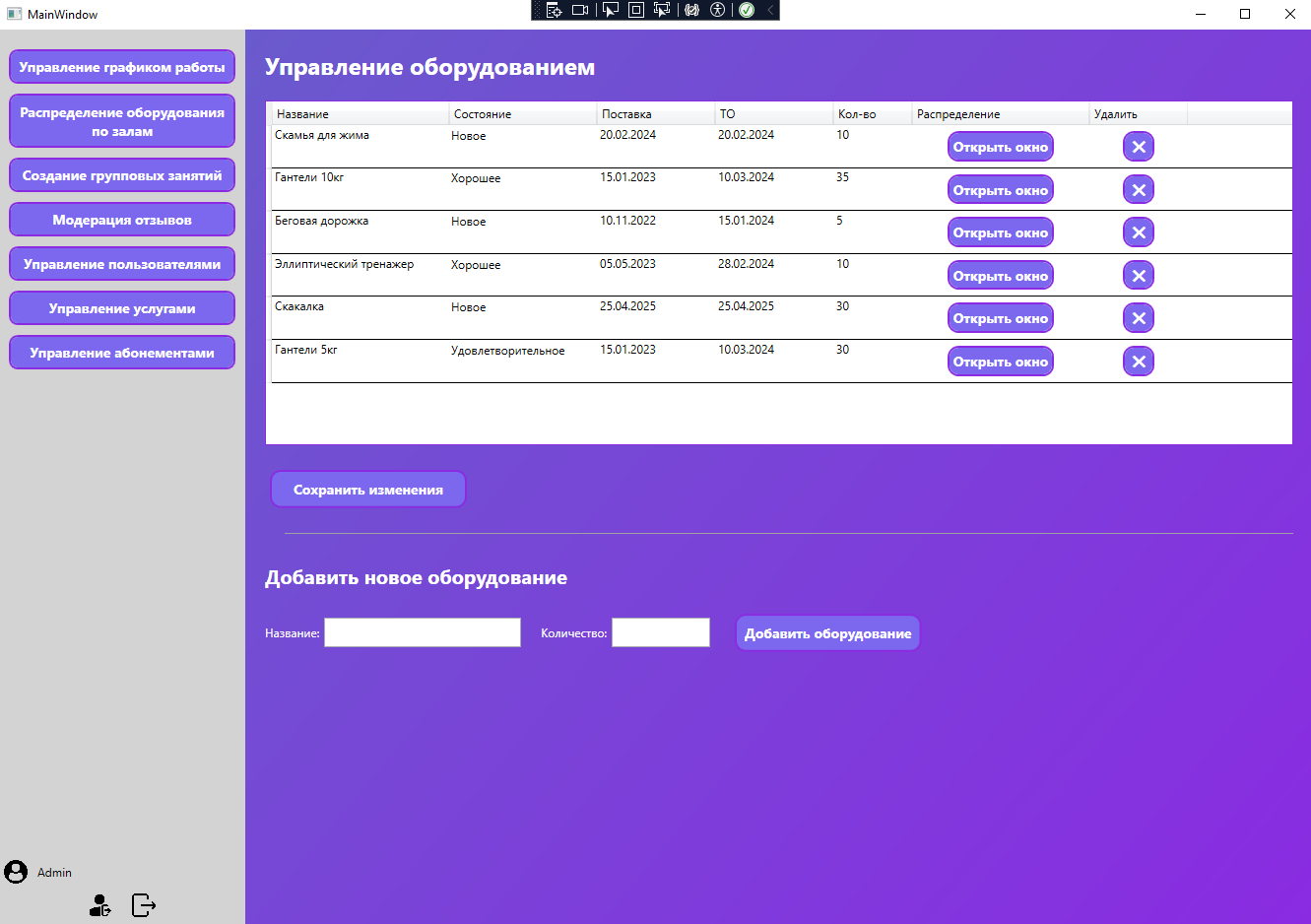


Рисунок 29 – страница «Управления оборудованием»

Страница управления оборудованием (см. Рисунок 29) предназначена для администраторов и позволяет выполнять полный цикл работы с инвентарем фитнес-клуба. На этой странице отображается таблица, содержащая сведения об оборудовании, включая название, состояние, дату поставки, дату последнего технического обслуживания, общее количество, а также действия по распределению и удалению.

В таблице доступны следующие поля. Название оборудования отображается, но не подлежит редактированию. Состояние оборудования задается с помощью выпадающего списка, элементы которого загружаются из справочной таблицы EquipmentConditions. Дата поставки отображается как справочная информация и также недоступна для редактирования. Дата последнего технического обслуживания может быть изменена вручную, при этом система проверяет корректность введенного значения. Количество оборудования указывается вручную, и любое изменение становится частью сохраняемой информации.

Для применения внесенных изменений используется кнопка «Сохранить изменения». При ее нажатии выполняется проверка корректности формата даты и сохранение новых данных в базу. Пользователь получает соответствующее уведомление о результате операции.

Кнопка «Открыть окно» в колонке распределения запускает модальное окно AssignEquipmentWindow (см. Рисунок 30). В этом окне отображается список залов с возможностью указать, сколько единиц оборудования выделяется на каждый из них. Отображается информация о том, сколько единиц доступно к распределению. Если общее количество назначенного оборудования превышает имеющееся, система показывает предупреждение. После сохранения новые данные записываются в таблицу HallEquipments, и окно автоматически закрывается. (листинг кода представлен в приложение Щ)

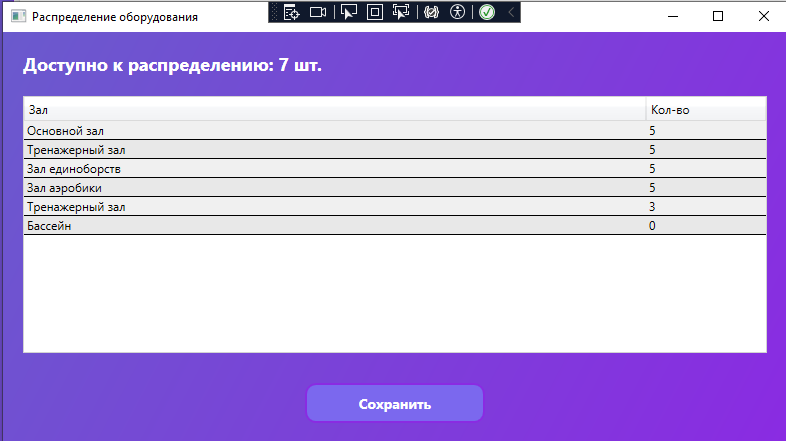


Рисунок 30 – окно «Распределение оборудования»

Кнопка с иконкой удаления в последнем столбце вызывает подтверждение. В случае согласия оборудование удаляется из таблицы Equipment, а также из всех связанных записей распределения по залам. После этого таблица обновляется, и пользователь получает уведомление о выполнении действия.

Под таблицей находится форма для добавления нового оборудования. В нее необходимо ввести название оборудования и количество. Допускается только положительное целое число. Название обязательно для заполнения. После подтверждения данные сохраняются в таблицу Equipment. Устанавливаются значения по умолчанию для даты поставки и ТО. Также автоматически присваивается состояние с минимальным идентификатором из таблицы EquipmentConditions. После сохранения форма очищается, и появляется сообщение об успешном добавлении.

Изменения в таблице можно вносить напрямую, щелкая по соответствующим ячейкам. Это делает работу с оборудованием удобной и быстрой. Комбинация редактируемых полей и встроенных окон для распределения позволяет администратору централизованно управлять инвентарем клуба без необходимости перехода на другие экраны.

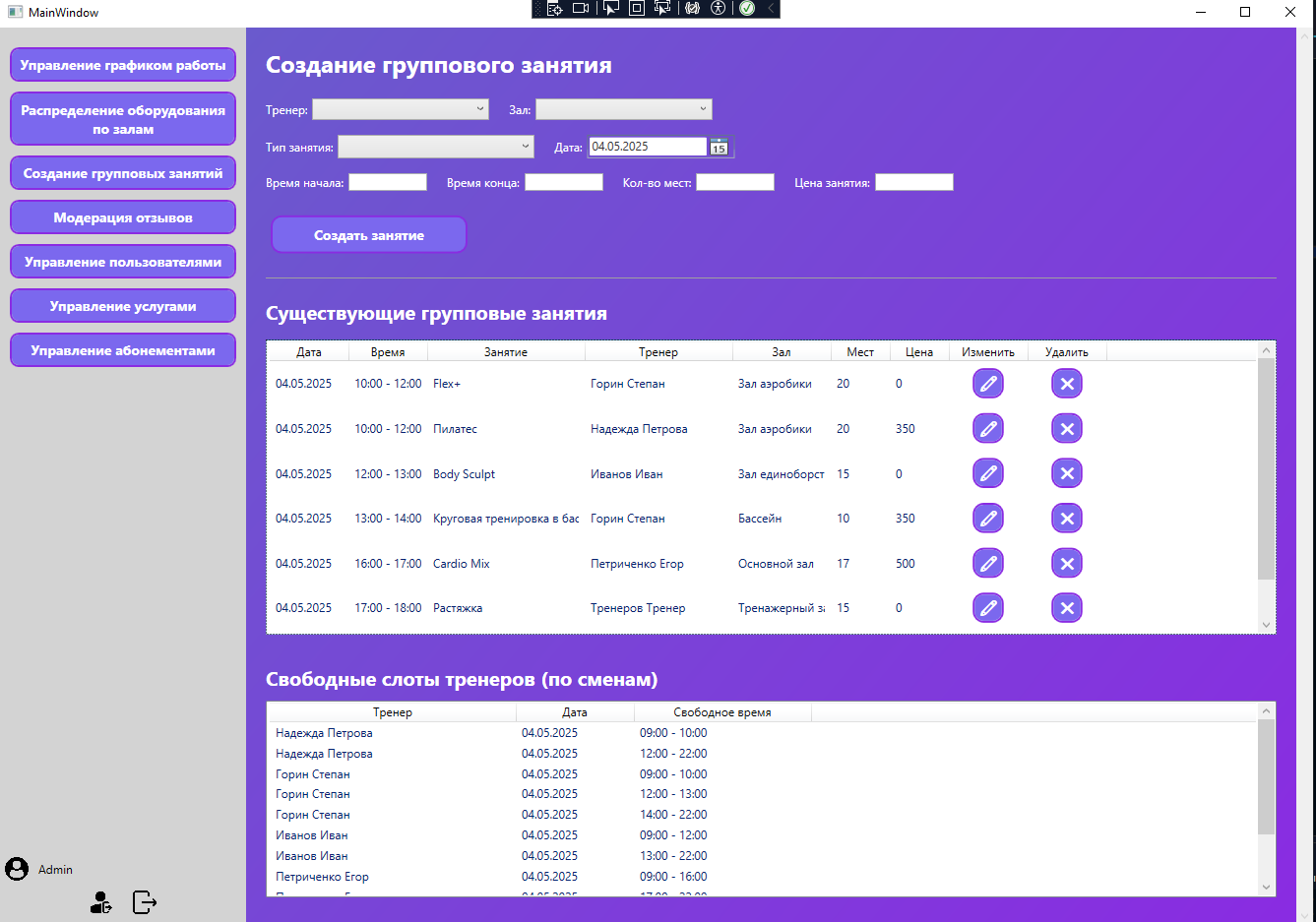


Рисунок 31 – страница «Создание групповых занятий»

На странице создания групповых занятий (см. Рисунок 31) администратор может добавлять, редактировать и удалять групповые тренировки, назначая их на свободные временные слоты тренеров. Форма создания нового занятия включает в себя обязательные поля: выбор тренера, выбор зала, тип занятия, дата проведения, время начала и окончания, количество мест, а также стоимость. Все поля должны быть корректно заполнены — в противном случае система не позволит завершить создание и выдаст сообщение об ошибке.

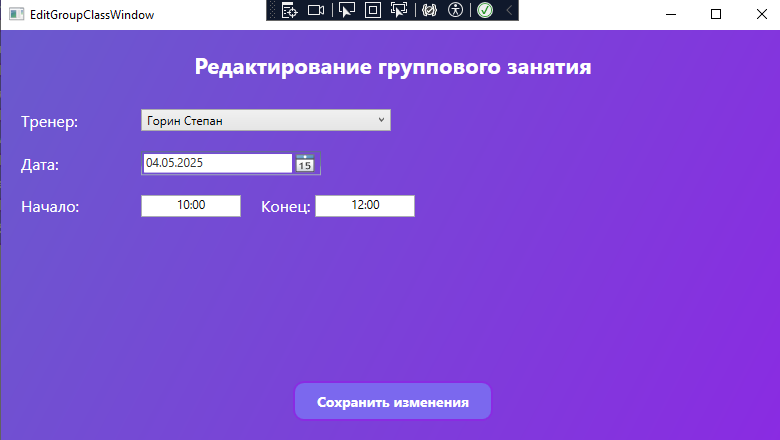


Рисунок 32 – окно «Изменение группового занятия»

Для редактирования занятия предусмотрено отдельное окно. (см. Рисунок 32)

Для облегчения планирования реализован механизм отображения свободных слотов тренеров. В нижней части страницы отображается список доступных временных окон по каждому тренеру, рассчитанных на основе их графика работы и уже назначенных занятий. При двойном клике на такой слот автоматически подставляются значения тренера, даты и временного интервала начала и конца — это существенно ускоряет процесс планирования занятия.

Создание нового занятия возможно только при условии, что в выбранную дату у тренера действительно есть смена. Также выполняется проверка на пересечение с другими уже назначенными занятиями в рамках той же смены. Если новое занятие по времени пересекается с существующим, система уведомляет пользователя об ошибке и не позволяет сохранить запись.

Ниже формы добавления выводится список уже существующих занятий. Для каждого занятия указаны дата, время, наименование, тренер, зал, количество мест и стоимость. Сопровождающие кнопки позволяют отредактировать выбранное занятие или удалить его из базы. При нажатии на кнопку редактирования открывается отдельное окно, где можно изменить дату, время и тренера. Проверка на наличие смены у нового тренера также выполняется и в этом окне.

Таким образом, страница предоставляет полный инструментарий для эффективного управления групповыми занятиями с учётом расписания тренеров, доступности залов и контроля возможных пересечений во времени. (листинг кода представлен в приложение Э)



Рисунок 33 – страница «Модерация отзывов»

На странице модерации(см. Рисунок 33) отзывов администратор получает доступ к предварительной проверке отзывов, оставленных пользователями о тренерах или занятиях. Отображается имя клиента, который оставил отзыв, объект, к которому он относится (например, конкретный тренер или занятие), а также сам текст отзыва и оценка по шкале от 1 до 5.

Интерфейс страницы представляет каждый отзыв в виде отдельного блока, снабжённого двумя кнопками: одобрить и отклонить. При нажатии на кнопку "Одобрить", отзыв становится доступным для просмотра другим пользователям, в том числе и в личном кабинете тренера. Кнопка "Отклонить" удаляет отзыв, и он не сохраняется в базе.

Такой подход позволяет избежать публикации неподобающего контента и обеспечивает контроль качества обратной связи, которая отображается пользователям. Страница выполнена в простом и понятном формате, что позволяет быстро обрабатывать входящие отзывы. (листинг кода представлен в приложение Я)

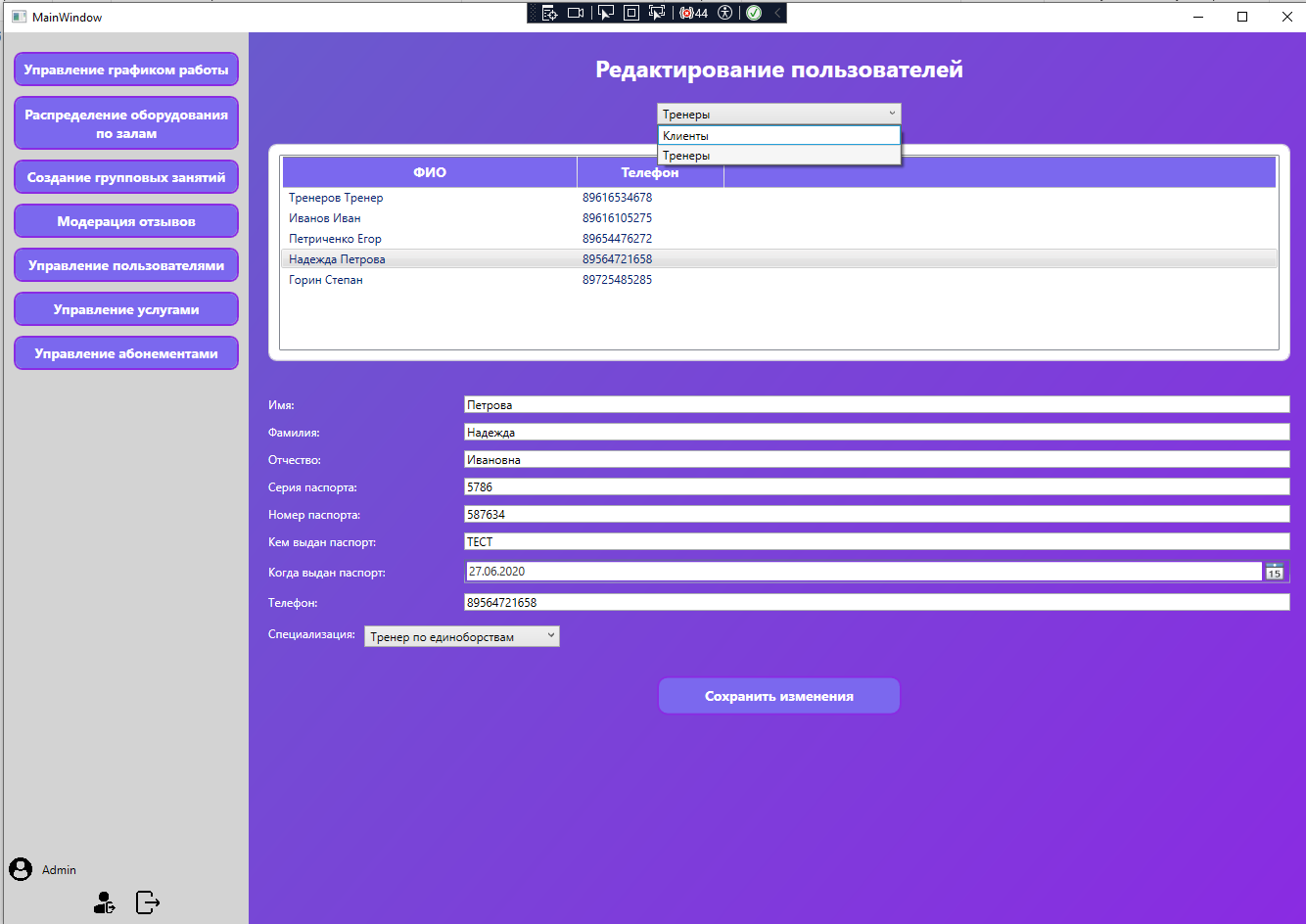


Рисунок 34 – страница «Управление пользователями»

На странице управления пользователями (см. Рисунок 34) администратор получает возможность просматривать и редактировать персональные данные зарегистрированных пользователей системы. В верхней части интерфейса расположен выпадающий список, с помощью которого можно фильтровать отображаемых пользователей по роли: клиенты или тренеры. Ниже отображается таблица с ФИО и номером телефона, позволяющая выбрать нужного пользователя для редактирования.

При выборе пользователя его данные автоматически подгружаются в поля редактирования. Для клиентов можно изменить следующие параметры:

* имя
* фамилию
* отчество
* серию и номер паспорта
* кем выдан паспорт
* дату выдачи паспорта
* номер телефона

Для тренеров список доступных для редактирования данных расширен. Помимо всего перечисленного выше, можно также изменить их специализацию, например, указать, что тренер является фитнес-тренером или инструктором по йоге.

После внесения изменений администратор может сохранить их нажатием кнопки "Сохранить изменения". Эта форма обеспечивает удобное и централизованное управление данными пользователей, снижая вероятность ошибок и обеспечивая актуальность информации.

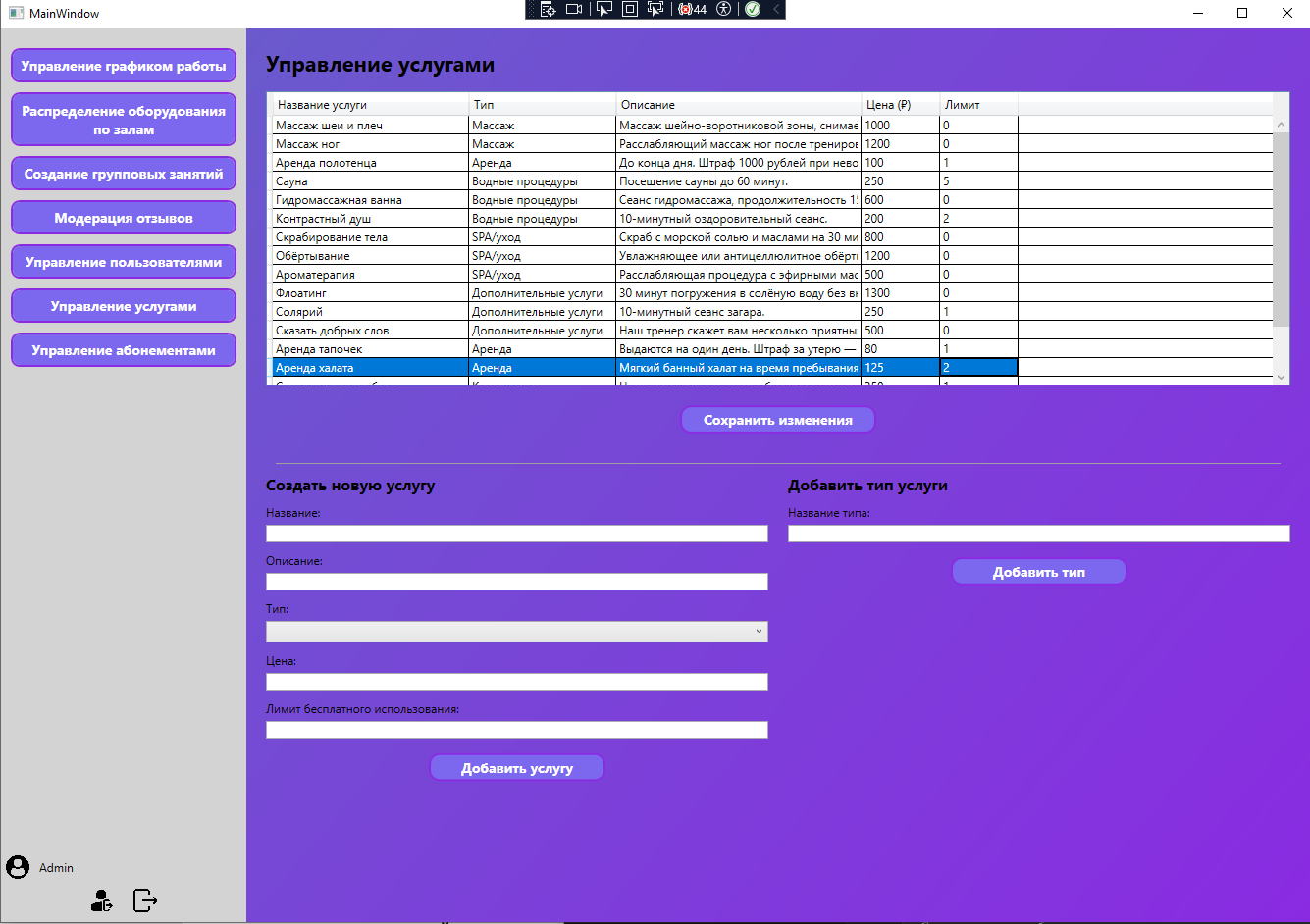


Рисунок 35 – страница «Управление услугами»

На странице управления услугами (см. Рисунок 35) администратор может редактировать, добавлять и структурировать услуги, доступные клиентам. Основная таблица отображает список всех существующих услуг с их названием, типом, описанием, стоимостью и лимитом бесплатного использования.

Редактирование осуществляется прямо в таблице. При двойном клике на соответствующую ячейку становится доступно изменение описания, цены или лимита использования. Это позволяет быстро обновлять данные без необходимости открывать отдельные формы. После внесения правок необходимо нажать кнопку "Сохранить изменения", чтобы изменения были сохранены в базе данных.

Ниже таблицы размещены два блока. В первом можно создать новую услугу, заполнив следующие поля:

* название
* описание
* тип (выбирается из существующих типов)
* цена
* лимит бесплатного использования

Все поля проходят обязательную валидацию. Например, цена должна быть числом, лимит — неотрицательным числом, а все текстовые поля — заполненными. После ввода данных администратор нажимает кнопку "Добавить услугу".

Во втором блоке можно добавить новый тип услуги. Для этого указывается только название типа, после чего он становится доступным для выбора при создании новых услуг.

Таким образом, страница предоставляет удобный инструмент для ведения каталога дополнительных сервисов клуба и поддержания его в актуальном состоянии.

На странице управление абонементами (см. Рисунок 36) аналогично управлению услугами: можно редактировать описание, цену и длительность абонемента двойным кликом по ячейке, сохранять изменения, добавлять новые абонементы с валидацией полей, а также создавать новые типы абонементов.

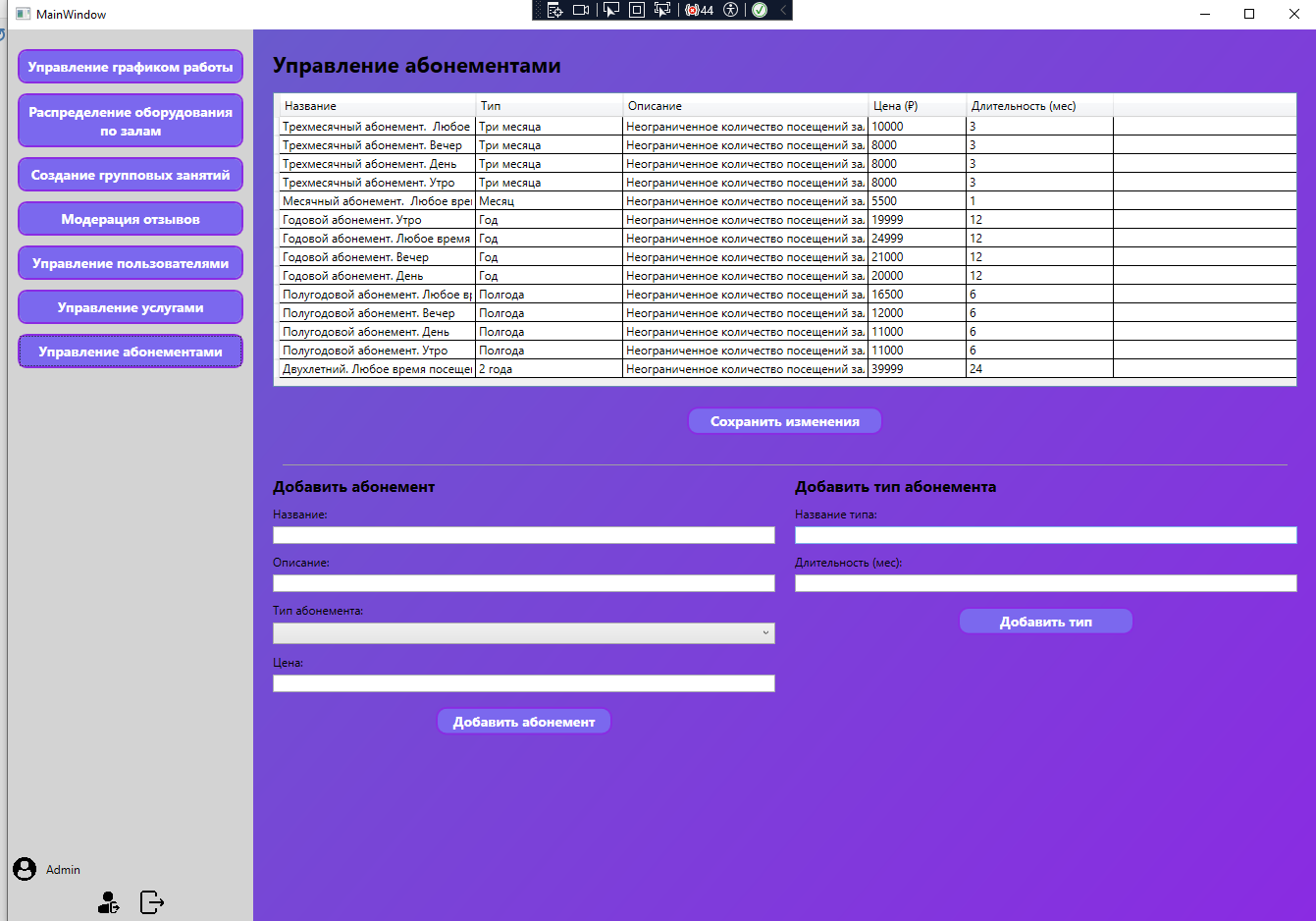


Рисунок 36 – страница «Управление абонементами»

## Тестирование разработанной автоматизированной системы

Для подтверждения работоспособности системы также рассмотрим все возможные варианты ввода входных данных, тем самым имитируются все возможные ситуации, которые могут произойти при дальнейшей эксплуатации программы. (листинг кода представлен в приложение Ю)

Рассмотрим вариант, при котором пользователь пробует войти в программу используя неправильный логин. В этом случае пользователь получит сообщение «Пользователь с таким логином не найден». (см. Рисунок 37)

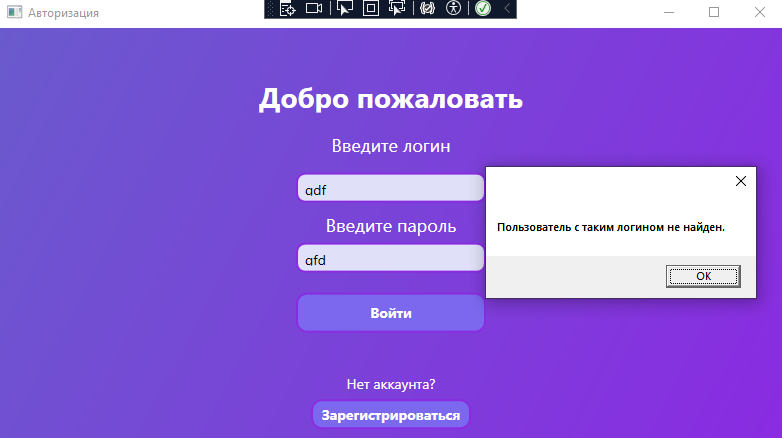


Рисунок 37 – сообщение «Пользователь с таким логином не найден»

Рассмотрим вариант, при котором пользователь пробует войти в программу используя правильный логин, но неправильный пароль. В этом случае пользователь получит сообщение «Неверный пароль». (см. Рисунок 38)

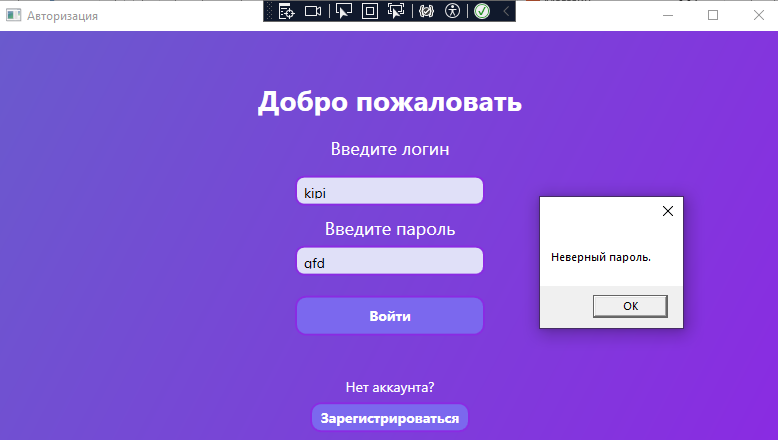


Рисунок 38 – сообщение «Неверный пароль»

Рассмотрим вариант, при котором пользователь пытается зарегистрироваться, заполнив не все поля. (см. Рисунок 38)

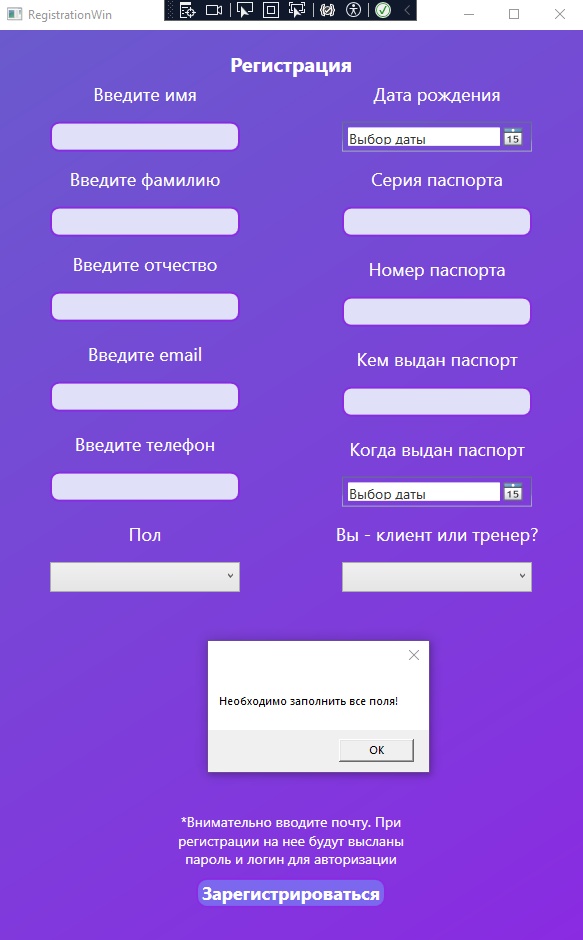


Рисунок 39 – сообщение «Необходимо заполнить все поля»

Подобным способом проверяется заполнение всех полей в системе.

Рассмотрим вариант, при котором пользователь вводит номер телефона в неверном формате при редактировании профиля и получает сообщение «Телефон должен быть в формате +7XXXXXXXXXX или 8XXXXXXXXXX» (см. Рисунок 40)

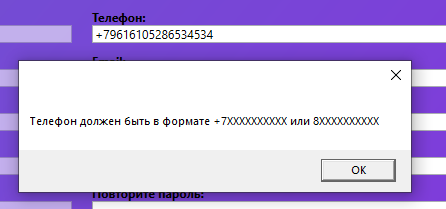


Рисунок 40 – сообщение «Телефон должен быть в формате +7XXXXXXXXXX или 8XXXXXXXXXX»

Подобным образом проверяются все поля, где важен формат ввода внутри системы.

Рассмотрим следующий пример. Клиент сохраняет изменения внесенные в профиль и в случае, если все данные введены в верном формате ему выдается сообщение «Изменения успешно сохранены!» (см. Рисунок 41)

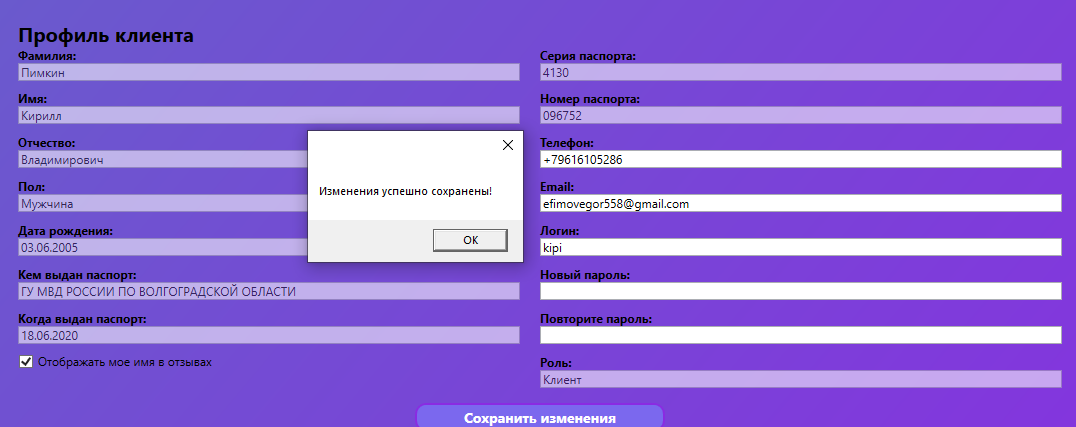


Рисунок 41 – сообщение «Изменения успешно сохранены!»

Подобным образом выдаются все сообщения об успешных действиях пользователя в системе.

Рассмотрим следующий пример. При оплате клиенту недостаточно баланса и ему выдается следующее сообщение «У вас недостаточно средств на балансе.» (см. Рисунок 42)



Рисунок 42 – сообщение «У вас недостаточно средств на балансе.»

Рассмотрим ситуацию, когда клиент успешно оплатил тот или иной объект. В случае успешной оплаты ему выдается сообщение, где указано «Вы успешно оплатили ‘{название объекта}’», название объекта передается со страницы оплаты и подставляется в окно сообщения. (см. Рисунок 43)

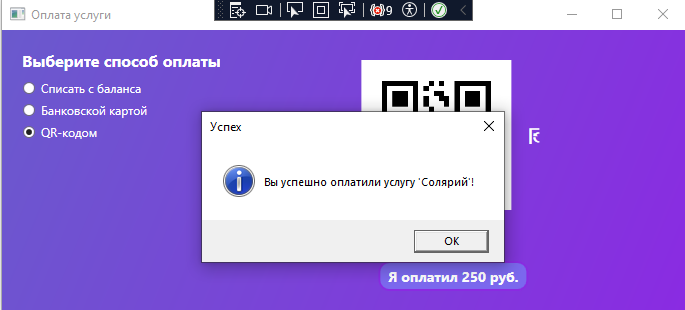


Рисунок 43 – «Вы успешно оплатили ‘{название  
объекта}’!»

Рассмотрим следующую ситуацию, клиент хочет отменить оформленную услугу, но дата использования услуги равна сегодня или была в прошлом. В таком случае клиент получит следующее сообщение «Нельзя отменить услугу, которая уже использована» (см. Рисунок 44)

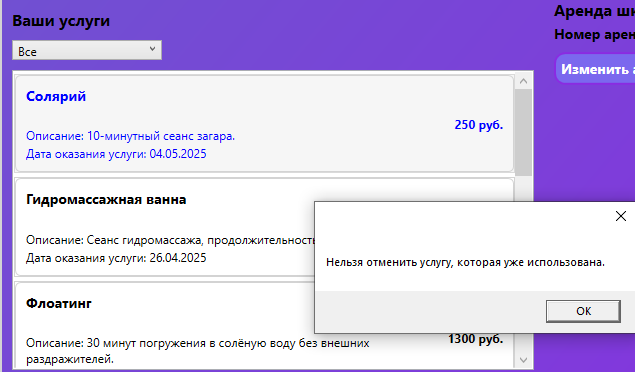


Рисунок 44 – сообщение «Нельзя отменить услугу, которая уже использована.»

Подобным образом проверяются возвраты за занятия. Нельзя отменить занятие, если стоит отметка «Посещено».

Рассмотрим следующую ситуацию. Клиент хочет оформить услугу в прошлом. В таком случае он получает сообщение «Выберите дату предоставления услуги (возможно вы указываете дату в прошлом» (см. Рисунок 45)

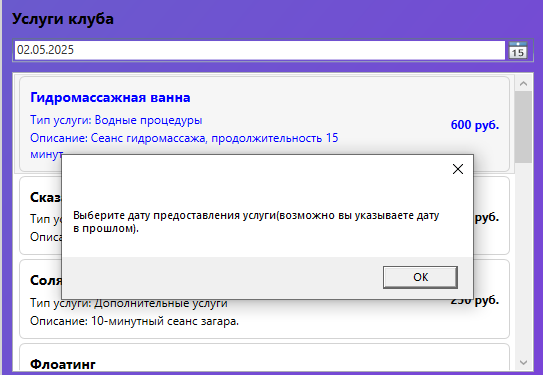


Рисунок 45 – сообщение «Выберите дату предоставления услуги (возможно вы указываете дату в прошлом»

Подобные сообщения выдаются пользователю в случае выбора даты окончания аренды шкафчика больше даты окончания абонемента или даты в прошлом.

Рассмотрим следующую ситуацию. Клиент записывается на занятие и выбирает дату и время занятия в то время, когда у него уже есть другое занятие в это время. В таком случае он получает сообщение «У вас уже есть занятие в это время!» (см. Рисунок 46)

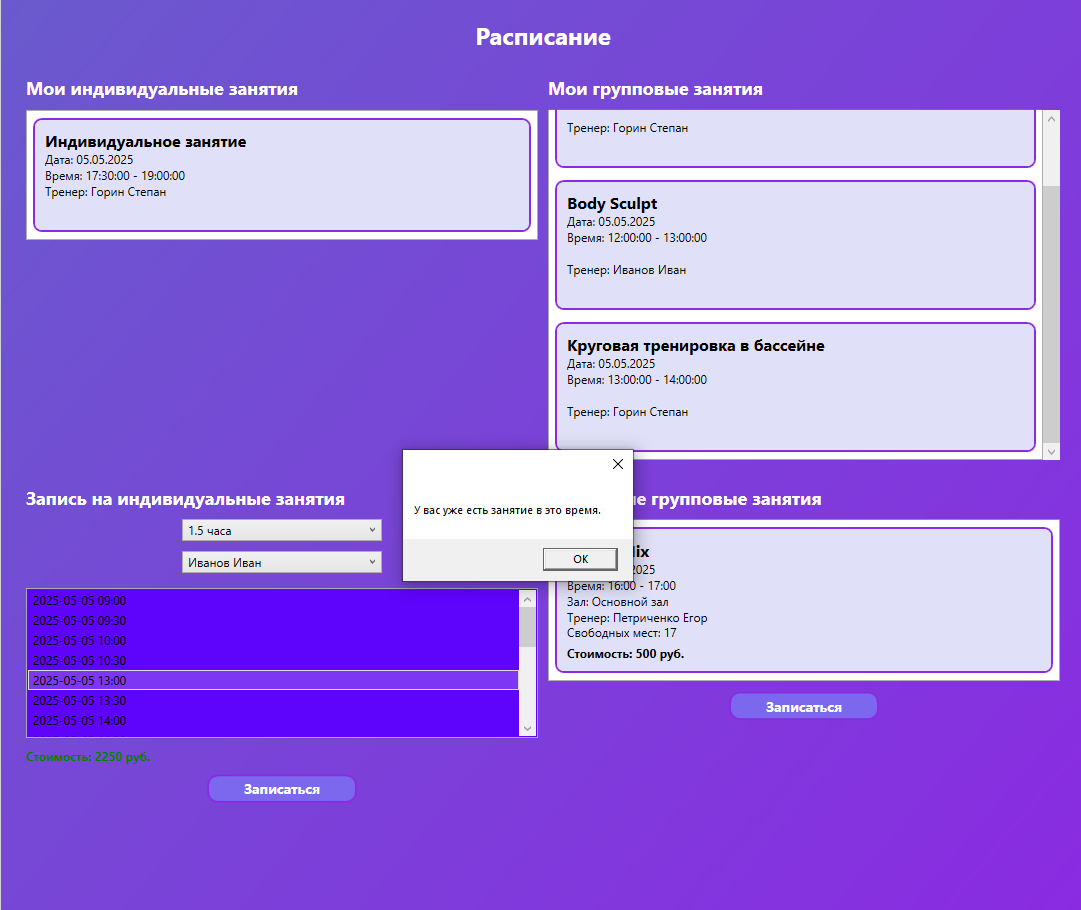


Рисунок 46 – сообщение «У вас уже есть занятие в это время!»

Рассмотрим следующую ситуацию. Тренер хочет сделать отметку о посещаемости спустя 3 и более дней после занятия. В таком случае он получит сообщение «Прошло более 3 дней с момента проведения занятия. Отметка недоступна.» (см. Рисунок 47)

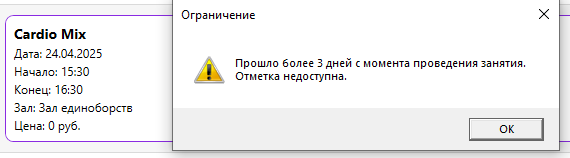


Рисунок 47 – сообщение «Прошло более 3 дней с момента проведения занятия. Отметка недоступна.»

Рассмотрим следующую ситуацию. Администратор пытается создать тренеру занятие, когда у него нет рабочей смены в это время. В таком случае он получает сообщение «У выбранного тренера нет смены на эту дату». (см. Рисунок 48)

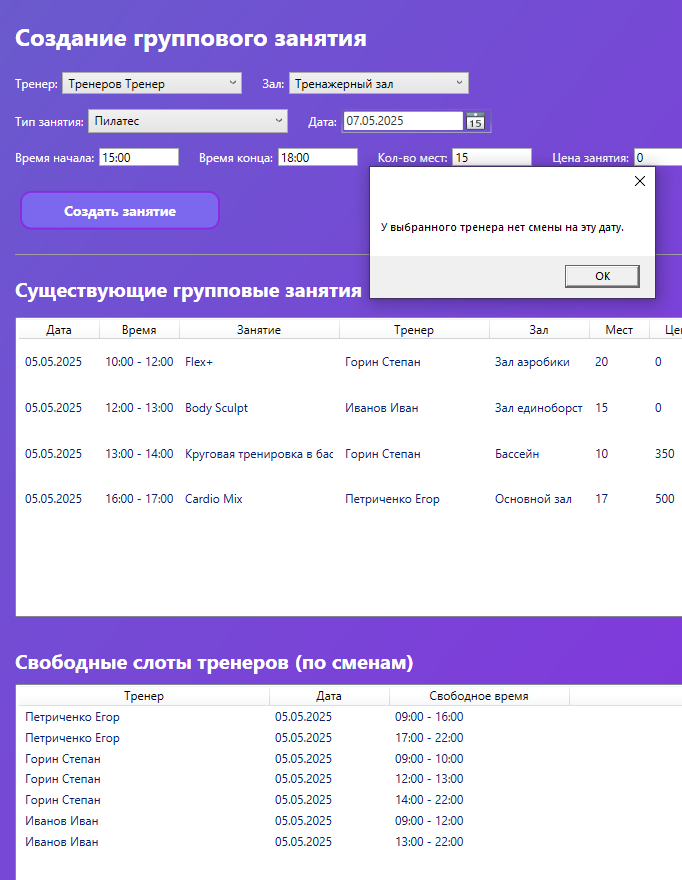


Рисунок 48 – сообщение «У выбранного тренера нет смены на эту дату»

# ЗАКЛЮЧЕНИЕ

Учитывая, что для разработки решения была использована не профессиональная база, а учебная, некоторые данные не были представлены в качестве сущностей.

Разработанная информационная система отвечает всем требованиям предметной области, таблицы созданной базы данных отвечают требованиям нормализации, что позволяет обеспечить целостность и непротиворечивость информации.

Созданная информационная система позволяет решать все задачи, сформулированные в задании на курсовой проект. Это позволяет сделать вывод, что задание выполнено полностью.

Разработанная база данных легко дополняется при необходимости разработки профессиональной базы данных и может быть использована в дальнейших проектах.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ И ЛИТЕРАТУРЫ

1. Васильев А.Н. Программирование на C# для начинающих: учеб. пособие / А.Н. Васильев. – М.: Бомбора, 2023. – 528 с.
2. Домбровская Г.Р., Новиков Б.Ю., Бейликова А.А. Оптимизация запросов PostgreSQL: практическое руководство / Г.Р. Домбровская, Б.Ю. Новиков, А.А. Бейликова. – М.: ДМК Пресс, 2022. – 278 с.
3. Прайс М.Дж. C# 8 и .NET Core. Разработка и оптимизация / М.Дж. Прайс. – СПб.: Питер, 2021. – 816 с.
4. Смит Дж.П. Entity Framework Core в действии / Дж.П. Смит. – М.: ДМК Пресс, 2023. – 576 с.
5. Стиллмен Э., Грин Д. Head First. Изучаем C# / Э. Стиллмен, Д. Грин. – 4-е изд. – СПб.: Питер, 2022. – 768 с.
6. Фримен А. Entity Framework Core 2 для ASP.NET Core MVC для профессионалов / А. Фримен. – М.: Диалектика, 2019. – 816 с.
7. Обзор языка C# – руководство по C# [Электронный ресурс]. – Режим доступа: <https://docs.microsoft.com/ru-ru/dotnet/csharp/tour-of-csharp/> (дата обращения: 05.05.2025).
8. Справочник по Transact-SQL (компонент Database Engine) [Электронный ресурс]. – Режим доступа: <https://docs.microsoft.com/ru-ru/sql/t-sql/language-reference?view=sql-server-ver15> (дата обращения: 05.05.2025).
9. Общие сведения о Visual Studio [Электронный ресурс]. – Режим доступа: <https://docs.microsoft.com/ru-ru/visualstudio/get-started/visual-studio-ide?view=vs-2019> (дата обращения: 05.05.2025).

# ПРИЛОЖЕНИЕ

Приложение А. Скрипт на создание БД

CREATE FUNCTION public.count\_visited\_classes(client\_id integer) RETURNS integer

LANGUAGE plpgsql

AS $$

BEGIN

RETURN (

SELECT COUNT(\*)

FROM classvisits cv

JOIN clientmembership cm ON cv.client\_membership\_id = cm.client\_membership\_id

WHERE cm.client\_id = client\_id AND cv.visited = true

);

END;

$$;

CREATE FUNCTION public.delete\_user\_when\_client\_deleted() RETURNS trigger

LANGUAGE plpgsql

AS $$

BEGIN

IF OLD.user\_id IS NOT NULL THEN

IF NOT EXISTS (

SELECT 1 FROM trainer

WHERE user\_id = OLD.user\_id

) THEN

DELETE FROM users

WHERE user\_id = OLD.user\_id;

END IF;

END IF;

RETURN OLD;

END;

$$;

CREATE FUNCTION public.delete\_user\_when\_trainer\_deleted() RETURNS trigger

LANGUAGE plpgsql

AS $$

BEGIN

IF OLD.user\_id IS NOT NULL THEN

IF NOT EXISTS (

SELECT 1 FROM client

WHERE user\_id = OLD.user\_id

) THEN

DELETE FROM users

WHERE user\_id = OLD.user\_id;

END IF;

END IF;

RETURN OLD;

END;

$$;

CREATE FUNCTION public.empty\_review\_check() RETURNS trigger

LANGUAGE plpgsql

AS $$

BEGIN

IF NEW.review\_content IS NULL OR TRIM(NEW.review\_content) = '' THEN

NEW.review\_content := 'Отзыв без подробностей';

END IF;

RETURN NEW;

END;

$$;

CREATE FUNCTION public.get\_trainer\_avg\_rating(trainer\_id integer) RETURNS double precision

LANGUAGE plpgsql

AS $$

BEGIN

RETURN (

SELECT AVG(review\_grade)

FROM trainer\_reviews

WHERE trainer\_id = trainer\_id AND moderated = true

);

END;

$$;

CREATE FUNCTION public.insert\_user\_on\_client() RETURNS trigger

LANGUAGE plpgsql

AS $$

DECLARE

new\_user\_id INT;

BEGIN

INSERT INTO users (login, password, role\_id)

VALUES ('', '', NULL)

RETURNING user\_id INTO new\_user\_id;

NEW.user\_id := new\_user\_id;

RETURN NEW;

END;

$$;

CREATE FUNCTION public.limit\_equipment\_quantity() RETURNS trigger

LANGUAGE plpgsql

AS $$

BEGIN

IF NEW.quantity > 50 THEN

NEW.quantity := 50;

END IF;

RETURN NEW;

END;

$$;

CREATE FUNCTION public.locker\_days\_left(client\_id integer) RETURNS integer

LANGUAGE plpgsql

AS $$

DECLARE

end\_date DATE;

BEGIN

SELECT rl.end\_date

INTO end\_date

FROM rented\_lockers rl

JOIN clientmembership cm ON rl.client\_membership\_id = cm.client\_membership\_id

WHERE cm.client\_id = client\_id

ORDER BY rl.end\_date DESC

LIMIT 1;

IF end\_date IS NULL THEN

RETURN 0;

ELSE

RETURN GREATEST(0, end\_date - CURRENT\_DATE);

END IF;

END;

$$;

CREATE FUNCTION public.membership\_days\_left(client\_id integer) RETURNS integer

LANGUAGE plpgsql

AS $$

DECLARE

end\_date DATE;

BEGIN

SELECT end\_date INTO end\_date

FROM clientmembership

WHERE client\_id = client\_id

ORDER BY end\_date DESC

LIMIT 1;

IF end\_date IS NULL THEN

RETURN 0;

ELSE

RETURN GREATEST(0, end\_date - CURRENT\_DATE);

END IF;

END;

$$;

CREATE FUNCTION public.set\_maintenance\_date() RETURNS trigger

LANGUAGE plpgsql

AS $$

BEGIN

IF NEW.last\_maintenance\_date IS NULL THEN

NEW.last\_maintenance\_date := NEW.delivery\_date;

END IF;

RETURN NEW;

END;

$$;

CREATE FUNCTION public.unpaid\_services\_sum(client\_id integer) RETURNS numeric

LANGUAGE plpgsql

AS $$

BEGIN

RETURN (

SELECT COALESCE(SUM(price), 0)

FROM services\_payments

WHERE client\_id = client\_id AND payment\_date IS NULL

);

END;

$$;

CREATE TABLE public.class (

class\_id integer NOT NULL PRIMARY KEY,

work\_schedule\_id integer NOT NULL REFERENCES public.workschedule(work\_schedule\_id),

start\_time time without time zone NOT NULL,

end\_time time without time zone NOT NULL,

hall\_id integer NOT NULL REFERENCES public.hall(hall\_id),

class\_type\_id integer NOT NULL REFERENCES public.classtype(class\_type\_id),

people\_quantity integer NOT NULL,

price double precision,

class\_info\_id integer REFERENCES public.class\_info(class\_info\_id),

trainer\_checked boolean DEFAULT false NOT NULL

);

CREATE TABLE public.class\_info (

class\_info\_id integer NOT NULL PRIMARY KEY,

class\_name character varying(100) NOT NULL,

description text

);

CREATE TABLE public.class\_payments (

class\_payment\_id integer NOT NULL PRIMARY KEY,

class\_id integer REFERENCES public.class(class\_id),

payment\_date date,

price double precision,

client\_id integer REFERENCES public.client(client\_id)

);

CREATE TABLE public.class\_reviews (

review\_id integer NOT NULL PRIMARY KEY,

class\_info\_id integer REFERENCES public.class\_info(class\_info\_id) ON DELETE SET NULL,

author\_id integer REFERENCES public.client(client\_id) ON DELETE SET NULL,

review\_grade integer,

review\_content text,

moderated boolean DEFAULT false,

CONSTRAINT reviews\_review\_grade\_check CHECK (((review\_grade >= 1) AND (review\_grade <= 5)))

);

CREATE TABLE public.classtype (

class\_type\_id integer NOT NULL PRIMARY KEY,

class\_type\_name character varying(50) NOT NULL

);

CREATE TABLE public.classvisits (

visit\_class\_id integer NOT NULL PRIMARY KEY,

class\_id integer NOT NULL REFERENCES public.class(class\_id),

client\_membership\_id integer NOT NULL REFERENCES public.clientmembership(client\_membership\_id),

visited boolean DEFAULT false

);

CREATE TABLE public.client (

client\_id integer NOT NULL PRIMARY KEY,

last\_name character varying(50) NOT NULL,

first\_name character varying(50) NOT NULL,

patronymic character varying(50),

gender\_id character(2) NOT NULL REFERENCES public.gender(gender\_id),

birth\_date date NOT NULL,

passport\_series character(4) NOT NULL,

passport\_number character(6) NOT NULL,

phone\_number character(15) NOT NULL,

user\_id integer REFERENCES public.users(user\_id) ON DELETE CASCADE,

email character varying(256) NOT NULL,

passport\_kem\_vidan character varying(250) NOT NULL,

passport\_kogda\_vidan date NOT NULL,

add\_author\_name boolean DEFAULT true,

balance numeric(10,2) DEFAULT 0.00 NOT NULL,

CONSTRAINT client\_birth\_date\_check1 CHECK ((birth\_date <= (CURRENT\_DATE - '14 years'::interval))),

CONSTRAINT uk\_client\_passport UNIQUE (passport\_series, passport\_number)

);

CREATE TABLE public.client\_transaction (

transaction\_id integer NOT NULL PRIMARY KEY,

client\_id integer NOT NULL REFERENCES public.client(client\_id),

operation\_description text,

payment\_way character varying(50),

amount numeric(10,2) NOT NULL,

transaction\_type character varying(20),

transaction\_date timestamp without time zone DEFAULT CURRENT\_TIMESTAMP

);

CREATE TABLE public.clientmembership (

client\_membership\_id integer NOT NULL PRIMARY KEY,

client\_id integer NOT NULL REFERENCES public.client(client\_id),

membership\_id integer NOT NULL REFERENCES public.membership(membership\_id),

start\_date date DEFAULT CURRENT\_DATE NOT NULL,

end\_date date NOT NULL

);

CREATE TABLE public.equipment (

equipment\_id integer NOT NULL PRIMARY KEY,

equipment\_name character varying(100) NOT NULL,

equipment\_condition\_id integer NOT NULL REFERENCES public.equipmentcondition(equipment\_condition\_id),

delivery\_date date NOT NULL,

last\_maintenance\_date date NOT NULL,

quantity integer NOT NULL

);

CREATE TABLE public.equipmentcondition (

equipment\_condition\_id integer NOT NULL PRIMARY KEY,

equipment\_condition\_description character varying(50) NOT NULL

);

CREATE TABLE public.gender (

gender\_id character(2) NOT NULL PRIMARY KEY,

gender\_name character varying(10) NOT NULL

);

CREATE TABLE public.hall (

hall\_id integer NOT NULL PRIMARY KEY,

hall\_name character varying(100) NOT NULL,

capacity integer NOT NULL,

description text,

area double precision NOT NULL

);

CREATE TABLE public.hallequipment (

equipment\_id integer NOT NULL REFERENCES public.equipment(equipment\_id),

hall\_id integer NOT NULL REFERENCES public.hall(hall\_id),

quantity integer NOT NULL,

PRIMARY KEY (equipment\_id, hall\_id)

);

CREATE TABLE public.lockers (

locker\_id integer NOT NULL PRIMARY KEY,

locker\_zone character varying(50),

month\_price double precision

);

CREATE TABLE public.membership (

membership\_id integer NOT NULL PRIMARY KEY,

membership\_name character varying(100) NOT NULL,

price integer NOT NULL,

membership\_type\_id integer NOT NULL REFERENCES public.membershiptype(membership\_type\_id),

membership\_description text

);

CREATE TABLE public.membership\_payments (

membership\_payment\_id integer NOT NULL PRIMARY KEY,

membership\_id integer REFERENCES public.membership(membership\_id),

payment\_date date,

price double precision,

client\_id integer REFERENCES public.client(client\_id)

);

CREATE TABLE public.membership\_services (

membershipservice\_id integer NOT NULL PRIMARY KEY,

client\_membership\_id integer REFERENCES public.clientmembership(client\_membership\_id) ON DELETE CASCADE,

service\_id integer REFERENCES public.services(service\_id) ON DELETE CASCADE,

provision\_date date

);

CREATE TABLE public.membershiptype (

membership\_type\_id integer NOT NULL PRIMARY KEY,

membership\_type\_name character varying(50) NOT NULL,

duration\_months integer

);

CREATE TABLE public.rented\_lockers (

rent\_locker\_id integer NOT NULL PRIMARY KEY,

locker\_id integer REFERENCES public.lockers(locker\_id) ON DELETE CASCADE,

client\_membership\_id integer REFERENCES public.clientmembership(client\_membership\_id) ON DELETE CASCADE,

start\_date date,

end\_date date,

rent\_price integer,

payment\_date date

);

CREATE TABLE public.roles (

role\_id integer NOT NULL PRIMARY KEY,

role\_name character varying(255) NOT NULL UNIQUE

);

CREATE TABLE public.service\_type (

service\_type\_id integer NOT NULL PRIMARY KEY,

type\_name character varying(100) NOT NULL

);

CREATE TABLE public.services (

service\_id integer NOT NULL PRIMARY KEY,

service\_type\_id integer REFERENCES public.service\_type(service\_type\_id) ON DELETE CASCADE,

service\_name character varying(100) NOT NULL,

description text,

price double precision,

free\_usage\_limit integer DEFAULT 0

);

CREATE TABLE public.services\_payments (

service\_payment\_id integer NOT NULL PRIMARY KEY,

service\_id integer REFERENCES public.services(service\_id),

payment\_date date,

price double precision,

client\_id integer REFERENCES public.client(client\_id)

);

CREATE TABLE public.specialization (

specialization\_id integer NOT NULL PRIMARY KEY,

specialization\_name character varying(100) NOT NULL,

achievements character varying(200),

education character varying(200)

);

CREATE TABLE public.trainer (

trainer\_id integer NOT NULL PRIMARY KEY,

photo text,

last\_name character varying(50) NOT NULL,

first\_name character varying(50) NOT NULL,

patronymic character varying(50),

gender\_id character(2) NOT NULL REFERENCES public.gender(gender\_id),

birth\_date date NOT NULL,

passport\_number character(6) NOT NULL,

passport\_series character(4) NOT NULL,

phone\_number character varying(15) NOT NULL,

specialization\_id integer REFERENCES public.specialization(specialization\_id),

user\_id integer REFERENCES public.users(user\_id) ON DELETE CASCADE,

email character varying(256) NOT NULL,

passport\_kogda\_vidan date NOT NULL,

passport\_kem\_vidan character varying(250) NOT NULL,

individual\_price double precision,

date\_of\_employment date,

CONSTRAINT trainer\_birth\_date\_check CHECK ((birth\_date <= (CURRENT\_DATE - '18 years'::interval))),

CONSTRAINT uk\_trainer\_passport UNIQUE (passport\_series, passport\_number)

);

CREATE TABLE public.trainer\_reviews (

trainer\_review\_id integer NOT NULL PRIMARY KEY,

trainer\_id integer REFERENCES public.trainer(trainer\_id),

author\_id integer REFERENCES public.client(client\_id) ON DELETE SET NULL,

review\_grade integer,

review\_content text,

moderated boolean DEFAULT false,

CONSTRAINT trainer\_reviews\_review\_grade\_check CHECK (((review\_grade >= 1) AND (review\_grade <= 5)))

);

CREATE TABLE public.training\_plan (

training\_plan\_id integer NOT NULL PRIMARY KEY,

client\_id integer NOT NULL REFERENCES public.client(client\_id) ON DELETE CASCADE,

trainer\_id integer NOT NULL REFERENCES public.trainer(trainer\_id) ON DELETE CASCADE,

plan text NOT NULL,

created\_at timestamp without time zone DEFAULT now() NOT NULL

);

CREATE TABLE public.users (

user\_id integer NOT NULL PRIMARY KEY,

login character varying(255) UNIQUE,

password character varying(255),

role\_id integer REFERENCES public.roles(role\_id) ON DELETE CASCADE

);

CREATE TABLE public.workschedule (

work\_schedule\_id integer NOT NULL PRIMARY KEY,

trainer\_id integer NOT NULL REFERENCES public.trainer(trainer\_id),

work\_date date NOT NULL,

start\_time time without time zone NOT NULL,

end\_time time without time zone NOT NULL

);

CREATE TRIGGER trg\_client\_delete\_user AFTER DELETE ON public.client FOR EACH ROW EXECUTE FUNCTION public.delete\_user\_when\_client\_deleted();

CREATE TRIGGER trg\_empty\_review\_class BEFORE INSERT ON public.class\_reviews FOR EACH ROW EXECUTE FUNCTION public.empty\_review\_check();

CREATE TRIGGER trg\_empty\_review\_trainer BEFORE INSERT ON public.trainer\_reviews FOR EACH ROW EXECUTE FUNCTION public.empty\_review\_check();

CREATE TRIGGER trg\_insert\_user BEFORE INSERT ON public.client FOR EACH ROW EXECUTE FUNCTION public.insert\_user\_on\_client();

CREATE TRIGGER trg\_limit\_hall\_equipment BEFORE INSERT ON public.hallequipment FOR EACH ROW EXECUTE FUNCTION public.limit\_equipment\_quantity();

CREATE TRIGGER trg\_set\_maintenance\_date BEFORE INSERT ON public.equipment FOR EACH ROW EXECUTE FUNCTION public.set\_maintenance\_date();

Приложение Б. Листинг кода AutorizationWin.xaml.cs

public partial class AuthorizationWin : Window

{

UserFromDb userFromDB = new UserFromDb();

public static Users currentUser { get; set; } = null;

public AuthorizationWin()

{

InitializeComponent();

}

private void registrationWinOpenBtn\_Click(object sender, RoutedEventArgs e)

{

RegistrationWin regWin = new RegistrationWin();

regWin.ShowDialog();

}

private void enterBtn\_Click(object sender, RoutedEventArgs e)

{

if (!(passwordTb.Text != "" && loginTb.Text != ""))

{

MessageBox.Show("Введите данные"); return;

}

else

{

currentUser = userFromDB.GetUser(loginTb.Text, passwordTb.Text);

if (currentUser != null)

{

MainWindow mainWin = new MainWindow();

mainWin.Show();

this.Hide();

}

}

}

}

Приложение В. Листинг кода UserFromDb.cs

using System;

using System.Linq;

using System.Windows;

using fitness\_club.Classes;

using Microsoft.EntityFrameworkCore;

using Microsoft.EntityFrameworkCore.Metadata.Builders;

namespace fitness\_club.Model

{

public class UserFromDb

{

public bool CheckPassword(string password)

{

if (password.Length < 6)

{

MessageBox.Show("Длина пароля не может быть короче 6 символов");

return false;

}

bool hasDigit = false, hasUpper = false, hasSpecial = false;

string specialChars = "!@#$^";

foreach (char c in password)

{

if (char.IsDigit(c)) hasDigit = true;

if (char.IsUpper(c)) hasUpper = true;

if (specialChars.Contains(c)) hasSpecial = true;

}

if (!hasDigit || !hasUpper)

{

MessageBox.Show("Пароль должен содержать хотя бы одну цифру и одну заглавную букву!");

return false;

}

if (!hasSpecial)

{

MessageBox.Show("Пароль должен содержать один из символов !@#$^");

return false;

}

return true;

}

public void AddTrainerAndUpdateUser(

string login,

string password,

int roleId,

string lastName,

string firstName,

string patronymic,

DateTime birthDate,

string phoneNumber,

string email,

string genderId,

string passportSeries,

string passportNumber,

string passportKemVidan,

DateTime passportKogdaVidan)

{

using (var db = new AppDbContext())

{

using (var transaction = db.Database.BeginTransaction())

{

try

{

// 1. Создаём пользователя

var user = new Users

{

Login = login,

Password = Verification.GetSHA512Hash(password),

RoleId = roleId

};

db.Users.Add(user);

db.SaveChanges(); // чтобы получить user.UserId

// 2. Создаём тренера

var trainer = new Trainer

{

LastName = lastName,

FirstName = firstName,

Patronymic = patronymic,

BirthDate = birthDate,

PhoneNumber = phoneNumber,

Email = email,

GenderId = genderId,

PassportSeries = passportSeries,

PassportNumber = passportNumber,

PassportKemVidan = passportKemVidan,

PassportKogdaVidan = passportKogdaVidan,

DateOfEmployment = DateTime.Today,

SpecializationId = 1, // или выбрать из UI

UserId = user.UserId

};

db.Trainers.Add(trainer);

db.SaveChanges();

transaction.Commit();

}

catch (Exception ex)

{

transaction.Rollback();

string inner = ex.InnerException?.Message ?? "Нет деталей";

throw new Exception($"Ошибка при добавлении тренера: {ex.Message}\nINNER: {inner}");

}

}

}

}

public bool CheckUser(string login)

{

using (var db = new AppDbContext())

{

try

{

bool userExists = db.Users.Any(u => u.Login == login);

if (userExists)

{

MessageBox.Show("Такой логин уже есть");

return false;

}

return true;

}

catch (Exception ex)

{

MessageBox.Show($"Ошибка при проверке логина: {ex.Message}");

return false;

}

}

}

public Users GetUser(string login, string password)

{

using (var db = new AppDbContext())

{

try

{

var user = db.Users

.FirstOrDefault(u => u.Login == login);

if (user == null)

{

MessageBox.Show("Пользователь с таким логином не найден.");

return null;

}

// Проверка пароля

if (!Verification.VerifySHA512Hash(password, user.Password))

{

MessageBox.Show("Неверный пароль.");

return null;

}

if (user.RoleId == 1)

{

var trainer = db.Users

.Include(u => u.Trainer)

.FirstOrDefault(u => u.Login == login);

return trainer;

}

else if (user.RoleId == 2)

{

var client = db.Users

.Include(u => u.Client)

.FirstOrDefault(u => u.Login == login);

return client;

}

else

{

var admin = db.Users

.FirstOrDefault(u => u.Login == login);

return admin;

}

}

catch (Exception ex)

{

MessageBox.Show($"Ошибка при получении пользователя: {ex.Message}");

return null;

}

}

}

public void AddClientAndUpdateUser(

string login,

string password,

int roleId,

string lastName,

string firstName,

string patronymic,

DateTime birthDate,

string phoneNumber,

string email,

string genderId,

string passportSeries,

string passportNumber,

string passportKemVidan,

DateTime passportKogdaVidan)

{

using (var db = new AppDbContext())

{

try

{

using (var transaction = db.Database.BeginTransaction())

{

// 1. Добавляем клиента

var client = new Client

{

LastName = lastName,

FirstName = firstName,

Patronymic = patronymic,

BirthDate = birthDate,

PhoneNumber = phoneNumber,

Email = email,

GenderId = genderId,

PassportSeries = passportSeries,

PassportNumber = passportNumber,

PassportKemVidan = passportKemVidan,

PassportKogdaVidan = passportKogdaVidan

};

db.Clients.Add(client);

db.SaveChanges();

// 2. Берем последнего пользователя (считаем что это он)

var user = db.Users

.OrderByDescending(u => u.UserId)

.FirstOrDefault();

if (user == null)

throw new Exception("Не найден ни один пользователь");

// 3. Обновляем его поля

user.Login = login;

user.Password = Verification.GetSHA512Hash(password);

user.RoleId = roleId;

db.SaveChanges();

transaction.Commit();

}

}

catch (Exception ex)

{

var inner = ex.InnerException?.Message ?? "Нет деталей";

throw new Exception($"Ошибка при добавлении клиента: {ex.Message}\nINNER: {inner}");

}

}

}

}

}

Приложение Г. Листинг кода Verification.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Security.Cryptography;

namespace fitness\_club.Classes

{

class Verification

{

public static string GetSHA512Hash(string input)

{

// Создаем новый экземпляр объекта MD5CryptoServiceProvider.

SHA512CryptoServiceProvider SHA512Hasher = new SHA512CryptoServiceProvider();

// Преобразуем входную строку в байтовый массив и вычисляем хеш.

byte[] data = SHA512Hasher.ComputeHash(Encoding.Default.GetBytes(input));

// Создаем новый Stringbuilder для сбора байтов и создания строки.

StringBuilder sBuilder = new StringBuilder();

// Перебираем каждый байт хэшированных данных

// и форматируем каждый как шестнадцатеричную строку.

for (int i = 0; i < data.Length; i++)

{

sBuilder.Append(data[i].ToString("x2"));

}

// Возвращаем шестнадцатеричную строку.

return sBuilder.ToString();

}

// Проверяет хэш (из базы данных) на соответствие строке (из поля ввода)

public static bool VerifySHA512Hash(string input, string hash)

{

// Хэшируем ввод.

string hashOfInput = GetSHA512Hash(input);

// Создаем StringComparer для сравнения хэшей.

StringComparer comparer = StringComparer.OrdinalIgnoreCase;

// Сравниваем два хэша.

if (0 == comparer.Compare(hashOfInput, hash))

{

return true; // Хэши равны.

}

else

{

return false; // Не равны.

}

}

public static bool CheckPasswordMatch(string currentPassword, string enteredPassword)

{

// Сравниваем хэшированные пароли

return currentPassword == enteredPassword;

}

}

}

Приложение Д. Листинг кода ClientSchedulePage.cs

using fitness\_club.Classes;

using fitness\_club.Model;

using fitness\_club.Windows;

using Microsoft.EntityFrameworkCore;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Navigation;

using System.Windows.Shapes;

namespace fitness\_club.Pages.ClientPages

{

/// <summary>

/// Логика взаимодействия для ClientSchedulePage.xaml

/// </summary>

public partial class ClientSchedulePage : Page

{

Users currentUser = null;

public ClientSchedulePage()

{

InitializeComponent();

currentUser = AuthorizationWin.currentUser;

LoadIndividualClasses();

LoadGroupClasses();

LoadTrainerList();

LoadAvailableGroupClasses();

}

private void LoadTrainerList()

{

using (var db = new AppDbContext())

{

TrainerComboBox.ItemsSource = db.Trainers

.Select(t => new

{

t.TrainerId,

FullName = t.LastName + " " + t.FirstName

})

.ToList();

}

}

public void LoadIndividualClasses()

{

try

{

using (var db = new AppDbContext())

{

var clientId = db.Clients

.Where(c => c.UserId == currentUser.UserId)

.Select(c => c.ClientId)

.FirstOrDefault();

var individualClasses = db.ClassVisits

.Include(cv => cv.Class)

.ThenInclude(c => c.ClassType)

.Include(cv => cv.Class)

.ThenInclude(c => c.WorkSchedule)

.ThenInclude(ws => ws.Trainer)

.Include(cv => cv.Class)

.ThenInclude(c => c.ClassInfo)

.Where(cv =>

cv.ClientMembership.ClientId == clientId &&

cv.Class.ClassType.ClassTypeName == "Индивидуальные" &&

cv.Class.WorkSchedule.WorkDate >= DateTime.Today)

.Select(cv => new

{

cv.Class.ClassId,

ClassName = cv.Class.ClassInfo.ClassName,

Date = cv.Class.WorkSchedule.WorkDate,

Time = $"{cv.Class.StartTime} - {cv.Class.EndTime}",

TrainerName = cv.Class.WorkSchedule.Trainer.LastName + " " + cv.Class.WorkSchedule.Trainer.FirstName

})

.ToList();

AuthorizedIndividualList.ItemsSource = individualClasses;

}

}

catch (Exception ex)

{

MessageBox.Show($"Ошибка загрузки индивидуальных занятий: {ex.Message}");

}

}

public void LoadGroupClasses()

{

try

{

using (var db = new AppDbContext())

{

var clientId = db.Clients

.Where(c => c.UserId == currentUser.UserId)

.Select(c => c.ClientId)

.FirstOrDefault();

var groupClasses = db.ClassVisits

.Include(cv => cv.Class)

.ThenInclude(c => c.ClassType)

.Include(cv => cv.Class)

.ThenInclude(c => c.WorkSchedule)

.ThenInclude(ws => ws.Trainer)

.Include(cv => cv.Class)

.ThenInclude(c => c.ClassInfo)

.Where(cv =>

cv.ClientMembership.ClientId == clientId &&

cv.Class.ClassType.ClassTypeName != "Индивидуальные" &&

cv.Class.WorkSchedule.WorkDate >= DateTime.Today)

.Select(cv => new

{

cv.Class.ClassId,

ClassName = cv.Class.ClassInfo.ClassName,

Date = cv.Class.WorkSchedule.WorkDate,

Time = $"{cv.Class.StartTime} - {cv.Class.EndTime}",

TrainerName = cv.Class.WorkSchedule.Trainer.LastName + " " + cv.Class.WorkSchedule.Trainer.FirstName

})

.ToList();

AuthorizedGroupList.ItemsSource = groupClasses;

}

}

catch (Exception ex)

{

MessageBox.Show($"Ошибка загрузки групповых занятий: {ex.Message}");

}

}

private void DurationComboBox\_SelectionChanged(object sender, SelectionChangedEventArgs e)

{

UpdateAvailableTimeSlots();

}

private void TrainerComboBox\_SelectionChanged(object sender, SelectionChangedEventArgs e)

{

UpdateAvailableTimeSlots();

}

private void UpdateAvailableTimeSlots()

{

TimeSlotsListBox.ItemsSource = null;

PriceTextBlock.Text = "";

if (TrainerComboBox.SelectedValue == null || DurationComboBox.SelectedItem == null)

return;

int trainerId = (int)TrainerComboBox.SelectedValue;

double duration = GetDurationFromComboBox();

var durationTime = TimeSpan.FromHours(duration);

try

{

using (var db = new AppDbContext())

{

// Получаем данные клиента

var clientId = db.Clients

.Where(c => c.UserId == currentUser.UserId)

.Select(c => c.ClientId)

.FirstOrDefault();

// Получаем тренера и его цену

var trainer = db.Trainers.FirstOrDefault(t => t.TrainerId == trainerId);

if (trainer?.IndividualPrice == null)

{

PriceTextBlock.Text = "Не указана цена тренировки";

PriceTextBlock.Foreground = Brushes.Red;

return;

}

// Получаем все рабочие расписания тренера (ближайшие 14 дней)

var today = DateTime.Today;

var schedules = db.WorkSchedules

.Where(ws => ws.TrainerId == trainerId &&

ws.WorkDate >= today &&

ws.WorkDate <= today.AddDays(14))

.ToList();

// Получаем все занятия тренера

var trainerClasses = db.Class

.Include(c => c.WorkSchedule)

.Where(c => c.WorkSchedule.TrainerId == trainerId)

.Select(c => new

{

c.WorkSchedule.WorkDate,

Start = c.StartTime,

End = c.EndTime

})

.ToList();

// Получаем все занятия клиента

var clientClasses = db.ClassVisits

.Include(cv => cv.Class)

.ThenInclude(c => c.WorkSchedule)

.Where(cv => cv.ClientMembership.ClientId == clientId)

.Select(cv => new

{

cv.Class.WorkSchedule.WorkDate,

Start = cv.Class.StartTime,

End = cv.Class.EndTime

})

.ToList();

var availableSlots = new List<string>();

foreach (var schedule in schedules)

{

var currentTime = schedule.StartTime;

while (currentTime + durationTime <= schedule.EndTime)

{

var slotEnd = currentTime + durationTime;

bool isTrainerBusy = trainerClasses.Any(tc =>

tc.WorkDate == schedule.WorkDate &&

currentTime < tc.End && slotEnd > tc.Start);

if (!isTrainerBusy)

{

availableSlots.Add($"{schedule.WorkDate:yyyy-MM-dd} {currentTime:hh\\:mm}");

}

currentTime = currentTime.Add(TimeSpan.FromMinutes(30));

}

}

TimeSlotsListBox.ItemsSource = availableSlots;

if (availableSlots.Any())

{

PriceTextBlock.Text = $"Стоимость: {trainer.IndividualPrice \* duration} руб.";

PriceTextBlock.Foreground = Brushes.Green;

}

else

{

PriceTextBlock.Text = "Нет доступных слотов";

PriceTextBlock.Foreground = Brushes.Red;

}

}

}

catch (Exception ex)

{

MessageBox.Show($"Ошибка при загрузке слотов: {ex.Message}");

}

}

private double GetDurationFromComboBox()

{

if (DurationComboBox.SelectedItem is ComboBoxItem item)

{

return item.Content.ToString() switch

{

"1.5 часа" => 1.5,

"2 часа" => 2.0,

\_ => 1.0

};

}

return 1.0;

}

private bool CanTrainerTakeClass(int trainerId, DateTime date, TimeSpan start, TimeSpan end, bool isIndividual)

{

using (var db = new AppDbContext())

{

var schedule = db.WorkSchedules.FirstOrDefault(ws =>

ws.TrainerId == trainerId &&

ws.WorkDate == date &&

ws.StartTime <= start &&

ws.EndTime >= end);

if (schedule == null)

{

MessageBox.Show("Тренер не работает в выбранное время.");

return false;

}

var overlappingClass = db.Class

.Include(c => c.ClassType)

.Include(c => c.WorkSchedule)

.Where(c =>

c.WorkSchedule.TrainerId == trainerId &&

c.WorkSchedule.WorkDate == date &&

(

(start >= c.StartTime && start < c.EndTime) ||

(end > c.StartTime && end <= c.EndTime) ||

(start <= c.StartTime && end >= c.EndTime)

))

.ToList();

foreach (var c in overlappingClass)

{

bool existingIsIndividual = c.ClassType.ClassTypeName == "Индивидуальные";

if (existingIsIndividual || isIndividual)

{

MessageBox.Show("У тренера уже есть занятие в это время.");

return false;

}

}

return true;

}

}

private void AuthorizedIndividual\_Click(object sender, RoutedEventArgs e)

{

if (TrainerComboBox.SelectedValue == null || TimeSlotsListBox.SelectedItem == null || DurationComboBox.SelectedItem == null)

{

MessageBox.Show("Выберите тренера, длительность и время.");

return;

}

int trainerId = (int)TrainerComboBox.SelectedValue;

double duration = GetDurationFromComboBox();

string slot = TimeSlotsListBox.SelectedItem.ToString();

var split = slot.Split(' ');

DateTime date = DateTime.Parse(split[0]);

TimeSpan startTime = TimeSpan.Parse(split[1]);

TimeSpan endTime = startTime.Add(TimeSpan.FromHours(duration));

using (var db = new AppDbContext())

{

int clientId = db.Clients.First(c => c.UserId == AuthorizationWin.currentUser.UserId).ClientId;

var membership = db.ClientMemberships

.FirstOrDefault(cm => cm.ClientId == clientId && cm.EndDate.Date >= DateTime.UtcNow.Date && cm.StartDate <= DateTime.UtcNow.Date);

if (membership == null)

{

MessageBox.Show("Нет активного абонемента.");

return;

}

if (IsClientBusyAt(clientId, date, startTime, endTime))

{

MessageBox.Show("У вас уже есть занятие в это время.");

return;

}

if (!CanTrainerTakeClass(trainerId, date, startTime, endTime, true))

{

return;

}

var schedule = db.WorkSchedules.First(ws =>

ws.TrainerId == trainerId &&

ws.WorkDate == date &&

ws.StartTime <= startTime &&

ws.EndTime >= endTime);

var trainer = db.Trainers.First(t => t.TrainerId == trainerId);

double totalPrice = trainer.IndividualPrice.Value \* duration;

var newClass = new Class

{

ClassInfoId = 1,

ClassTypeId = 2,

HallId = 5,

WorkScheduleId = schedule.WorkScheduleId,

PeopleQuantity = 1,

StartTime = startTime,

EndTime = endTime,

Price = totalPrice

};

db.Class.Add(newClass);

db.SaveChanges();

db.ClassVisits.Add(new ClassVisits

{

ClassId = newClass.ClassId,

ClientMembershipId = membership.ClientMembershipId

});

db.SaveChanges();

var classPayment = new ClassPayments

{

ClassId = newClass.ClassId,

PaymentDate = null,

Price = totalPrice,

ClientId = clientId

};

db.ClassPayments.Add(classPayment);

db.SaveChanges();

MessageBox.Show("Вы успешно записались!");

LoadIndividualClasses();

UpdateAvailableTimeSlots();

}

}

private bool IsClientBusyAt(int clientId, DateTime date, TimeSpan start, TimeSpan end)

{

using (var db = new AppDbContext())

{

var busyVisits = db.ClassVisits

.Include(cv => cv.Class)

.ThenInclude(c => c.WorkSchedule)

.Where(cv =>

cv.ClientMembership.ClientId == clientId &&

cv.Class.WorkSchedule.WorkDate == date)

.ToList();

foreach (var visit in busyVisits)

{

var existingStart = visit.Class.StartTime;

var existingEnd = visit.Class.EndTime;

// Проверка на пересечение

if (start < existingEnd && end > existingStart)

{

return true;

}

}

return false;

}

}

private void AuthorizedGroup\_Click(object sender, RoutedEventArgs e)

{

var selectedClass = AvailableGroupList.SelectedItem;

if (selectedClass != null)

SignUpOnGroupClass(selectedClass);

else

MessageBox.Show("Вы не выбрали занятие для записи");

}

private void SignUpOnGroupClass(dynamic selectedClass)

{

if (AvailableGroupList.SelectedItem == null)

{

MessageBox.Show("Выберите занятие из списка.");

return;

}

int classId = selectedClass.ClassId;

using (var db = new AppDbContext())

{

int clientId = db.Clients

.Where(c => c.UserId == AuthorizationWin.currentUser.UserId)

.Select(c => c.ClientId)

.FirstOrDefault();

var membership = db.ClientMemberships

.FirstOrDefault(cm => cm.ClientId == clientId && cm.EndDate.Date >= DateTime.UtcNow.Date && cm.StartDate.Date <= DateTime.UtcNow.Date);

if (membership == null)

{

MessageBox.Show("У вас нет активного абонемента.");

return;

}

var classObj = db.Class

.Include(c => c.WorkSchedule)

.Include(c => c.ClassVisits)

.FirstOrDefault(c => c.ClassId == classId);

if (classObj == null)

{

MessageBox.Show("Занятие не найдено.");

return;

}

// Проверка: клиент уже записан?

bool alreadySigned = db.ClassVisits.Any(cv =>

cv.ClientMembership.ClientId == clientId &&

cv.ClassId == classId);

if (alreadySigned)

{

MessageBox.Show("Вы уже записаны на это занятие.");

return;

}

var clientVisits = db.ClassVisits

.Include(cv => cv.Class)

.ThenInclude(c => c.WorkSchedule)

.Where(cv => cv.ClientMembership.ClientId == clientId &&

cv.Class.WorkSchedule.WorkDate == classObj.WorkSchedule.WorkDate)

.ToList();

var newStart = classObj.StartTime;

var newEnd = classObj.EndTime;

bool hasOverlap = clientVisits.Any(v =>

{

var existing = v.Class;

return newStart < existing.EndTime && newEnd > existing.StartTime;

});

if (hasOverlap)

{

MessageBox.Show("У вас уже есть занятие в это время.");

return;

}

// Проверка на свободные места

if (classObj.ClassVisits.Count >= classObj.PeopleQuantity)

{

MessageBox.Show("На это занятие нет свободных мест.");

return;

}

// Запись

db.ClassVisits.Add(new ClassVisits

{

ClassId = classObj.ClassId,

ClientMembershipId = membership.ClientMembershipId

});

db.SaveChanges();

DateTime classDate = db.Class

.Where(c => c.ClassId == classId)

.Select(c => c.WorkSchedule.WorkDate)

.FirstOrDefault();

//Добавление записи в таблицу ClassPayments

var classPayment = new ClassPayments

{

ClassId = classObj.ClassId,

PaymentDate = classObj.Price == 0 ? DateTime.SpecifyKind(classDate, DateTimeKind.Utc) : null,

Price = classObj.Price ?? 0,

ClientId = clientId

};

db.ClassPayments.Add(classPayment);

db.SaveChanges();

MessageBox.Show("Вы успешно записались на групповое занятие!");

LoadGroupClasses();

LoadAvailableGroupClasses();

}

}

private void LoadAvailableGroupClasses()

{

using (var db = new AppDbContext())

{

int clientId = db.Clients

.Where(c => c.UserId == AuthorizationWin.currentUser.UserId)

.Select(c => c.ClientId)

.FirstOrDefault();

var today = DateTime.Today;

var availableGroupClasses = db.Class

.Include(c => c.ClassType)

.Include(c => c.WorkSchedule).ThenInclude(ws => ws.Trainer)

.Include(c => c.Hall)

.Include(c => c.ClassInfo)

.Include(c => c.ClassVisits).ThenInclude(cv => cv.ClientMembership)

.Where(c =>

c.ClassType.ClassTypeName != "Индивидуальные" &&

c.WorkSchedule.WorkDate >= today &&

!c.ClassVisits.Any(cv => cv.ClientMembership != null && cv.ClientMembership.ClientId == clientId))

.Select(c => new

{

c.ClassId,

ClassName = c.ClassInfo.ClassName,

Date = c.WorkSchedule.WorkDate,

Time = $"{c.StartTime:hh\\:mm} - {c.EndTime:hh\\:mm}",

HallName = c.Hall.HallName,

TrainerName = $"{c.WorkSchedule.Trainer.LastName} {c.WorkSchedule.Trainer.FirstName}",

Price = c.Price,

FreeSpots = c.PeopleQuantity - c.ClassVisits.Count

})

.ToList();

AvailableGroupList.ItemsSource = availableGroupClasses;

}

}

private void OpenClassDetails(dynamic selectedItem)

{

try

{

if (selectedItem != null)

{

int classId = selectedItem.ClassId;

ClassDetailsWindow cdw = new ClassDetailsWindow(classId, "schedulePage");

cdw.Show();

}

else

{

MessageBox.Show("Вы не выбрали занятие");

}

}

catch (Exception ex)

{

MessageBox.Show($"Не удалось загрузить информацию о занятии: {ex.Message}");

}

}

private dynamic GetSelectedItemFromContext(object sender)

{

var menuItem = sender as MenuItem;

if (menuItem == null) return null;

var contextMenu = menuItem.Parent as ContextMenu;

if (contextMenu == null) return null;

var border = contextMenu.PlacementTarget as Border;

if (border == null) return null;

// Ищем родительский ListViewItem

var listViewItem = FindVisualParent<ListViewItem>(border);

if (listViewItem == null) return null;

return listViewItem.DataContext;

}

private T FindVisualParent<T>(DependencyObject child) where T : DependencyObject

{

var parentObject = VisualTreeHelper.GetParent(child);

if (parentObject == null) return null;

if (parentObject is T parent)

return parent;

else

return FindVisualParent<T>(parentObject);

}

private void AuthorizedIndividualList\_MouseDoubleClick(object sender, MouseButtonEventArgs e)

{

dynamic selectedItem = AuthorizedIndividualList.SelectedItem;

if (selectedItem != null)

OpenClassDetails(selectedItem);

else

MessageBox.Show("Вы не выбрали занятие, о котором хотите загрузить информацию");

}

private void AuthorizedGroupList\_MouseDoubleClick(object sender, MouseButtonEventArgs e)

{

dynamic selectedItem = AuthorizedGroupList.SelectedItem;

if (selectedItem != null)

OpenClassDetails(selectedItem);

else

MessageBox.Show("Вы не выбрали занятие, о котором хотите загрузить информацию");

}

private void AvailableGroupList\_MouseDoubleClick(object sender, MouseButtonEventArgs e)

{

dynamic selectedItem = AvailableGroupList.SelectedItem;

if (selectedItem != null)

OpenClassDetails(selectedItem);

else

MessageBox.Show("Вы не выбрали занятие, о котором хотите загрузить информацию");

}

private void SignUpGroupClassContext\_Click(object sender, RoutedEventArgs e)

{

var selectedClass = GetSelectedItemFromContext(sender);

if (selectedClass != null)

SignUpOnGroupClass(selectedClass);

else

MessageBox.Show("Вы не выбрали занятие для записи");

}

private void AvailableGroupDetailsContext\_Click(object sender, RoutedEventArgs e)

{

var selectedItem = GetSelectedItemFromContext(sender);

if (selectedItem != null)

OpenClassDetails(selectedItem);

else

MessageBox.Show("Вы не выбрали занятие, о котором хотите загрузить информацию");

}

private void IndividualClassDetailsContext\_Click(object sender, RoutedEventArgs e)

{

var selectedItem = GetSelectedItemFromContext(sender);

if (selectedItem != null)

OpenClassDetails(selectedItem);

else

MessageBox.Show("Вы не выбрали занятие, о котором хотите загрузить информацию");

}

private void GroupClassDetailsContext\_Click(object sender, RoutedEventArgs e)

{

var selectedItem = GetSelectedItemFromContext(sender);

if (selectedItem != null)

OpenClassDetails(selectedItem);

else

MessageBox.Show("Вы не выбрали занятие, о котором хотите загрузить информацию");

}

private void CancelClientClass(int classId, bool isIndividual)

{

try

{

using var db = new AppDbContext();

var clientId = AuthorizationWin.currentUser.Client.ClientId;

var todayUtc = DateTime.SpecifyKind(DateTime.UtcNow, DateTimeKind.Utc);

var classVisit = db.ClassVisits

.Include(cv => cv.ClientMembership)

.FirstOrDefault(cv => cv.ClassId == classId && cv.ClientMembership.ClientId == clientId);

if (classVisit == null)

{

MessageBox.Show("Запись не найдена.");

return;

}

if (classVisit.Visited)

{

MessageBox.Show("Нельзя отменить занятие, которое уже посещено.");

return;

}

var payment = db.ClassPayments

.FirstOrDefault(cp => cp.ClassId == classId && cp.ClientId == clientId);

if (payment != null && payment.PaymentDate != null)

{

var client = db.Clients.FirstOrDefault(c => c.ClientId == clientId);

if (client != null)

{

client.Balance += (decimal)payment.Price;

db.ClientTransactions.Add(new ClientTransaction

{

ClientId = client.ClientId,

OperationDescription = isIndividual

? "Возврат за отмену индивидуального занятия"

: "Возврат за отмену группового занятия",

PaymentWay = "На баланс",

Amount = (decimal)payment.Price,

TransactionType = "возврат",

TransactionDate = todayUtc

});

}

}

if (payment != null)

db.ClassPayments.Remove(payment);

db.ClassVisits.Remove(classVisit);

if (isIndividual)

{

var classToDelete = db.Class.FirstOrDefault(c => c.ClassId == classId);

if (classToDelete != null)

db.Class.Remove(classToDelete);

}

db.SaveChanges();

if (isIndividual)

{

LoadIndividualClasses();

UpdateAvailableTimeSlots();

}

else

{

LoadGroupClasses();

LoadAvailableGroupClasses();

}

MessageBox.Show("Запись на занятие отменена.");

}

catch (Exception ex)

{

MessageBox.Show($"Ошибка при отмене: {ex.Message}");

}

}

private void CancelIndividualClassContext\_Click(object sender, RoutedEventArgs e)

{

if (sender is MenuItem menuItem && menuItem.CommandParameter is int classId)

{

CancelClientClass(classId, isIndividual: true);

}

}

private void CancelGroupClassContext\_Click(object sender, RoutedEventArgs e)

{

if (sender is MenuItem menuItem && menuItem.CommandParameter is int classId)

{

CancelClientClass(classId, isIndividual: false);

}

}

}

}

Приложение Е. Листинг кода ClientMembershipPage.xaml.cs

using fitness\_club.Model;

using fitness\_club.Classes;

using fitness\_club.Windows;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Navigation;

using Microsoft.EntityFrameworkCore;

using System.Windows.Shapes;

namespace fitness\_club.Pages.ClientPages

{

/// <summary>

/// Логика взаимодействия для ClientMembershipPage.xaml

/// </summary>

public partial class ClientMembershipPage : Page

{

DateTime today = DateTime.SpecifyKind(DateTime.Today, DateTimeKind.Utc);

public ClientMembershipPage()

{

InitializeComponent();

LoadMemberships();

LoadMembershipTypes();

}

public void LoadMembershipTypes()

{

using(var db = new AppDbContext())

{

var membershipTypes = db.MembershipTypes.Where(mt=>mt.MembershipTypeName != "Пробная тренировка").OrderBy(mt=>mt.MembershipTypeId).ToList();

membershipDurationCmb.ItemsSource = membershipTypes;

membershipDurationCmb.DisplayMemberPath = "MembershipTypeName";

membershipDurationCmb.SelectedValuePath = "MembershipTypeId";

membershipDurationCmb.SelectedValue = 1;

}

}

public void SuccessfullPayment(int membershipId)

{

MessageBoxResult result = MessageBox.Show(

"Вы успешно приобрели абонемент. Для оплаты перейдите на страницу оплаты!",

"Успех",

MessageBoxButton.OK,

MessageBoxImage.Information);

if (result == MessageBoxResult.OK)

{

using (var db = new AppDbContext())

{

var currentClientMembership = db.ClientMemberships

.Where(cm => cm.ClientId == AuthorizationWin.currentUser.Client.ClientId &&

cm.EndDate >= today)

.OrderByDescending(cm => cm.EndDate)

.FirstOrDefault();

var membership = db.Memberships

.FirstOrDefault(m => m.MembershipId == membershipId);

int? membershipDuration = db.MembershipTypes.Where(mt => membership.MembershipTypeId == mt.MembershipTypeId).Select(mt => mt.DurationMonths).FirstOrDefault();

if (currentClientMembership == null)

{

var newClientMembership = new ClientMembership

{

MembershipId = membershipId,

ClientId = AuthorizationWin.currentUser.Client.ClientId,

StartDate = today,

EndDate = today.AddMonths((int)membershipDuration)

};

db.ClientMemberships.Add(newClientMembership);

db.SaveChanges();

db.MembershipPayments.Add(new MembershipPayments

{

MembershipId = membershipId,

PaymentDate = null,

Price = membership.Price,

ClientId = AuthorizationWin.currentUser.Client.ClientId,

});

db.SaveChanges();

}

else

{

var newClientMembership = new ClientMembership

{

MembershipId = membershipId,

ClientId = AuthorizationWin.currentUser.Client.ClientId,

StartDate = DateTime.SpecifyKind(currentClientMembership.EndDate.AddDays(1), DateTimeKind.Utc),

EndDate = DateTime.SpecifyKind(

currentClientMembership.EndDate.AddDays(1).AddMonths((int)membershipDuration),

DateTimeKind.Utc)

};

db.ClientMemberships.Add(newClientMembership);

db.SaveChanges();

db.MembershipPayments.Add(new MembershipPayments

{

MembershipId = membershipId,

PaymentDate = null,

Price = membership.Price,

ClientId = AuthorizationWin.currentUser.Client.ClientId,

});

db.SaveChanges();

}

}

LoadMemberships();

}

}

public void LoadMemberships()

{

using (var db = new AppDbContext())

{

var userId = AuthorizationWin.currentUser.UserId;

var client = db.Clients.FirstOrDefault(c => c.UserId == userId);

if (client == null) return;

var data = db.ClientMemberships

.Where(cm => cm.ClientId == client.ClientId)

.Include(cm => cm.Membership)

.ThenInclude(m => m.MembershipType)

.ToList()

.Select(cm => new

{

cm.Membership.MembershipName,

cm.Membership.MembershipDescription,

MembershipType = cm.Membership.MembershipType.MembershipTypeName,

cm.StartDate,

cm.EndDate,

Price = cm.Membership.Price + " ₽",

Status = cm.EndDate >= DateTime.Today ? "Активный" : "Истёкший"

})

.ToList();

MembershipGrid.ItemsSource = data;

}

}

public void LoadAvailableMemberships()

{

using (var db = new AppDbContext())

{

if (membershipDurationCmb.SelectedValue is int selectedTypeId)

{

var memberships = db.Memberships

.Include(m => m.MembershipType)

.Where(m => m.MembershipTypeId == selectedTypeId)

.OrderByDescending(m => m.Price)

.ToList();

membershipList.ItemsSource = memberships;

}

}

}

private void membershipDurationCmb\_SelectionChanged(object sender, SelectionChangedEventArgs e)

{

costTb.Visibility = Visibility.Collapsed;

LoadAvailableMemberships();

}

private void buyMemembershipBtn\_Click(object sender, RoutedEventArgs e)

{

dynamic selectedItem = membershipList.SelectedItem;

SuccessfullPayment(selectedItem.MembershipId);

}

private void membershipList\_SelectionChanged(object sender, SelectionChangedEventArgs e)

{

dynamic selectedItem = membershipList.SelectedItem;

if(selectedItem != null)

{

costTb.Visibility = Visibility.Visible;

costTb.Text = $"К оплате: {selectedItem.Price} руб.";

costTb.Foreground = Brushes.Green;

}

}

}

}

Приложение Ж. Листинг кода ClientPaymentPage.xaml.cs

using fitness\_club.Model;

using fitness\_club.Windows;

using Microsoft.EntityFrameworkCore;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Navigation;

using System.Windows.Shapes;

namespace fitness\_club.Pages.ClientPages

{

public partial class ClientPaymentPage : Page

{

DateTime todayUtc = DateTime.SpecifyKind(DateTime.Today, DateTimeKind.Utc);

int membershipId = 0;

public ClientPaymentPage()

{

InitializeComponent();

GridLoaded();

}

public void GridLoaded()

{

GetMemberShipId();

LoadServicesDebts();

LoadMembershipDebts();

LoadLockerDebt();

LoadClassDebts();

}

public void GetMemberShipId()

{

using (var db = new AppDbContext())

{

membershipId = db.ClientMemberships

.Where(cm => cm.ClientId == AuthorizationWin.currentUser.Client.ClientId && cm.EndDate >= todayUtc && cm.StartDate <= todayUtc)

.Select(cm => cm.ClientMembershipId)

.FirstOrDefault();

}

}

public void LoadServicesDebts()

{

using (var db = new AppDbContext())

{

var servicesDebts = db.ServicesPayments

.Include(sp => sp.Service)

.ThenInclude(s => s.ServiceType)

.Where(sp => sp.ClientId == AuthorizationWin.currentUser.Client.ClientId && sp.PaymentDate == null)

.Select(sp => new ServiceDebt

{

ServiceId = sp.ServiceId,

ServiceName = sp.Service.ServiceName,

ServiceType = sp.Service.ServiceType.TypeName,

Description = sp.Service.Description,

Price = sp.Price,

ProvisionDate = sp.Service.MembershipServices.Where(ms=>ms.ServiceId == sp.ServiceId && ms.ClientMembershipId == membershipId)

.Select(ms => ms.ProvisionDate)

.FirstOrDefault()

})

.ToList();

if(servicesDebts.Count != 0)

{

unPayedServicesList.ItemsSource = servicesDebts;

double totalDebt = servicesDebts.Sum(d => d.Price);

payForAllServicesDebts.Content = $"Оплатить все ({totalDebt} руб.)";

}

else

{

unPayedServicesList.Visibility = Visibility.Collapsed;

payForAllServicesDebts.Visibility = Visibility.Collapsed;

servicesToBeProceedTb.Text = "Нет услуг для оплаты";

}

}

}

public void LoadMembershipDebts()

{

using (var db = new AppDbContext())

{

var clientId = AuthorizationWin.currentUser.Client.ClientId;

// Загружаем неоплаченные платежи

var debts = db.MembershipPayments

.Include(mp => mp.Membership)

.ThenInclude(m => m.MembershipType)

.Where(mp => mp.ClientId == clientId && mp.PaymentDate == null)

.ToList();

// Получаем client\_membership для всех этих membershipId

var membershipIds = debts.Select(mp => mp.MembershipId).ToList();

var membershipsInfo = db.ClientMemberships

.Where(cm => cm.ClientId == clientId && membershipIds.Contains(cm.MembershipId))

.ToList();

// Объединяем вручную

var result = debts.Select(mp =>

{

var cm = membershipsInfo.FirstOrDefault(x => x.MembershipId == mp.MembershipId);

return new MembershipDebt

{

MembershipId = mp.MembershipId,

MembershipName = mp.Membership.MembershipName,

MembershipTypeName = mp.Membership.MembershipType.MembershipTypeName,

StartDate = cm?.StartDate ?? DateTime.MinValue,

EndDate = cm?.EndDate ?? DateTime.MinValue,

Price = mp.Price

};

}).ToList();

// Вывод

if (result.Any())

{

unPayedMembershipList.ItemsSource = result;

double total = result.Sum(x => x.Price);

payForAllMembershipDebts.Content = $"Оплатить все ({total} руб.)";

}

else

{

unPayedMembershipList.Visibility = Visibility.Collapsed;

payForAllMembershipDebts.Visibility = Visibility.Collapsed;

membershipsToBeProceedTb.Text = "Нет абонементов для оплаты";

}

}

}

public void UpdatePaidServices(List<int> serviceIds)

{

using (var db = new AppDbContext())

{

// Находим все неоплаченные платежи для выбранных услуг

var paymentsToUpdate = db.ServicesPayments

.Where(sp => sp.ClientId == AuthorizationWin.currentUser.Client.ClientId &&

sp.PaymentDate == null &&

serviceIds.Contains(sp.ServiceId))

.ToList();

// Помечаем как оплаченные

foreach (var payment in paymentsToUpdate)

{

payment.PaymentDate = todayUtc;

}

db.SaveChanges();

// Обновляем список

LoadServicesDebts();

}

}

public void UpdatePaidMemberships(List<int> membershipIds)

{

using (var db = new AppDbContext())

{

// Получаем сегодняшнюю дату в UTC

DateTime todayUtc = DateTime.SpecifyKind(DateTime.Today, DateTimeKind.Utc);

// Находим все неоплаченные платежи по указанным абонементам

var paymentsToUpdate = db.MembershipPayments

.Where(mp => mp.ClientId == AuthorizationWin.currentUser.Client.ClientId &&

mp.PaymentDate == null &&

membershipIds.Contains(mp.MembershipId))

.ToList();

// Помечаем как оплаченные

foreach (var payment in paymentsToUpdate)

{

payment.PaymentDate = todayUtc;

}

db.SaveChanges();

// Обновляем отображение задолженности

LoadMembershipDebts();

}

}

public void LoadLockerDebt()

{

using (var db = new AppDbContext())

{

var lockerDebt = db.RentedLockers

.Include(rl => rl.Locker)

.Include(rl => rl.ClientMembership)

.Where(rl => rl.ClientMembership.ClientId == AuthorizationWin.currentUser.Client.ClientId &&

rl.PaymentDate == null)

.Select(rl => new LockerDebt

{

RentLockerId = rl.RentLockerId,

LockerId = rl.LockerId,

LockerZone = rl.Locker.LockerZone,

StartDate = rl.StartDate,

EndDate = rl.EndDate,

RentPrice = rl.RentPrice

})

.FirstOrDefault();

if (lockerDebt != null)

{

lockerPaymentPanel.DataContext = lockerDebt;

}

else

{

lockerPaymentPanel.Visibility = Visibility.Collapsed;

lockerTitle.Text = "Нет задолженности по шкафчику";

}

}

}

public void LoadClassDebts()

{

using (var db = new AppDbContext())

{

var classDebts = db.ClassPayments

.Include(cp => cp.Class)

.ThenInclude(c => c.ClassInfo)

.Include(cp => cp.Class)

.ThenInclude(c => c.ClassType)

.Include(cp => cp.Class)

.ThenInclude(c => c.WorkSchedule)

.Where(cp => cp.ClientId == AuthorizationWin.currentUser.Client.ClientId &&

cp.PaymentDate == null &&

cp.Class.WorkSchedule != null &&

cp.Class.WorkSchedule.WorkDate != null &&

cp.Class.WorkScheduleId != null)

.Select(cp => new ClassDebt

{

ClassId = cp.ClassId,

ClassName = cp.Class.ClassInfo.ClassName,

Description = cp.Class.ClassInfo.Description,

TrainerName = $"{cp.Class.WorkSchedule.Trainer.LastName} {cp.Class.WorkSchedule.Trainer.LastName} {cp.Class.WorkSchedule.Trainer.Patronymic}",

Date = cp.Class.WorkSchedule.WorkDate,

StartTime = cp.Class.StartTime,

Price = cp.Price

})

.ToList();

if (classDebts.Any())

{

unPayedClassList.ItemsSource = classDebts;

double total = classDebts.Sum(x => x.Price);

payAllClassesBtn.Content = $"Оплатить все ({total} руб.)";

}

else

{

unPayedClassList.Visibility = Visibility.Collapsed;

payAllClassesBtn.Visibility = Visibility.Collapsed;

classTitle.Text = "Нет занятий для оплаты";

}

}

}

public void UpdateLockerPayment(int rentLockerId)

{

using (var db = new AppDbContext())

{

var locker = db.RentedLockers

.FirstOrDefault(rl => rl.RentLockerId == rentLockerId &&

rl.ClientMembership.ClientId == AuthorizationWin.currentUser.Client.ClientId &&

rl.PaymentDate == null);

if (locker != null)

{

locker.PaymentDate = DateTime.SpecifyKind(DateTime.Today, DateTimeKind.Utc);

db.SaveChanges();

LoadLockerDebt();

}

}

}

public void UpdateMultipleClassPayments(List<int> classIds)

{

using (var db = new AppDbContext())

{

var classPayments = db.ClassPayments

.Where(cp => cp.ClientId == AuthorizationWin.currentUser.Client.ClientId &&

cp.PaymentDate == null &&

classIds.Contains(cp.ClassId))

.ToList();

var todayUtc = DateTime.SpecifyKind(DateTime.Today, DateTimeKind.Utc);

foreach (var cp in classPayments)

{

cp.PaymentDate = todayUtc;

}

db.SaveChanges();

LoadClassDebts();

}

}

private void payService\_Click(object sender, RoutedEventArgs e)

{

var button = sender as Button;

var selectedItem = button?.DataContext as ServiceDebt;

if (selectedItem != null)

{

var paymentWindow = new PaymentWindow($"услугу '{selectedItem.ServiceName}'", (int)selectedItem.Price);

if (paymentWindow.ShowDialog() == true && paymentWindow.PaymentSuccess)

{

// Обновляем только одну оплаченную услугу

UpdatePaidServices(new List<int> { selectedItem.ServiceId });

}

}

}

private void payForAllServicesDebts\_Click(object sender, RoutedEventArgs e)

{

var allDebts = unPayedServicesList.ItemsSource as List<ServiceDebt>;

if (allDebts == null || !allDebts.Any())

{

MessageBox.Show("Нет услуг для оплаты");

return;

}

double totalSum = allDebts.Sum(d => d.Price);

var paymentWindow = new PaymentWindow("все услуги", (int)totalSum);

if (paymentWindow.ShowDialog() == true && paymentWindow.PaymentSuccess)

{

UpdatePaidServices(allDebts.Select(d => d.ServiceId).ToList());

payForAllServicesDebts.Visibility = Visibility.Collapsed;

}

}

private void payMembership\_Click(object sender, RoutedEventArgs e)

{

var button = sender as Button;

var selectedItem = button?.DataContext as MembershipDebt;

if (selectedItem != null)

{

var paymentWindow = new PaymentWindow($"'{selectedItem.MembershipName}'", (int)selectedItem.Price);

if (paymentWindow.ShowDialog() == true && paymentWindow.PaymentSuccess)

{

// Обновляем только один оплаченный абонемент

UpdatePaidMemberships(new List<int> { selectedItem.MembershipId});

}

}

}

private void payForAllMembershipDebts\_Click(object sender, RoutedEventArgs e)

{

var allDebts = unPayedMembershipList.ItemsSource as List<MembershipDebt>;

if (allDebts == null || !allDebts.Any())

{

MessageBox.Show("Нет абонементов для оплаты");

return;

}

double totalSum = allDebts.Sum(d => d.Price);

var paymentWindow = new PaymentWindow("все абонементы", (int)totalSum);

if (paymentWindow.ShowDialog() == true && paymentWindow.PaymentSuccess)

{

UpdatePaidMemberships(allDebts.Select(d => d.MembershipId).ToList());

payForAllMembershipDebts.Visibility = Visibility.Collapsed;

}

}

private void payLockerBtn\_Click(object sender, RoutedEventArgs e)

{

var debt = lockerPaymentPanel.DataContext as LockerDebt;

if (debt != null)

{

var paymentWindow = new PaymentWindow($"аренду шкафчика #{debt.LockerId}", debt.RentPrice);

if (paymentWindow.ShowDialog() == true && paymentWindow.PaymentSuccess)

{

UpdateLockerPayment(debt.RentLockerId);

}

}

}

private void payAllClassesBtn\_Click(object sender, RoutedEventArgs e)

{

var allDebts = unPayedClassList.ItemsSource as List<ClassDebt>;

if (allDebts == null || !allDebts.Any())

{

MessageBox.Show("Нет занятий для оплаты");

return;

}

double totalSum = allDebts.Sum(d => d.Price);

var paymentWindow = new PaymentWindow("все занятия", (int)totalSum);

if (paymentWindow.ShowDialog() == true && paymentWindow.PaymentSuccess)

{

UpdateMultipleClassPayments(allDebts.Select(d => d.ClassId).ToList());

payAllClassesBtn.Visibility = Visibility.Collapsed;

}

}

private void payClass\_Click(object sender, RoutedEventArgs e)

{

var button = sender as Button;

var debt = button?.DataContext as ClassDebt;

if (debt != null)

{

var paymentWindow = new PaymentWindow($"занятие '{debt.ClassName}'", (int)debt.Price);

if (paymentWindow.ShowDialog() == true && paymentWindow.PaymentSuccess)

{

UpdateMultipleClassPayments(new List<int> { debt.ClassId });

}

}

}

}

public class ServiceDebt

{

public int ServiceId { get; set; }

public string ServiceName { get; set; }

public string ServiceType { get; set; }

public string Description { get; set; }

public double Price { get; set; }

public DateTime? ProvisionDate { get; set; }

}

public class MembershipDebt

{

public int MembershipId { get; set; }

public string MembershipName { get; set; }

public string MembershipTypeName { get; set; }

public DateTime StartDate { get; set; }

public DateTime EndDate { get; set; }

public double Price { get; set; }

}

public class LockerDebt

{

public int RentLockerId { get; set; }

public int LockerId { get; set; }

public string LockerZone { get; set; }

public DateTime StartDate { get; set; }

public DateTime EndDate { get; set; }

public int RentPrice { get; set; }

}

public class ClassDebt

{

public int ClassId { get; set; }

public string ClassName { get; set; }

public string Description { get; set; }

public string TrainerName { get; set; }

public DateTime Date { get; set; }

public TimeSpan StartTime { get; set; }

public double Price { get; set; }

}

}

Приложение И. Листинг кода ClientProfilePage.xaml.cs

using fitness\_club.Classes;

using fitness\_club.Model;

using fitness\_club.Windows;

using Microsoft.EntityFrameworkCore;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Text.RegularExpressions;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Navigation;

using System.Windows.Shapes;

namespace fitness\_club.Pages.ClientPages

{

/// <summary>

/// Логика взаимодействия для ClientProfilePage.xaml

/// </summary>

public partial class ClientProfilePage : Page

{

public ClientProfilePage()

{

InitializeComponent();

LoadClientData();

}

private void LoadClientData()

{

var user = AuthorizationWin.currentUser;

using (var db = new AppDbContext())

{

var fullUser = db.Users

.Include(u => u.Role)

.Include(u => u.Client)

.ThenInclude(c => c.Gender)

.FirstOrDefault(u => u.UserId == user.UserId);

if (fullUser == null || fullUser.Client == null)

{

MessageBox.Show("Пользователь или клиент не найден.");

return;

}

// Левая колонка

LastNameTb.Text = fullUser.Client.LastName;

FirstNameTb.Text = fullUser.Client.FirstName;

PatronymicTb.Text = fullUser.Client.Patronymic;

GenderTb.Text = fullUser.Client.Gender?.GenderName ?? "Не указано";

BirthDateTb.Text = fullUser.Client.BirthDate.ToShortDateString();

KemVidanTb.Text = fullUser.Client.PassportKemVidan;

KogdaVidanTb.Text = fullUser.Client.PassportKogdaVidan.ToShortDateString();

addAuthorNameCb.IsChecked = fullUser.Client.IsAddAuthorName;

// Правая колонка

PassportSeriesTb.Text = fullUser.Client.PassportSeries;

PassportNumberTb.Text = fullUser.Client.PassportNumber;

PhoneTb.Text = fullUser.Client.PhoneNumber.Trim();

EmailTb.Text = fullUser.Client.Email;

LoginTb.Text = fullUser.Login;

RoleTb.Text = fullUser.Role?.RoleName ?? "Не указана";

}

}

private void SaveChanges\_Click(object sender, RoutedEventArgs e)

{

using (var db = new AppDbContext())

{

var user = db.Users

.Include(u => u.Client)

.FirstOrDefault(u => u.UserId == AuthorizationWin.currentUser.UserId);

if (user == null || user.Client == null)

{

MessageBox.Show("Пользователь или клиент не найден.");

return;

}

// Телефон

string phone = PhoneTb.Text.Trim().Replace(" ", "").Replace("-", "").Replace("(", "").Replace(")", "");

if (!(Regex.IsMatch(phone, @"^\+7\d{10}$") || Regex.IsMatch(phone, @"^8\d{10}$")))

{

MessageBox.Show("Телефон должен быть в формате +7XXXXXXXXXX или 8XXXXXXXXXX");

return;

}

// Email

string email = EmailTb.Text.ToLower();

if (!Regex.IsMatch(email, @"^[a-z0-9]+@[a-z0-9]+\.[a-z]+$"))

{

MessageBox.Show("Введите корректный email.");

return;

}

// Логин

string login = LoginTb.Text.Trim();

if (login.Length < 4)

{

MessageBox.Show("Логин должен быть не короче 4 символов.");

return;

}

// Пароль

string newPassword = NewPasswordPb.Password.Trim();

string repeatPassword = RepeatPasswordPb.Password.Trim();

if (!string.IsNullOrEmpty(newPassword))

{

if (newPassword != repeatPassword)

{

MessageBox.Show("Пароли не совпадают.");

return;

}

var uf = new UserFromDb();

if (!uf.CheckPassword(newPassword)) return;

user.Password = Verification.GetSHA512Hash(newPassword);

}

// Обновляем

user.Login = login;

user.Client.PhoneNumber = phone;

user.Client.Email = email;

user.Client.IsAddAuthorName = addAuthorNameCb.IsChecked ?? false;

db.SaveChanges();

MessageBox.Show("Изменения успешно сохранены!");

}

}

}

}

Приложение К. Листинг кода ClientServicesPage.xaml.cs

using fitness\_club.Model;

using fitness\_club.Windows;

using fitness\_club.Classes;

using Microsoft.EntityFrameworkCore;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Media;

namespace fitness\_club.Pages.ClientPages

{

public partial class ClientServicesPage : Page

{

int clientMemberShipId = 0;

private List<FreeServiceViewModel> freeServicesGlobal;

int rentedLockerId = 0;

DateTime todayUtc = DateTime.SpecifyKind(DateTime.Today, DateTimeKind.Utc);

string? clientGender = "";

public ClientServicesPage()

{

InitializeComponent();

GridLoaded();

}

public void GridLoaded()

{

GetMemberShipClientId();

clientGender = CheckClientGender();

pickServiceProvisionDate.SelectedDate = todayUtc;

pickServiceDateForMembership.SelectedDate = todayUtc;

servicesHistoryComboBox.SelectedIndex = 0;

//Методы для услуг

LoadAvailableServicesByMemberShip();

LoadAvailableServices();

LoadClientsServices(servicesHistoryComboBox.SelectedIndex);

//Методы для шкафчиков

HasUserRentedLocker();

rentEndDp.SelectedDate = todayUtc.AddDays(1);

lockerZoneComboBox.ItemsSource = LoadLockerZone();

if (clientGender == "Мужской")

lockerZoneComboBox.SelectedItem = "Мужская раздевалка";

else if (clientGender == "Женский")

lockerZoneComboBox.SelectedItem = "Женская раздевалка";

else

lockerZoneComboBox.SelectedIndex = 0;

if (lockerZoneComboBox.SelectedItem == null)

return;

if (rentedLockerId == 0)

{

LoadLockers();

}

}

public void HasUserRentedLocker()

{

using (var db = new AppDbContext())

{

var rentedLocker = db.RentedLockers

.Where(rl => rl.ClientMembershipId == clientMemberShipId && rl.EndDate > todayUtc)

.FirstOrDefault();

if (rentedLocker != null)

{

rentedLockerId = rentedLocker.LockerId;

rentLockerSP.Visibility = Visibility.Collapsed;

rentedLockerIdTb.Text = $"Номер арендованного шкафчика: {rentedLockerId}";

userHasRentedLocker.Visibility = Visibility.Visible;

}

}

}

public string? CheckClientGender()

{

using (var db = new AppDbContext())

{

string? clientGender = db.Clients

.Where(c => c.ClientId == AuthorizationWin.currentUser.Client.ClientId)

.Select(c => c.Gender.GenderName)

.FirstOrDefault();

return clientGender;

}

}

public List<string> LoadLockerZone()

{

using (var db = new AppDbContext())

{

if (clientGender == "Мужчина")

{

return db.Lockers

.Select(l => l.LockerZone)

.Distinct()

.Where(l => l != "Женская раздевалка")

.ToList();

}

else if (clientGender == "Женщина")

{

return db.Lockers

.Select(l => l.LockerZone)

.Distinct()

.Where(l => l != "Мужская раздевалка")

.ToList();

}

else

{

return db.Lockers

.Select(l => l.LockerZone)

.Distinct()

.ToList();

}

}

}

public void LoadLockers()

{

string selectedZone = lockerZoneComboBox.SelectedItem.ToString();

using (var db = new AppDbContext())

{

if (rentedLockerId != 0)

{

rentEndDp.SelectedDate = db.RentedLockers

.Where(rl => rl.LockerId == rentedLockerId)

.Select(rl => rl.EndDate)

.FirstOrDefault();

}

var rentedLockerIds = db.RentedLockers

.Where(rl => rl.EndDate >= todayUtc)

.Select(rl => rl.LockerId)

.ToHashSet();

var availableLockers = db.Lockers

.Where(l => l.LockerZone == selectedZone && !rentedLockerIds.Contains(l.LockerId))

.ToList();

availableLockersList.ItemsSource = availableLockers;

}

}

public int CalculateLockerPrice(dynamic selectedLocker)

{

var endDate = rentEndDp.SelectedDate.HasValue

? DateTime.SpecifyKind(rentEndDp.SelectedDate.Value, DateTimeKind.Utc)

: throw new Exception("Дата не выбрана");

if (endDate <= todayUtc)

MessageBox.Show("Дата окончания должна быть позже даты начала");

int days = (endDate - todayUtc).Days;

float dailyPrice = selectedLocker.MonthPrice / 30;

float total = dailyPrice \* days;

return (int)Math.Floor(total);

}

public void RentLocker(dynamic selectedLocker)

{

using (var db = new AppDbContext())

{

if (rentedLockerId != 0)

{

db.RentedLockers

.Remove(db.RentedLockers

.Where(rl => rl.ClientMembershipId == clientMemberShipId && rl.EndDate > todayUtc)

.FirstOrDefault());

}

var rentEndDate = rentEndDp.SelectedDate.HasValue

? DateTime.SpecifyKind(rentEndDp.SelectedDate.Value, DateTimeKind.Utc)

: throw new Exception("Дата не выбрана");

if (rentEndDate > db.ClientMemberships

.Where(cm => cm.ClientMembershipId == clientMemberShipId)

.Select(cm => cm.EndDate)

.FirstOrDefault())

{

MessageBox.Show("Дата окончания аренды не может быть позже даты окончания абонемента.");

}

else

{

var rentedLocker = new RentedLocker

{

ClientMembershipId = clientMemberShipId,

LockerId = selectedLocker.LockerId,

StartDate = todayUtc,

EndDate = rentEndDate,

RentPrice = CalculateLockerPrice(selectedLocker),

PaymentDate = null

};

db.RentedLockers.Add(rentedLocker);

db.SaveChanges();

if (MessageBox.Show("Шкафчик успешно арендован.", "Успешно", MessageBoxButton.OK, MessageBoxImage.Information) == MessageBoxResult.OK)

{

HasUserRentedLocker();

}

}

}

}

public void LoadAvailableServicesByMemberShip()

{

using (var db = new AppDbContext())

{

freeServicesGlobal = db.Services

.Include(s => s.ServiceType)

.Where(s => s.FreeUsageLimit > 0)

.Select(s => new FreeServiceViewModel

{

ServiceId = s.ServiceId,

ServiceType = s.ServiceType.TypeName,

ServiceName = s.ServiceName,

Description = s.Description,

UsageCount = db.MembershipServices.Count(ms => ms.ServiceId == s.ServiceId && ms.ClientMembershipId == clientMemberShipId) > s.FreeUsageLimit

? s.FreeUsageLimit

: db.MembershipServices.Count(ms => ms.ServiceId == s.ServiceId && ms.ClientMembershipId == clientMemberShipId),

RemainingCount = s.FreeUsageLimit - db.MembershipServices.Count(ms => ms.ServiceId == s.ServiceId && ms.ClientMembershipId == clientMemberShipId) < 0 ? 0

: s.FreeUsageLimit - db.MembershipServices.Count(ms => ms.ServiceId == s.ServiceId && ms.ClientMembershipId == clientMemberShipId)

})

.ToList();

servicesAvailableByMembership.ItemsSource = freeServicesGlobal;

}

}

public void GetMemberShipClientId()

{

using (var db = new AppDbContext())

{

int clientmsId = db.ClientMemberships

.Where(cm => cm.ClientId == AuthorizationWin.currentUser.Client.ClientId &&

cm.EndDate >= todayUtc && cm.StartDate <= todayUtc)

.Select(cm => cm.ClientMembershipId)

.FirstOrDefault();

clientMemberShipId = clientmsId;

}

}

public void LoadClientsServices(int selectionIndex)

{

using (var db = new AppDbContext())

{

IQueryable<MembershipService> baseQuery = db.MembershipServices

.Include(ms => ms.Service)

.Include(ms => ms.ClientMembership)

.Where(ms => ms.ClientMembershipId == clientMemberShipId);

switch (selectionIndex)

{

case 0: // Все услуги

servicesHistoryList.ItemsSource = baseQuery

.OrderByDescending(ms => ms.ProvisionDate)

.Select(ms => new

{

ms.ServiceId,

ms.Service.ServiceName,

ms.Service.Description,

ms.Service.Price,

ms.ProvisionDate,

})

.ToList();

break;

case 1: // Использованные услуги (до текущей даты)

servicesHistoryList.ItemsSource = baseQuery

.Where(ms => ms.ProvisionDate < todayUtc)

.OrderByDescending(ms => ms.ProvisionDate)

.Select(ms => new

{

ms.ServiceId,

ms.Service.ServiceName,

ms.Service.Description,

ms.Service.Price,

ms.ProvisionDate,

})

.ToList();

break;

case 2: // Неиспользованные услуги (текущая дата и позже)

servicesHistoryList.ItemsSource = baseQuery

.Where(ms => ms.ProvisionDate >= todayUtc)

.OrderBy(ms => ms.ProvisionDate)

.Select(ms => new

{

ms.ServiceId,

ms.Service.ServiceName,

ms.Service.Description,

ms.Service.Price,

ms.ProvisionDate,

})

.ToList();

break;

}

}

}

public void LoadAvailableServices()

{

using (var db = new AppDbContext())

{

var paidOnly = db.Services

.Include(s => s.ServiceType)

.Where(s => s.FreeUsageLimit == 0)

.Select(s => new

{

s.ServiceName,

s.Description,

s.Price,

ServiceType = s.ServiceType.TypeName

})

.ToList();

var exhaustedFreeServices = freeServicesGlobal

.Where(f => f.RemainingCount <= 0)

.Select(f => new

{

f.ServiceName,

f.Description,

Price = db.Services.Where(s => s.ServiceId == f.ServiceId).Select(s => s.Price).FirstOrDefault(),

f.ServiceType

})

.ToList();

var combined = paidOnly

.Concat(exhaustedFreeServices)

.OrderBy(s => s.ServiceType)

.ThenBy(s => s.ServiceName)

.ToList();

availableServicesList.ItemsSource = combined;

}

}

public void OrderService(dynamic selectedItem, DateTime provisionDate, bool isServiceFree)

{

try

{

using (var db = new AppDbContext())

{

if (selectedItem != null)

{

string serviceName = selectedItem.ServiceName;

var service = db.Services

.Where(s => s.ServiceName == serviceName)

.FirstOrDefault();

db.MembershipServices.Add(new MembershipService

{

ServiceId = service.ServiceId,

ClientMembershipId = clientMemberShipId,

ProvisionDate = provisionDate,

});

db.SaveChanges();

if (!isServiceFree)

{

db.ServicesPayments.Add(new ServicesPayments

{

ServiceId = service.ServiceId,

PaymentDate = null,

Price = (double)service.Price,

ClientId = AuthorizationWin.currentUser.Client.ClientId

});

db.SaveChanges();

}

else

{

db.ServicesPayments.Add(new ServicesPayments

{

ServiceId = service.ServiceId,

PaymentDate = provisionDate,

Price = 0,

ClientId = AuthorizationWin.currentUser.Client.ClientId

});

db.SaveChanges();

}

MessageBox.Show("Вы успешно оформили услугу");

LoadClientsServices(servicesHistoryComboBox.SelectedIndex);

LoadAvailableServices();

LoadAvailableServicesByMemberShip();

}

}

}

catch (Exception ex)

{

var inner = ex.InnerException?.Message ?? "Нет дополнительной информации.";

MessageBox.Show("Ошибка при оформлении услуги:\n" + ex.Message + "\n\n" + inner);

}

}

private void orderService\_Click(object sender, RoutedEventArgs e)

{

dynamic selectedItem = availableServicesList.SelectedItem;

if (pickServiceProvisionDate.SelectedDate == null || pickServiceProvisionDate.SelectedDate < todayUtc)

{

MessageBox.Show("Выберите дату предоставления услуги(возможно вы указываете дату в прошлом).");

return;

}

else

{

var provisionDate = pickServiceProvisionDate.SelectedDate.HasValue

? DateTime.SpecifyKind(pickServiceProvisionDate.SelectedDate.Value, DateTimeKind.Utc)

: throw new Exception("Дата не выбрана");

if (selectedItem == null)

{

MessageBox.Show("Выберите услугу.");

return;

}

else

{

OrderService(selectedItem, provisionDate, false);

}

}

}

private void servicesHistoryComboBox\_SelectionChanged(object sender, SelectionChangedEventArgs e)

{

LoadClientsServices(servicesHistoryComboBox.SelectedIndex);

}

private void useServiceButton\_Click(object sender, RoutedEventArgs e)

{

var button = sender as Button;

var selectedItem = button?.DataContext as FreeServiceViewModel;

if (pickServiceDateForMembership.SelectedDate == null || pickServiceDateForMembership.SelectedDate < todayUtc)

{

MessageBox.Show("Выберите дату предоставления услуги(возможно вы указываете дату в прошлом).");

return;

}

else

{

var provisionDate = pickServiceDateForMembership.SelectedDate.HasValue

? DateTime.SpecifyKind(pickServiceDateForMembership.SelectedDate.Value, DateTimeKind.Utc)

: throw new Exception("Дата не выбрана");

if (selectedItem == null)

{

MessageBox.Show("Выберите услугу.");

return;

}

else

{

OrderService(selectedItem, provisionDate, true);

}

}

}

private void lockerZoneComboBox\_SelectionChanged(object sender, SelectionChangedEventArgs e)

{

LoadLockers();

}

private void rentLockerBtn\_Click(object sender, RoutedEventArgs e)

{

dynamic selectedLocker = availableLockersList.SelectedItem as Locker;

if (selectedLocker != null)

{

if (rentEndDp.SelectedDate == null || rentEndDp.SelectedDate < todayUtc)

{

MessageBox.Show("Выберите дату окончания аренды(возможно вы указываете дату в прошлом).");

return;

}

else

{

RentLocker(selectedLocker);

}

}

else

{

MessageBox.Show("Выберите шкафчик для аренды.");

}

}

private void rentEndDp\_SelectionChanged(object sender, SelectionChangedEventArgs e)

{

dynamic selectedLocker = availableLockersList.SelectedItem as Locker;

if (selectedLocker != null)

lockerPriceForPerion.Text = $"Цена аренды: {CalculateLockerPrice(selectedLocker)}";

}

private void availableLockersList\_SelectionChanged(object sender, SelectionChangedEventArgs e)

{

dynamic selectedLocker = availableLockersList.SelectedItem as Locker;

if (selectedLocker != null)

lockerPriceForPerion.Text = $"Цена аренды: {CalculateLockerPrice(selectedLocker)}";

}

private void rentLockerUntilMembershipEnds\_Click(object sender, RoutedEventArgs e)

{

using (var db = new AppDbContext())

{

rentEndDp.SelectedDate = db.ClientMemberships

.Where(cm => cm.ClientMembershipId == clientMemberShipId)

.Select(cm => cm.EndDate)

.FirstOrDefault();

}

}

private void changeRentedLockerBtn\_Click(object sender, RoutedEventArgs e)

{

userHasRentedLocker.Visibility = Visibility.Collapsed;

rentLockerSP.Visibility = Visibility.Visible;

lockerPriceForPerion.Visibility = Visibility.Collapsed;

rentEndDp.IsEnabled = false;

rentLockerBtn.Content = "Изменить ящик";

rentLockerUntilMembershipEnds.IsEnabled = false;

LoadLockers();

}

private void CancelServiceClassContext\_Click(object sender, RoutedEventArgs e)

{

if (sender is MenuItem menuItem && menuItem.CommandParameter is int serviceId)

{

try

{

using (var db = new AppDbContext())

{

var clientId = AuthorizationWin.currentUser.Client.ClientId;

var membershipService = db.MembershipServices

.FirstOrDefault(ms => ms.ServiceId == serviceId && ms.ClientMembershipId == clientMemberShipId);

if (membershipService == null)

{

MessageBox.Show("Услуга не найдена.");

return;

}

if (membershipService.ProvisionDate <= todayUtc)

{

MessageBox.Show("Нельзя отменить услугу, которая уже использована.");

return;

}

var payment = db.ServicesPayments

.Include(sp => sp.Service)

.FirstOrDefault(sp => sp.ServiceId == serviceId && sp.ClientId == clientId);

if (payment != null)

{

if (payment.PaymentDate != null)

{

// Возврат на баланс

var client = db.Clients.FirstOrDefault(c => c.ClientId == clientId);

if (client != null)

{

client.Balance += (decimal)payment.Price;

db.ClientTransactions.Add(new ClientTransaction

{

ClientId = client.ClientId,

OperationDescription = $"Возврат за отмену услуги: {payment.Service.ServiceName}",

PaymentWay = "На баланс",

Amount = (decimal)payment.Price,

TransactionType = "возврат",

TransactionDate = todayUtc

});

}

}

db.ServicesPayments.Remove(payment);

}

db.MembershipServices.Remove(membershipService);

db.SaveChanges();

LoadClientsServices(servicesHistoryComboBox.SelectedIndex);

LoadAvailableServices();

LoadAvailableServicesByMemberShip();

MessageBox.Show("Услуга успешно отменена.", "Успех",

MessageBoxButton.OK, MessageBoxImage.Information);

}

}

catch (Exception ex)

{

MessageBox.Show($"Ошибка при отмене услуги: {ex.Message}", "Ошибка",

MessageBoxButton.OK, MessageBoxImage.Error);

}

}

else

{

MessageBox.Show("Пожалуйста, выберите услугу для отмены.", "Информация",

MessageBoxButton.OK, MessageBoxImage.Information);

}

}

}

}

public class FreeServiceViewModel

{

public int ServiceId { get; set; }

public string ServiceType { get; set; }

public string ServiceName { get; set; }

public string Description { get; set; }

public int UsageCount { get; set; }

public int RemainingCount { get; set; }

}

Приложение Л. Листинг кода ClientTrainerListPage.xaml.cs

using System;

using fitness\_club.Model;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Navigation;

using System.Windows.Shapes;

using Microsoft.EntityFrameworkCore;

using System.Collections;

using fitness\_club.Classes;

namespace fitness\_club.Pages.ClientPages

{

/// <summary>

/// Логика взаимодействия для ClientTrainerListPage.xaml

/// </summary>

public partial class ClientTrainerListPage : Page

{

string zaglushka = System.IO.Path.GetFullPath("../../../images/zagluska.png");

public ClientTrainerListPage()

{

InitializeComponent();

LoadTrainers();

}

public void LoadTrainers()

{

using (var db = new AppDbContext())

{

var trainersRaw = db.Trainers

.Include(t => t.TrainerReviews)

.Include(t => t.Specialization)

.ToList();

var trainers = trainersRaw.Select(t => new

{

t.TrainerId,

FullName = $"{t.FirstName} {t.LastName} {t.Patronymic}",

TrainerAge = $"{DateTime.Now.Year - t.BirthDate.Year}",

t.PhoneNumber,

Experience = CalculateExperience(t.DateOfEmployment),

AverageRating = CalculateAvgRating(t.TrainerId) != 0 ? $"{CalculateAvgRating(t.TrainerId)}" : "Нет оценок",

Photo = !string.IsNullOrWhiteSpace(t.Photo)

? System.IO.Path.GetFullPath(t.Photo)

: zaglushka,

t.IndividualPrice,

Specialization = t.Specialization.SpecializationName

}).ToList();

TrainersItemsControl.ItemsSource = trainers;

}

}

public double CalculateAvgRating(int trainerId)

{

using (var db = new AppDbContext())

{

var trainer = db.Trainers

.Include(t => t.TrainerReviews)

.FirstOrDefault(t => t.TrainerId == trainerId);

if (trainer != null && trainer.TrainerReviews.Count > 0)

{

double avgRating = trainer.TrainerReviews.Average(r => r.ReviewGrade);

return avgRating;

}

else

{

return 0;

}

}

}

private string CalculateExperience(DateTime startDate)

{

DateTime today = DateTime.Now;

int years = today.Year - startDate.Year;

int months = today.Month - startDate.Month;

int days = today.Day - startDate.Day;

if (days < 0)

{

months--;

days += DateTime.DaysInMonth(today.Year, today.Month);

}

if (months < 0)

{

years--;

months += 12;

}

return $"{years} лет {months} месяцев {days} дней";

}

private void TrainerCard\_Click(object sender, MouseButtonEventArgs e)

{

if (sender is Border border)

{

dynamic selectedItem = border.DataContext;

ClientTrainerDetailsWindow trainerDetailsWindow = new ClientTrainerDetailsWindow(selectedItem.TrainerId);

trainerDetailsWindow.Show();

}

}

}

}

Приложение М. Листинг кода ClientTrainingPlanFromTrainer.xaml.cs

using fitness\_club.Model;

using fitness\_club.Windows;

using Microsoft.EntityFrameworkCore;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Navigation;

using System.Windows.Shapes;

namespace fitness\_club.Pages.ClientPages

{

public partial class ClientTrainingPlanFromTrainer : Page

{

public ClientTrainingPlanFromTrainer()

{

InitializeComponent();

GetTrainingPlan();

}

public void GetTrainingPlan()

{

using(var db = new AppDbContext())

{

var trainingPlan = db.TrainingPlans

.Include(tp=>tp.Trainer)

.Where(tp => tp.ClientId == AuthorizationWin.currentUser.Client.ClientId)

.FirstOrDefault();

if(trainingPlan != null)

{

trainingPlanTbox.Text = trainingPlan.Plan;

trainerNameTb.Text = $"Ваш тренеровочный план от тренера {trainingPlan.Trainer.FirstName} {trainingPlan.Trainer.LastName}";

}

else

{

trainingPlanTbox.Text = "У вас пока нет тренировочного плана от тренера.";

}

}

}

}

}

Приложение Н. Листинг кода ClientTransactionsPage.xaml.cs

using fitness\_club.Model;

using fitness\_club.Windows;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Shapes;

namespace fitness\_club.Pages.ClientPages

{

/// <summary>

/// Логика взаимодействия для ClientTransactions.xaml

/// </summary>

public partial class ClientTransactionsPage : Page

{

public ClientTransactionsPage()

{

InitializeComponent();

LoadTransactions();

}

private void LoadTransactions()

{

try

{

using var db = new AppDbContext();

int clientId = AuthorizationWin.currentUser.Client.ClientId;

var transactions = db.ClientTransactions

.Where(t => t.ClientId == clientId)

.OrderByDescending(t => t.TransactionDate)

.Select(t => new

{

t.OperationDescription,

t.PaymentWay,

t.Amount,

t.TransactionType,

t.TransactionDate

})

.ToList();

TransactionsListView.ItemsSource = transactions;

}

catch (Exception ex)

{

MessageBox.Show($"Ошибка при загрузке транзакций: {ex.Message}", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);

}

}

}

}

Приложение П. Листинг кода PaymentWindow.xaml.cs

using fitness\_club.Model;

using fitness\_club.Classes;

using QRCoder;

using System;

using System.Collections.Generic;

using System.Drawing;

using System.Linq;

using System.Security.Policy;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Controls.Primitives;

using System.Windows.Data;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Shapes;

using System.Windows.Threading;

using Brushes = System.Windows.Media.Brushes;

using Color = System.Windows.Media.Color;

using fitness\_club.Windows;

namespace fitness\_club.Pages.ClientPages

{

public partial class PaymentWindow : Window

{

public bool PaymentSuccess { get; private set; } = false;

string paymentType;

int paymentSum;

decimal clientBalance = 0;

DateTime todayUTC = DateTime.SpecifyKind(DateTime.Today, DateTimeKind.Utc);

public PaymentWindow(string paymentType, int paymentSum)

{

InitializeComponent();

GetClientsBalance();

proceedAmoutTbox.Text = 0.ToString();

this.paymentType = paymentType;

this.paymentSum = paymentSum;

}

public void GetClientsBalance()

{

using(var db = new AppDbContext())

{

clientBalance = db.Clients.Where(c => c.ClientId == AuthorizationWin.currentUser.Client.ClientId).Select(c => c.Balance).FirstOrDefault();

}

clientsBalanceTb.Text = $"Ваш баланс: {clientBalance}";

}

private string GenerateRandomString()

{

const string chars = "ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789";

var random = new Random();

return new string(Enumerable.Repeat(chars, 16)

.Select(s => s[random.Next(s.Length)])

.ToArray());

}

private void GenerateRandomQrCode()

{

string randomData = GenerateRandomString();

using (var qrGenerator = new QRCodeGenerator())

using (var qrCodeData = qrGenerator.CreateQrCode(randomData, QRCodeGenerator.ECCLevel.Q))

using (var qrCode = new QRCode(qrCodeData))

{

var qrCodeBitmap = qrCode.GetGraphic(20);

QrCodeImage.Source = ConvertToBitmapImage(qrCodeBitmap);

}

}

private BitmapImage ConvertToBitmapImage(Bitmap bitmap)

{

using (var memory = new System.IO.MemoryStream())

{

bitmap.Save(memory, System.Drawing.Imaging.ImageFormat.Png);

memory.Position = 0;

var bitmapImage = new BitmapImage();

bitmapImage.BeginInit();

bitmapImage.StreamSource = memory;

bitmapImage.CacheOption = BitmapCacheOption.OnLoad;

bitmapImage.EndInit();

return bitmapImage;

}

}

public static bool IsValidCardNumber(string cardNumber)

{

string cleanCardNumber = new string(cardNumber.Where(char.IsDigit).ToArray());

if (cleanCardNumber.Length < 13 || cleanCardNumber.Length > 19)

return false;

int sum = 0;

bool alternate = false;

for (int i = cleanCardNumber.Length - 1; i >= 0; i--)

{

int digit = cleanCardNumber[i] - '0';

if (alternate)

{

digit \*= 2;

if (digit > 9)

digit = (digit % 10) + 1;

}

sum += digit;

alternate = !alternate;

}

return sum % 10 == 0;

}

private void GenerateQr\_Click(object sender, RoutedEventArgs e)

{

GenerateRandomQrCode();

}

private void payBtn\_Click(object sender, RoutedEventArgs e)

{

if (cardRb.IsChecked == true)

{

if (string.IsNullOrWhiteSpace(cardNumberTb.Text) || string.IsNullOrWhiteSpace(cvvTb.Text) ||

string.IsNullOrWhiteSpace(expiryDateMonthTb.Text) || string.IsNullOrWhiteSpace(expiryDateYearTb.Text))

{

MessageBox.Show("Пожалуйста, заполните все поля", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);

}

else

{

if (IsValidCardNumber(cardNumberTb.Text))

{

if (CheckExpirationDate())

{

if (cvvTb.Text.Length == 3 && int.TryParse(cvvTb.Text, out int cvv) && cvv >= 1 && cvv <= 999)

{

using (var db = new AppDbContext())

{

var clientId = AuthorizationWin.currentUser.Client.ClientId;

db.ClientTransactions.Add(new ClientTransaction

{

ClientId = clientId,

OperationDescription = $"Оплата: {paymentType}",

PaymentWay = "Карта",

Amount = -paymentSum,

TransactionType = "списание",

TransactionDate = todayUTC

});

db.SaveChanges();

}

MessageBoxResult result = MessageBox.Show(

$"Вы успешно оплатили {paymentType}!",

"Успех",

MessageBoxButton.OK,

MessageBoxImage.Information);

if (result == MessageBoxResult.OK)

{

PaymentSuccess = true;

this.DialogResult = true;

this.Close();

}

}

else

{

MessageBox.Show("Некорректный CVV код. Пожалуйста, проверьте введенные данные.", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);

}

}

}

else

{

MessageBox.Show("Пожалуйста, проверьте номер карты!", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);

}

}

}

else if (qrCodeRb.IsChecked == true)

{

using (var db = new AppDbContext())

{

var clientId = AuthorizationWin.currentUser.Client.ClientId;

db.ClientTransactions.Add(new ClientTransaction

{

ClientId = clientId,

OperationDescription = $"Оплата: {paymentType}",

PaymentWay = "QR-код",

Amount = -paymentSum,

TransactionType = "списание",

TransactionDate = todayUTC

});

db.SaveChanges();

}

MessageBoxResult result = MessageBox.Show(

$"Вы успешно оплатили {paymentType}!",

"Успех",

MessageBoxButton.OK,

MessageBoxImage.Information);

if (result == MessageBoxResult.OK)

{

PaymentSuccess = true;

this.DialogResult = true;

this.Close();

}

}

else

{

MessageBox.Show("Пожалуйста, выберите способ оплаты", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);

}

}

public bool CheckExpirationDate()

{

DateTime today = DateTime.Today;

int currentYear = today.Year;

int currentMonth = today.Month;

if (string.IsNullOrWhiteSpace(expiryDateMonthTb.Text) ||

string.IsNullOrWhiteSpace(expiryDateYearTb.Text))

{

MessageBox.Show("Пожалуйста, заполните все поля даты", "Ошибка",

MessageBoxButton.OK, MessageBoxImage.Error);

return false;

}

if (!int.TryParse(expiryDateMonthTb.Text, out int expirationMonth) ||

!int.TryParse(expiryDateYearTb.Text, out int expirationYearShort))

{

MessageBox.Show("Некорректный формат даты. Используйте только цифры", "Ошибка",

MessageBoxButton.OK, MessageBoxImage.Error);

return false;

}

int expirationYear = 2000 + expirationYearShort;

if (expirationMonth < 1 || expirationMonth > 12)

{

MessageBox.Show("Месяц должен быть от 01 до 12", "Ошибка",

MessageBoxButton.OK, MessageBoxImage.Error);

return false;

}

if (expirationYearShort < 0 || expirationYearShort > 99)

{

MessageBox.Show("Год должен быть от 00 до 99", "Ошибка",

MessageBoxButton.OK, MessageBoxImage.Error);

return false;

}

if (expirationYear > currentYear)

{

return true;

}

else if (expirationYear == currentYear && expirationMonth >= currentMonth)

{

return true;

}

MessageBox.Show("Срок действия карты истек", "Ошибка",

MessageBoxButton.OK, MessageBoxImage.Error);

return false;

}

private void payMethods\_Checked(object sender, RoutedEventArgs e)

{

RadioButton selectedRadio = (RadioButton)sender;

switch (selectedRadio.Name)

{

case "cardRb":

payByBalanceSp.Visibility = Visibility.Collapsed;

payByQRCodeSp.Visibility = Visibility.Collapsed;

payByCardSp.Visibility = Visibility.Visible;

payBtn.Visibility = Visibility.Visible;

payBtn.Content = $"Оплатить {paymentSum} руб.";

break;

case "qrCodeRb":

payByBalanceSp.Visibility = Visibility.Collapsed;

payByCardSp.Visibility = Visibility.Collapsed;

GenerateRandomQrCode();

payByQRCodeSp.Visibility = Visibility.Visible;

payBtn.Visibility = Visibility.Visible;

payBtn.Content = $"Я оплатил {paymentSum} руб.";

break;

case "balanceRb":

payByBalanceSp.Visibility = Visibility.Visible;

payByCardSp.Visibility = Visibility.Collapsed;

payByQRCodeSp.Visibility = Visibility.Collapsed;

payBtn.Visibility = Visibility.Collapsed;

payByBalanceBtn.Content = $"Оплатить {paymentSum} руб.";

break;

}

}

private void payByBalanceBtn\_Click(object sender, RoutedEventArgs e)

{

if (!int.TryParse(proceedAmoutTbox.Text, out int proceedAmount) || proceedAmount <= 0)

{

MessageBox.Show("Некорректный ввод. Введите положительное число.");

return;

}

using var db = new AppDbContext();

var client = db.Clients

.FirstOrDefault(c => c.ClientId == AuthorizationWin.currentUser.Client.ClientId);

if (client == null)

{

MessageBox.Show("Клиент не найден.");

return;

}

if (proceedAmount > client.Balance)

{

MessageBox.Show("У вас недостаточно средств на балансе.");

return;

}

if (paymentSum > proceedAmount) // Частичная оплата

{

paymentSum -= proceedAmount;

client.Balance -= proceedAmount;

db.ClientTransactions.Add(new ClientTransaction

{

ClientId = client.ClientId,

OperationDescription = $"Частичная оплата: {paymentType}",

PaymentWay = "С баланса",

Amount = -proceedAmount,

TransactionType = "списание",

TransactionDate = todayUTC

});

db.SaveChanges();

MessageBox.Show($"Списано {proceedAmount} руб. Остаток: {client.Balance} руб. " +

"Для оплаты оставшейся части заказа выберите другой способ оплаты!");

payByBalanceBtn.Content = $"Оплатить {paymentSum} руб.";

}

else // Полная оплата

{

client.Balance -= paymentSum;

db.ClientTransactions.Add(new ClientTransaction

{

ClientId = client.ClientId,

OperationDescription = $"Полная оплата: {paymentType}",

PaymentWay = "С баланса",

Amount = -paymentSum,

TransactionType = "списание",

TransactionDate = todayUTC

});

db.SaveChanges();

MessageBox.Show($"Списано {paymentSum} руб. Остаток: {client.Balance} руб.");

MessageBoxResult result = MessageBox.Show(

$"Вы успешно оплатили {paymentType}! С баланса списано {paymentSum} руб.\nОстаток: {client.Balance} руб.",

"Успешная оплата",

MessageBoxButton.OK,

MessageBoxImage.Information);

if (result == MessageBoxResult.OK)

{

PaymentSuccess = true;

this.DialogResult = true;

this.Close();

}

}

}

private void fullAmountOfPrice\_Click(object sender, RoutedEventArgs e)

{

proceedAmoutTbox.Text = paymentSum.ToString();

}

private void proceedAmoutTbox\_TextChanged(object sender, TextChangedEventArgs e)

{

payByBalanceBtn.Content = $"Оплатить {proceedAmoutTbox.Text} руб.";

}

}

}

Приложение Р. Листинг кода ClassVisitorsListPage.xaml.cs

using fitness\_club.Model;

using fitness\_club.Classes;

using Microsoft.EntityFrameworkCore;

using System.Collections.Generic;

using System.Linq;

using System.Windows;

using System.Windows.Controls;

namespace fitness\_club.Pages.TrainerPages

{

public partial class ClassVisitorsListPage : Page

{

private readonly int classId;

// простой список для сохранения

private List<VisitRow> visitors;

public ClassVisitorsListPage(int classId)

{

InitializeComponent();

this.classId = classId;

LoadVisitors();

}

private void LoadVisitors()

{

using var db = new AppDbContext();

classNameTb.Text = $"Участники занятия {db.Class

.Where(c => c.ClassId == classId)

.Select(c => c.ClassInfo.ClassName)

.FirstOrDefault()}";

visitors = db.ClassVisits

.Include(cv => cv.ClientMembership)

.ThenInclude(cm => cm.Client)

.Where(cv => cv.ClassId == classId)

.Select(cv => new VisitRow

{

ClientMembershipId = cv.ClientMembershipId,

FullName = cv.ClientMembership.Client.LastName + " " + cv.ClientMembership.Client.FirstName,

Visited = cv.Visited

})

.ToList();

VisitorsDataGrid.ItemsSource = visitors;

}

private void SaveButton\_Click(object sender, RoutedEventArgs e)

{

using var db = new AppDbContext();

foreach (var v in visitors)

{

var visit = db.ClassVisits

.FirstOrDefault(x => x.ClassId == classId && x.ClientMembershipId == v.ClientMembershipId);

if (visit != null)

{

visit.Visited = v.Visited;

}

}

// Обновляем trainerChecked

var training = db.Class.FirstOrDefault(c => c.ClassId == classId);

if (training != null)

{

training.TrainerChecked = true;

}

db.SaveChanges();

MessageBox.Show("Статусы посещения сохранены.");

}

private void BackButton\_Click(object sender, RoutedEventArgs e)

{

this.NavigationService.GoBack();

}

}

public class VisitRow

{

public int ClientMembershipId { get; set; }

public string FullName { get; set; }

public bool Visited { get; set; }

}

}

Приложение С. Листинг кода TrainerClassesPage.xaml.cs

using fitness\_club.Model;

using fitness\_club.Windows;

using Microsoft.EntityFrameworkCore;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Controls.Primitives;

using System.Windows.Data;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Navigation;

using System.Windows.Shapes;

namespace fitness\_club.Pages.TrainerPages

{

/// <summary>

/// Логика взаимодействия для TrainerClassesPage.xaml

/// </summary>

public partial class TrainerClassesPage : Page

{

private readonly int trainerId;

DateTime todayUtc = DateTime.SpecifyKind(DateTime.Today, DateTimeKind.Utc);

public TrainerClassesPage()

{

InitializeComponent();

trainerId = AuthorizationWin.currentUser.Trainer.TrainerId;

LoadClasses();

}

private void LoadClasses()

{

var todayUtc = DateTime.UtcNow.Date;

using var db = new AppDbContext();

var classes = db.Class

.Include(c => c.ClassInfo)

.Include(c => c.Hall)

.Include(c => c.ClassType)

.Include(c => c.WorkSchedule)

.ThenInclude(ws => ws.Trainer)

.Include(c => c.ClassVisits)

.ThenInclude(cv => cv.ClientMembership.Client)

.Where(c => c.WorkSchedule.TrainerId == trainerId &&

c.WorkSchedule.WorkDate > todayUtc)

.ToList();

var individual = classes

.Where(c => c.ClassTypeId == 2)

.Select(c => new

{

c.ClassId,

ClassName = c.ClassInfo.ClassName,

Date = c.WorkSchedule.WorkDate,

StartTime = c.StartTime,

EndTime = c.EndTime,

HallName = c.Hall.HallName,

Price = c.Price ?? 0,

ClientName = c.ClassVisits.FirstOrDefault()?.ClientMembership?.Client != null

? $"{c.ClassVisits.First().ClientMembership.Client.LastName} {c.ClassVisits.First().ClientMembership.Client.FirstName}"

: "Нет клиента"

})

.ToList();

var group = classes

.Where(c => c.ClassTypeId != 2)

.Select(c => new

{

c.ClassId,

ClassName = c.ClassInfo.ClassName,

Date = c.WorkSchedule.WorkDate,

StartTime = c.StartTime,

EndTime = c.EndTime,

HallName = c.Hall.HallName,

Price = c.Price ?? 0,

VisitorsCount = c.ClassVisits.Count

})

.ToList();

IndividualSessionsList.ItemsSource = individual;

GroupSessionsList.ItemsSource = group;

}

}

}

Приложение Т. Листинг кода TrainerClientsPage.xaml.cs

using fitness\_club.Classes;

using fitness\_club.Model;

using fitness\_club.Windows;

using Microsoft.EntityFrameworkCore;

using System.Linq;

using System.Windows;

using System.Windows.Controls;

namespace fitness\_club.Pages.TrainerPages

{

public partial class TrainerClientsPage : Page

{

private int trainerId = AuthorizationWin.currentUser.Trainer.TrainerId;

public TrainerClientsPage()

{

InitializeComponent();

LoadClients();

}

private void LoadClients()

{

using var db = new AppDbContext();

var clients = db.Class

.Include(c => c.ClassVisits)

.ThenInclude(cv => cv.ClientMembership)

.ThenInclude(cm => cm.Client)

.Include(c => c.WorkSchedule)

.Where(c => c.WorkSchedule.TrainerId == trainerId && c.ClassTypeId == 2)

.SelectMany(c => c.ClassVisits)

.Select(cv => new

{

ClientMembershipId = cv.ClientMembershipId,

ClientId = cv.ClientMembership.Client.ClientId,

FullName = cv.ClientMembership.Client.LastName + " " + cv.ClientMembership.Client.FirstName

})

.Distinct()

.ToList();

ClientsListView.ItemsSource = clients;

}

private void ClientsListView\_MouseDoubleClick(object sender, System.Windows.Input.MouseButtonEventArgs e)

{

dynamic selected = ClientsListView.SelectedItem;

if (selected != null)

{

int clientId = selected.ClientId;

var window = new WritePlanWindow(trainerId, clientId);

window.ShowDialog();

}

}

}

}

Приложение У. Листинг кода TrainerPastClasses.xaml.cs

using fitness\_club.Model;

using fitness\_club.Windows;

using Microsoft.EntityFrameworkCore;

using System;

using System.Linq;

using System.Windows;

using System.Windows.Controls;

namespace fitness\_club.Pages.TrainerPages

{

public partial class TrainerPastClassesPage : Page

{

private int trainerId = AuthorizationWin.currentUser.Trainer.TrainerId;

private DateTime today = DateTime.UtcNow.Date;

public TrainerPastClassesPage()

{

InitializeComponent();

LoadClasses();

FilterComboBox.SelectedIndex = 0;

}

private void FilterComboBox\_SelectionChanged(object sender, SelectionChangedEventArgs e)

{

LoadClasses();

}

private void LoadClasses()

{

using var db = new AppDbContext();

var classes = db.Class

.Include(c => c.ClassInfo)

.Include(c => c.Hall)

.Include(c => c.WorkSchedule)

.Where(c => c.WorkSchedule.TrainerId == trainerId &&

c.WorkSchedule.WorkDate <= today)

.ToList();

string selected = (FilterComboBox.SelectedItem as ComboBoxItem)?.Content as string;

if (selected == "Только отмеченные")

{

classes = classes.Where(c => c.TrainerChecked).ToList();

}

else if (selected == "Только неотмеченные")

{

classes = classes.Where(c => !c.TrainerChecked && (today - c.WorkSchedule.WorkDate).TotalDays < 3).ToList();

}

PastClassesListView.ItemsSource = classes

.Select(c => new

{

c.ClassId,

ClassName = c.ClassInfo.ClassName,

Date = c.WorkSchedule.WorkDate,

StartTime = c.StartTime,

EndTime = c.EndTime,

HallName = c.Hall.HallName,

Price = c.Price ?? 0,

TrainerChecked = c.TrainerChecked

})

.OrderByDescending(x => x.Date)

.ToList();

}

private void PastClassesListView\_MouseDoubleClick(object sender, System.Windows.Input.MouseButtonEventArgs e)

{

dynamic selected = PastClassesListView.SelectedItem;

if (selected != null)

{

DateTime classDate = selected.Date;

if ((today - classDate).TotalDays > 3)

{

MessageBox.Show("Прошло более 3 дней с момента проведения занятия. Отметка недоступна.", "Ограничение", MessageBoxButton.OK, MessageBoxImage.Warning);

return;

}

NavigationService?.Navigate(new ClassVisitorsListPage(selected.ClassId));

}

}

}

}

Приложение Ф. Листинг кода TrainerProfilePage.xaml.cs

using fitness\_club.Classes;

using fitness\_club.Model;

using fitness\_club.Windows;

using Microsoft.EntityFrameworkCore;

using Microsoft.Win32;

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Net;

using System.Text;

using System.Text.RegularExpressions;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Navigation;

using System.Windows.Shapes;

namespace fitness\_club.Pages.TrainerPages

{

/// <summary>

/// Логика взаимодействия для TrainerProfilePage.xaml

/// </summary>

public partial class TrainerProfilePage : Page

{

private string projectImageFolderPath;

private string relativeImagePath;

public TrainerProfilePage()

{

InitializeComponent();

projectImageFolderPath = System.IO.Path.Combine(AppDomain.CurrentDomain.BaseDirectory, @"..\..\..\images");

LoadTrainerData();

}

private void LoadTrainerData()

{

var user = AuthorizationWin.currentUser;

using (var db = new AppDbContext())

{

var fullUser = db.Users

.Include(u => u.Role)

.Include(u => u.Trainer)

.ThenInclude(t => t.Gender)

.Include(u => u.Trainer.Specialization)

.FirstOrDefault(u => u.UserId == user.UserId);

if (fullUser?.Trainer == null)

{

MessageBox.Show("Тренер не найден.");

return;

}

var trainer = fullUser.Trainer;

// Левая колонка

dateOfEmployment.Text = trainer.DateOfEmployment.ToShortDateString();

individualPrice.Text = trainer.IndividualPrice?.ToString();

// Средняя колонка

LastNameTb.Text = trainer.LastName;

FirstNameTb.Text = trainer.FirstName;

PatronymicTb.Text = trainer.Patronymic;

GenderTb.Text = trainer.Gender?.GenderName ?? "Не указано";

BirthDateTb.Text = trainer.BirthDate.ToShortDateString();

KemVidanTb.Text = trainer.PassportKemVidan;

KogdaVidanTb.Text = trainer.PassportKogdaVidan.ToShortDateString();

SpecializationTb.Text = trainer.Specialization?.SpecializationName ?? "Не указана";

// Правая колонка

PassportSeriesTb.Text = trainer.PassportSeries;

PassportNumberTb.Text = trainer.PassportNumber;

PhoneTb.Text = trainer.PhoneNumber;

EmailTb.Text = trainer.Email;

LoginTb.Text = fullUser.Login;

RoleTb.Text = fullUser.Role?.RoleName ?? "Не указана";

// Аватар

if (!string.IsNullOrEmpty(trainer.Photo))

{

string fullImagePath = System.IO.Path.Combine(AppDomain.CurrentDomain.BaseDirectory, trainer.Photo);

if (File.Exists(fullImagePath))

{

AvatarPreview.Source = new BitmapImage(new Uri(fullImagePath));

}

}

}

}

private void SelectImage\_Click(object sender, RoutedEventArgs e)

{

OpenFileDialog openFileDialog = new OpenFileDialog

{

Filter = "Image Files (\*.png;\*.jpg;\*.jpeg)|\*.png;\*.jpg;\*.jpeg"

};

if (openFileDialog.ShowDialog() == true)

{

string selectedFilePath = openFileDialog.FileName;

string fileName = System.IO.Path.GetFileName(selectedFilePath);

string destFilePath = System.IO.Path.Combine(projectImageFolderPath, fileName);

// Копируем изображение в папку проекта (images)

File.Copy(selectedFilePath, destFilePath, true);

// Отображаем превью

AvatarPreview.Source = new BitmapImage(new Uri(destFilePath));

// Формируем относительный путь для БД

relativeImagePath = System.IO.Path.Combine("../../../images", fileName).Replace("\\", "/");

}

}

private void SaveChanges\_Click(object sender, RoutedEventArgs e)

{

using (var db = new AppDbContext())

{

var user = db.Users

.Include(u => u.Trainer)

.FirstOrDefault(u => u.UserId == AuthorizationWin.currentUser.UserId);

if (user?.Trainer == null)

{

MessageBox.Show("Пользователь или тренер не найден.");

return;

}

var trainer = user.Trainer;

// Проверки телефона, email, логина

string phone = PhoneTb.Text.Trim();

string email = EmailTb.Text.Trim().ToLower();

string login = LoginTb.Text.Trim();

if (!Regex.IsMatch(phone, @"^\+7\d{10}$") && !Regex.IsMatch(phone, @"^8\d{10}$"))

{

MessageBox.Show("Телефон должен быть в формате +7XXXXXXXXXX или 8XXXXXXXXXX");

return;

}

if (!Regex.IsMatch(email, @"^[a-z0-9]+@[a-z0-9]+**\.**[a-z]+$"))

{

MessageBox.Show("Введите корректный email.");

return;

}

if (login.Length < 4)

{

MessageBox.Show("Логин должен быть не короче 4 символов.");

return;

}

string newPassword = NewPasswordPb.Password.Trim();

string repeatPassword = RepeatPasswordPb.Password.Trim();

if (!string.IsNullOrEmpty(newPassword))

{

if (newPassword != repeatPassword)

{

MessageBox.Show("Пароли не совпадают.");

return;

}

var uf = new UserFromDb();

if (!uf.CheckPassword(newPassword)) return;

user.Password = Verification.GetSHA512Hash(newPassword);

}

// Обновляем поля

user.Login = login;

trainer.PhoneNumber = phone;

trainer.Email = email;

if (double.TryParse(individualPrice.Text.Trim(), out double price))

{

trainer.IndividualPrice = price;

}

// Сохраняем аватар, если был выбран

if (!string.IsNullOrEmpty(relativeImagePath))

{

trainer.Photo = relativeImagePath;

}

db.SaveChanges();

MessageBox.Show("Изменения успешно сохранены!");

}

}

}

}

Приложение Х. Листинг кода ClientVisitsPage.xaml.cs

using fitness\_club.Model;

using fitness\_club.Windows;

using Microsoft.EntityFrameworkCore;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Navigation;

using System.Windows.Shapes;

namespace fitness\_club.Pages.ClientPages

{

/// <summary>

/// Логика взаимодействия для ClientVisitsPage.xaml

/// </summary>

public partial class ClientVisitsPage : Page

{

public ClientVisitsPage()

{

InitializeComponent();

LoadClientPastVisits();

}

private void LoadClientPastVisits()

{

using (var db = new AppDbContext())

{

int clientId = db.Clients

.Where(c => c.UserId == AuthorizationWin.currentUser.UserId)

.Select(c => c.ClientId)

.FirstOrDefault();

var today = DateTime.Today;

var pastVisits = db.ClassVisits

.Include(cv => cv.Class)

.ThenInclude(c => c.WorkSchedule)

.ThenInclude(ws => ws.Trainer)

.Include(cv => cv.Class)

.ThenInclude(c => c.Hall)

.Include(cv => cv.Class)

.ThenInclude(c => c.ClassInfo)

.Where(cv =>

cv.ClientMembership.ClientId == clientId &&

cv.Class.WorkSchedule.WorkDate < today)

.Select(cv => new

{

cv.Class.ClassId,

ClassName = cv.Class.ClassInfo.ClassName,

Date = cv.Class.WorkSchedule.WorkDate,

Time = $"{cv.Class.StartTime:hh\\:mm} - {cv.Class.EndTime:hh\\:mm}",

HallName = cv.Class.Hall.HallName,

TrainerName = $"{cv.Class.WorkSchedule.Trainer.LastName} {cv.Class.WorkSchedule.Trainer.FirstName}",

Price = cv.Class.Price,

Visited = cv.Visited ? "Да" : "Нет"

})

.OrderByDescending(x => x.Date)

.ToList();

ClientVisitsList.ItemsSource = pastVisits;

}

}

private void RefundVisitContext\_Click(object sender, RoutedEventArgs e)

{

if (sender is not MenuItem menuItem || menuItem.CommandParameter is not int classId)

{

MessageBox.Show("Ошибка при определении занятия для возврата.", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);

return;

}

using var db = new AppDbContext();

int clientId = AuthorizationWin.currentUser.Client.ClientId;

// Подгружаем посещение вместе со всей нужной информацией

var visit = db.ClassVisits

.Include(cv => cv.Class)

.ThenInclude(c => c.ClassType)

.Include(cv => cv.Class)

.ThenInclude(c => c.ClassInfo)

.Include(cv => cv.ClientMembership)

.FirstOrDefault(cv => cv.ClassId == classId && cv.ClientMembership.ClientId == clientId);

if (visit == null)

{

MessageBox.Show("Посещение не найдено. Возможно, вы уже отменили занятие.", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Warning);

return;

}

// Проверка: посещено ли занятие

if (visit.Visited)

{

MessageBox.Show("Нельзя вернуть средства за уже посещённое занятие.", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Warning);

return;

}

// Получаем информацию об оплате

var payment = db.ClassPayments

.FirstOrDefault(cp => cp.ClassId == classId && cp.ClientId == clientId);

if (payment == null)

{

MessageBox.Show("Оплата за это занятие не найдена. Возможно, оно было бесплатным.", "Информация", MessageBoxButton.OK, MessageBoxImage.Information);

db.ClassVisits.Remove(visit); // просто удаляем посещение без возврата

db.SaveChanges();

LoadClientPastVisits();

return;

}

if (payment.PaymentDate == null)

{

MessageBox.Show("Занятие ещё не оплачено. Сначала оплатите, чтобы можно было оформить возврат.", "Информация", MessageBoxButton.OK, MessageBoxImage.Warning);

return;

}

var client = db.Clients.FirstOrDefault(c => c.ClientId == clientId);

if (client == null)

{

MessageBox.Show("Клиент не найден.", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);

return;

}

// Возврат средств

client.Balance += (decimal)payment.Price;

db.ClientTransactions.Add(new ClientTransaction

{

ClientId = client.ClientId,

OperationDescription = $"Возврат за занятие: {visit.Class.ClassInfo.ClassName}",

PaymentWay = "На баланс",

Amount = (decimal)payment.Price,

TransactionType = "возврат",

TransactionDate = DateTime.SpecifyKind(DateTime.UtcNow, DateTimeKind.Utc)

});

db.ClassPayments.Remove(payment);

db.ClassVisits.Remove(visit);

// Удаляем занятие, если индивидуальное

if (visit.Class.ClassType.ClassTypeId == 2)

{

var individualClass = db.Class.FirstOrDefault(c => c.ClassId == classId);

if (individualClass != null)

db.Class.Remove(individualClass);

}

db.SaveChanges();

MessageBox.Show($"Занятие отменено, средства возвращены на ваш баланс. Текущий баланс: {client.Balance}", "Успешно", MessageBoxButton.OK, MessageBoxImage.Information);

LoadClientPastVisits();

}

private void OpenClassDetails(dynamic selectedItem)

{

try

{

if (selectedItem != null)

{

int classId = selectedItem.ClassId;

ClassDetailsWindow cdw = new ClassDetailsWindow(classId, "visitsPage");

cdw.Show();

}

else

{

MessageBox.Show("Вы не выбрали занятие");

}

}

catch (Exception ex)

{

MessageBox.Show($"Не удалось загрузить информацию о занятии: {ex.Message}");

}

}

private void ClientVisitsList\_MouseDoubleClick(object sender, MouseButtonEventArgs e)

{

dynamic selectedItem = ClientVisitsList.SelectedItem;

if (selectedItem != null)

OpenClassDetails(selectedItem);

else

MessageBox.Show("Вы не выбрали занятие, о котором хотите загрузить информацию");

}

}

}

Приложение Ц. Листинг кода TrainerReviewsPage.xaml.cs

using fitness\_club.Model;

using fitness\_club.Windows;

using Microsoft.EntityFrameworkCore;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Navigation;

using System.Windows.Shapes;

namespace fitness\_club.Pages.TrainerPages

{

/// <summary>

/// Логика взаимодействия для TrainerReviewsPage.xaml

/// </summary>

public partial class TrainerReviewsPage : Page

{

private int trainerId;

public TrainerReviewsPage()

{

InitializeComponent();

trainerId = AuthorizationWin.currentUser.Trainer.TrainerId;

LoadTrainerReviews();

CalculateAvgRating();

}

private void LoadTrainerReviews()

{

using (var db = new AppDbContext())

{

var reviews = db.TrainerReviews

.Where(tr => tr.TrainerId == trainerId)

.Include(r => r.Author)

.Select(r => new

{

r.ReviewContent,

r.ReviewGrade,

ClientName = r.Author != null && r.Author.IsAddAuthorName

? $"{r.Author.LastName} {r.Author.FirstName}"

: "Аноним"

})

.ToList();

reviewsListView.ItemsSource = reviews;

}

}

private void CalculateAvgRating()

{

using (var db = new AppDbContext())

{

var trainer = db.Trainers

.Include(t => t.TrainerReviews)

.FirstOrDefault(t => t.TrainerId == trainerId);

double avgRating = 0;

if (trainer != null && trainer.TrainerReviews.Any())

{

avgRating = trainer.TrainerReviews.Average(r => r.ReviewGrade);

}

avgTrainerRating.Text = $"Ваша средняя оценка: {avgRating:F1} / 5";

}

}

}

}

Приложение Ш. Листинг кода TrainerWorkSchedule.xaml.cs

using fitness\_club.Model;

using fitness\_club.Windows;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Navigation;

using System.Windows.Shapes;

namespace fitness\_club.Pages.TrainerPages

{

/// <summary>

/// Логика взаимодействия для TrainerWorkSchedule.xaml

/// </summary>

public partial class TrainerWorkSchedule : Page

{

public TrainerWorkSchedule()

{

InitializeComponent();

LoadTrainerSchedule();

}

private void LoadTrainerSchedule()

{

using var db = new AppDbContext();

var trainerId = AuthorizationWin.currentUser.Trainer.TrainerId;

var today = DateTime.Today;

var inTwoWeeks = today.AddDays(14);

var schedule = db.WorkSchedules

.Where(ws => ws.TrainerId == trainerId &&

ws.WorkDate >= today &&

ws.WorkDate <= inTwoWeeks)

.OrderBy(ws => ws.WorkDate)

.Select(ws => new

{

ws.WorkDate,

StartTime = ws.StartTime.ToString(@"hh\:mm"),

EndTime = ws.EndTime.ToString(@"hh\:mm")

})

.ToList();

TrainerScheduleGrid.ItemsSource = schedule;

}

}

}

Приложение Щ. Листинг кода AdminEquipmentManagmentPage.xaml.cs

using fitness\_club.Classes;

using fitness\_club.Model;

using Microsoft.EntityFrameworkCore;

using System;

using System.Collections.Generic;

using System.Globalization;

using System.Linq;

using System.Windows;

using System.Windows.Controls;

namespace fitness\_club.Pages.AdminPages

{

public partial class AdminEquipmentManagementPage : Page

{

private List<EquipmentCondition> conditions;

private List<EquipmentEditableRow> equipmentData;

public AdminEquipmentManagementPage()

{

InitializeComponent();

LoadConditions();

LoadEquipment();

}

private void LoadConditions()

{

using var db = new AppDbContext();

conditions = db.EquipmentConditions.ToList();

ConditionColumn.ItemsSource = conditions;

}

private void LoadEquipment()

{

using var db = new AppDbContext();

equipmentData = db.Equipment

.Select(e => new EquipmentEditableRow

{

EquipmentId = e.EquipmentId,

EquipmentName = e.EquipmentName,

EquipmentConditionId = e.EquipmentConditionId,

DeliveryDate = e.DeliveryDate.ToString("dd.MM.yyyy"),

LastMaintenanceDate = e.LastMaintenanceDate.ToString("dd.MM.yyyy"),

Quantity = e.Quantity

}).ToList();

EquipmentGrid.ItemsSource = equipmentData;

}

private void SaveChanges\_Click(object sender, RoutedEventArgs e)

{

using var db = new AppDbContext();

foreach (var row in equipmentData)

{

var entity = db.Equipment.FirstOrDefault(e => e.EquipmentId == row.EquipmentId);

if (entity != null)

{

entity.EquipmentConditionId = row.EquipmentConditionId;

if (DateTime.TryParseExact(row.LastMaintenanceDate, "dd.MM.yyyy", CultureInfo.InvariantCulture, DateTimeStyles.None, out var date))

{

entity.LastMaintenanceDate = date;

}

entity.Quantity = row.Quantity;

}

}

db.SaveChanges();

MessageBox.Show("Изменения успешно сохранены!", "Готово", MessageBoxButton.OK, MessageBoxImage.Information);

}

private void AssignEquipment\_Click(object sender, RoutedEventArgs e)

{

if (sender is Button button && button.Tag is int id)

{

var win = new AssignEquipmentWindow(id);

win.ShowDialog();

LoadEquipment();

}

}

private void AddEquipment\_Click(object sender, RoutedEventArgs e)

{

string equipmentName = NewEquipmentNameTb.Text.Trim();

if (!int.TryParse(NewEquipmentQtyTb.Text, out int quantity) || quantity <= 0)

{

MessageBox.Show("Введите корректное количество (>0)", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);

return;

}

if (string.IsNullOrWhiteSpace(equipmentName))

{

MessageBox.Show("Введите название оборудования.", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);

return;

}

using var db = new AppDbContext();

// По умолчанию ставим состояние "Новое" или 1

int defaultConditionId = db.EquipmentConditions

.OrderBy(ec => ec.EquipmentConditionId)

.Select(ec => ec.EquipmentConditionId)

.FirstOrDefault();

var newEquipment = new Equipment

{

EquipmentName = equipmentName,

EquipmentConditionId = defaultConditionId,

DeliveryDate = DateTime.UtcNow,

LastMaintenanceDate = DateTime.UtcNow,

Quantity = quantity

};

db.Equipment.Add(newEquipment);

db.SaveChanges();

MessageBox.Show("Оборудование успешно добавлено!", "Успех", MessageBoxButton.OK, MessageBoxImage.Information);

// Обновляем таблицу

LoadEquipment();

// Очищаем поля

NewEquipmentNameTb.Text = "";

NewEquipmentQtyTb.Text = "";

}

private void DeleteEquipment\_Click(object sender, RoutedEventArgs e)

{

if (sender is Button button && button.Tag is int equipmentId)

{

var result = MessageBox.Show("Вы уверены, что хотите удалить это оборудование?\nОно также будет удалено из всех залов.",

"Подтверждение удаления",

MessageBoxButton.YesNo,

MessageBoxImage.Warning);

if (result != MessageBoxResult.Yes)

return;

try

{

using var db = new AppDbContext();

// Удаляем все связи в hall\_equipment

var hallEquipments = db.HallEquipments.Where(he => he.EquipmentId == equipmentId).ToList();

db.HallEquipments.RemoveRange(hallEquipments);

// Удаляем оборудование

var equipment = db.Equipment.FirstOrDefault(e => e.EquipmentId == equipmentId);

if (equipment != null)

{

db.Equipment.Remove(equipment);

db.SaveChanges();

MessageBox.Show("Оборудование успешно удалено.", "Успех", MessageBoxButton.OK, MessageBoxImage.Information);

LoadEquipment(); // Обновить таблицу

}

else

{

MessageBox.Show("Оборудование не найдено.", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);

}

}

catch (Exception ex)

{

MessageBox.Show($"Ошибка при удалении оборудования: {ex.Message}", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);

}

}

}

}

public class EquipmentEditableRow

{

public int EquipmentId { get; set; }

public string EquipmentName { get; set; }

public int EquipmentConditionId { get; set; }

public string DeliveryDate { get; set; }

public string LastMaintenanceDate { get; set; }

public int Quantity { get; set; }

}

}

Приложение Э. Листинг кода AdminManageGroupClassesPage.xaml.cs

using fitness\_club.Model;

using fitness\_club.Classes;

using Microsoft.EntityFrameworkCore;

using System;

using System.Globalization;

using System.Linq;

using System.Windows;

using System.Windows.Controls;

namespace fitness\_club.Pages.AdminPages

{

public partial class AdminManageGroupClassesPage : Page

{

public AdminManageGroupClassesPage()

{

InitializeComponent();

LoadData();

LoadClasses();

//LoadWorkSchedules();

LoadFreeSlots();

}

private void LoadFreeSlots()

{

using var db = new AppDbContext();

var today = DateTime.Today;

var workSchedules = db.WorkSchedules

.Include(ws => ws.Trainer)

.Where(ws => ws.WorkDate >= today)

.OrderBy(ws => ws.WorkDate)

.ThenBy(ws => ws.StartTime)

.ToList();

var groupClasses = db.Class

.Include(c => c.WorkSchedule)

.Where(c => c.ClassTypeId == 1 && c.WorkSchedule.WorkDate >= today)

.ToList();

var freeSlots = new List<object>();

foreach (var ws in workSchedules)

{

var trainerId = ws.TrainerId;

var date = ws.WorkDate;

var start = ws.StartTime;

var end = ws.EndTime;

var classes = groupClasses

.Where(c => c.WorkSchedule.TrainerId == trainerId && c.WorkSchedule.WorkDate == date)

.OrderBy(c => c.StartTime)

.ToList();

var current = start;

foreach (var cls in classes)

{

if (current < cls.StartTime)

{

freeSlots.Add(new

{

Trainer = ws.Trainer.LastName + " " + ws.Trainer.FirstName,

Date = date.ToShortDateString(),

Slot = $"{current:hh\\:mm} - {cls.StartTime:hh\\:mm}"

});

}

current = cls.EndTime > current ? cls.EndTime : current;

}

if (current < end)

{

freeSlots.Add(new

{

Trainer = ws.Trainer.LastName + " " + ws.Trainer.FirstName,

Date = date.ToShortDateString(),

Slot = $"{current:hh\\:mm} - {end:hh\\:mm}"

});

}

}

FreeSlotsListView.ItemsSource = freeSlots;

}

private void LoadData()

{

using var db = new AppDbContext();

TrainerComboBox.ItemsSource = db.Trainers

.Select(t => new

{

t.TrainerId,

FullName = t.LastName + " " + t.FirstName

}).ToList();

HallComboBox.ItemsSource = db.Halls

.Select(h => new

{

h.HallId,

h.HallName

}).ToList();

ClassInfoComboBox.ItemsSource = db.ClassInfo

.Where(c=>c.ClassName!= "Индивидуальное занятие")

.Select(c => new

{

c.ClassInfoId,

c.ClassName

}).ToList();

DatePicker.SelectedDate = DateTime.Today;

}

private void CreateClass\_Click(object sender, RoutedEventArgs e)

{

if (TrainerComboBox.SelectedValue is not int trainerId ||

HallComboBox.SelectedValue is not int hallId ||

ClassInfoComboBox.SelectedValue is not int classInfoId ||

!DatePicker.SelectedDate.HasValue ||

!TimeSpan.TryParseExact(StartTimeBox.Text, "hh\\:mm", CultureInfo.InvariantCulture, out var startTime) ||

!TimeSpan.TryParseExact(EndTimeBox.Text, "hh\\:mm", CultureInfo.InvariantCulture, out var endTime) ||

!int.TryParse(PeopleQtyBox.Text, out int peopleQty) || !double.TryParse(priceBox.Text, out double price))

{

MessageBox.Show("Пожалуйста, заполните все поля корректно.");

return;

}

if (startTime >= endTime)

{

MessageBox.Show("Время начала должно быть раньше времени окончания.");

return;

}

var date = DatePicker.SelectedDate.Value;

using var db = new AppDbContext();

var workSchedule = db.WorkSchedules

.FirstOrDefault(ws => ws.TrainerId == trainerId && ws.WorkDate == date);

if (workSchedule == null)

{

MessageBox.Show("У выбранного тренера нет смены на эту дату.");

return;

}

var newClass = new Class

{

ClassInfoId = classInfoId,

ClassTypeId = 1, // групповые занятия

HallId = hallId,

WorkScheduleId = workSchedule.WorkScheduleId,

StartTime = startTime,

EndTime = endTime,

PeopleQuantity = peopleQty,

Price = price

};

bool overlaps = db.Class.Any(c =>

c.WorkScheduleId == workSchedule.WorkScheduleId &&

startTime < c.EndTime &&

endTime > c.StartTime);

if (overlaps)

{

MessageBox.Show("Нельзя создать занятие: время пересекается с другим занятием у этого тренера.");

return;

}

db.Class.Add(newClass);

db.SaveChanges();

MessageBox.Show("Занятие успешно создано.");

LoadFreeSlots();

LoadClasses();

}

private void LoadClasses()

{

using var db = new AppDbContext();

var today = DateTime.Today;

var groupClasses = db.Class

.Include(c => c.ClassInfo)

.Include(c => c.Hall)

.Include(c => c.WorkSchedule)

.ThenInclude(ws => ws.Trainer)

.Where(c => c.ClassTypeId == 1 && c.WorkSchedule.WorkDate >= today)

.OrderBy(c => c.WorkSchedule.WorkDate)

.ThenBy(c => c.StartTime)

.Select(c => new

{

c.ClassId,

Date = c.WorkSchedule.WorkDate.ToShortDateString(),

Time = $"{c.StartTime:hh\\:mm} - {c.EndTime:hh\\:mm}",

ClassName = c.ClassInfo.ClassName,

TrainerName = c.WorkSchedule.Trainer.LastName + " " + c.WorkSchedule.Trainer.FirstName,

HallName = c.Hall.HallName,

PeopleQty = c.PeopleQuantity,

c.Price

})

.ToList();

GroupClassesListView.ItemsSource = groupClasses;

}

private void DeleteClass\_Click(object sender, RoutedEventArgs e)

{

if (sender is Button button && button.Tag is int classId)

{

using var db = new AppDbContext();

var selected = db.Class.Find(classId);

if (selected != null)

{

db.Class.Remove(selected);

db.SaveChanges();

MessageBox.Show("Занятие удалено.");

LoadClasses();

}

}

}

private void EditClass\_Click(object sender, RoutedEventArgs e)

{

if (sender is Button button && button.Tag is int classId)

{

var win = new EditGroupClassWindow(classId);

win.ShowDialog();

LoadClasses();

}

}

private void FreeSlotsListView\_MouseDoubleClick(object sender, System.Windows.Input.MouseButtonEventArgs e)

{

dynamic selected = FreeSlotsListView.SelectedItem;

if (selected!=null)

{

// Название тренера из слота

string trainerName = selected.Trainer as string;

string dateString = selected.Date as string;

string timeSlot = selected.Slot as string;

// Разделим время на начало и конец

var timeParts = timeSlot.Split(" - ");

if (timeParts.Length == 2)

{

StartTimeBox.Text = timeParts[0];

EndTimeBox.Text = timeParts[1];

}

// Установим дату

if (DateTime.TryParseExact(dateString, "dd.MM.yyyy", CultureInfo.InvariantCulture, DateTimeStyles.None, out var parsedDate))

{

DatePicker.SelectedDate = parsedDate;

}

// Найдём тренера в ComboBox по совпадению ФИО

foreach (var item in TrainerComboBox.Items)

{

dynamic trainer = item;

if ((trainer.FullName as string).Trim() == trainerName.Trim())

{

TrainerComboBox.SelectedItem = trainer;

break;

}

}

MessageBox.Show("Слот выбран. Вы можете сразу нажать 'Создать занятие'.", "Подсказка", MessageBoxButton.OK, MessageBoxImage.Information);

}

}

}

}

Приложение Ю. Листинг кода AdminMembershipManagementPage.xaml.cs

using fitness\_club.Classes;

using fitness\_club.Model;

using Microsoft.EntityFrameworkCore;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Navigation;

using System.Windows.Shapes;

namespace fitness\_club.Pages.AdminPages

{

/// <summary>

/// Логика взаимодействия для AdminMembershipManagementPage.xaml

/// </summary>

public partial class AdminMembershipManagementPage : Page

{

private List<MembershipEditableRow> memberships;

public AdminMembershipManagementPage()

{

InitializeComponent();

LoadMembershipTypes();

LoadMemberships();

}

private void LoadMembershipTypes()

{

using var db = new AppDbContext();

MembershipTypeComboBox.ItemsSource = db.MembershipTypes.ToList();

}

private void LoadMemberships()

{

using var db = new AppDbContext();

memberships = db.Memberships

.Select(m => new MembershipEditableRow

{

MembershipId = m.MembershipId,

MembershipName = m.MembershipName,

MembershipDescription = m.MembershipDescription,

Price = m.Price,

MembershipTypeName = m.MembershipType.MembershipTypeName,

DurationMonths = m.MembershipType.DurationMonths ?? 0

})

.ToList();

MembershipGrid.ItemsSource = memberships;

}

private void SaveChanges\_Click(object sender, RoutedEventArgs e)

{

using var db = new AppDbContext();

foreach (var row in memberships)

{

var membership = db.Memberships

.Include(m => m.MembershipType)

.FirstOrDefault(m => m.MembershipId == row.MembershipId);

if (membership != null)

{

membership.MembershipDescription = row.MembershipDescription;

membership.Price = row.Price;

if (membership.MembershipType != null)

{

membership.MembershipType.DurationMonths = row.DurationMonths;

}

}

}

db.SaveChanges();

MessageBox.Show("Изменения сохранены!");

LoadMemberships();

}

private void AddMembership\_Click(object sender, RoutedEventArgs e)

{

if (string.IsNullOrWhiteSpace(NewMembershipNameBox.Text) ||

!int.TryParse(NewMembershipPriceBox.Text, out int price) ||

MembershipTypeComboBox.SelectedValue == null)

{

MessageBox.Show("Пожалуйста, заполните все поля корректно.");

return;

}

using var db = new AppDbContext();

db.Memberships.Add(new Membership

{

MembershipName = NewMembershipNameBox.Text.Trim(),

MembershipDescription = NewMembershipDescriptionBox.Text?.Trim(),

Price = price,

MembershipTypeId = (int)MembershipTypeComboBox.SelectedValue

});

db.SaveChanges();

MessageBox.Show("Абонемент добавлен!");

LoadMemberships();

}

private void AddMembershipType\_Click(object sender, RoutedEventArgs e)

{

var name = NewMembershipTypeNameBox.Text.Trim();

if (string.IsNullOrWhiteSpace(name) ||

!int.TryParse(NewDurationBox.Text, out int duration))

{

MessageBox.Show("Введите корректные данные.");

return;

}

using var db = new AppDbContext();

db.MembershipTypes.Add(new MembershipType

{

MembershipTypeName = name,

DurationMonths = duration

});

db.SaveChanges();

MessageBox.Show("Тип абонемента добавлен!");

LoadMembershipTypes();

}

public class MembershipEditableRow

{

public int MembershipId { get; set; }

public string MembershipName { get; set; }

public string MembershipTypeName { get; set; }

public string MembershipDescription { get; set; }

public int Price { get; set; }

public int DurationMonths { get; set; }

}

}

}

Приложение Я. Листинг кода AdminModerationPage.xaml.cs

using fitness\_club.Model;

using Microsoft.EntityFrameworkCore;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Navigation;

using System.Windows.Shapes;

namespace fitness\_club.Pages.AdminPages

{

public partial class AdminModerationPage : Page

{

public AdminModerationPage()

{

InitializeComponent();

LoadReviews();

}

private void LoadReviews()

{

using var db = new AppDbContext();

var trainerReviews = db.TrainerReviews

.Include(r => r.Trainer)

.Include(r => r.Author)

.Where(r => !r.Moderated)

.ToList()

.Select(r => new ModerationItem

{

ReviewType = "trainer",

ReviewId = r.TrainerReviewId,

ObjectName = $"Тренер: {r.Trainer.LastName} {r.Trainer.FirstName}",

ReviewContent = r.ReviewContent,

Grade = r.ReviewGrade,

AuthorName = r.Author.IsAddAuthorName ? $"{r.Author.LastName} {r.Author.FirstName}" : "Аноним"

});

var classReviews = db.ClassReviews

.Include(r => r.ClassInfo)

.Include(r => r.Client)

.Where(r => !r.Moderated)

.ToList()

.Select(r => new ModerationItem

{

ReviewType = "class",

ReviewId = r.ReviewId,

ObjectName = $"Занятие: {r.ClassInfo.ClassName}",

ReviewContent = r.ReviewContent,

Grade = r.ReviewGrade,

AuthorName = r.Client.IsAddAuthorName ? $"{r.Client.LastName} {r.Client.FirstName}" : "Аноним"

});

var allReviews = trainerReviews.Concat(classReviews)

.OrderBy(r => r.ObjectName)

.ToList();

ReviewsListView.ItemsSource = allReviews;

}

private void ApproveReview\_Click(object sender, RoutedEventArgs e)

{

if (sender is Button button && button.Tag is ModerationItem review)

{

using var db = new AppDbContext();

if (review.ReviewType == "trainer")

{

var r = db.TrainerReviews.Find(review.ReviewId);

if (r != null)

r.Moderated = true;

}

else if (review.ReviewType == "class")

{

var r = db.ClassReviews.Find(review.ReviewId);

if (r != null)

r.Moderated = true;

}

db.SaveChanges();

MessageBox.Show("Отзыв одобрен.");

LoadReviews();

}

}

private void RejectReview\_Click(object sender, RoutedEventArgs e)

{

if (sender is Button button && button.Tag is ModerationItem review)

{

using var db = new AppDbContext();

if (review.ReviewType == "trainer")

{

var r = db.TrainerReviews.Find(review.ReviewId);

if (r != null)

db.TrainerReviews.Remove(r);

}

else if (review.ReviewType == "class")

{

var r = db.ClassReviews.Find(review.ReviewId);

if (r != null)

db.ClassReviews.Remove(r);

}

db.SaveChanges();

MessageBox.Show("Отзыв отклонён и удалён.");

LoadReviews();

}

}

}

public class ModerationItem

{

public string ReviewType { get; set; }

public int ReviewId { get; set; }

public string ObjectName { get; set; }

public string ReviewContent { get; set; }

public int Grade { get; set; }

public string AuthorName { get; set; }

}

}