**Содержание**

[Введение 3](#_Toc166673644)

[1 Постановка задачи 4](#_Toc166673645)

[1.1. Анализ прототипов 4](#_Toc166673646)

[1.1.1 Zona531 4](#_Toc166673647)

[1.1.2 Fitness-club.by 4](#_Toc166673648)

[1.2. Требования к проекту 5](#_Toc166673649)

[1.3. Вывод 5](#_Toc166673650)

[2 Проектирование web-приложения 6](#_Toc166673651)

[2.1 Архитектура приложения 6](#_Toc166673652)

[2.2 UML- диаграммы 7](#_Toc166673653)

[2.3 Проектирование базы данных 9](#_Toc166673654)

[2.4 Вывод 13](#_Toc166673655)

[3 Разработка web-приложения 14](#_Toc166673656)

[3.1 Разработка серверной части 14](#_Toc166673657)

[3.2 Разработка клиентской части 19](#_Toc166673658)

[3.3 Вывод 20](#_Toc166673659)

[4 Тестирование web приложения 21](#_Toc166673660)

[4.1 Авторизация и регистрация 21](#_Toc166673661)

[4.2 Функции клиентской части 22](#_Toc166673662)

[4.3 Вывод 23](#_Toc166673663)

[5 Руководство пользователя 24](#_Toc166673664)

[Заключение 28](#_Toc166673665)

[Список литературы 29](#_Toc166673666)

[Приложение А.Пример использования react router dom 30](#_Toc166673667)

# **Введение**

В настоящее время забота о здоровье и фитнесе стала одной из наиболее актуальных тем. Многие люди желают улучшить свое физическое состояние и поддерживать здоровье в хорошей форме, поэтому спрос на услуги фитнес-центров и спортивных залов неуклонно растет.

В современном быстротемповом мире, где все жизненные процессы автоматизируются, покупка абонементов в фитнес-центр становиться очень приятным предложением, так как не нужно покидать свое жилье, чтобы ознакомиться с предлагаемыми сервисами, тренерами и ценами, а сделать все дистанционно.

В этом контексте web-приложение для «Фитнес-центра» представляет собой инновационное и полезное решение. Приложение объединяет преимущества современных технологий с необходимостью эффективного создания карточек абонементов, которые включают в себе множество различных аспектов.

Одной из ключевых особенностей приложения является возможность просмотра абонементов в виде списка с удобной фильтрацией по ценам и поиску по названию. Что помогает клиенту удобно искать необходимую единицу.

Другой полезной функцией приложения является просмотр тренеров фитнес-центра с возможностью оставить комментарий. Это полезная функция, так как люди смогут видеть профессионализм тренеров, которым они доверяют свое здоровье.

Кроме того, приложение предлагает возможность редактирования профиля и просмотра истории купленных абонементов. Присутствует приятное дополнение в виде оповещений клиентов об окончании действия абонемента.  
 И конечно, стоит выделить удобный интерфейс создания и редактирования карточек абонементов и тренеров для пользователя-администратора.

Важное преимущество данного проекта состоит в его простоте использования и интуитивном интерфейсе. Пользователи могут легко просматривать, фильтровать и покупать абонементы. Это снижает барьеры в использовании приложения и делает его доступным для новых пользователей.

Таким образом, данное web-приложение «Фитнес-центр» представляет собой инновационное и актуальное решение для эффективного управления фитнес-центра. Оно объединяет преимущества современных технологий в организации единиц фитнес-индустрии, предоставляя пользователям удобный и интуитивно понятный интерфейс для достижения успеха в управлении своими абонементами.

# **1 Постановка задачи**

## **Анализ прототипов**

Анализ аналогов является важным этапом при разработке веб-приложения, так как позволяет изучить опыт других проектов и применить наработки для создания более успешного и конкурентоспособного продукта, а также выявить сильные и слабые стороны существующих проектов и найти возможности для инноваций и улучшений собственного проекта.

### **1.1.1 Zona531**

«Zona531» – это web-приложение для автоматизации процессов одноименного фитнес-центра, который предлагает широкий спектр функций, включая возможности для связи с тренерами, функции для просмотра и покупки различных услуг.

Преимущества: «Zona531» обладает отличными возможностями для ознакомления с услугами фитнес-центра, легок и комфортен в использовании.

Недостатки: «Zona531» не имеет функций для уведомления клиентов об окончании времени действия их абонемента или фитнес-услуги, приобретенной в определенный период времени в фитнес-центре.

«Zona531» начиналось как сайт-визитка для хранения текстовой и графической информации о фитнес-центре и со временем превратился в универсальное web-приложение для бронирования абонементов. Оно предлагает отличные возможности для ознакомления с тренерами и легко осваивается

Интерфейс интернет-ресурса представлен на рисунке 1.1.

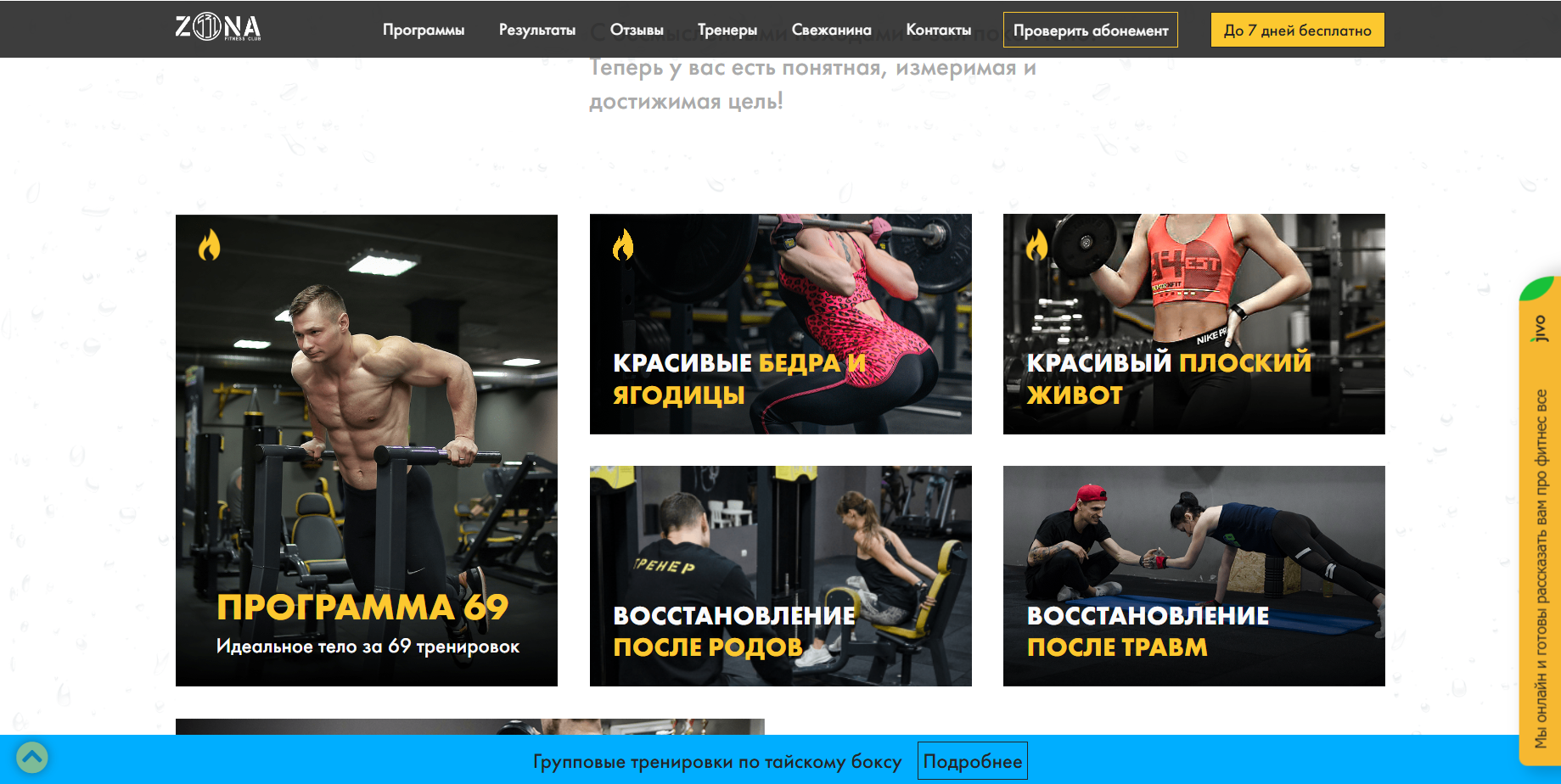


Рисунок 1.1 – Web-приложение «Zona531»

### **1.1.2 Fitness-club.by**

Еще одним альтернативным решением задач, поставленных в данном курсовом проекте, является интернет-ресурс «Fitness-club.by». [Среди функций Fitness-club.by можно выделить назначение плана тренировок в зависимости от выбранной услуги, временные скидки, интеграцию с другими веб-ресурсами, управление бюджетами проектов, а также наличие адаптивности для различных платформ](https://www.forbes.com/advisor/business/software/todoist-review/).

[Преимущества: простой и чистый интерфейс, предоставляет статистику по купленным товарам и абонементам, возможность связи с тренером.](https://www.pcmag.com/reviews/todoist)

[Недостатки: некоторые важные функции, такие как напоминания об окончании срока действия абонемента, недоступны для бесплатного плана](https://www.pcmag.com/reviews/todoist), отсутствует возможность оставить комментарии к тренерам.

Интерфейс данного интернет-ресурса представлен на рисунке 1.2.

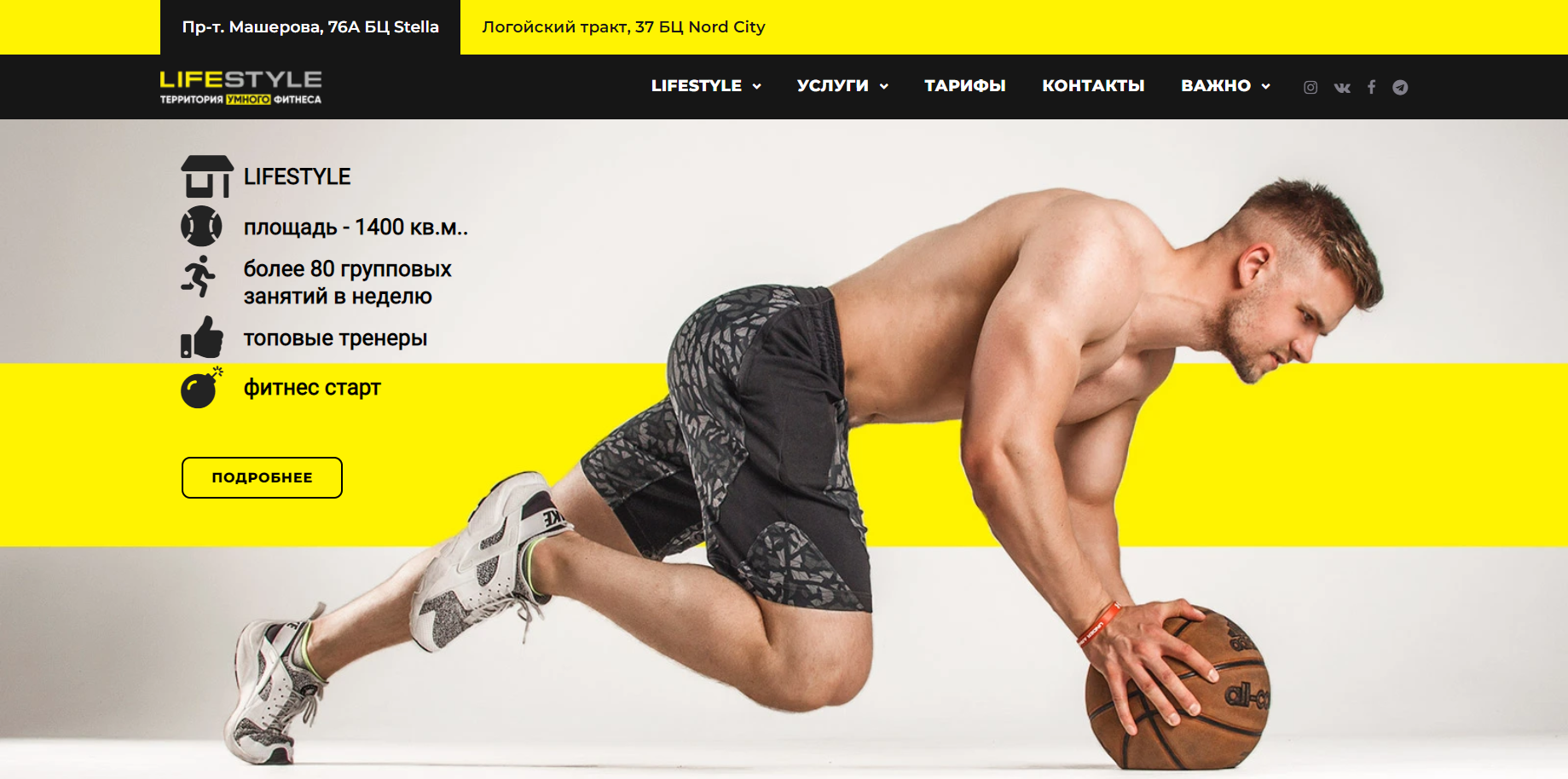


Рисунок 1.2 – Интернет-ресурс «Fitness-club.by»

## **Требования к проекту**

Проект должно обеспечивать выполнение следующих функций:

* обеспечивать возможность регистрации и авторизации;
* поддерживать роли администратора и пользователя;
* предоставлять подробную информацию о абонементах;
* позволять изменять информацию о абонементах;
* напоминать о истечении строка абонемента;
* позволять просматривать абонементы и осуществлять их покупку;
* предоставлять возможность поиска абонемента по различным критериям;
* позволять администратору управлять информацией о тренерах;
* позволять редактировать профиль пользователя

## **Вывод**

Были проанализированы цели и задачи, поставленные в данном курсовом проекте, а также рассмотрены аналогичные примеры их решений. На основании анализа всех достоинств и недостатков данных альтернативных решений были сформулированы требования к данному программному средству.

2 Проектирование web-приложения

2.1 Архитектура приложения

Архитектура программного решения для серверной части проекта, реализованного с использованием фреймворка Express.js, определяет структуру, организацию и взаимодействие его компонентов, а также распределение функций.

Express (ExpressJS) – это фреймворк для создания эффективных, масштабируемых Node.js серверных приложений. Он использует прогрессивный JavaScript, построен и полностью поддерживает TypeScript (но при этом позволяет разработчикам писать код на чистом JavaScript) и сочетает в себе элементы ООП (объектно-ориентированное программирование), FP (функциональное программирование) и FRP (функциональное реактивное программирование).

В данном проекте применяется трехуровневая архитектура. Эти три уровня включают контроллеры (controllers), сервисы (services) и модули (repositories).

Контроллеры (controllers) являются основным уровнем взаимодействия с клиентом и отвечают за обработку входящих запросов и отправку ответов. Они определяют конечные точки API и содержат логику обработки запросов. Они также служат связующим звеном между внешними запросами и сервисами приложения, принимая запросы и обрабатывая их, а затем передавая соответствующим сервисам для выполнения бизнес-логики. Контроллеры могут также выполнять валидацию данных, авторизацию и другие операции, связанные с обработкой запросов.

Сервисы (services) содержат бизнес-логику приложения и выполняют операции с данными, полученными от контроллеров. Они являются промежуточным уровнем между контроллерами и репозиториями (или другими сервисами). Сервисы могут включать операции CRUD (создание, чтение, обновление и удаление) над данными, выполнение сложных вычислений и манипуляцию с данными. Они также могут взаимодействовать с репозиториями для доступа к данным или с другими сервисами для выполнения определенных задач. Использование сервисов позволяет выделить и организовать бизнес-логику в отдельные компоненты, что способствует повторному использованию кода и улучшает его поддержку.

Репозитории (repositories) представляют собой слой, отвечающий за взаимодействие с базой данных или другими источниками хранения данных. Они инкапсулируют логику доступа к данным и обеспечивают абстракцию, позволяющую изолировать остальную часть приложения от конкретной технологии базы данных. Репозитории предоставляют методы для выполнения операций CRUD (создание, чтение, обновление, удаление), а также для выполнения сложных запросов. Использование репозиториев упрощает тестирование, так как можно легко заменять репозиторий моком или стабом. Репозитории могут работать напрямую с ORM (Object-Relational Mapping) для преобразования данных между объектами и таблицами.

Трехуровневая архитектура Express.js обеспечивает хорошую организацию компонентов приложения, улучшает его читаемость, поддержку и масштабируемость. Контроллеры, сервисы и модули работают совместно, чтобы обеспечить эффективное взаимодействие с клиентами, выполнение бизнес-логики и организацию кодовой базы приложения.

Для доступа к данным используется база данных SQLServer, развернутая на собственном ПК, которая обеспечивает надежное хранение и управление данными.

Взаимодействие между сервером и клиентом осуществляется посредством Web API. Сервер предоставляет API для обмена информацией с клиентским приложением, которое отправляет HTTPS-запросы на сервер и получает ответы в формате JSON. Это позволяет серверной и клиентской частям взаимодействовать и передавать данные между собой, обеспечивая функциональность приложения.

* 1. UML- диаграммы

UML (англ. Unified Modeling Language – унифицированный язык моделирования) – язык графического описания для объектного моделирования в области разработки программного обеспечения, для моделирования бизнес-процессов, системного проектирования и отображения организационных структур.

UML является языком широкого профиля, это – открытый стандарт, использующий графические обозначения для создания абстрактной модели системы, называемой UML-моделью. UML был создан для определения, визуализации, проектирования и документирования, программных систем. UML не является языком программирования, но на основании UML-моделей возможна генерация кода.

Также каждая диаграмма в UML имеет свою специфичную цель и фокус, но вместе они образуют полный набор инструментов для моделирования систем различного масштаба и сложности. UML-диаграммы могут быть использованы на различных этапах разработки программного обеспечения, от анализа и проектирования до реализации и тестирования.

Важно отметить, что UML не является жестким стандартом, и его использование может различаться в зависимости от контекста и предпочтений команды разработчиков. Однако, универсальность и широкое распространение UML делают его ценным инструментом для коммуникации и документирования во многих проектах разработки программного обеспечения.

Для эффективного использования системы важно определить роли пользователей и их сценарии использования. Сценарии описывают, как каждый пользователь будет взаимодействовать с системой в соответствии с их ролью. Это помогает определить доступные функции, доступные данные и организацию навигации в системе. Для визуализации взаимодействия между пользователями и системой используются диаграммы UML.

Роли пользователей определяют набор прав и привилегий, предоставляемых пользователям в системе. Каждая роль имеет свои уникальные возможности и доступ к функциям системы. В курсовом проекте роли пользователей включают неавторизованного пользователя, авторизованного пользователя и администратора.

Неавторизованный пользователь имеет возможность авторизоваться и зарегистрироваться в web-приложении.

Авторизованный пользователь имеет возможность просматривать абонементы, покупать абонементы, редактировать профиль.

Администратор имеет возможность редактировать тренеров, абонементы и всё остальное, что входит в эти понятия.

Далее определим, как будет происходить взаимодействие пользователя с системой. Для этого используем Use-case диаграмму. Диаграмма вариантов использования приведена на рисунке 2.1.



Рисунок 2.1 – Диаграмма вариантов использования

Диаграмма развёртывания моделирует физическое развёртывание компонентов и архитектуры распределённой программной системы на аппаратном обеспечении и содержит изображение размещения различных артефактов по отдельным узлам системы. Она визуализирует только те элементы, что существуют лишь на этапе её исполнения. При этом представляются только компоненты-экземпляры программы, являющиеся исполнимыми файлами или динамическими библиотеками. Те компоненты, которые не используются на этапе исполнения, на диаграмме развертывания не показываются.

На диаграмме развертывания показывается структура системы в терминах узлов и соединений между ними. Компоненты системы представляются в виде узлов, а связи между ними показывают, как компоненты взаимодействуют между собой и внешними ресурсами (например, с базами данных или сетями). Данный тип диаграммы позволяет лучше разобраться в архитектуре, физической конфигурации системы и её взаимодействие с окружающей средой.

Диаграмма развёртывания приведена на рисунке 2.2.

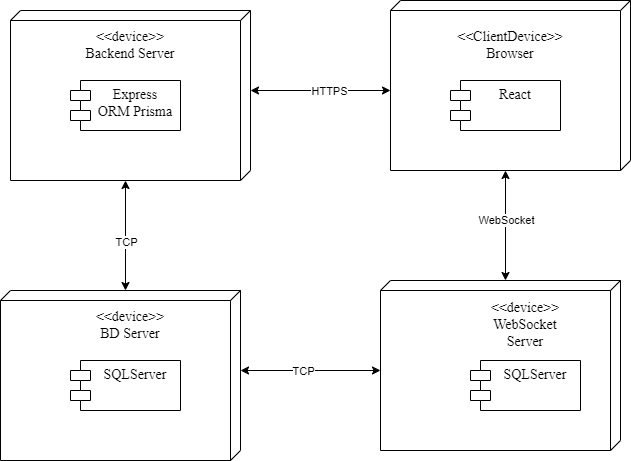


Рисунок 2.2 – Диаграмма развёртывания

Клиент использует браузер для отправки запросов к web-серверу. Клиент и сервер могут обмениться сообщениям по протоколам HTTPS.

Web-сервер обрабатывает запросы при помощи сервера, созданного при помощи фреймворка Express. В случае необходимости сервер отправляет запросы к базе данных, находящейся на том компьютере, что и непосредственно сам сервер и находящейся под управлением СУБД SQLServer.

WS сервер обмениться с клиентом сообщениям по протоколу WebSocket.

Для обеспечения безопасного взаимодействия между клиентом и сервером используется протокол HTTPS, который обеспечивает шифрование данных и аутентификацию с помощью сертификата. Это позволяет защитить передаваемую информацию от несанкционированного доступа.

Взаимодействие между сервером и базой данных осуществляется посредством протокола TCP, который обеспечивает надежную передачу данных между ними. Это позволяет серверу выполнять операции чтения и записи данных в базу данных.

2.3 Проектирование базы данных

Проектирование баз данных – это процесс создания структуры и организации базы данных, которая будет эффективно хранить и управлять данными, соответствуя требованиям и потребностям конкретной системы или приложения. Оно включает в себя определение сущностей и их атрибутов, определение связей между сущностями, оптимизацию структуры для обеспечения эффективного доступа к данным и поддержку необходимых операций.

Проектирование базы данных проводится в два основных этапа: концептуальное и логическое проектирования.

В рамках данного web-приложения ограничимся основными таблицами, необходимыми для просматривания абонементов, добавления абонементов, редактирования абонементов, удаления абонементов, создания тренеров, создания комментариев к тренерам. Диаграмма базы данных приведена на рисунке 2.3.

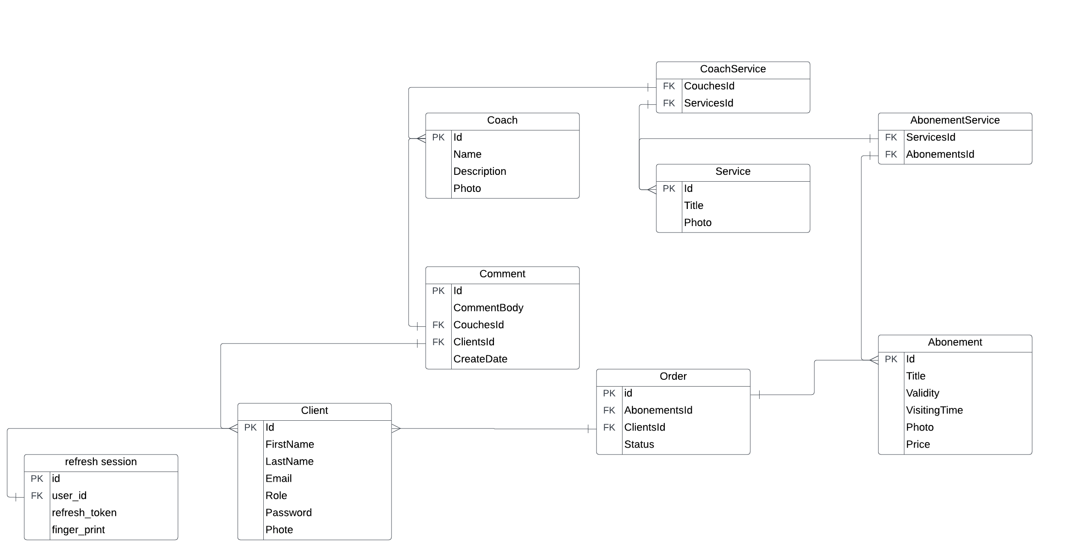


Рисунок 2.3 – Логическая схема базы данных

Теперь необходимо описать таблицы базы данных, а именно их названия, типы данных, описание.

Первой к рассмотрению подлежит такая таблицу, как Client. В этой таблице содержится вся необходимая информация для функционирования такой самостоятельной сущности, как пользователь или же клиент приложения фитнес-центра. Подробное описание приведено в таблице 2.1

Таблица 2.1 – Содержание таблицы Client (Пользователь)

|  |  |  |
| --- | --- | --- |
| Название | Тип данных | Описание |
| Id | uniqueidentifier | Идентификатор пользователя, является первичным ключом |
| FirstName | nvarchar(MAX) | Имя пользователя |
| LastName | nvarchar(MAX) | Фамилия пользователя |
| Email | nvarchar(MAX) | Мыло пользователя |
| Password | nvarchar(MAX) | Пароль пользователя |

Продолжение таблицы 2.1

|  |  |  |
| --- | --- | --- |
| Photo | INT | Путь до фото пользователя |

Таблица refresh\_sessions (Refresh сессия) нужна для безопасного хранения данных необходимых для доступа к защищенным ресурсам. Детальная информация о ней приведена в таблице 2.2.

Таблица 2.2 – Содержание таблицы refresh\_sessions (Refresh сессия)

|  |  |  |
| --- | --- | --- |
| Название | Тип данных | Описание |
| id | uniqueidentifier | Идентификатор сессии, является первичным ключом |
| user\_id | uniqueidentifier | Идентификатор пользователя |
| refresh\_token | varchar(400) | Refresh токен |
| finger\_print | varchar(32) | Отпечаток браузера |

Таблица Abonement содержит всю информацию об абонементе: название, период валидности, время посещения, фото, цену. Подробная структура таблицы представлена в таблице 2.3.

Таблица 2.3 – Содержание таблицы Abonement (Абонемент)

|  |  |  |
| --- | --- | --- |
| Название | Тип данных | Описание |
| Id | uniqueidentifier | Идентификатор абонемента, является первичным ключом |
| Title | nvarchar(MAX) | Название абонемента |
| Validity | nvarchar(MAX) | Период валидности абонемента |
| VisitingTime | nvarchar(MAX) | Время посещения |
| Photo | nvarchar(MAX) | Фото |
| Price | INT | Цена |

Для работы с комментариями была создана таблица Comment. Данная таблица содержит всю основную информацию о комментарии. Структура данной таблицы содержится в таблице 2.4.

Таблица 2.4 – Содержание таблицы Comment (Комментарий)

|  |  |  |
| --- | --- | --- |
| Название | Тип данных | Описание |
| Id | uniqueidentifier | Идентификатор комментария, является первичным ключом |
| CommentBody | nvarchar(MAX) | Тело |

Продолжение таблицы 2.4

|  |  |  |
| --- | --- | --- |
| ClientsId | uniqueidentifier | Идентификатор клиента, является внешним ключом |
| CouchesId | uniqueidentifier | Идентификатор тренера, является внешним ключом |
| CreateDate | datetime | Дата создания |

Для тренеров была создана таблица Coach. Текущая структура таблица представлена в таблице 2.5.

Таблица 2.5 – Содержание таблицы Coach (Тренер)

|  |  |  |
| --- | --- | --- |
| Название | Тип данных | Описание |
| Id | uniqueidentifier | Идентификатор тренера, является первичным ключом |
| Name | nvarchar(MAX) | Имя тренера |
| Description | nvarchar(MAX) | Описание тренера |
| Photo | nvarchar(MAX) | Фото |
| Pomodoro\_session\_id | INT | Идентификатор сессии, является внешним ключом |

Для работы с сервисами была создана таблица Service. Данная таблица содержит всю основную информацию о услуге. Текущая структура таблица представлена в таблице 2.6.

Таблица 2.6 – Содержание таблицы Service (Услуга)

|  |  |  |
| --- | --- | --- |
| Название | Тип данных | Описание |
| Id | uniqueidentifier | Идентификатор услуги, является первичным ключом |
| Title | nvarchar(MAX) | Название услуги |
| Phote | nvarchar(MAX) | Фотоуслуги |

Для работы с заказами была создана таблица Order. Данная таблица содержит всю основную информацию о заказе: статус заказа и идентификаторы клиента, абонемента и услуги. Текущая структура таблицы представлена в таблице 2.7.

Таблица 2.7 – Содержание таблицы Order (Заказ)

|  |  |  |
| --- | --- | --- |
| Название | Тип данных | Описание |
| Id | uniqueidentifier | Идентификатор услуги, является первичным ключом |

Продолжение таблицы 2.7

|  |  |  |
| --- | --- | --- |
| AbonementsId | uniqueidentifier | Идентификатор абонемента, является внешним ключом |
| ClientsId | uniqueidentifier | Идентификатор клиента, является внешним ключом |
| Status | int | Статус заказа |

Каждая таблица служит для выполнения определенной цели, обеспечивая целостное хранение данных в приложении.

2.4 Вывод

Проектирование веб-приложения – это ключевой этап, который позволяет создать функциональное и эффективное приложение. Оно обеспечивает понимание внутренней структуры и взаимодействия компонентов, что важно для разработчиков. Правильное проектирование позволяет создавать веб-приложения, способные масштабироваться и обрабатывать растущую нагрузку.

Что касается фреймворка ExpressJs, его трехуровневая архитектура предоставляет удобный способ организации кода.

Построение UML-диаграмм (например, диаграммы Use-case и диаграммы развертывания) помогает визуализировать структуру приложения и взаимодействие между его компонентами.

Концептуальное и логическое проектирование баз данных также важно. Оно позволяет определить сущности, их атрибуты и связи между ними. Это облегчает создание физической модели базы данных и разработку запросов.

Нужно заполнить эту страницу ещё

3 Разработка web-приложения

3.1 Разработка серверной части

В разработке серверной части веб-приложения был выбран подход, начинающийся с написания таблиц и их связей в базе данных SQLServer и последующего преобразования таблиц в сущности-объекты при помощи ORM (Object-Relational Mapping) Prisma. Данный подход называется DataBase first. По мере роста проекта вносились изменения в сущности таблиц, которые переносились в саму базу данных при помощи миграций Prisma.

Одной из базовых проверок корректности написания моделей и их связи с базой данных является реализация CRUD-операций, которые обеспечивают создание (Create), чтение (Read), обновление (Update) и удаление (Delete) данных. Примером является удаление абонемента, которое представлено на листинге 3.1.

static async deleteAbonement(abonementId) {

const deletedAbonement = await prismaClient.abonement.delete({

where: {

Id: abonementId.toUpperCase()

}

})

return deletedAbonement;

}

Листинг 3.1 − Удаление абонемента

Помимо обычных CRUD-операций, для полноценной работы с базой данных часто требуются более специфические запросы, которые включают различные фильтры, сортировки и агрегацию данных из нескольких таблиц. Пример более сложного запроса представлен на листинге 3.2.

|  |
| --- |
| static async getOrdersByUser(clientId){  const userOrders = await prismaClient.order.findMany({  where:{  ClientsId: clientId  },  include: {  Abonement: {  include: {  AbonementService:{  include: {  Service: true  }  }  }  }  }  }) |

Продолжение листинга 3.2

|  |
| --- |
| return userOrders;  } |

Листинг 3.2 − Получение информации о заказах пользователя

После разработки моделей и связей, процесс разработки серверной части веб-приложения продолжается с созданием контроллеров. Одной из основных задач контроллеров является формирование корректного роутинга для эндпоинтов и проверка связи с бизнес-логикой приложения. На листинге 3.3 приведен пример контроллера.

|  |
| --- |
| router.post("/orders", TokenService.checkAccess, ResourcesValida-tor.postOrders, ResourcesController.createOrder);  static async createOrder(req, res){  try{  const user = await AuthService.getUserByToken(req);  const abonement = await AbonnementReposito-ry.getAbonnementById(req.body.abonementId);  if (!abonement) {  return res.status(404).json({ message: 'Abonement not exists' });  }  const orderId = uuidv4();  const abonementId = req.body.abonementId;  const clientId = user.id;  const order = await OrderRepository.createOrder({orderId, abonementId, clientId})  res.status(200).json(order)  }catch (e){  res.status(400).json({message: 'Order can\'t be create'})  } |

Листинг 3.3 − Метод, создающий access и refresh токены

На данном листинге можно наблюдать часть эндпоинтов, которые обрабатываются контроллером. У эндпоинтов указываются HTTP методы и URI, на которые они будут реагировать. Для защиты приложения от несанкционированного входа на защищенные части был использован JSON Web Token (JWT), который позволяет создать токены для обеспечения безопасной аутентификации и авторизации пользователей. JSON Web Token (JWT) является компактным и самодостаточным способом обмена информацией между сторонами в формате JSON. На листинге 3.4 указаны методы создания токенов.

|  |
| --- |
| static async generateAccessToken(payload) {  return await jwt.sign(payload, process.env.ACCESS\_TOKEN\_SECRET, { |

Продолжение листинга 3.4

|  |
| --- |
| static async checkAccess(req, \_, next) {  const authHeader = req.headers.authorization;  const token = authHeader?.split(" ")?.[1];  if (!token) {  return next(new Unauthorized());  }  try{  req.user = await TokenService.verifyAccessToken(token);  }catch (error){  console.log('error: '+JSON.stringify(error, null, 2));  return next(new Forbidden(error));  }  next();  } |

Листинг 3.4 − Метод, создающий access и refresh токены

Secret для JWT извлекается из .env путем использования process.env, который работает благодаря dotenv. Для проверки доступа к защищенным роутам был добавлен метод-middleware checkAccess, представленный на листинге 3.5.

|  |
| --- |
| static async checkAccess(req, \_, next) {  const authHeader = req.headers.authorization;  const token = authHeader?.split(" ")?.[1];  if (!token) {  return next(new Unauthorized());  }  try{  req.user = await TokenService.verifyAccessToken(token);  }catch (error){  console.log('error: '+JSON.stringify(error, null, 2));  return next(new Forbidden(error));  } |

Продолжение листинга 3.5

|  |
| --- |
| next();  } |

Рисунок 3.5 − Реализация класса checkAccess

WebSocket – это протокол связи между клиентом и сервером, который обеспечивает двустороннюю связь и позволяет установить постоянное соединение между клиентом и сервером. Их используют для реализации взаимодействия в режиме реального времени между клиентскими и серверными приложениями. В отличие от HTTP, который является протоколом запроса-ответа, WebSocket поддерживает постоянное соединение, что позволяет серверу передавать данные клиенту без необходимости повторных запросов. Часть реализации Websocket, которая используется в курсовом проекте представлена в листинге 3.6.

|  |
| --- |
| io.on('connection', (socket) => {  console.log(User connected:' ${socket.id});  const clientId = socket.user.id;  clientsAbonnements.forEach(item => {  if (item.clientId === clientId) {  item.socketId = socket.id;  }  })  if (!intervalId) {  console.log('setInterval')  intervalId = setInterval(checkAbonnements, 1000 \* 5);  }  socket.on('disconnect', () => {  console.log(User disconnected: ${socket.id});  });  socket.on('startTimer',  validateParams(socket, async (data) => {  try {  const {orderId, abonementTitle} = data;  const orderWithAbonement = await prismaCli-ent.order.findUnique({  where: {  Id: orderId  },  include:{  Abonement: true  }  })  let abonementValidity; |

Продолжение листинга 3.6

|  |
| --- |
| if(orderWithAbonement.Abonement.Title === abonementTi-tle){  abonementValidity = orderWithAbo-nement.Abonement.Validity;  clientsAbonnements.push({  socketId: socket.id,  orderId: orderId,  clientId: clientId,  abonementTitle: abonementTitle,  expirationDate: new Date(Date.now() + abo-nementValidity \* expirationDateConst)  })  }  else{  socket.emit('error', 'no such abonement')  }  } catch (err) {  console.log('Socket err: ' + JSON.stringify(err, null, 2))  socket.emit('error', err)  }  }))  }) |

Листинг 3.6 − Реализации слушателя SocketIO

SocketIO предоставляет удобный и интуитивно понятный способ работы с WebSocket. Он предоставляет слушатели событий и методы для создания WebSocket-серверов, для управления соединениями и передачи данных. Это делает разработку приложений, использующих SocketIO, быстрой и эффективной.

Для взаимодействия с БД был разработан PrismaClient. Реализация этого скрипта представлена в листинге 3.7.

|  |
| --- |
| import { PrismaClient } from '@prisma/client';  const prismaClient = new PrismaClient();  export default prismaClient; |

Листинг 3.7 − Реализация скрипта PrismaClient

Также, в современном мире веб-разработки одной из важнейших задач является защита пользовательских данных и обеспечение конфиденциальности обмена информацией между клиентом и сервером.

Тут ещё можно, чтобы страничку до конца заполнить

3.2 Разработка клиентской части

При разработке клиентской части веб-приложения было принято решение использовать библиотеку React, которая позволяет легко создавать многокомпонентные слабосвязанные пользовательские интерфейсы. В контексте реализации роутов для различных страниц на клиенте применялась библиотке react router dom. Также в проекте используется библиотека redux для хранения глобальных состояний. Помимо всего этого использовалась библиотека materialUi для создания узнаваеиого пользовательского опыта. Примеры использования react router dom c защитой роутов для неавторизованных пользователей представлен в приложении А.

Для взаимодействия с сервером используется библиотека axios. Axios – это популярная библиотека JavaScript для выполнения HTTP-запросов, которая предоставляет удобный API для отправки запросов и обработки ответов. В проекте настроены специальные перехватчики запросов, которые обеспечивают requests необходимыми заголовками для доступа к защищенным ресурсам сервера. Использование представлено в листинге 3.8.

|  |
| --- |
| export const Resource = axios.create({  baseURL: ${config.API\_URL}/resources,  withCredentials: true,  })  Resource.get('/ordersByUser')  .then(response => {  console.log('get.ordersByUser response.data: '+JSON.stringify(response.data, null, 2))  setOrders(response.data)  })  .catch(error => {  console.error('Failed to fetch orders:', error);  }); |

Листинг 3.8 − Экземпляр axios c авторизацией

Для взаимодействия через websocket с сервером на клиентской стороне были разработаны методы, который позволяет подключиться к сокету, прослушивать канал, отправлять и получать сообщения. Данные методы вызываются при покупке абонемента и обновлении страницы

Часть реализации взаимодействия представлена в листинге 3.9.

|  |
| --- |
| const response = await Resource.post('/orders', postOrdersData);  if (response.status === 200) {  showSuccessMessage("Abonement bought successfully");  const createdOrderId = response.data.Id;  const socketStartTimerData = {  orderId: createdOrderId, |

Продолжение листинга 3.9

|  |
| --- |
| abonementTitle: abonne-ment.Title,  abonementValidity: abonne-ment.Validity  if(user.socket){  us-er.socket.emit('startTimer', socketStartTimerData);  else{  const accessToken = inMemor-yJWT.getToken();  let socket = await io(config.API\_WEBSOCKET\_URL, {  reconnection: true,  reconnectionAttempts: 5,  reconnectionDelay: 1000,  extraHeaders: {  Authorization: Bearer ${accessToken} |

Листинг 3.10 − Реализация взаимодействия frontend и backend по websocket

Итого были рассмотрены основные аспекты клиентской части приложения.

**3.3 Вывод**

В целом, эти инструменты и библиотеки обеспечивают разработчикам гибкие возможности для создания интерактивных веб-приложений. Они позволяют создавать сложные пользовательские интерфейсы, обеспечивают эффективное взаимодействие с серверной частью и управление данными.

Дописать)

4 Тестирование web приложения

4.1 Авторизация и регистрация

В приложении есть роль – пользователь. Для пользователя были разработаны следующие страницы: авторизация и регистрация, главная, абонементы, тренера, редактировать профиль. Страницы авторизации и регистрации представлены на рисунке 4.1.

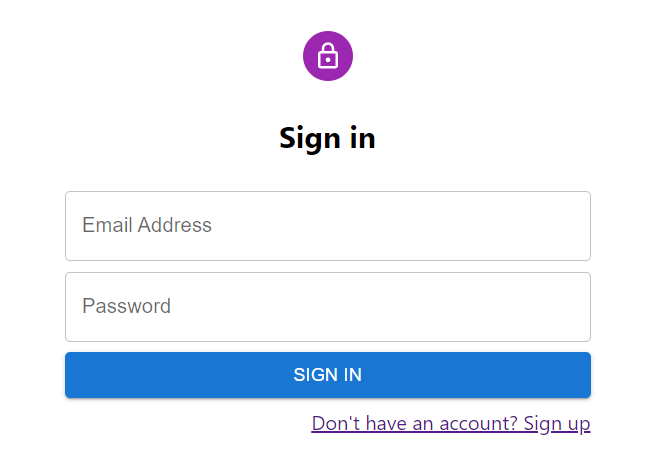
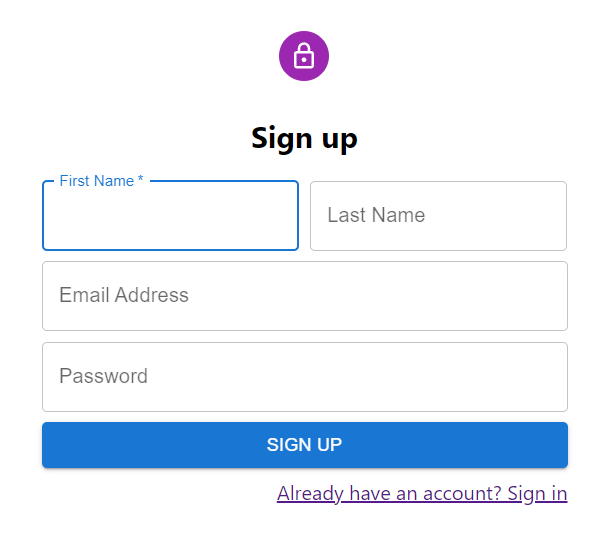
 

Рисунок 4.1 − Страница «Авторизации» и «Регистрации»

Для этой страницы разработана валидация: проверка на уникальность email, и если он не уникальный, то генерируется соответствующее исключение и на экран выводится обработанное сообщение об ошибке.

Попытка доступа к ресурсу, незарегистрированным пользователем приведет к переадресации на страницу со входом/регистрацией. Данное поведение можно наблюдать на рисунке 4.2.

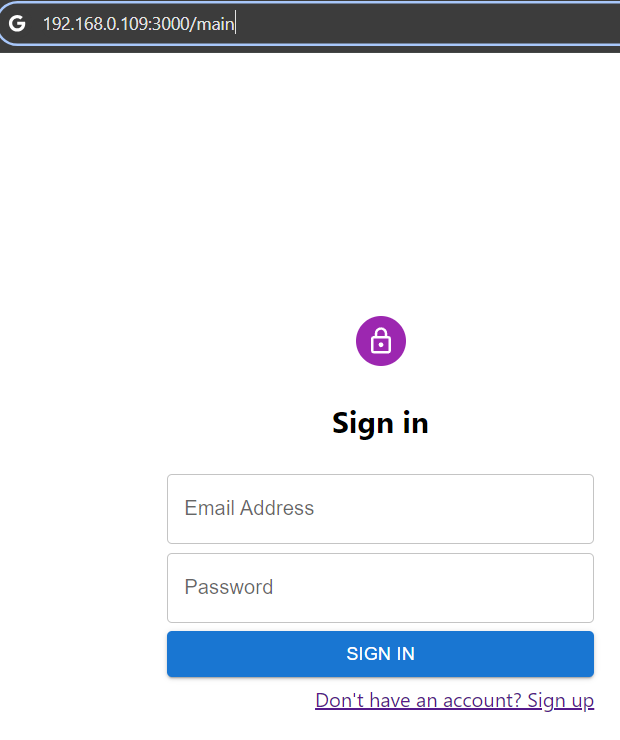


Рисунок 4.2 − Попытка доступа к ресурсу неавторизованным пользователем

Достигнуто это путем создания условий в react router dom запросов и перенаправления пользователей в зависимости от их статуса аутентификации.

4.2 Функции клиентской части

После успешной авторизации появится сразу страница main. На ней находится слайдер, а вверху картинка пользователя и имя и фамилия которые мы ввели при регистрации или логине. Страница main представлена на рисунке 4.3.

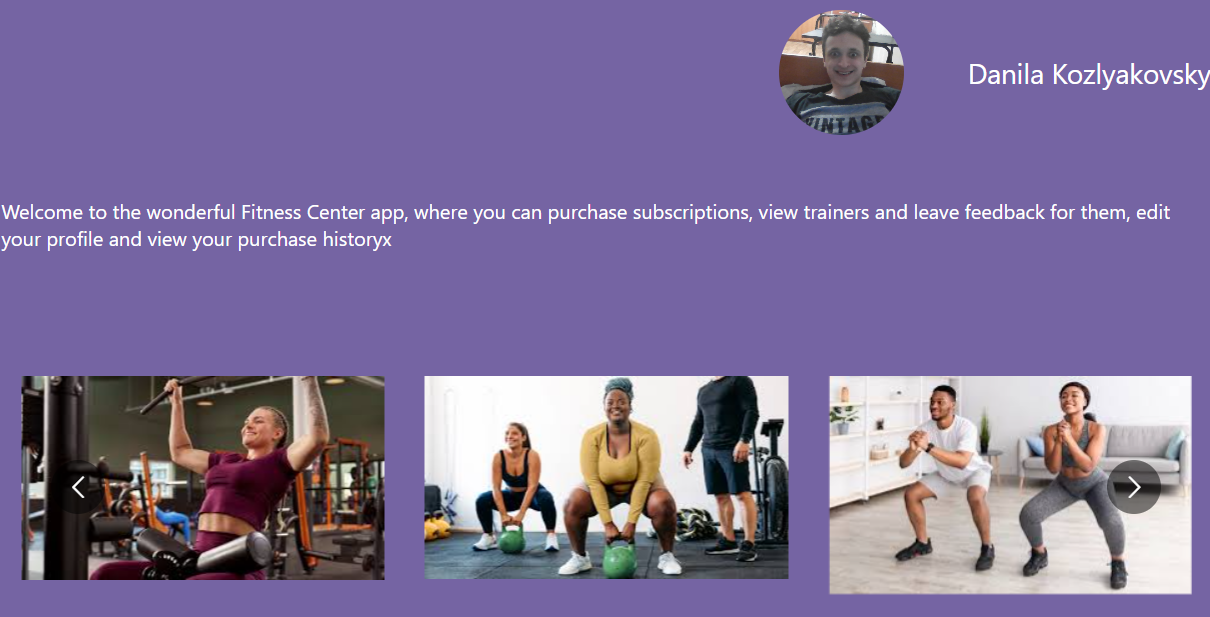


Рисунок 4.3 – Страница «Main»

При нажатии на кнопку Abonements, пользователя переадресует на страницу абонементов, где возможно фильтровать и искать абонементы по цене и названию. Тестирывание представлено на рисунке 4.4.

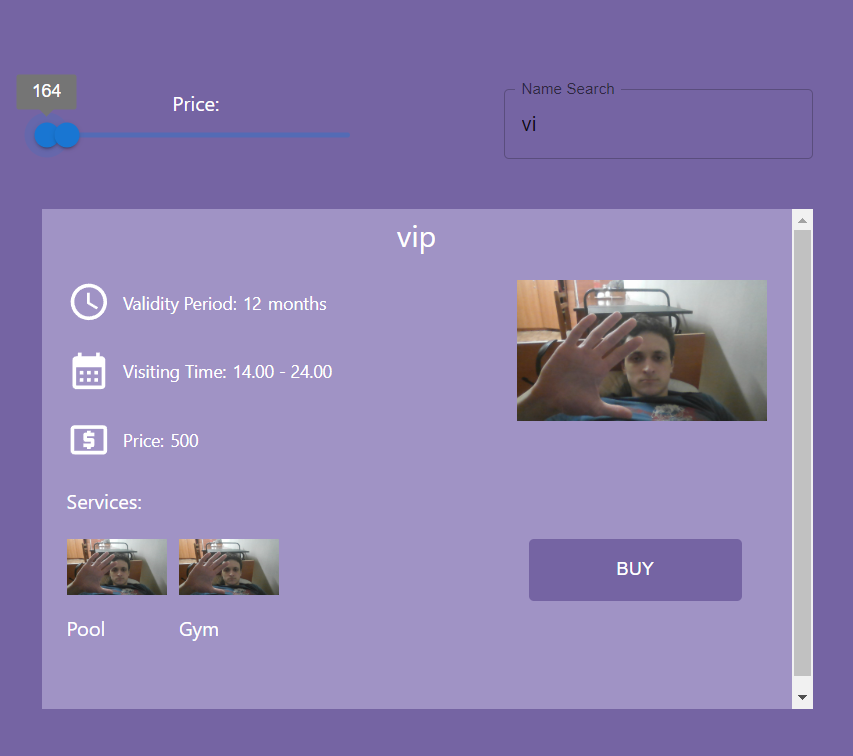


Рисунок 4.4 – Пример поиска абонемента

При нажатии на кнопку купить, абонемент перенесется в историю покупок, которая располагается в профиле пользователя.

По истечению строка действия абонемента в приложение придет оповещение о том, что срок действия абонемента истек, а сам абонемент станет подсвечиваться как не валидным в профиле пользователя, что представлено на рисунок 4.5

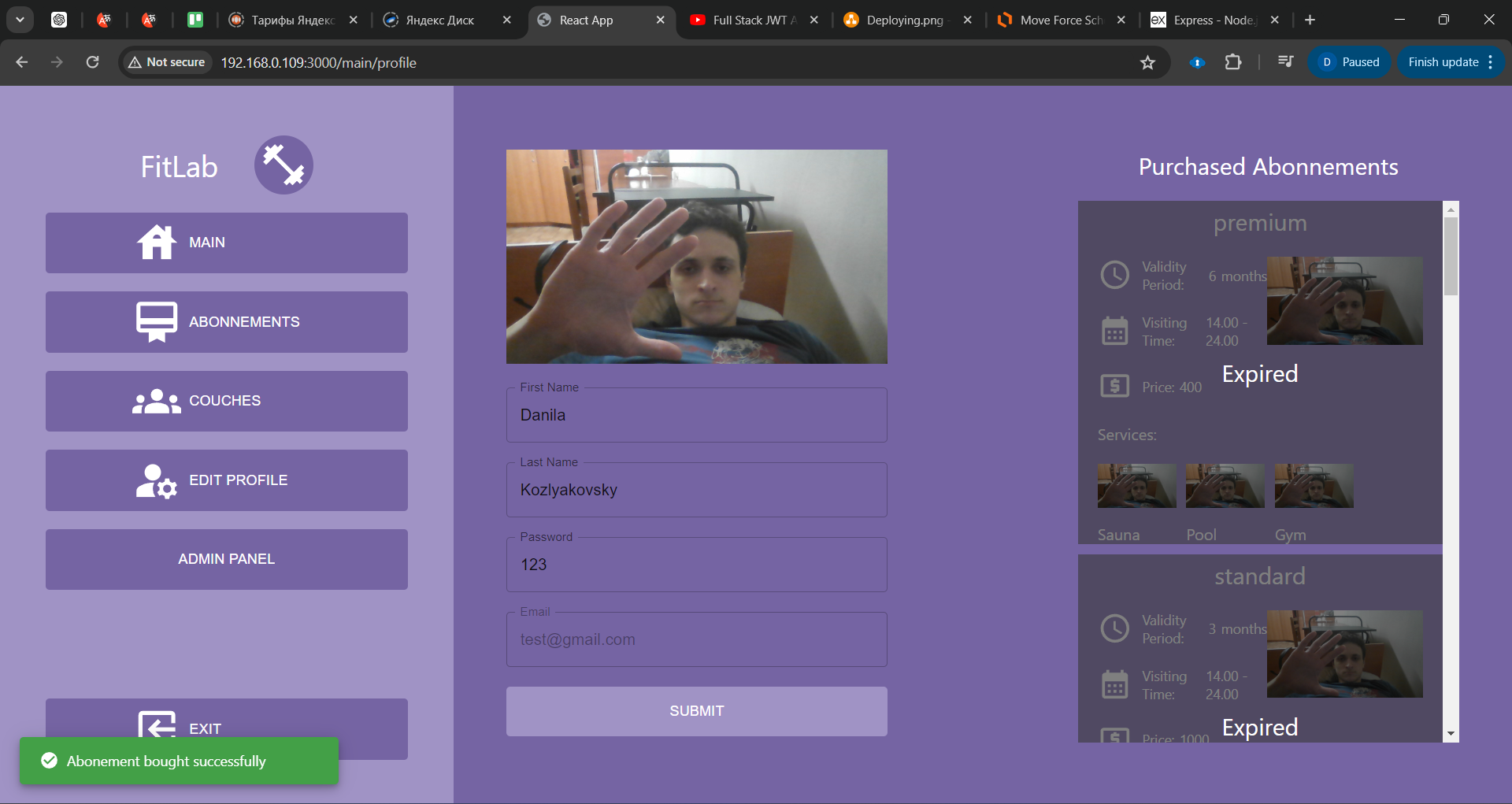


Рисунок 4.4 – Пример истечения строка абонемента

4.3 Вывод

Целью тестирования было проверить функциональность, надежность и производительность приложения, а также выявить и исправить возможные ошибки и проблемы. В результате тестирования не было выявлено ошибок и проблем. Приложение успешно прошло все тесты и продемонстрировало свою функциональность, надежность и производительность.

Тоже дописать чего-нибудь желательно

5 Руководство пользователя

При входе на веб-приложение пользователь попадает на страницу «Вход». Для входа необходимо нажать на кнопку «Вход», если данные верные, то открывается главное окно, если нет, выводит сообщение с указанием ошибки. Если аккаунта нету, то нужно перейти по ссылке на регистрацию. Появится форма регистрации. Страницы «Вход» и «Регистрация» представлены на рисунке 5.1.

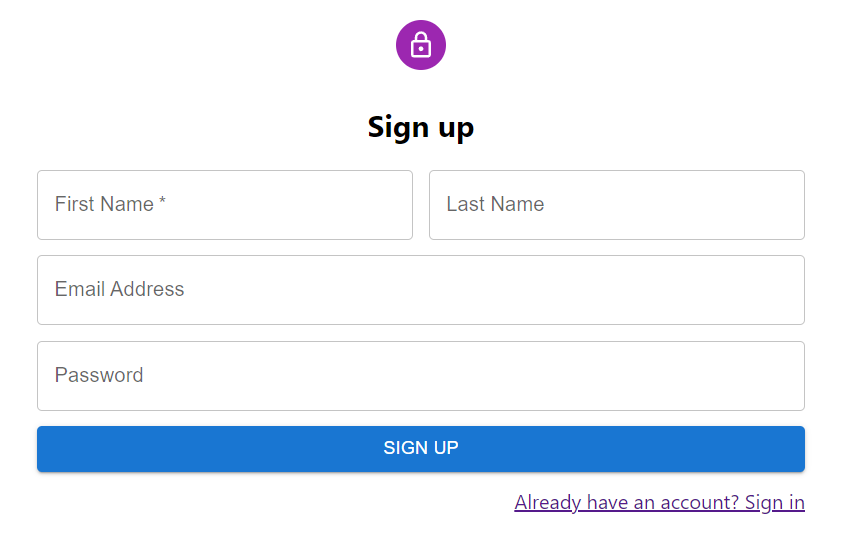
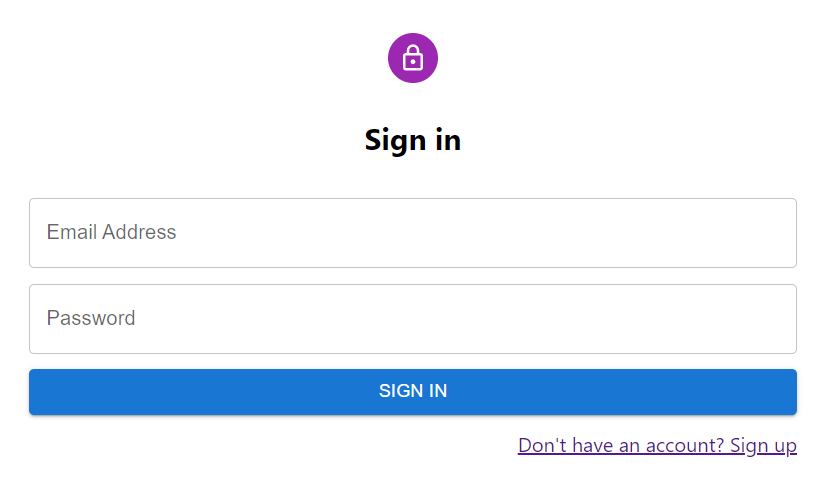


Рисунок 5.1 − Страницы «Вход» и «Регистрация»

Если все условия имя, emal и пароль соответствуют условиям, то происходит регистрация и вход в аккаунт.

После авторизации пользователь сразу попадает на страницу Main. На ней находится картинка, имя клиента и слайдер. Main представлена на рисунке 5.2.

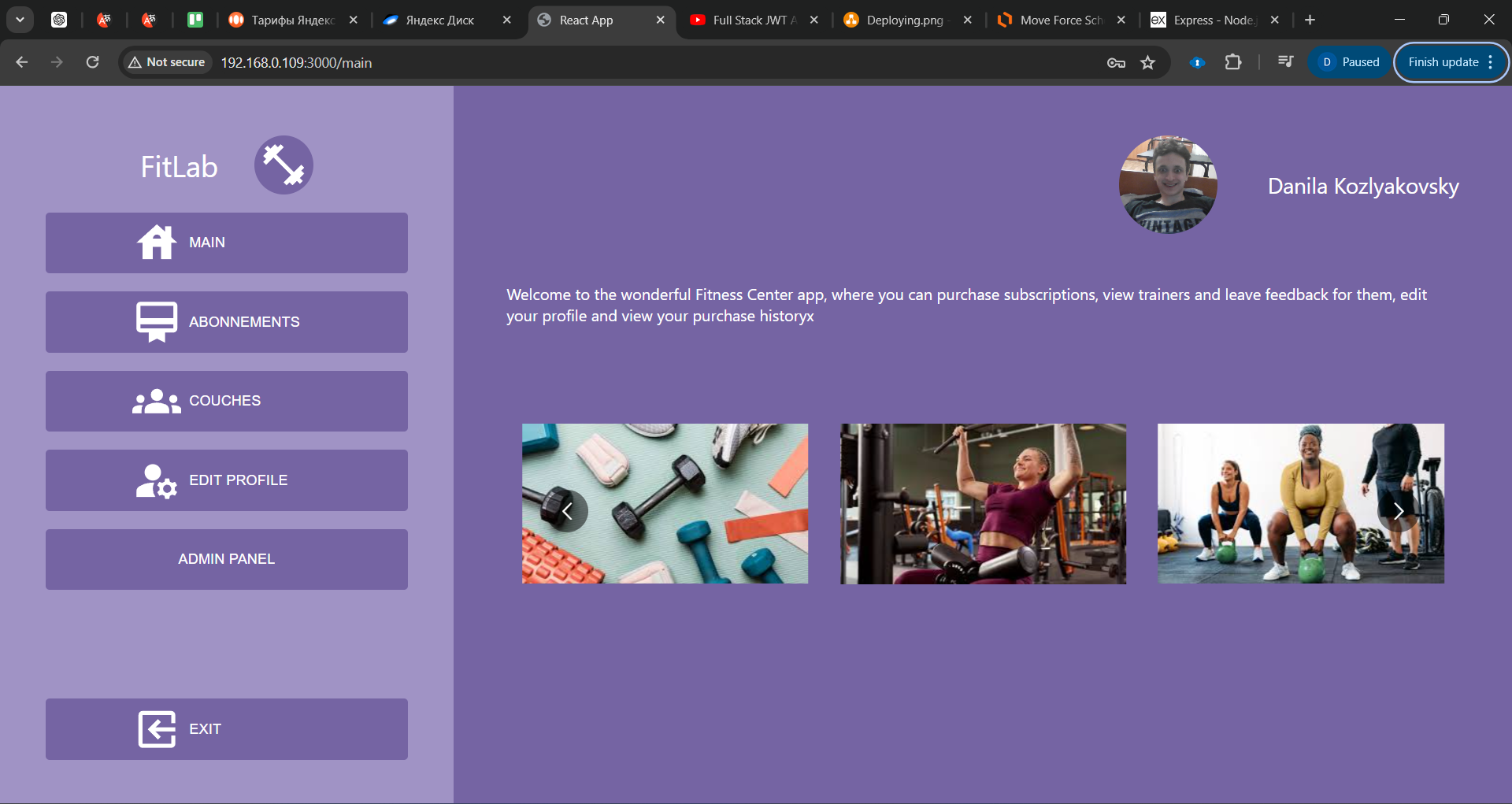


Рисунок 5.2 – Страница Main

Слайдер перелистывается автоматически, но пользователь также может взаимодействовать с ним вручную.

Сбоку на каждой странице располагается меню, которое состоит из ссылок на все страницы данного web-приложения. Ссылка состоит из названия и иконки. В меню представлены следующие разделы: «Main», «Abonnements», «Coaches», «Edit Profile», «Admin Panel» и «Exit».

После страница «Abonnements» на ней пользователь может искать абонементы по цене и названию, а также приобретать их. Страница «Abonnements» представлена на рисунке 5.3.

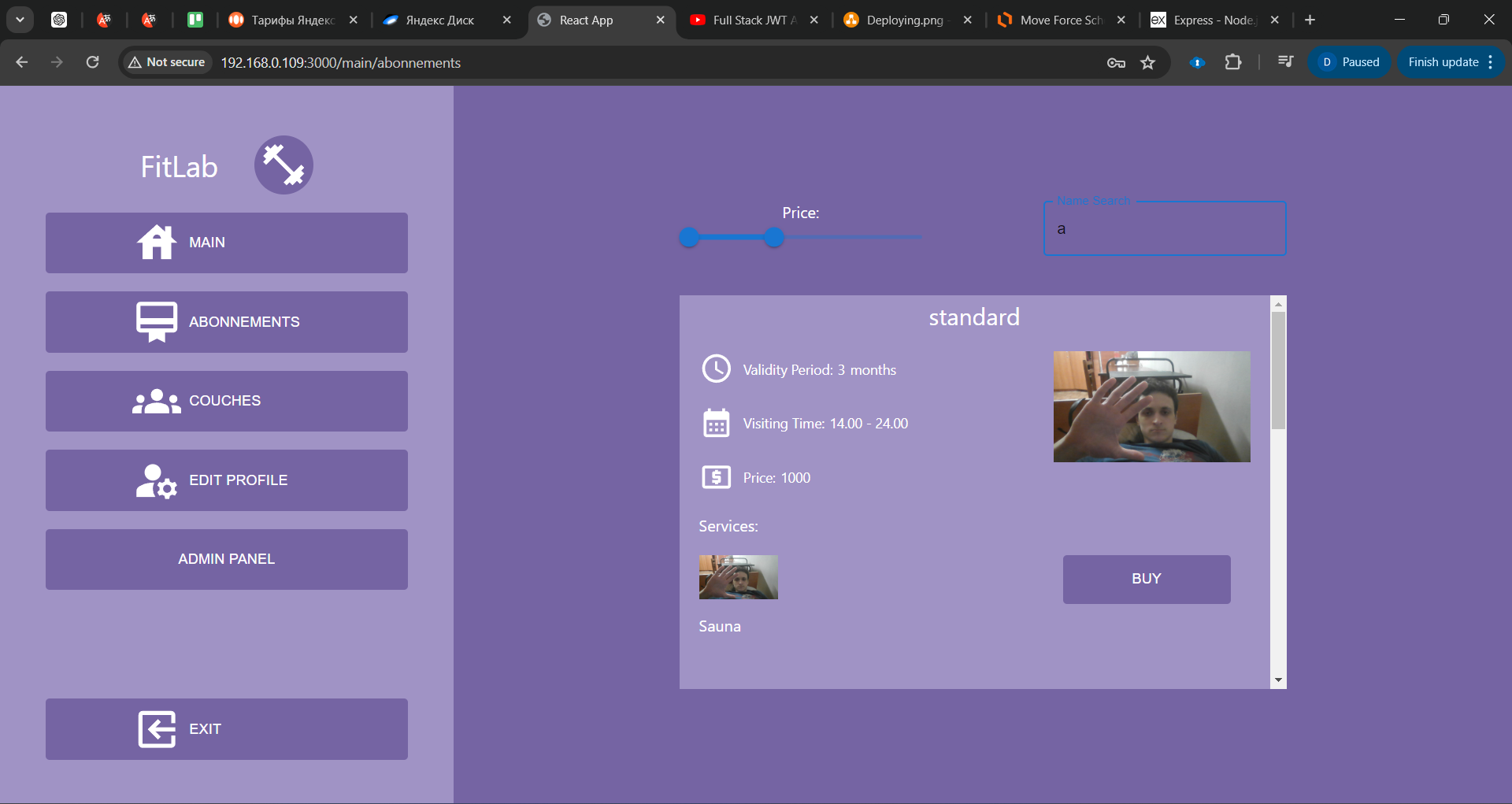


Рисунок 5.3 – Страница «Abonnements»

На странице «Coaches» пользователь может посматривать список тренеров с их услугами и может углубиться в подробности о конкретном тренере. Страница «Coaches» представлена на рисунке 5.4.

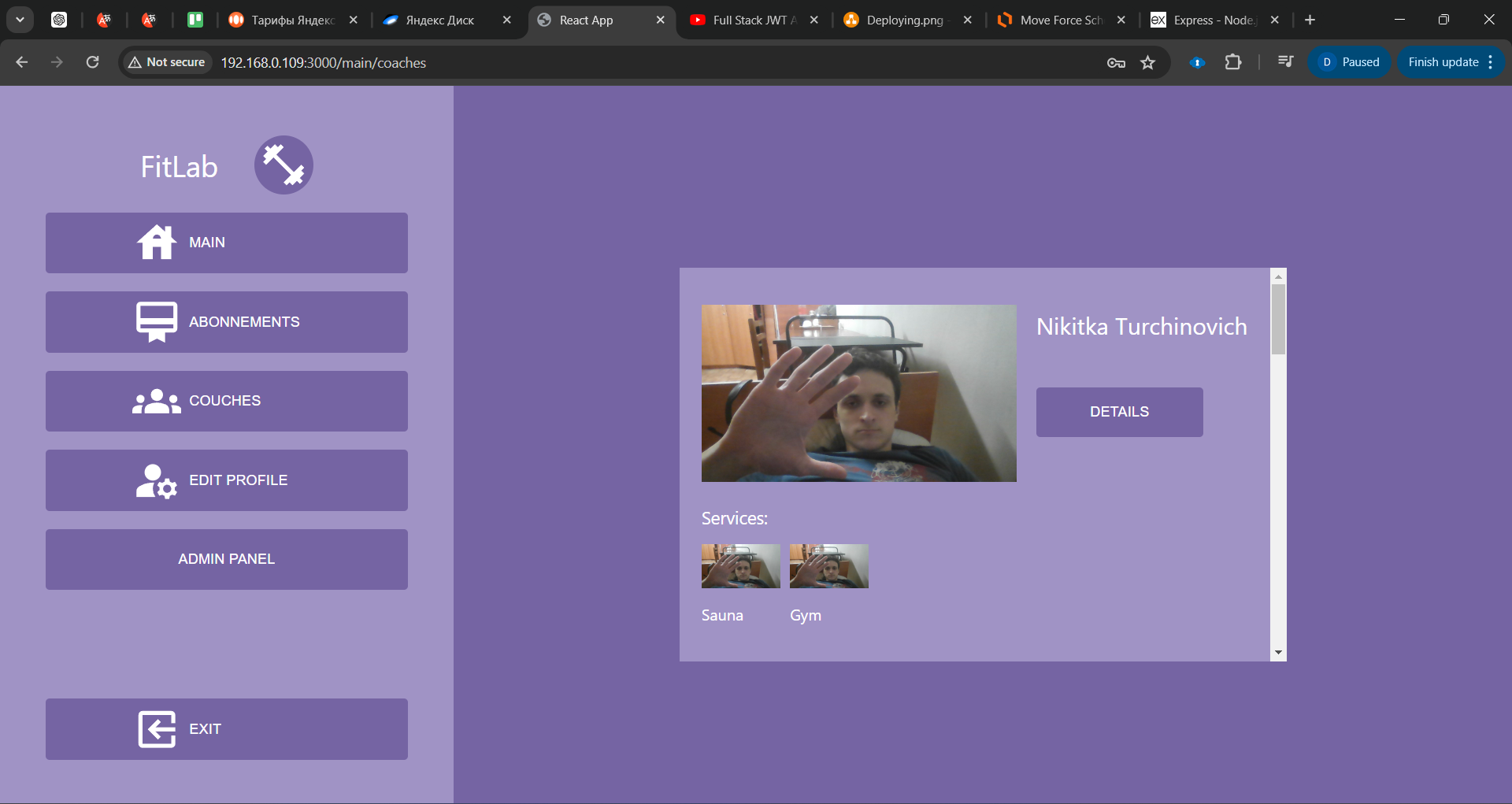


Рисунок 5.4 – Страница «Coaches»

Если углубиться в подробности о конкретном тренере нажав при это кнопку.

То пользователя перекинет на страницу «CoachesDeatails», где располагается более подробная информация о конкретном выбранном тренере. Таже присутствует возможность оставить комментарий к тренеру и посмотреть список всех оставленных комментариев от лица других пользователей. Страница «CoachesDeatails» представлена на рисунке 5.5

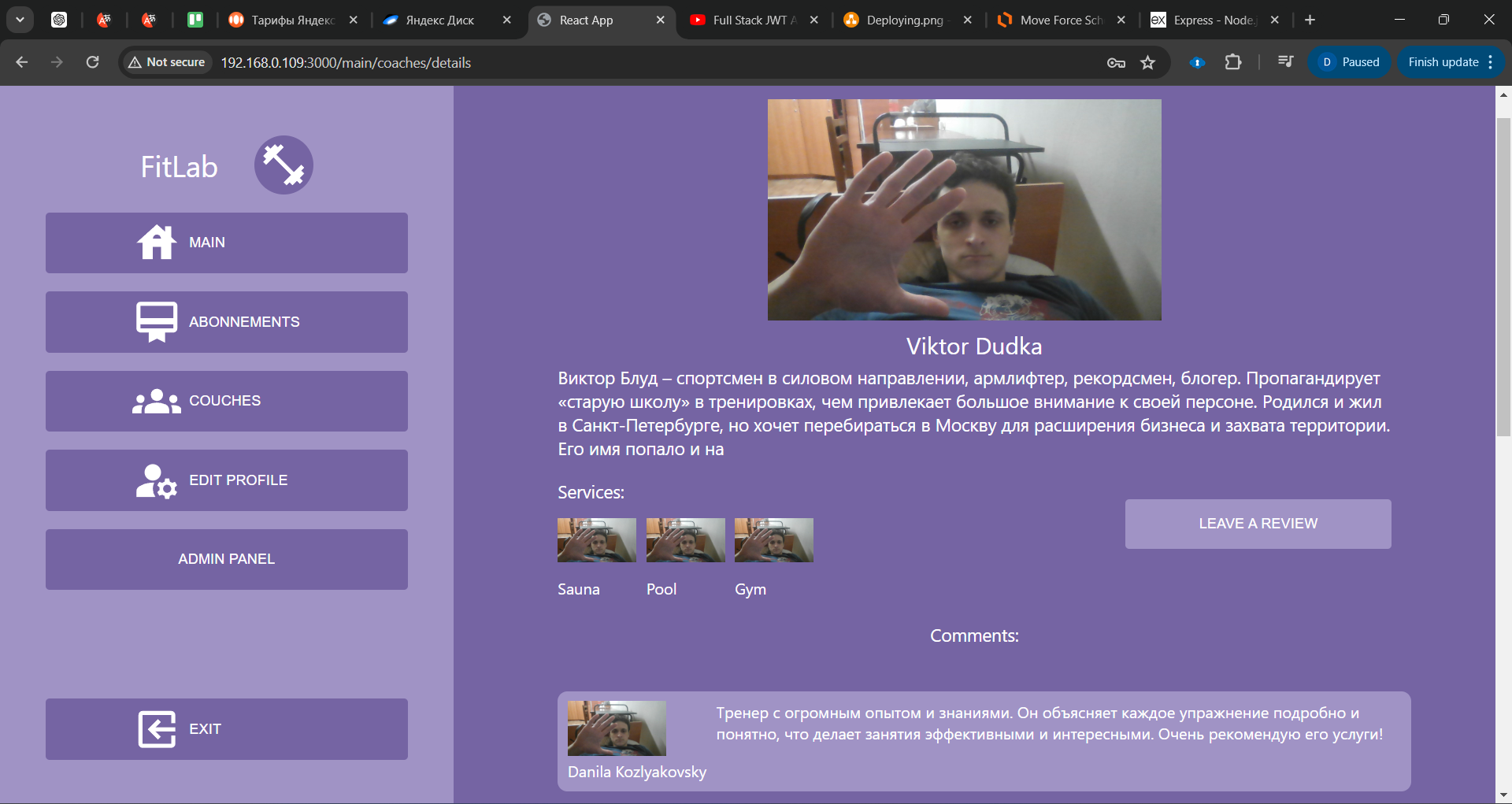


Рисунок 5.5 – Страница «CoachesDeatails»

Страница «Edit Profile» демонстрирует профиль пользователя, под которым клиент вошел в приложение. Присутствуют возможности просмотра таких данных, как имя, фамилия, пароль и почта. Всех их можно обновлять, за исключением поля email. Более того, справа от пользовательской информации располагается история покупок, где клиент может посмотреть свои валидные и истекшие абонементы.

При истечении абонемента пользователь оповещается всплывающим зеленым оповещением. Помимо всего прочего стоит отметить, что любые успешные и неуспешные действия в проекте оповещаются всплывающими подсказками в нижнем левом углу экрана. Страница «Edit Profile» представлена на рисунке 5.6.

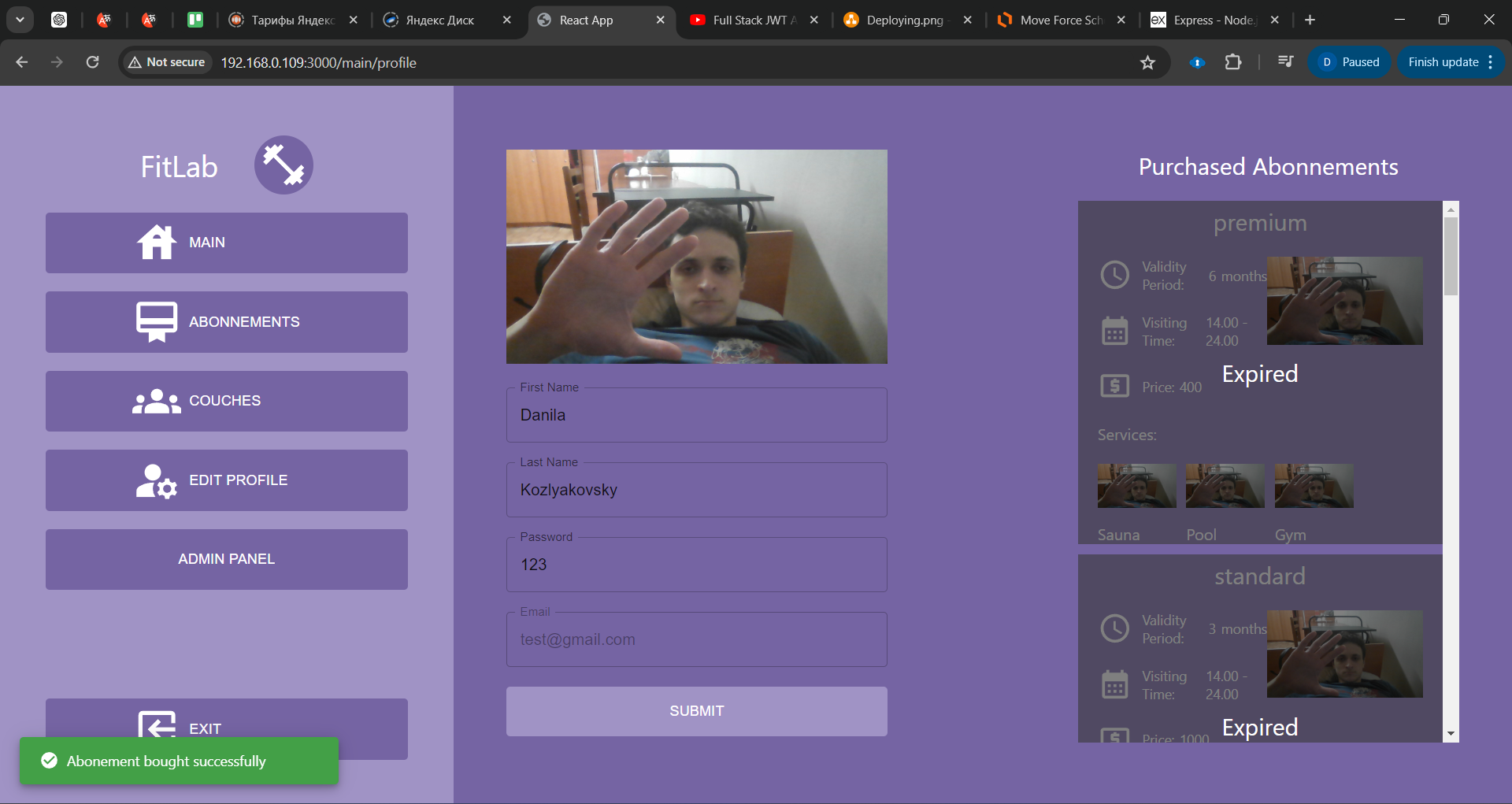


Рисунок 5.6 – Страница «Edit Profile»

Ну и на конец последняя кнопка в навигационной панели – это «Edit Profile», которая разлогинивает пользователя и перенаправляет его на страницу «login», после нажатия на эту кнопку у пользователя не будет возможности вернутся на предыдущие страницы, до повторного входа в аккаунт. Последствия нажатия на кнопку «Exit» представлены на рисунке 5.7.

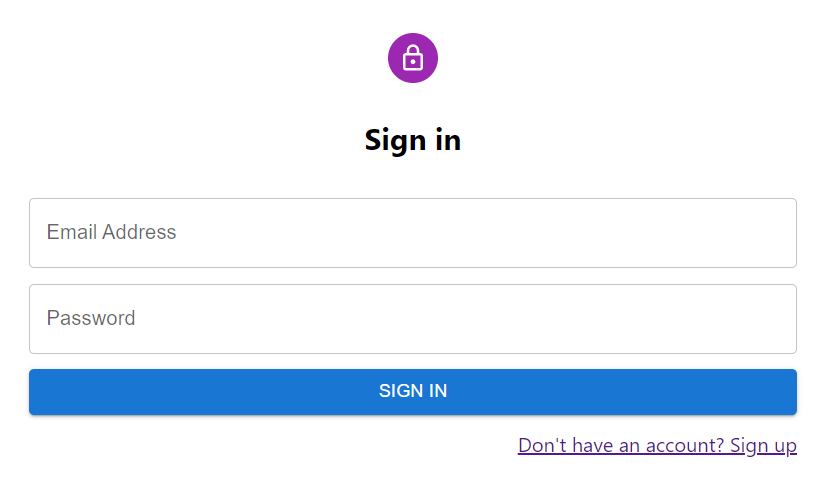


Рисунок 5.7 – Страница «login»

Как итог мы успешно вышли из аккаунта и не имеем доступа к прошлым страницам, которые были подробно описаны в прошлых пунктах данного раздела.

Дописать надо, чтобы было 70% заполнено страницы

**Заключение**

Разработанное веб-приложение на основе Express.js, React.js и SQLServer успешно реализует все ключевые функции, предусмотренные в исходном задании:

– обеспечивать возможность регистрации и авторизации;

– поддерживать роли администратора и пользователя;

– предоставлять подробную информацию о абонементах;

– позволять изменять информацию о абонементах;

– напоминать о истечении строка абонемента;

– позволять просматривать доступные абонементы и осуществлять их покупку;

– предоставлять возможность поиска абонемента по различным критериям;

– позволять администратору управлять информацией о тренерах;

– позволять редактировать профиль пользователя;

В приложении реализован оповещатель, на основе Websocket, что позволяет оповещать клиентов об истечении срока их абонементов.

Также была спроектирована база данных, которая может быть расширена без особых изменений в структуре.

В целом, веб-приложение предоставляет удобный инструмент для фитнец-индустрии. Его стабильная работа и отзывчивость гарантируют удовлетворение пользовательских потребностей.

Дальнейшее развитие проекта может включать в себя расширение функционала, улучшение производительности и внедрение новых возможностей для улучшения опыта пользователей. Тщательное тестирование всех компонентов подтверждает исправную работу приложения и выполнение всех основных требований.

# **Список литературы**

1. METANIT.COM [Электронный ресурс]. – Электронные данные. – Режим доступа: https://metanit.com.
2. Express.js [Электронный ресурс]. – Электронные данные. – Режим доступа: https://expressjs.com.
3. Wikipedia.org [Электронный ресурс]. – Электронные данные. – Режим доступа: https://ru.wikipedia.org/wiki/UML.

4 Enabling HTTPS on express.js [Электронный ресурс]. – Электронные данные. – Режим доступа: https://stackoverflow.com/questions/11744975/enabling-https-on-express-js.

# **Приложение А**

**Пример использования react router dom**

|  |
| --- |
| export function AppRouter() {  const {isUserLogged} = useContext(AuthContext);  return (  <Routes>  {!isUserLogged ? (  <>  <Route  path='/signin'  element={<SignIn/>}  />  <Route  path='/signup'  element={<SignUp/>}  />  </>  ) : (  <>  <Route  path='/main'  element={<MainNavHome/>}  />  <Route  path='/main/abonnements'  element={<MainNavAbonnements/>}  />  <Route  path='/main/profile'  element={<MainNavProfile/>}  />  <Route  path='/main/coaches'  element={<MainNavCoaches/>}  />  <Route  path='/main/coaches/details'  element={<MainNavCoachDetailCard/>}  />  <Route  path='/adminPanel'  element={<MainNavAdminPanel/>}  />  </>  )}  <Route path="\*" element={<Navigate to={isUserLogged ? "/main" : "/signin"}/>}></Route>  </Routes>  );  } |