МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных Технологий

Кафедра Информационных систем и технологий

Специальность 1-40 01 01 «Программное обеспечение информационных технологий»

Специализация 1-40 01 01 10 «Программное обеспечение информационных технологий (программирование интернет-приложений)»

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**к курсовому проекту на тему:**

Web-приложение «Запись на тренировки в конно-спортивный клуб»

Выполнила студентка Ожередова Полина Андреевна

(Ф.И.О.)

Руководитель проекта асс. Дубовик М.В.

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

Заведующий кафедрой к.т.н., доц. Смелов В.В.

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

Консультанты асс. Дубовик М.В.

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

Нормоконтролер асс. Дубовик М.В.

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

Курсовой проект защищен с оценкой

Минск 2021

Содержание

[Введение 3](#_Toc91533634)

[1 Постановка задачи 4](#_Toc91533635)

[2 Проектирование программного средства 6](#_Toc91533636)

[2.1 Анализ требований к программному средству и разработка функциональных требований 6](#_Toc91533637)

[2.2 Общая структура программного средства 7](#_Toc91533638)

[2.3 Cтруктура базы данных 7](#_Toc91533639)

[3 Разработка программного средства 12](#_Toc91533640)

[3.1 Реализация сервера 12](#_Toc91533641)

[3.2. Работа с базой данных 15](#_Toc91533642)

[3.3. Проектирование структуры клиентской части 17](#_Toc91533643)

[4 Тестирование программного средства 20](#_Toc91533644)

[5 Руководство пользователя 24](#_Toc91533645)

[5.1. Руководство для неавторизованного пользователя 24](#_Toc91533646)

[5.2. Руководство для аутентифицированного пользователя 24](#_Toc91533647)

[5.3. Руководство для тренера 25](#_Toc91533648)

[5.4. Руководство для администратора 26](#_Toc91533649)

[Заключение 31](#_Toc91533650)

[Список используемых источников 32](#_Toc91533651)

[ПРИЛОЖЕНИЕ А 33](#_Toc91533652)

[ПРИЛОЖЕНИЕ Б 35](#_Toc91533653)

[ПРИЛОЖЕНИЕ В 37](#_Toc91533654)

[ПРИЛОЖЕНИЕ Г 39](#_Toc91533655)

Введение

В век технологий очень удобно использовать интернет во многих сферах жизнедеятельности: покупка вещей и продуктов, оформление заказа, запись на услуги в различные центры, поиск информации и многое другое.

В современном мире нам постоянно не хватает свободного времени, онлайн-запись позволяет его экономить. По статистике, с развитием различных интернет сервисов около 70% людей предпочитают не звонить, а записываться онлайн, а среди мужчин эта цифра достигает 90%. Если компании и предприятия не используют онлайн-запись, то они попросту пренебрегают огромным количеством потенциальных клиентов, а так же не дают выбора существующим.

Онлайн-запись нужна для удобства, потому что потенциальные и существующие клиенты смогут, не отрываясь от работы и дел, записаться на услугу. Так же это удобно, потому что интернет доступен 24 часа в сутки, что добавляет возможность записываться не только в дневное время, но и в ночное, в не зависимости от рабочего дня компании.

Такие услуги как тренировки по конному спорту – это услуги, которые проводятся чаще всего за чертой города, и, чтобы записаться на тренировку, неудобно и не выгодно ездить так далеко и долго. Запись по телефону – тоже не всегда удобна сейчас. Телефон для записи может быть занят а то время, когда можете вы, а потом у вас не получается выделить минутку и клуб теряет потенциального клиента. Онлайн запись на тренировки предоставляет вам возможность записаться в то время, когда удобно именно вам. Есть время, чтобы подумать на счёт времени и даты, никто вас не торопит. Вы можете самостоятельно изучить всю необходимую информацию на сайте и спокойно оформить запись на тренировку.

Запись онлайн организована как автоматизированная система, которая работает сама по себе и не зависит от окружающих факторов. Ее использование повышает лояльность клиентов, что способствует большей посещаемости и популярности. Это очень удобная система, т.к. все необходимые данные о клиентах находятся в одном месте. Приложение для записи удобно как клиентам, так и тренерам, так и администратору. Таким образом целью курсового проекта является создание веб-приложения для записи на тренировки в конно-спортивный клуб, основными задачами которого являются:

* провести аналитический обзор литературы;
* разработать модель базы данных;
* разработать приложение с использованием многоуровневой архитектуры;
* провести тестирование;
* написать руководство пользователя.

**1** **Постановка задачи**

Moby Dick – приложение, имеющее функционал для записи на тренировки в тренажерный зал. Данное приложение позволяет записаться клиентам в тренажерный зал. Пользователь имеет возможность просмотреть подробную информацию о зале, ознакомиться с новостями и фотографиями зала. Можно просматривать профили тренеров и актуальные цены на тренировки. Есть возможность заходить на каждый профиль тренера, читать информацию непосредственно о нем и записаться именно к нему на тренировку, здесь так же есть цена тренировки у определенного тренера. Есть возможность позвонить для уточнения подробной информации либо же для записи через телефон. На странице просмотра информации о зале есть карта Google, с помощью которой можно посмотреть точное местонахождение зала и проложить маршрут к нему на разных видах транспорта. В разделе контакты можно просмотреть контактную информацию, чтобы можно было связаться с администратором зала, уточнить у него вопросы, которые у вас появились во время просмотра информации о зале, и записаться на тренировку. Существует страница с отзывами о зале, где можно почитать эмоции и мнения других посетителей. К преимуществам данного приложения можно отнести минималистичный и интуитивно понятный дизайн, быстрый отклик графического интерфейса и возможность входа через социальные сети. Графический интерфейс данного приложения представлен на рисунке 1.1.

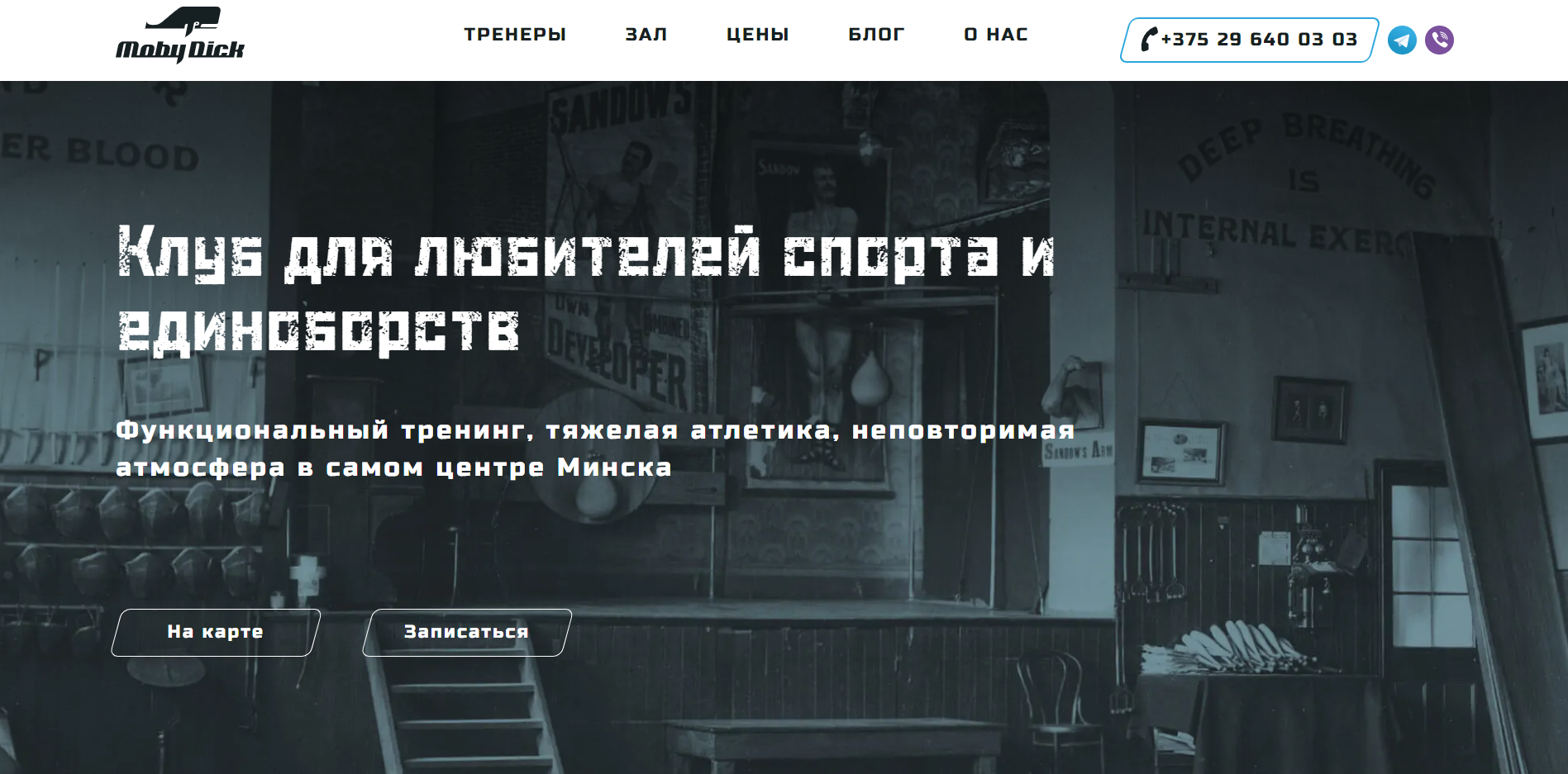
****

Рисунок 1.1 – Графический интерфейс Moby Dick

Однако несмотря на красивый дизайн и некоторый функционал, приложение не лишено недостатков:

* большое количество информации и страниц, в которых можно потеряться;
* недоработанные части сайта, где отображается лишний текст;
* нет возможности выбрать определенное время для тренировки;
* запись осуществлена не полностью через онлайн – администратор всегда должен звонить и уточнять детали.

Еще один пример схожего приложения – Титан. Здесь так же можно посмотреть информацию о зале, цены, предоставляемые услуги, контактную информацию и фотографии зала. Графический интерфейс данного приложения представлен на рисунке 1.2.

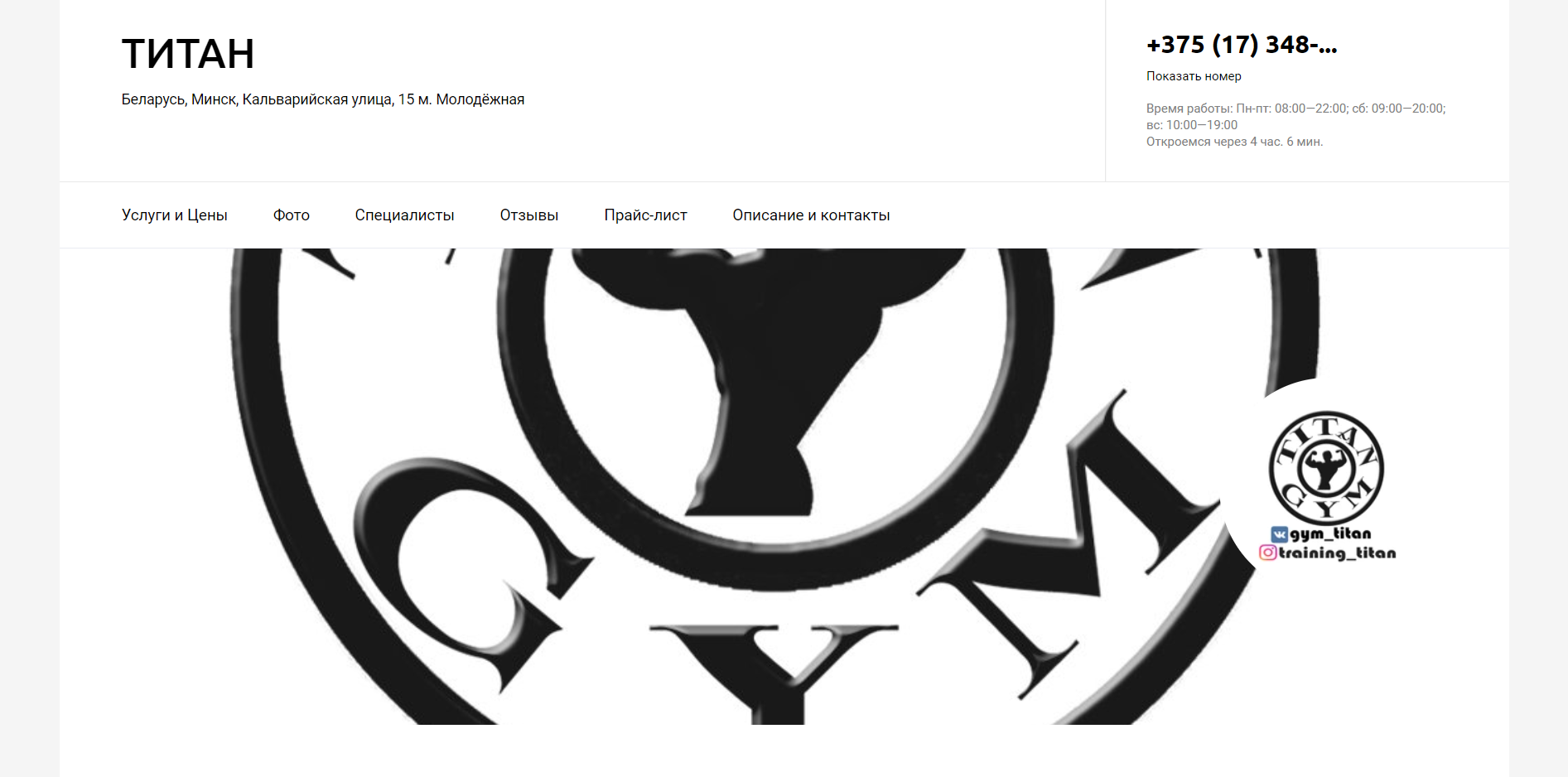


Рисунок 1.2 – Графический интерфейс Титан

Достоинства:

* светлый интерфейс, который не нагружает глаза;
* предоставление контактной информации о каждом тренере;
* широкий выбор фильтров для страницы с тренерами и отзывами;
* нет лишней шумной информации.

Недостатки:

* устаревшее оформление дизайна;
* нет возможности зарегистрироваться и просматривать свои записи;
* нет возможности записаться на тренировку на сайте;
* отсутствие адаптивности интерфейса под разные устройства.

Проанализировав программы на схожую тематику, были разработаны технические требования к разработке веб-приложения. Приложение должно:

* поддерживать возможность регистрации и входа в приложение;
* обращаться к базе данных, успешно извлекать, изменять и дополнять данные, осуществлять поиск по базе и автоматическую запись некоторых полей;
* разграничивать функционал в зависимости от роли пользователя;
* выглядеть аккуратно, не иметь лишней информации;
* поддерживать возможность клиента заказа звонка;
* осуществлять запись на тренировку по определенным критериям.

Эти пункты и составляют техническое задание разрабатываемого проекта.

2 Проектирование программного средства

## 2.1 Анализ требований к программному средству и разработка функциональных требований

Приложение должно иметь разделение ролей. Для каждой роли были разработаны различные возможности, которые представлены на UML-диаграмме вариантов использования. При первом открытии сайта пользователь не аутентифицирован. Общая диаграмма вариантов отображена на рисунке 2.1.

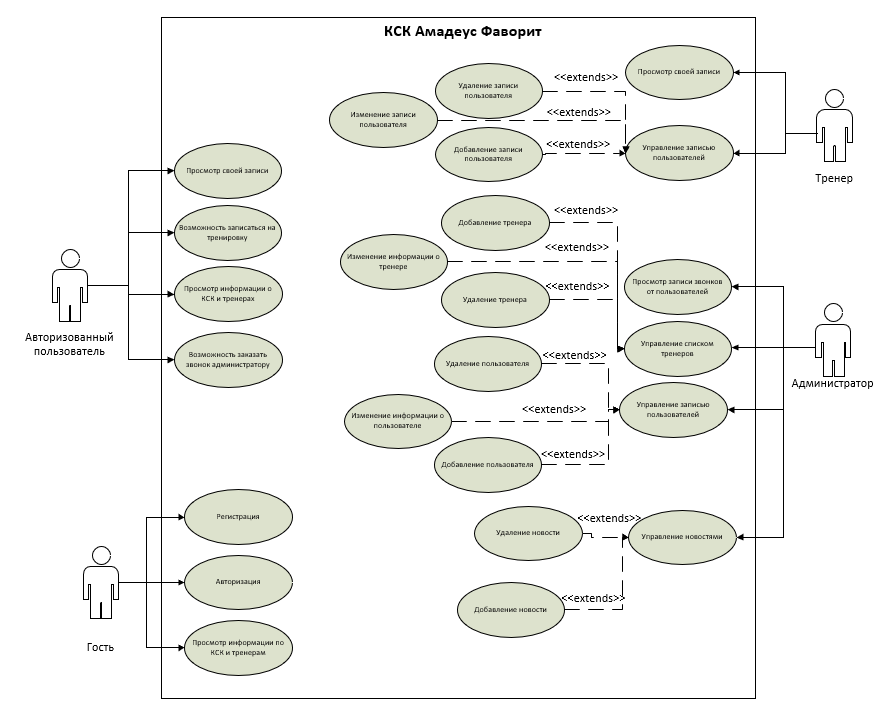


Рисунок 2.1 – Диаграмма вариантов использования

Как видно из диаграммы, не аутентифицированный пользователь(гость) имеет возможность войти и зарегистрироваться, а также посмотреть общую информацию о конно-спортивном клубе.

Итак, зарегистрировавшись и войдя в аккаунт, пользователь получает роль аутентифицированного пользователя. В данном приложении аутентифицированный пользователь – это клиент. После входа у клиента появляется возможность заказать звонок у администратора, записаться на тренировку и посмотреть тренировки, на которые он записан.

Исходя из диаграммы вариантов использования, можно заметить, что имеются еще 2 роли: администратор и тренер. Тренер имеет возможность просматривать свою запись и управлять записями клиентов, т.е. может добавлять, изменять информацию в записи и удалять запись клиента на тренировку. Администратор может добавлять и удалять пользователей в рамках системы, удалять и добавлять тренеров, а также изменять информацию о тренерах и клиентах. Также у администратора есть возможность добавлять и удалять новости клуба.

## 2.2 Общая структура программного средства

Для того чтобы построить качественную архитектуру программного средства были проанализированы проекты со схожим функционалом. Построенная архитектура изображена на рисунке 2.3.

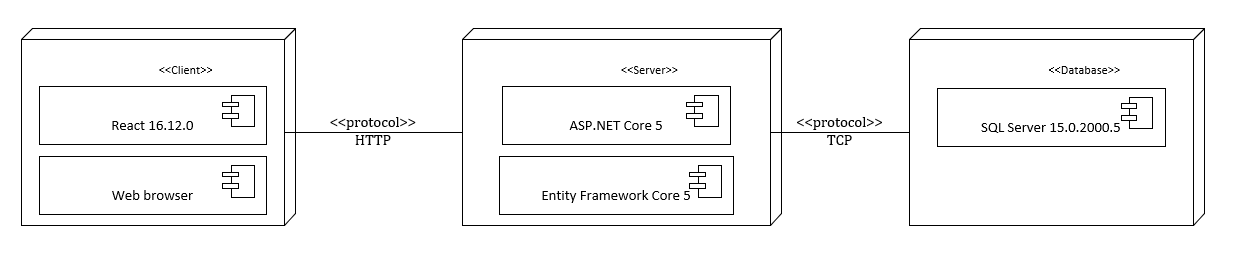


Рисунок 2.2 – Общая структура проекта

У нас представлены две рабочие станции: сервер (ASP.NET Core 5, Entity Framework Core 5), клиент (React.js, Browser), связанные по протоколу HTTP. А также присутствует компонент базы данных (SQL Server), который связан с сервером по протоколу TCP.

## 2.3 Cтруктура базы данных

В предыдущей главе были разработаны технические требования к программному средству. Для реализации разрабатываемого проекта были выбраны: фреймворк ASP.NET Core, база данных MSSQL и библиотека React.

Взаимодействие серверной части приложения с базой данных осуществляется с помощью объектно-ориентированной технологии доступа к данным Entity Framework Core. Благодаря данному подходу осуществлено моделирование данных приложения на основе объектов. EF Core позволяет абстрагироваться от самой базы данных и ее таблиц и работать с данными независимо от типа хранилища. Если на физическом уровне мы оперируем таблицами, индексами, первичными и внешними ключами, но на концептуальном уровне, который нам предлагает Entity Framework, мы уже работаем с объектами [2].

База данных состоит из 8 таблиц: Users, Users\_Information, Roles, Template, Training, Shedule, News, Calls.

Далее будут представлены все таблицы базы данных с их содержанием и краткой характеристикой.

Таблица Users, содержит информацию о пользователях:

* Id – уникальный идентификатор;
* Login – логин пользователя;
* Password – пароль пользователя;
* Name – имя пользователя;
* Surname – фамилия пользователя;
* Phone – телефон пользователя;
* Id\_role – идентификатор роли пользователя;

Таблица Users представлена на рисунке 2.3.

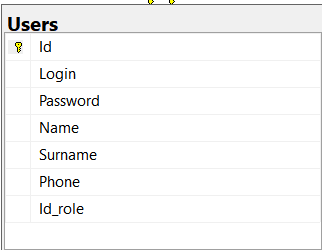


Рисунок 2.3 – Таблица Users

Таблица Roles содержит роли, предоставленные в приложении:

* Id – уникальный идентификатор;
* Role\_name – наименование роли;

Таблица Roles представлена на рисунке 2.4.

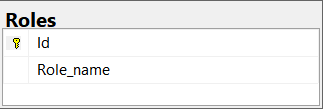


Рисунок 2.4 – Таблица Roles

Таблица Template содержит шаблоны, по которым производится запись пользователей на тренировки в клуб:

* Template\_name – уникальный наименование шаблона;
* Ozn – количество клиентов с уровнем Ozn;
* Z – количество клиентов с уровнем Z;
* FirstLevel – количество клиентов с 1-ым уровнем;
* SecondLevel – количество клиентов со 2-ым уровнем;
* Poni – количество клиентов с уровнем Poni.

Таблица Template представлена на рисунке 2.5.

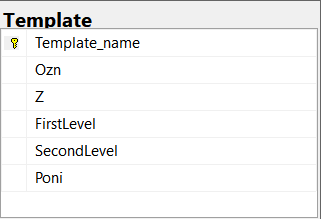


Рисунок 2.5 – Таблица Template

Таблица Training содержит записи клиентов на тренировки в клуб:

* Id – уникальный идентификатор;
* Id\_shedule – идентификатор тренировки у тренера;
* Id\_user – идентификатор пользователя, который записался;
* Status – статус тренировки пользователя;
* NeedEquipment – необходимость аренды снаряжения.

Таблица Training представлена на рисунке 2.6.

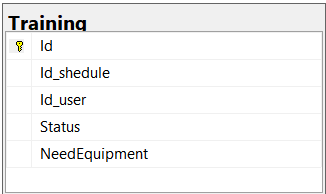


Рисунок 2.6 – Таблица Training

Таблица Shedule содержит тренировки у тренеров по часам.

* Id – уникальный идентификатор;
* Id\_trainer – идентификатор тренера, который ведет тренировку;
* Data – дата тренировки;
* Hours\_start – час начала тренировки;
* Hours\_end – час конца тренировки.

Таблица Shedule представлена на рисунке 2.7.

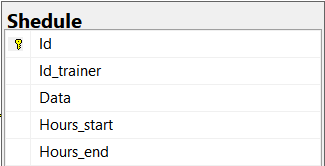


Рисунок 2.7 – Таблица Shedule

Таблица News содержит новости клуба.

* Id – уникальный идентификатор;
* NewsHeading – заголовок новости;
* NewsElement – основной текст новости.

Таблица News представлена на рисунке 2.8.

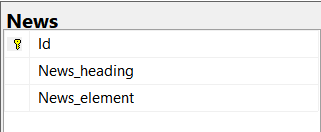


Рисунок 2.8 – Таблица News

Таблица Calls содержит информацию о заказанных звонках для администратора:

* Id – уникальный идентификатор;
* Name – имя клиента;
* Surname – фамилия клиента;
* Phone – телефон клиента.

Таблица Calls представлена на рисунке 2.9.

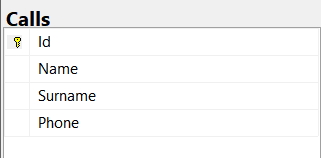


Рисунок 2.9 – Таблица Calls

Таблица Users\_Information имеет следующее содержание:

* Id\_user – идентификатор пользователя;
* Level\_status – уровень пользователя;
* Canceled\_training – количество отмененных тренировок пользователя;
* Amount\_training – количество пройденных тренировок;
* Trainer\_discription – описание тренера;
* Equipment – необходимость аренды снаряжения на тренировку.

Эта информация помогает контролировать статистику тренировок у клиентов.

Поле Trainer\_discription предоставляет описание тренера, которое можно посмотреть на странице с тренерами или изменить его, если есть такая необходимость.

Таблица Users\_Information представлена на рисунке 2.10.

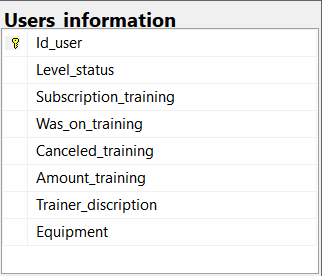


Рисунок 2.10 – Таблица Users\_Information

В таблицах базы данных Amadeus содержится вся необходимая информация для функционирования приложения для записи на тренировки в конно-спортивный клуб. На рисунке 2.11 представлена полная диаграмма базы данных Amadeus вместе со связями, которые установлены между таблицами данной базы данных. Исходя из диаграммы, можно увидеть, что некоторые таблицы существуют сами по себе, т.е. без какой-либо связи с остальными.



Рисунок 2.11 – Диаграмма базы данных Amadeus

# 3 Разработка программного средства

## 3.1 Реализация сервера

Класс Startup является входной точкой в приложение ASP.NET Core (Приложение А). Этот класс производит конфигурацию приложения, настраивает сервисы, которые приложение будет использовать, устанавливает компоненты для обработки запроса или middleware.

Класс Startup должен определять метод Configure(), и также опционально в Startup можно определить конструктор класса и метод ConfigureServices().При запуске приложения сначала срабатывает конструктор, затем метод ConfigureServices() и в конце метод Configure(). Эти методы вызываются средой выполнения ASP.NET [1].

На рисунке 3.1 отображена структура сервера, включающая в себя все файлы и директории, разработанные в процессе создания приложения.

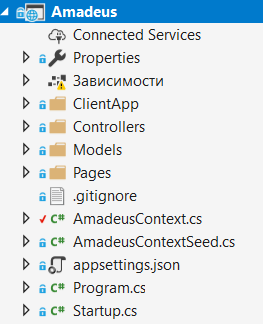


Рисунок 3.1 – Структура проекта сервера

Сервер написан на ASP.NET Core с применением EF для связи с БД. Сервер взаимодействует с клиентской частью приложения, используя формат JSON. На серверной стороне реализовано множество методов для работы приложения, в особенности для:

* регистрации пользователей;
* авторизации пользователей;
* взаимодействия с новостями клуба;
* оформления звонков;
* взаимодействия с клиентами;
* взаимодействия с тренерами.

Главным файлом приложения является Program.cs, с которого и начинается его выполнение. Код этого файла настраивает и запускает веб-хост, в рамках которого разворачивается приложение. Содержание файла Program.cs показано на листинге 3.1.

public class Program

{

public static void Main(string[] args)

{

var host = CreateHostBuilder(args).Build();

using (var scope = host.Services.CreateScope())

{

var services = scope.ServiceProvider;

try

{

var context = services.GetRequiredService<AmadeusContext>();

AmadeusContextSeed.SeedSampleData(context);

}

catch (Exception ex)

{

}

}

host.Run();

}

public static IHostBuilder CreateHostBuilder(string[] args) =>

Host.CreateDefaultBuilder(args)

.ConfigureWebHostDefaults(webBuilder =>

{

webBuilder.UseStartup<Startup>();

});

}

Листинг 3.1 – Содержание файла Program.cs

В директории «Сontrollers» содержатся контроллеры. Контроллеры – это классы, которые содержат логику обработки запроса , которые наследуются от абстрактного базового класса Microsoft.AspNetCore.Mvc.Controller.. По соглашениям об именовании названия контроллеров обычно оканчиваются на суффикс "Controller", остальная же часть до этого суффикса считается именем контроллера. Контроллеры вызывают методы сервера, в которых находится основная бизнес-логика приложения. Метод контроллера принимает в параметры объекты запроса или ответа.

Пример одного из методов контроллера, а именно добавление пользователя на тренировку представлен на листинге 3.2. В этом методе проходит проверка на предмет того, подходит ли выбранная тренировка для данного пользователя по его спортивному уровню.

[HttpPost]

[Route("addTrainingParticipant")]

public async Task<IActionResult> AddTrainingParticipant([FromBody]AddTraining model)

{

if (model.ScheduleId == 0 || model.UserId == 0)

{

return Json(new BadResponse("Некорректные данные"));

}

training training\_participant = \_context.training.Where(t => t.IdShedule == model.ScheduleId && t.IdUser == model.UserId).FirstOrDefault();

if (training\_participant != null)

{

return Json(new BadResponse("Пользователь уже записан"));

}

Shedule schedule = \_context.Shedules.Where(s => s.Id == model.ScheduleId).FirstOrDefault();

if (!DoesTrainingSuitForUser(schedule, model.UserId))

{

return Json(new BadResponse("Пользователь уже записан"));

}

\_context.training.Add(new training(model.ScheduleId, model.UserId, "Записан", model.NeedEquipment));

await \_context.SaveChangesAsync();

return Json("Запись на тренировку добавлена");

}

Листинг 3.2 – Метод контроллера для добавления пользователя на тренировку

Этот метод принимает данные из тела запроса. Далее эти данные проверяются на равность нулю и, если они равны, то возвращается ответ о том, что данные некорректны. После чего идет проверка, существует ли уже такая запись и, если запись не существует и пользователь подходит на эту тренировку по заранее определенным шаблонам для тренировок, то запись осуществляется.

В директории «Controllers» также содержатся контроллеры для работы администратора, тренера и контроллеры, которые обеспечивают правильное взаимодействие с клиентом.

В Приложении В и в приложении Г продемонстрированы фрагменты кода для поиска подходящей тренировки и отображение тренировок для разных ролей соответственно.

Одним из основных моментов безопасности приложения считается использование JWT токена. Метод получения JWT токена представлен на листинге 3.3.

private string GetJwtToken(User login, ClaimsIdentity identity)

{

var now = DateTime.UtcNow;

// создаем JWT-токен

var jwt = new JwtSecurityToken(

issuer: Options.ISSUER,

audience: Options.AUDIENCE,

notBefore: now,

claims: identity.Claims,

expires: now.Add(TimeSpan.FromMinutes(Options.LIFETIME)),

signingCredentials: new SigningCredentials(Options.GetSymmetricSecurityKey(), SecurityAlgorithms.HmacSha256));

var encodedJwt = new JwtSecurityTokenHandler().WriteToken(jwt);

return encodedJwt;

}

Листинг 3.3 – Метод получения JWT токена

Директория «Models» содержит файлы с классами-сущностями базы данных. Приложение использует ORM-технологию EF для взаимодействия с базой данных. Строка подключения к базе данных находится в файле AmadeusContext.cs. В реализации данного приложения была применен подход Database First, поэтому все классы-сущности и класс контекста генерировались на основе уже существующей базы данных.

Директория «ClientApp» содержит реализацию клиентской части этого приложения, т.к. при создании проекта был выбран вариант создания ASP.NET Core + React.

## Работа с базой данных

Взаимодействие с базой данных в Entity Framework Core происходит посредством специального класса – контекста данных. В данном приложении эту роль выполняем класс AmadeusContext.cs. Т.к. при создании и разработке приложения использовался подход Database First, класс контекста и все классы-сущности генерировались автоматически по ранее созданной в MSSQL базе данных. После этого с помощью консоли диспетчера пакетов и команды Scaffold-DbContext "Server=WIN-4PG4224I26S;Database=Amadeus;Trusted\_Connection=True;" Microsoft.EntityFrameworkCore.SqlServer в проект будут добавлены классы-сущности и класс контекста базы данных. В команде для генерации этих классов мы используем имя сервера, имя базы данных, которую нам надо добавить и устанавливаем Trusted\_Connection в true.

На листинге 3.4 представлена строка подключения сервера к базе данных Amadeus.

protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)

{

if (!optionsBuilder.IsConfigured)

{

optionsBuilder.UseSqlServer("Server=WIN-4PG4224I26S;Database=Amadeus;Trusted\_Connection=True;");

}

}

Рисунок 3.4 – Фрагмент кода AmadeusContext.cs

После того, как все сущности были перенесены мы можем спокойно работать с нашей базой данных, вставлять, изменять и удалять данные из нее. Для добавления объекта используется метод Add. Метод Add устанавливает значение Added в качестве состояния нового объекта. Поэтому метод db.SaveChanges() сгенерирует выражение INSERT для вставки модели в таблицу. Удаление производится с помощью метода Remove. Данный метод установит статус объекта в Deleted, благодаря чему Entity Framework при выполнении метода db.SaveChanges() сгенерирует SQL-выражение DELETE. При изменении объекта Entity Framework сам отслеживает все изменения, и когда вызывается метод SaveChanges(), будет сформировано SQL-выражение UPDATE для данного объекта, которое обновит объект в базе данных .

Структура модели пользователя показана на листинге 3.5.

public partial class User

{

public User()

{

Shedules = new HashSet<Shedule>();

training = new HashSet<training>();

}

public int Id { get; set; }

public string Login { get; set; }

public string Password { get; set; }

public string Name { get; set; }

public string Surname { get; set; }

public string Phone { get; set; }

public int? IdRole { get; set; }

public virtual Role IdRoleNavigation { get; set; }

public virtual UsersInformation UsersInformation { get; set; }

public virtual ICollection<Shedule> Shedules { get; set; }

public virtual ICollection<training> training { get; set; }

}

Рисунок 3.5 – Структура модели пользователя

Отличительной чертой Entity Framework Core, как технологии ORM, является использование запросов LINQ для выборки данных из БД. С помощью LINQ мы можем создавать различные запросы на выборку объектов, в том числе связанных различными ассоциативными связями. А Entity Framework при выполнение запроса транслирует выражения LINQ в выражения, понятные для конкретной СУБД (как правило, в выражения SQL) [2].

## Проектирование структуры клиентской части

Интерфейс пользователя разработан с использованием библиотеки React. Клиентская часть приложения представляет собой набор компонентов и сервисов для взаимодействия клиента с сервером. Компоненты представляют собой обычные js-модули, но за отрисовку и оптимизацию жизненных циклов компонентов отвечает библиотека React.

На рисунке 3.2 отображена структура клиентской части разрабатываемого программного средства.

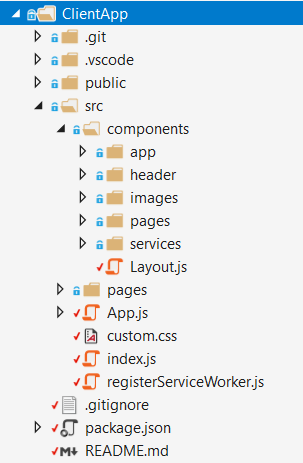


Рисунок 3.2 – Структура клиентской части приложения

Директория public содержит файл index.html который возвращается клиенту при первом обращении к серверу и является контейнером для всего SPA приложения.

Директория «src» содержит директорию «components», в которой находятся другие директории:

* директорию «app», в которой находится файл с маршрутами для клиентской части;
* директорию «header», которая содержит файлы для построение шапки приложения;
* директорию «pages», которая содержит страницы, из которых строится клиентская часть приложения;
* директорию «images», которая содержит изображения, использованные при демонстрации страниц;
* директорию «services», которая содержит файлы для связи непосредственно с сервером приложения.

В Приложении Б представлен фрагмент app.js, где представлены маршруты, которые доступны пользователям в зависимости от их роли.

При каждой отрисовке страницы, в зависимости от адреса, будет использоваться соответствующий компонент. Кроме этого библиотека React позволяет задавать несколько адресов на один компонент.

Вся функциональность библиотеки React построена на компонентах. Компоненты могут быть реализованы двумя способами. Первый способ – это создание компонента на основе класса, останавливаться на нем не будем так как в последних версиях библиотеки разработчики рекомендуют их не использовать. Второй способ – это реализация компонента на основе чистой функции, которая принимает на вход параметры и возвращает разметку. Важной особенностью функциональных компонентов является то, что они не могут изменять пришедшие через параметры значения. Все побочные эффекты, такие как отправка запросов на сервер или запуск таймеров должны выполняться внутри хуков. Хук – это специальная функция, внутри которой можно выполнять побочные эффекты, не опасаясь возникновения исключительных ситуаций [3]. Пример функционального компонента представлен на листинге 3.6.

function User(props) {

const server\_api = new Server\_api();

const [status,setStatus] = useState(0);

const [eq,setEq] = useState(0);

useEffect(()=>{

if(props.user){

setStatus(props.user.Status)

setEq(props.user.NeedEquipment)

}

},[])

const \_updateStatus = (e, user, scheduleId) => {

let form = {

scheduleId: scheduleId,

userId: user.UserId,

newStatus: e.target.value,

}

server\_api.сhangeStatus(form)

.then((data) => {

if (data.error) { alert(data.error) }

else {

alert(data);

}

})

.catch((error) => {

Листинг 3.6 – Фрагмент функциональной компоненты

Благодаря тому, что библиотека React строит виртуальное DOM дерево. Перерисовка DOM элементов происходит только тогда, когда это необходимо, что значительно повышает скорость отклика всего графического интерфейса.

Для взаимодействия с методами, которые лежат на стороне сервер используется файл server-api.js. Здесь мы указывает сервер, который используем и API позволяющие нам отправить данные в методы на стороне сервера, где эти данные обрабатываются и приходит ответ в формате Json. Ответ может приходить как объектом для последующей демонстрации данных из этого объекта, так и просто строкой в формате Json. Фрагмент файла server-api.js представлен на листинге 3.7

async get\_news(data) {

try {

const res = await fetch(`${SERVER}home/?page=${data.page}&limit=${data.limit}`, {

method: "GET",

headers: {

'Content-Type': 'application/json',

'Access-type': 'application/json'

}

})

if (!res.ok) {

throw 'Error with review'

}

return res.json();

} catch (error) {

}

}

Листинг 3.7 – Фрагмент файла server-api.js

В приложении представлены 2 вида компонент React: функциональная и классовая. Классовая компонента является более устаревшей. Для ее создания необходимо наследоваться от стандартного класса React.Component. Также в классовой компоненте присутствуют методы жизненного цикла: componentWillMount – вызывается при вымонтировании компоненты из DOM дерева. Обратным к этому методу является ComponentDidMount. Также существует метод ComponentDidUpdate, который вызывается при каждом изменении компоненты.

4 Тестирование программного средства

В разработанном программном средстве предусматриваются некоторые негативные сценарии.

При регистрации нового пользователя, если пользователь не заполнит все обязательные поля на форме либо введет некорректные данные, под соответствующим полем появится сообщение, изображенное на рисунке 4.1.

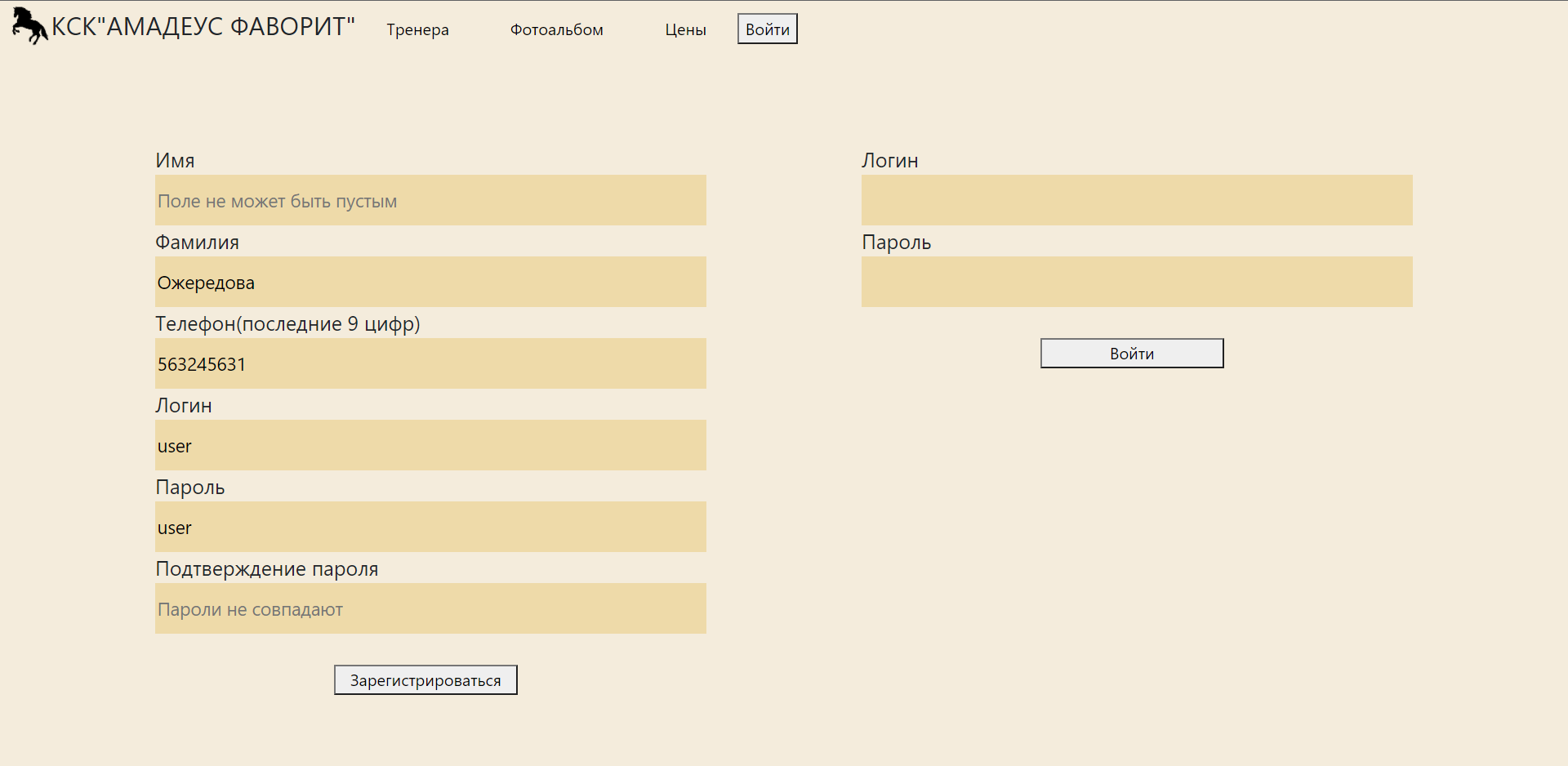


Рисунок 4.1 – Сообщения, возникающее при вводе некорректных данных

В данном приложении есть обработка запроса на удаление тренера из системы, если у этого тренера существуют активные записи на его расписание. Вариант обработки такого варианта событий изображена на рисунке 4.2.

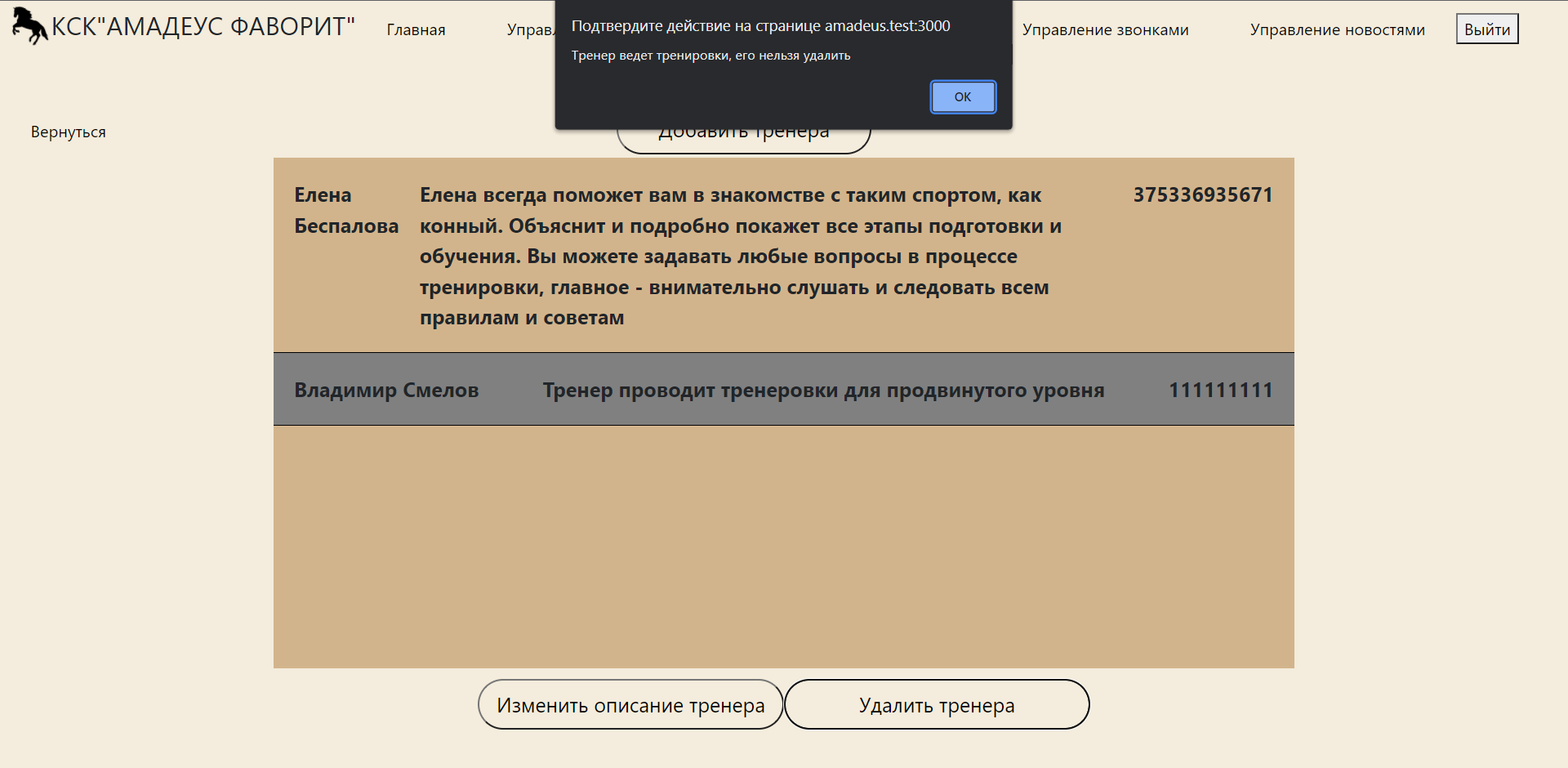


Рисунок 4.2 – Вариант обработки удаления тренера, у которого есть запись на тренировки

При нажатии на кнопку «Добавить тренера» появляется форма с полями для заполнения. В этой форме присутствует валидация, благодаря которой пользователь не может ввести некорректные данные.

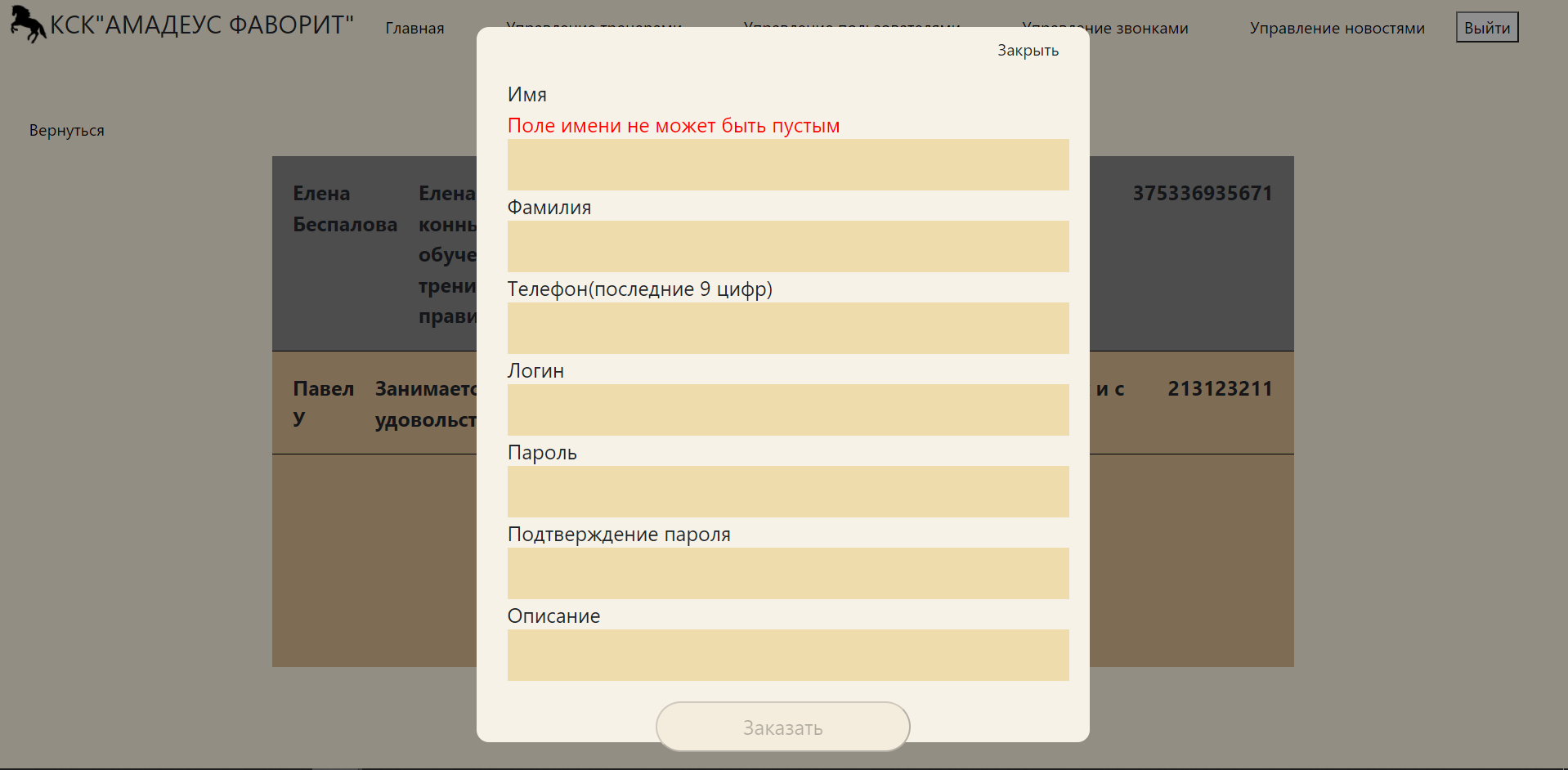


Рисунок 4.7 – Форма добавления нового тренера

У администратора есть возможность удалить запись графика в расписании тренеров, но, если на это время у тренера есть записанные клиенты, администратор не сможет удалить эту запись, пока тренер не удалит клиентов из записи. На странице 4.3 представлена обработка варианта удаления записи графика из расписания, если на это время уже записаны клиенты.

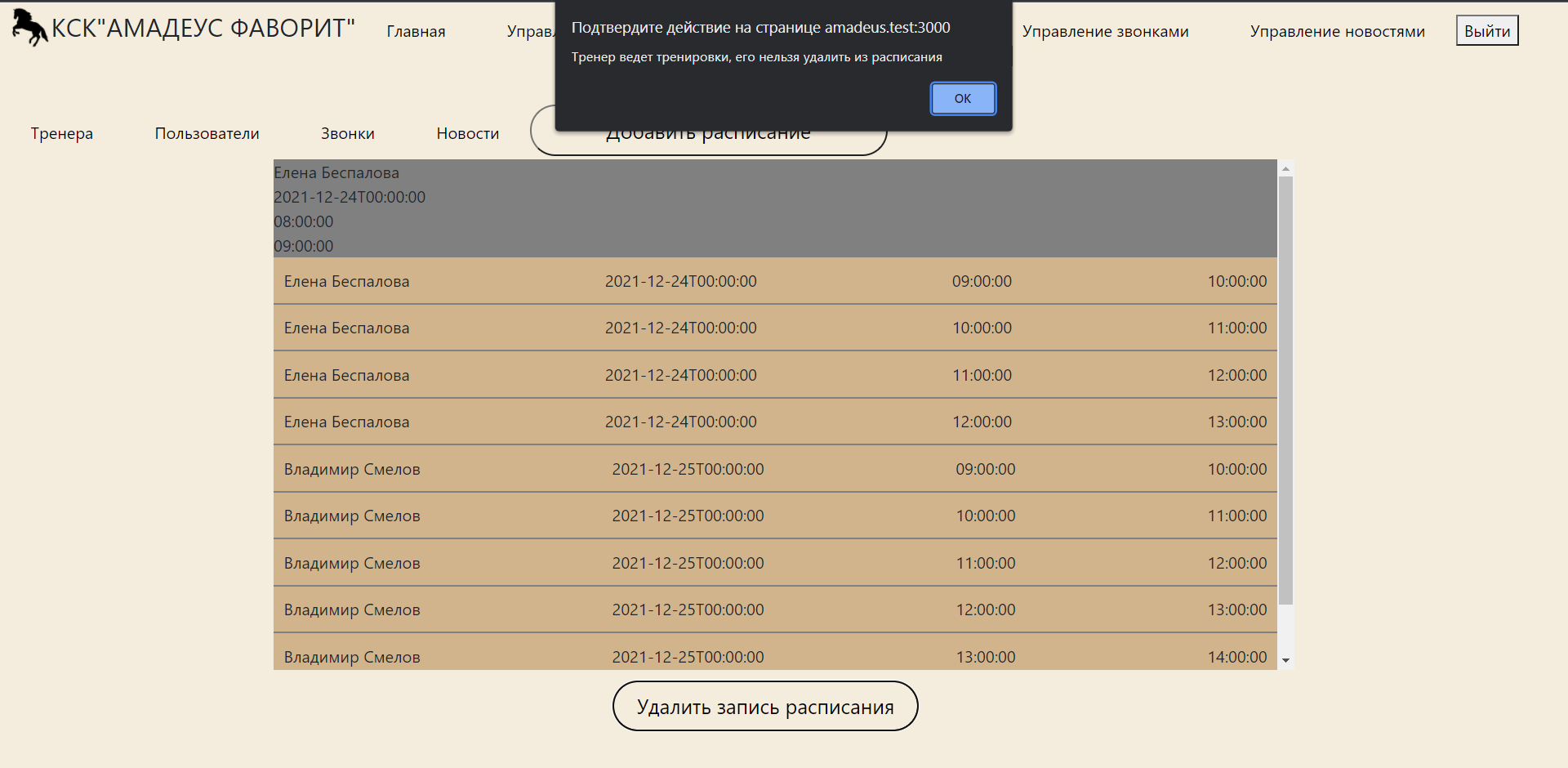


Рисунок 4.3 – Обработка удаления записи в расписании, если на это время записаны клиенты

На вкладке управления пользователями также есть возможность добавления нового пользователя. На рисунке 4.4 изображено окошко добавления нового пользователя и показана валидация при условии, что номер телефона введен неверно. Пока все данные не будут введены верно, кнопка «Добавить» не будет доступна.

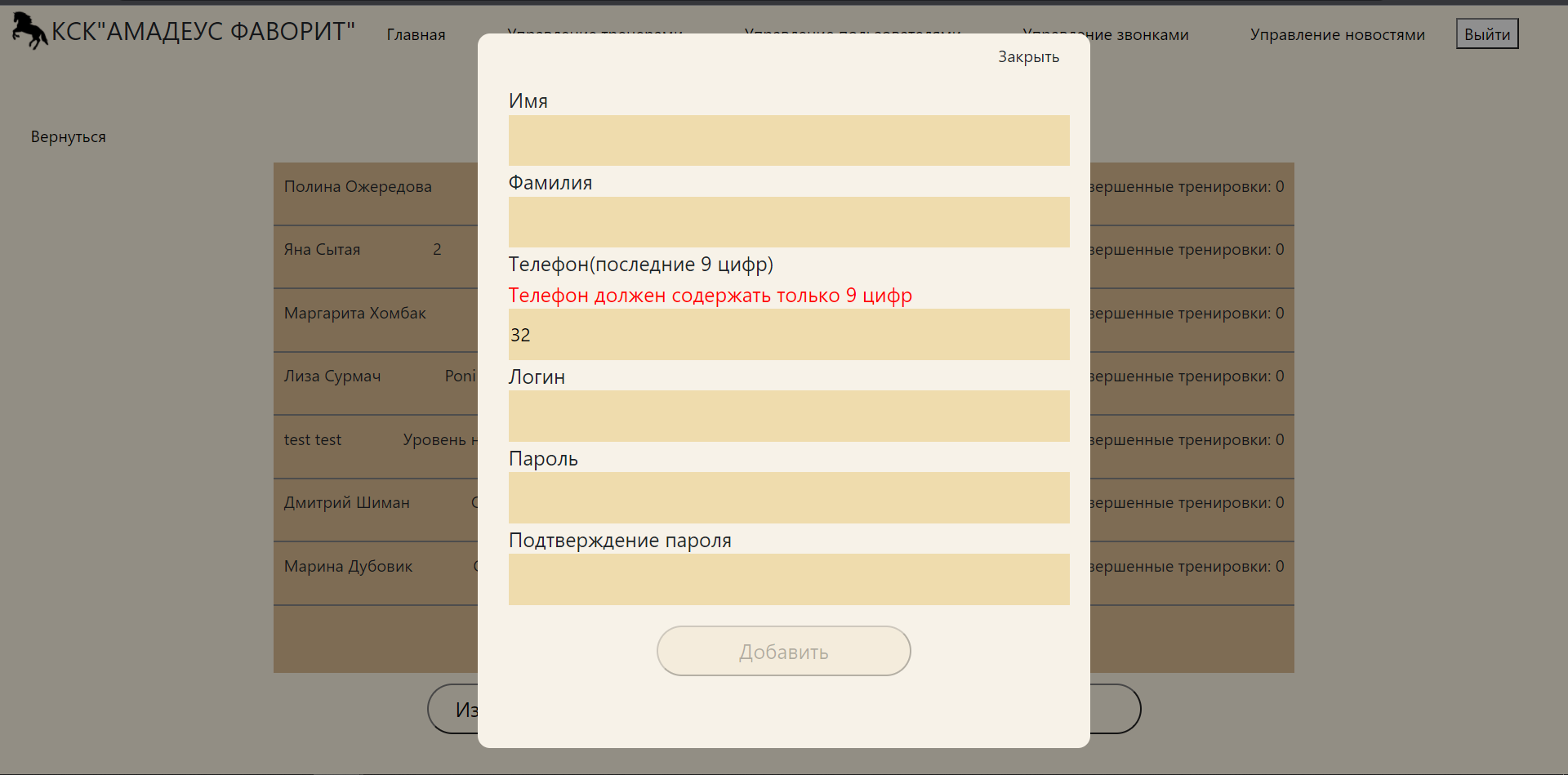


Рисунок 4.4 – Обработка неверно ведденого номера телефона в окошке добавления нового пользователя

На странице с новостями администратор может добавить новую новость через окошка добавления. Для добавление новой новости должны быть заполнены все поля, которые предоставлены в окошке. На рисунке 4.5 показан вариант обработки ситуации, если не все поля заполнены.

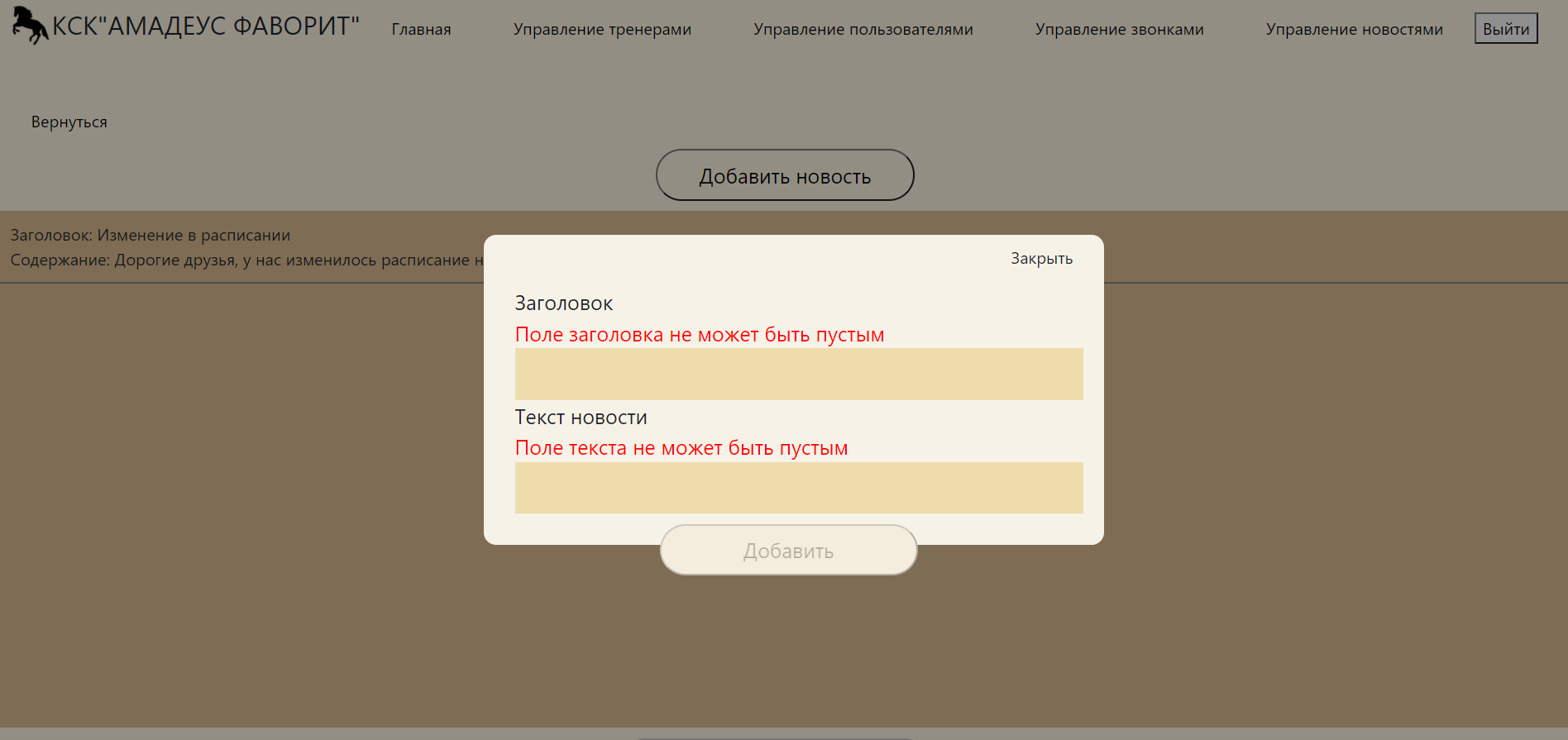


Рисунок 4.5 – Обработка данных при добавлении новой новости

# 5 Руководство пользователя

## Руководство для неавторизованного пользователя

Пользователь является неавторизованным в том случае если время жизни токенов которые находятся в локальном хранилище, истекло, либо токены отсутствуют.

Неавторизованный пользователь имеет возможность авторизироваться и зарегистрироваться. Формы регистрации и авторизации продемонстрированы в главе 4.

Для того чтобы зарегистрироваться пользователю необходимо заполнить все поля формы. После того, как все поля заполнены, нужно нажать кнопку «Зарегистрироваться». В случае успешной регистрации получит уведомление, что регистрация прошла успешно.

Для того чтобы войти в созданный аккаунт, необходимо заполнить все поля на странице входа. Если пользователь с указанным логином и паролем существует, произойдет перенаправление на главную страницу пользователя.

## Руководство для аутентифицированного пользователя

После успешной аутентификации пользователь попадает на свою главную страницу, где он может пролистать новости клуба используя пагинацию. Также здесь у него появляется возможность заказать звонок у администратора для уточнения информации, которой он не увидел в приложении. Для этого надо нажать на кнопку «Заказать звонок» и заявка будет автоматически добавлена. У пользователя появляется доступ к странице с тренировками, на которые он уже записан. На этой же странице есть кнопка для добавления новой тренировки. Пользователь не может самостоятельно удалить свою запись, чтобы не было спорных ситуаций. Страница с тренировками пользователя отображена на рисунке 5.1.

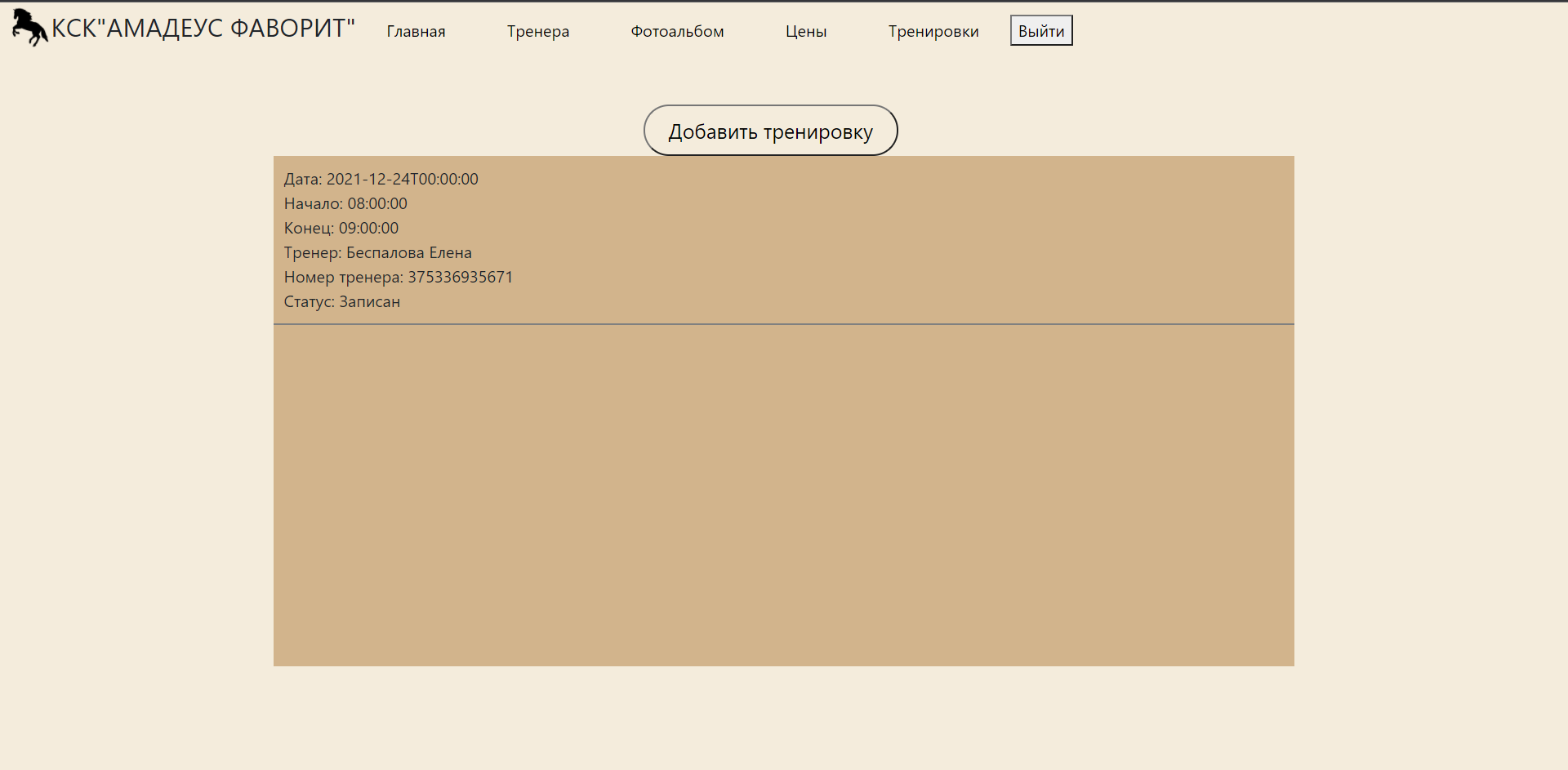


Рисунок 5.1 – Страница с тренировками пользователя

После нажатия на кнопку «Добавить тренировку» появляется окошко, где клиент может выбрать подходящую ему дату и предпочитаемого тренера, после чего ему будут предложены возможные по времени и его уровню тренировки. Окошко для добавления своей тренировки изображено на рисунке 5.2.

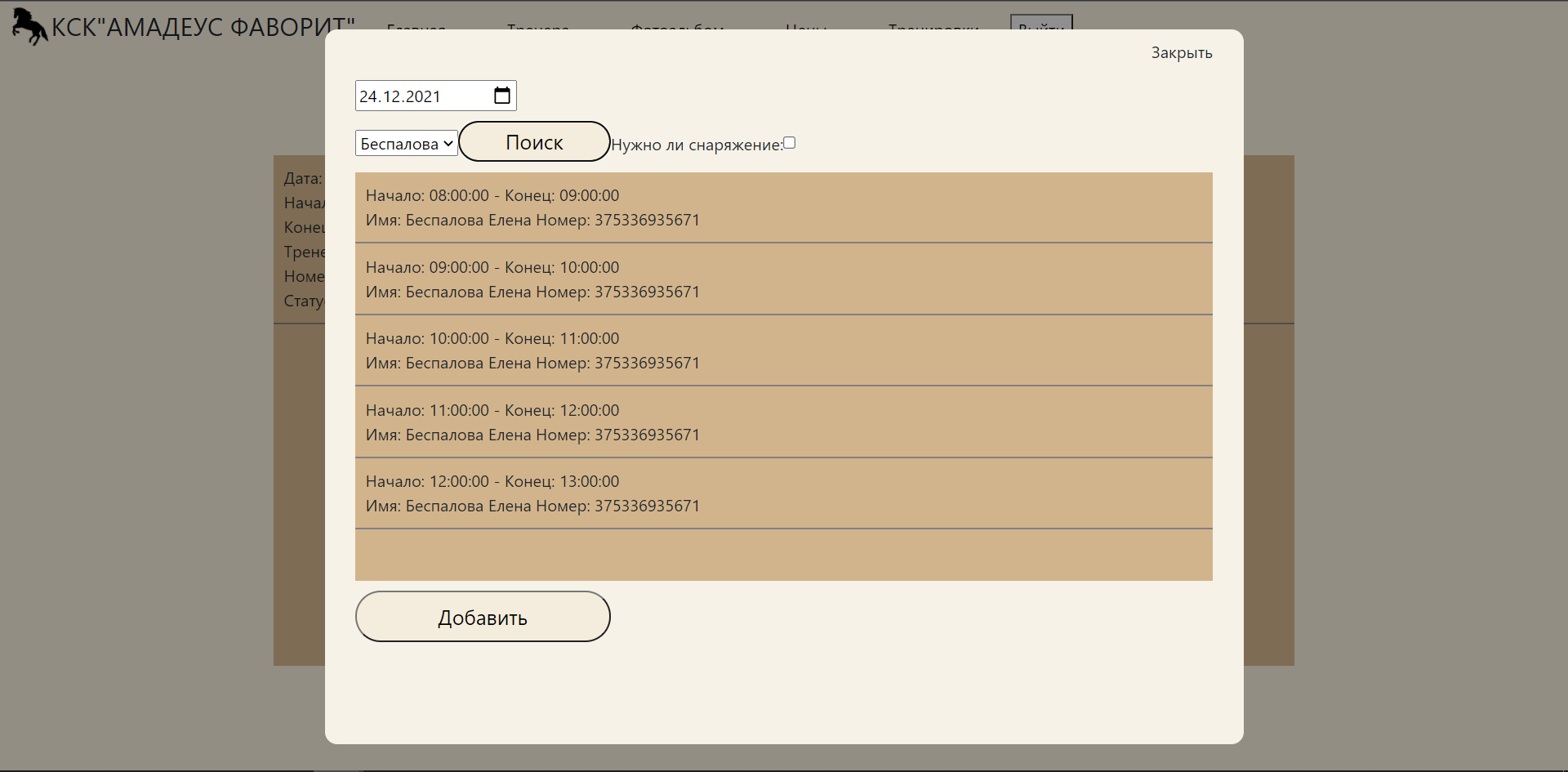


Рисунок 5.2 – Окошко для добавления тренировки

Таким образом пользователь выбирает подходящий ему вариант, отмечает необходимость аренды снаряжения и нажимает на кнопку «Добавить» для оформления записи. После этого запись на тренировку была оформлена и пользователь может увидеть ее в списке своих тренировок. Там отображается начало тренировки, ее конец, тренер, к которому записался пользователь, его телефон для связи и статус тренировки.

## Руководство для тренера

При авторизации тренера есть возможность перейти на страницу с тренировками этого тренера. В этом списке отображаются тренировки из расписания тренера, как с участниками, так и без них. Тренер имеет возможность посмотреть всех участников каждой тренировки, может изменить статус тренировки: «Завершен» – если клиент пришел на тренировку и она была проведена для него успешно, «Отменен» – если клиент отменился позднее, чем за сутки до начала тренировки, тем самым сорвав запись. При установке одного из статусов увеличивается количество либо успешно завершенных тренировок, либо количество отмененных тренировок. Эта информация отображается на странице у администратора для управления пользователями. Тренер может изменить необходимость аренды снаряжения для пользователя, если во время проведения тренировки снаряжение понадобилось в связи с различными обстоятельствами, и удалить запись пользователя на определённую тренировку. Страница тренера изображена на рисунке 5.3.

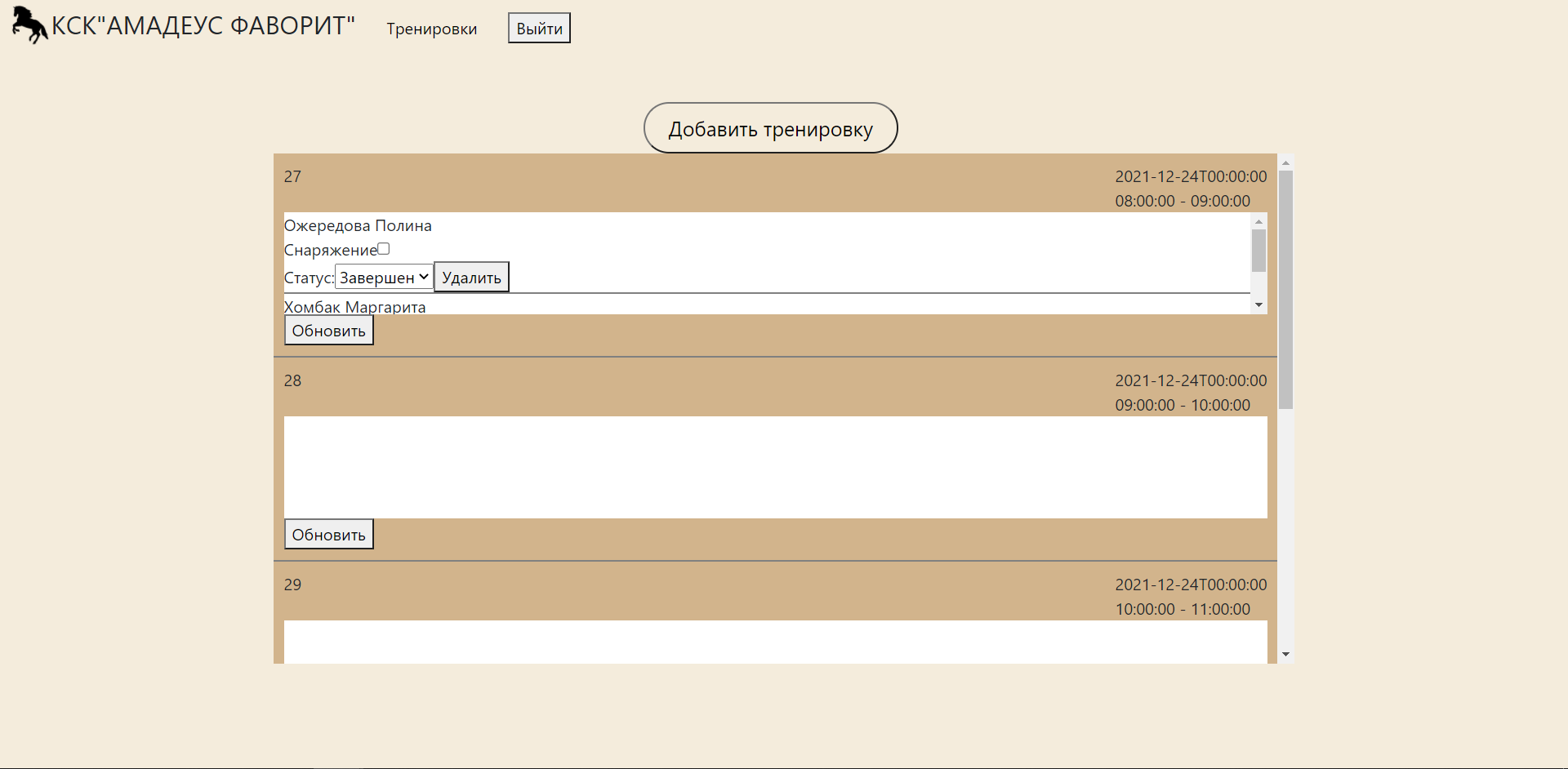


Рисунок 5.3 – Страница тренера

При нажатии на кнопку «Добавить тренировку» тренер может добавить новую запись клиента в свое расписание, заполнив все нужные поля.

Страница добавления тренировки изображена на рисунке 5.4.

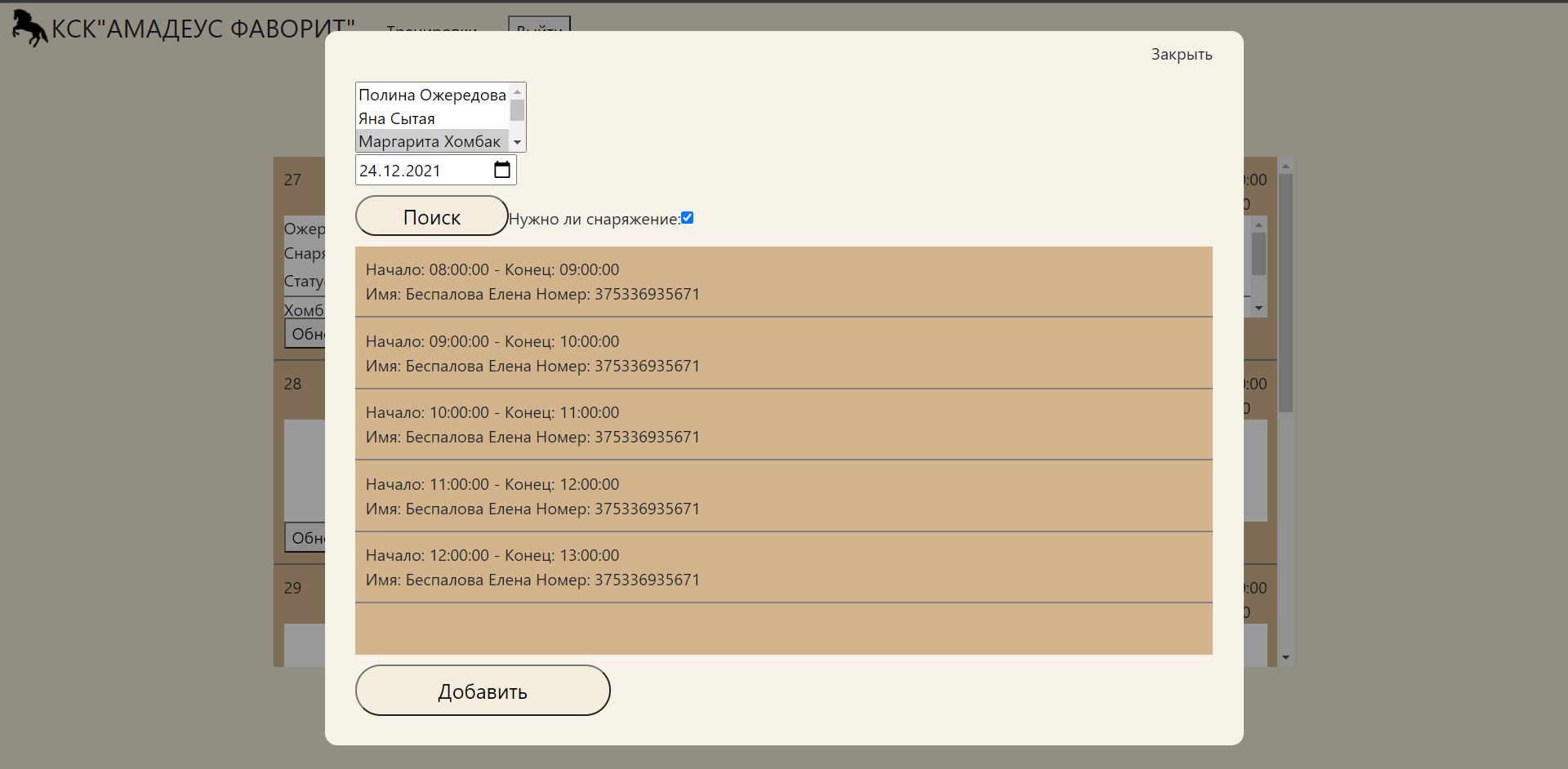


Рисунок 5.4 – Страница добавления тренировки

## Руководство для администратора

При в ходе в систему в качестве администратора вы попадаете на страницу с новостями. Тут представлены новости и изменения в расписании конно-спортивного клуба. Переключение между новостями осуществляется с помощью стрелочек. Страница с новостями изображена на рисунке 5.5.

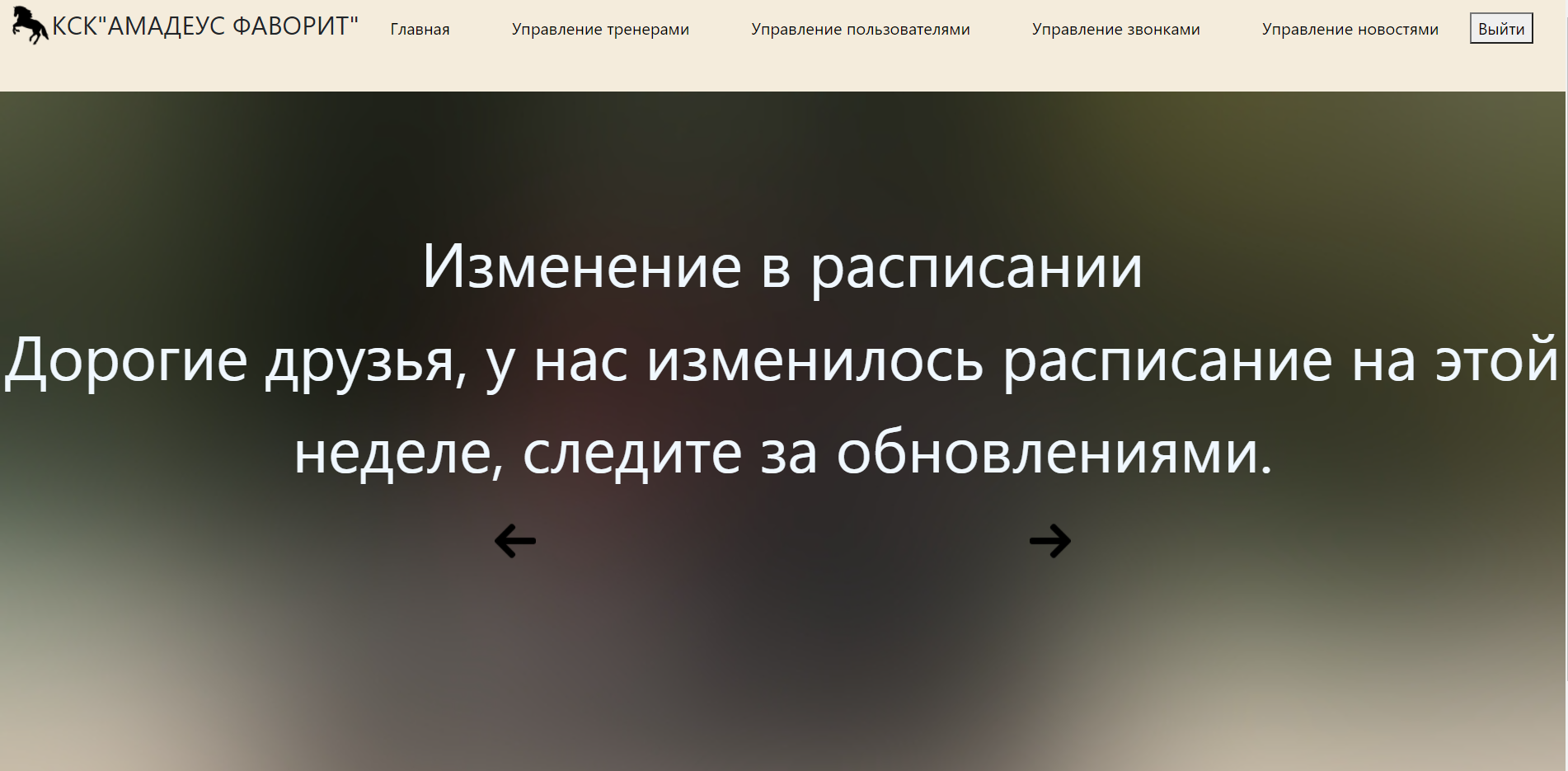


Рисунок 5.5 – Страница с новостями

У администратора есть 4 вкладки для управления данными в приложении. При нажатии на кнопку «Управление тренерами» вы попадаете на страницу, где отображаются все тренера, которые есть в системе. Здесь можно добавить нового тренера, удалить выбранного тренера либо изменить описание выбранного тренера. Страница администратора для управления тренерами показана на рисунке 5.6.

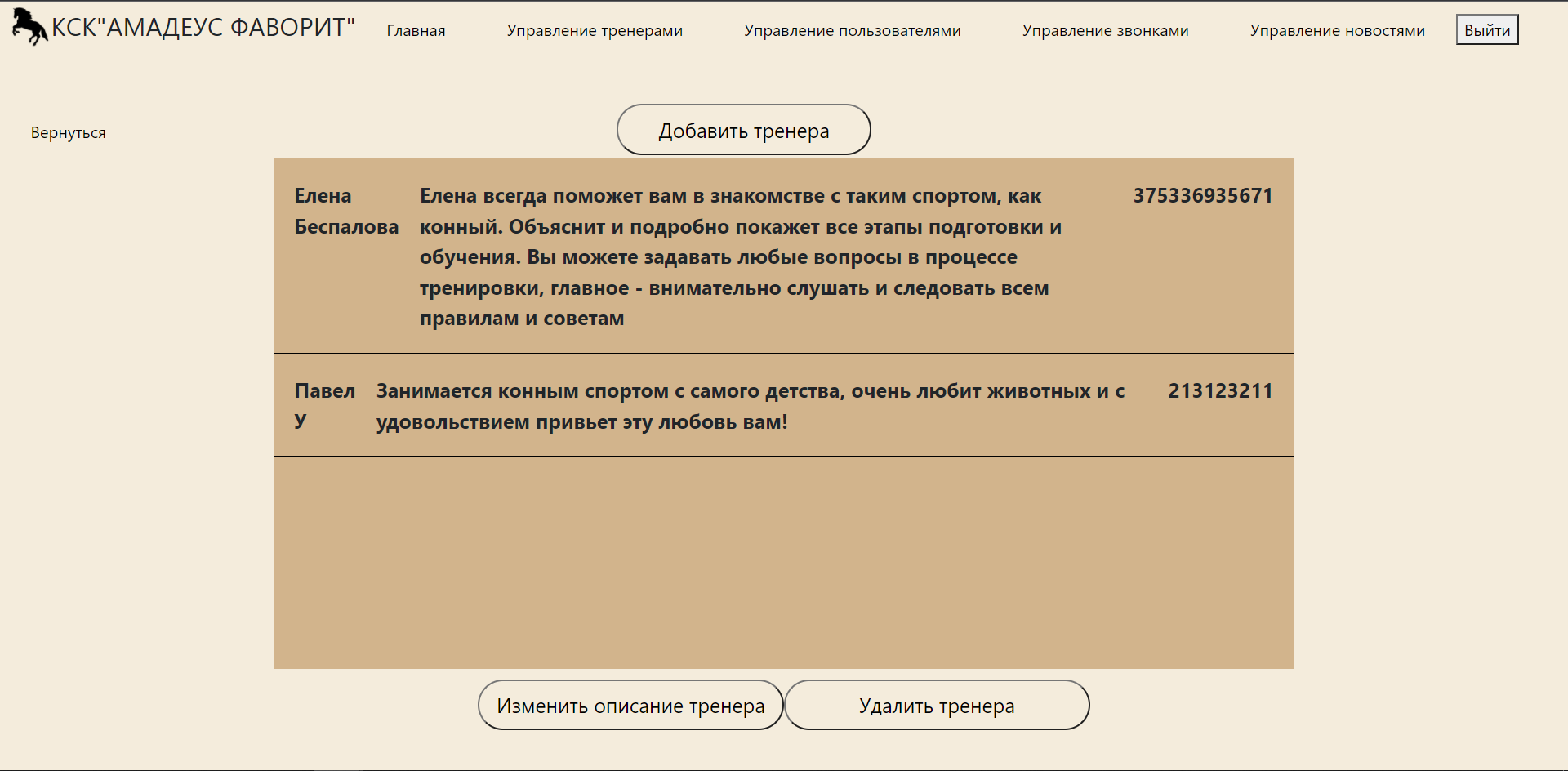


Рисунок 5.6 – Страница администратора для управления тренерами

При нажатии на кнопку для добавления тренера открывается окошко, в котором надо заполнить все поля. Поля в окошке имеют валидацию на количество символов или на определенный язык. Окошко для добавления нового тренера на рисунке 5.7.

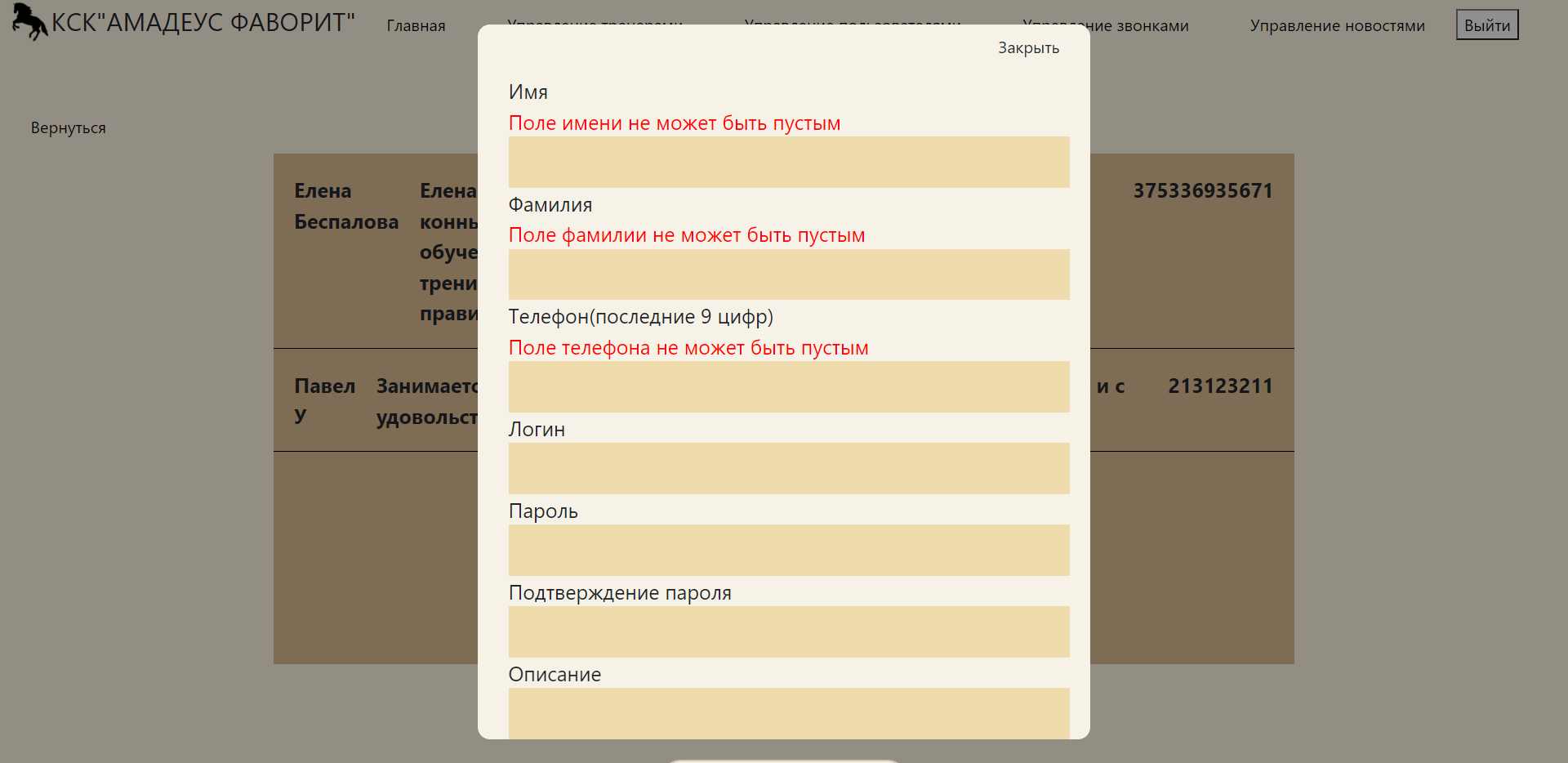


Рисунок 5.7 – Окошко для добавления нового тренера

При нажатии на определенного тренера и затем на кнопку «Изменить описание тренера» появляется окошко, где нужно ввести новое описание, чтобы оно отобразилось в описании выбранного тренера и нажать на кнопку «Добавить». На рисунке 5.8 отображено окошко для изменения описания тренера.

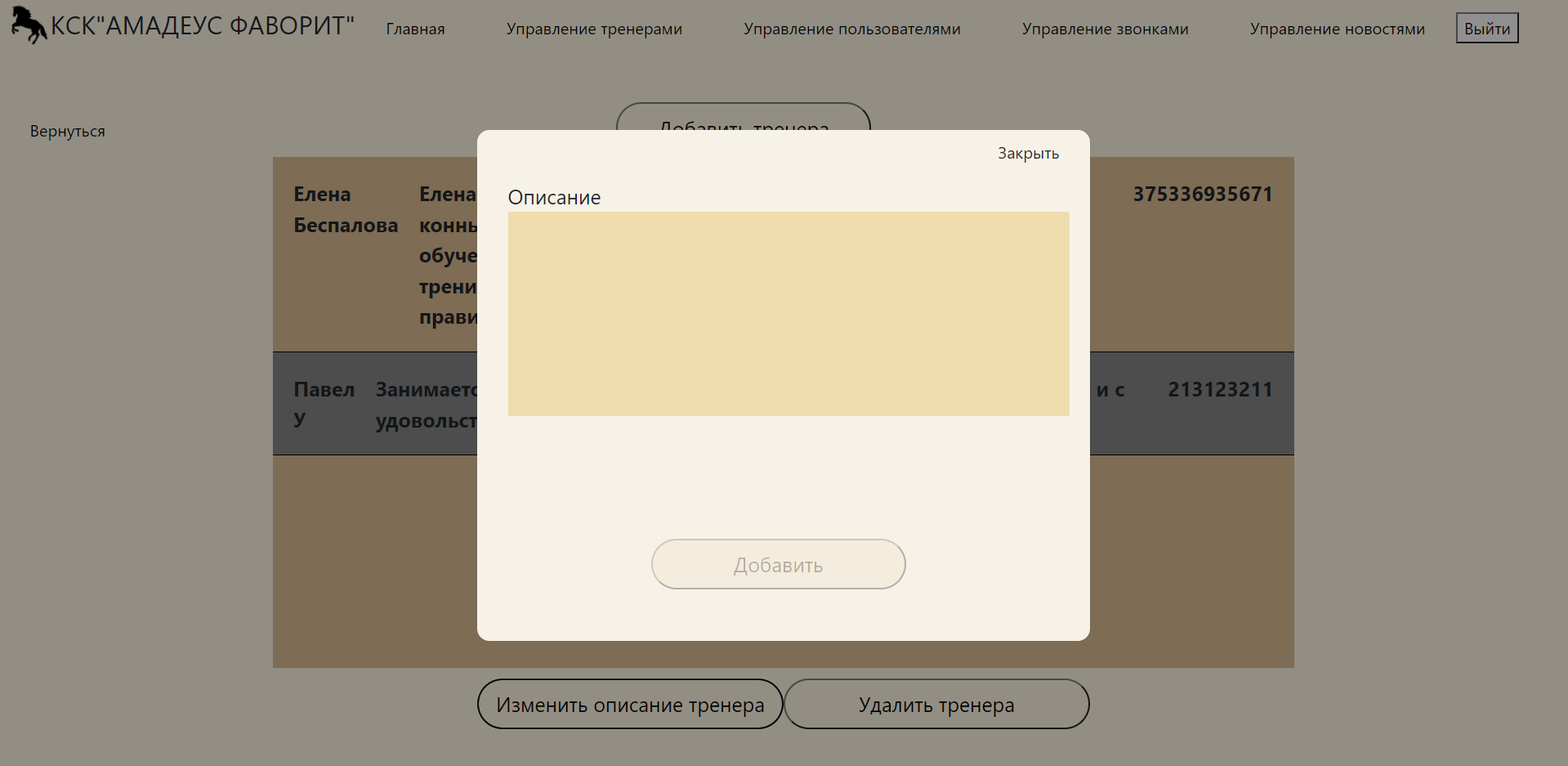


Рисунок 5.8 – Окошко для изменения описания тренера

При нажатии на тренера, а затем на кнопку «Удалить тренера» - выбранный тренер удалится.

При переходе на вкладку «Управления пользователями» мы видим список всех клиентов, которые зарегистрированы в системе. Здесь аналогичным образом мы можем добавить нового клиента, удалить выбранного и изменить информацию выбранного клиента. Страница администратора для управления пользователями представлена на рисунке 5.9.



Рисунок 5.9 – Страница администратора для управления пользователями

Вкладка «Управление звонками» демонстрирует список заявок на звонок. Можно выбрать определенную заявку и удалить, если она больше не актуальна. На рисунке 5.10 изображена страница для управления звонками.

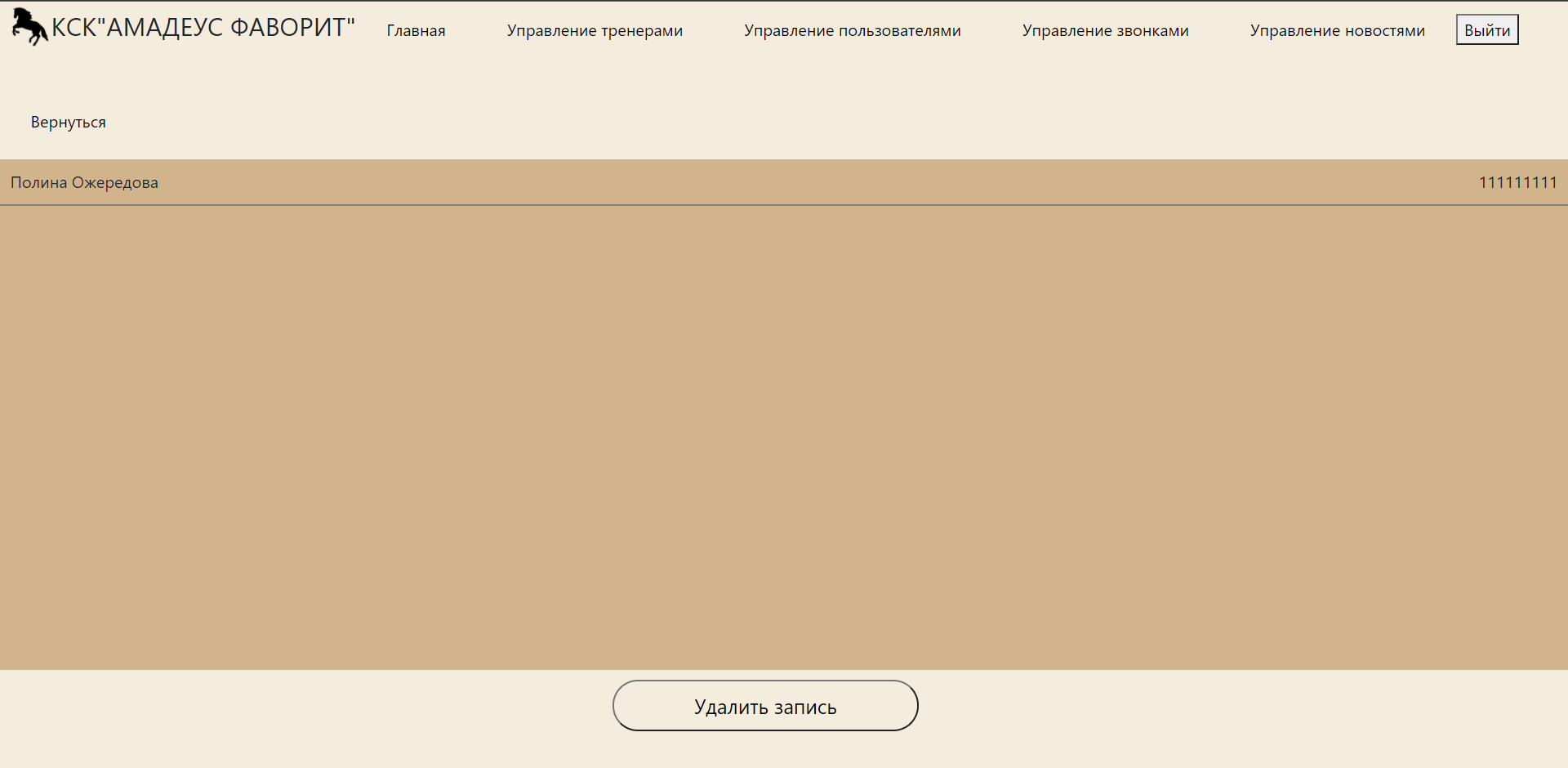


Рисунок 5.10 – Страница для управления звонками

При переходе на вкладку «Управление новостями» мы также получаем список. Это список уже добавленных новостей. При нажатии на кнопку «Добавить новость» отображается окошко для добавления новой новости. Здесь надо ввести заголовок новости и ее основной текст. Окошко для добавление новой новости на странице «Управление новостями» представлено на рисунке 5.11.

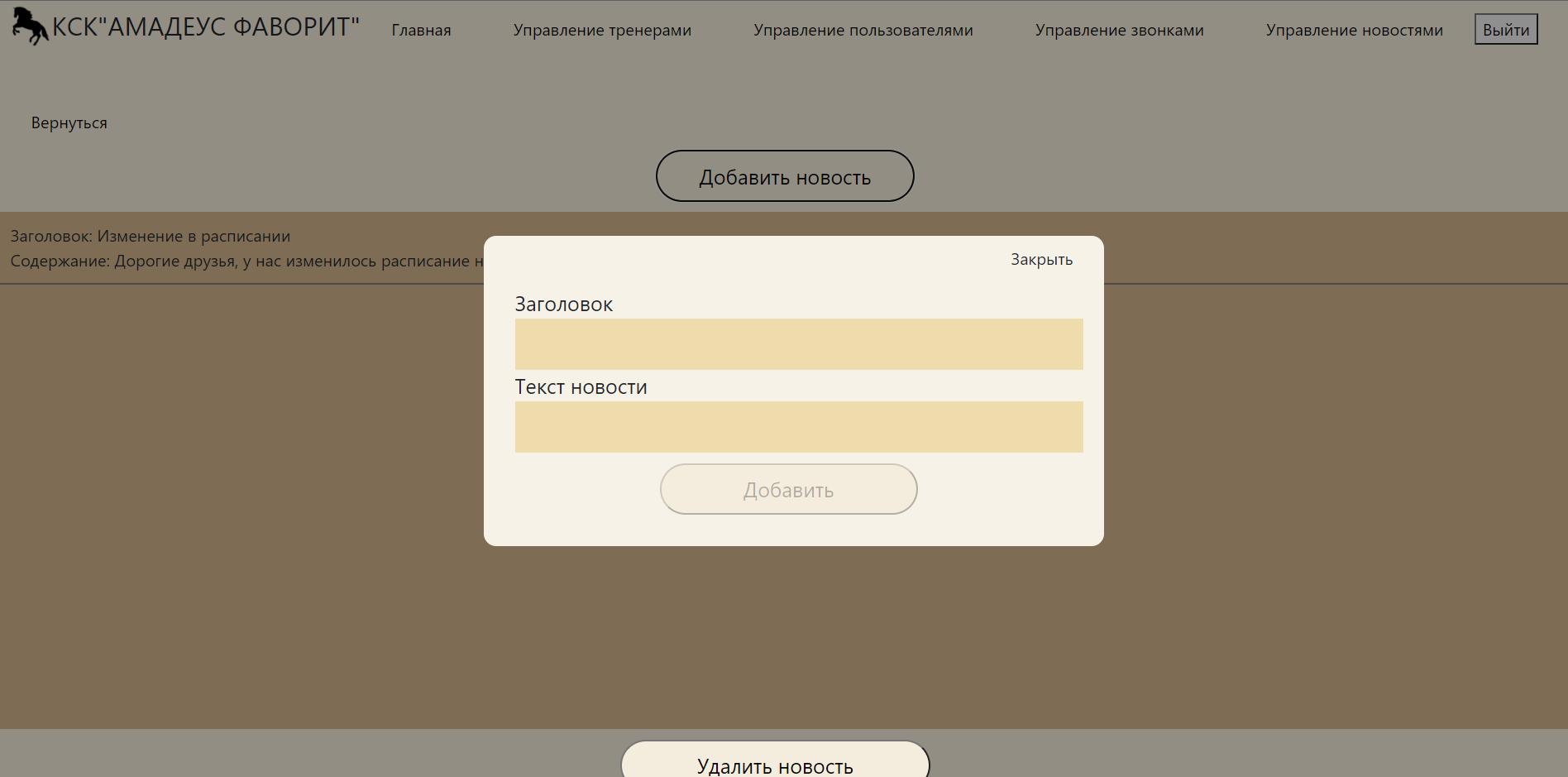


Рисунок 5.11 – Окошко для добавление новой новости

При нажатии на кнопку выйти с любого аккаунта и под любой ролью отображается страница регистрации/авторизации. Вид страницы регистрации/авторизации показан на рисунке 5.12.

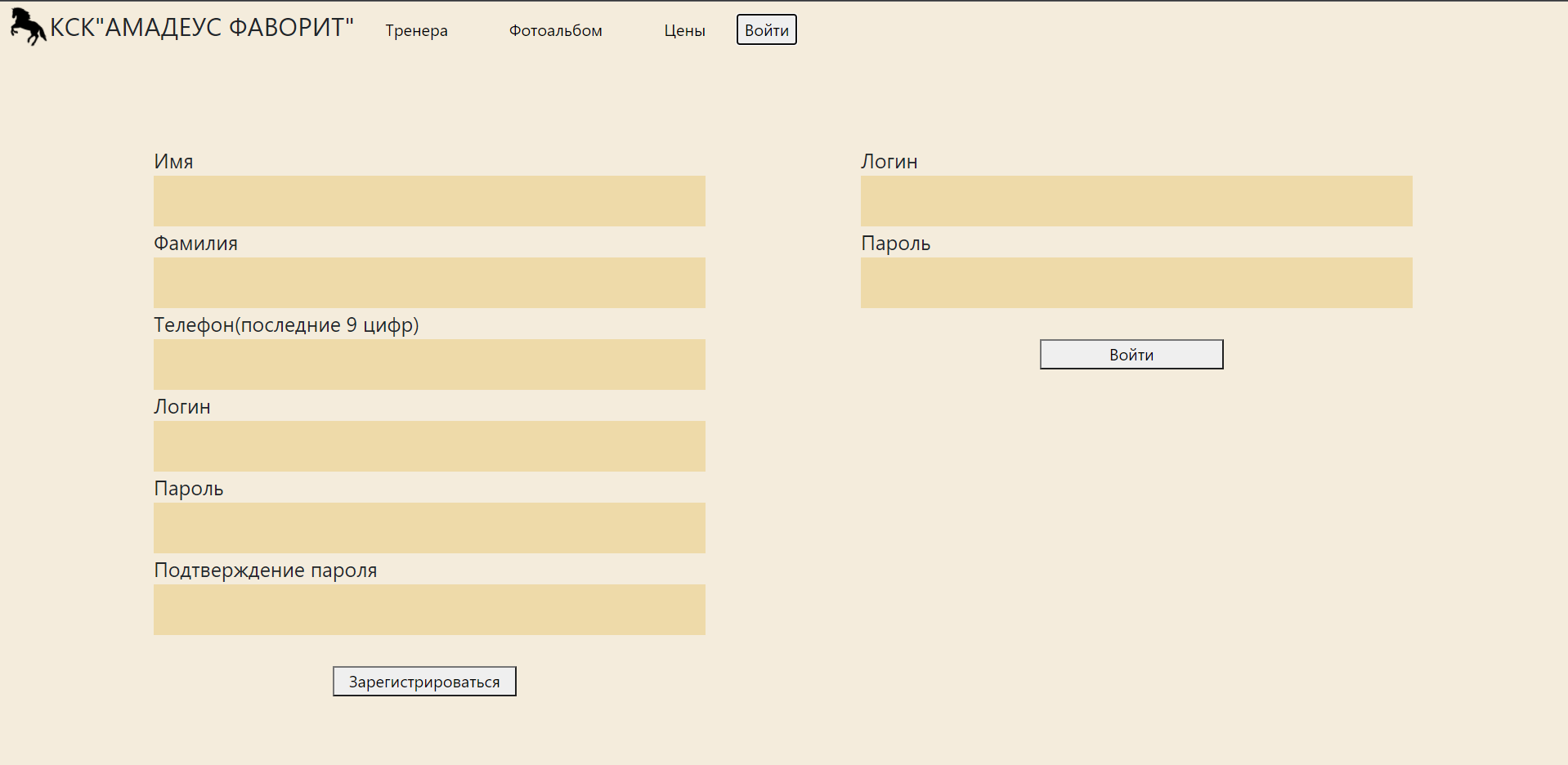


Рисунок 5.12 – Страница регистрации/авторизации

После выхода из аккаунта у пользователя есть возможность авторизоваться снова и продолжить работу.

Такой вид приложения удобен для отслеживания тренировок и записи на них. Клиентам предлагаются возможные варианты тренировок в зависимости от их уровня и заполненности записи.

# Заключение

Разработанный проект состоит из сервера, базы данных и клиентской части. Сервер реализован на фреймворке ASP.NET Core. Для передачи данных были использованы протоколы HTTP. Благодаря тому что при разработке был использован подход REST, клиентская часть слабо связана с сервером, что дает возможность легко заменить графический интерфейс приложения. Клиентская часть приложения разработана в виде многостраничного приложения, которое по мере необходимости загружает данные с сервера и обновляет интерфейс пользователя за счет использования библиотеки React. Валидация выполняется как на стороне сервера, так и на стороне клиента, это снижает возможность пользователю нарушить работу сервера. Взаимодействие с базой данных осуществляется при помощи Entity Framework, что позволяет легко работать с базой данных.

В результате курсовой работы было разработано приложение, которое реализует определенный функционал, а именно:

* поддерживание возможности регистрации и входа в приложение;
* предоставление ролей: пользователь, тренер, администратор;
* осуществление записи на тренировку по определенным критериям;
* предоставление общей информации о КСК и тренерах;
* предоставление тренерам возможности просматривать свою запись и добавлять/изменять/удалять записи клиентов;
* поддерживание роли администратора, который может добавлять/изменять/удалять информацию о тренерах;
* поддерживание роли администратора, который может добавлять/изменять/удалять информацию о клиентах;
* поддерживание возможности для клиента заказать звонка, чтобы уточнить подробную информацию, которая не была предоставлена в web-приложении.

Исходя из задач, которые были решены, проект представляет собойприложение для записи на тренировки в конно-спортивный клуб. Приложение было протестировано с помощью различных сценариев его применения, представляет конечный продукт, который готов к использованию.

# Список используемых источников

1. Руководство по ASP.NET Core 5 [Электронный ресурс] – Режим доступа: https://metanit.com / – Дата доступа: 20.12.2021.
2. Metanit [Электронный ресурс] – Режим доступа: https://metanit.com/sharp/entityframeworkcore – Дата доступа: 20.12.2021.
3. React js documentation [Электронный ресурс] – Режим доступа: <https://reactjs.org/docs/> – Дата доступа: 20.12.2021.

# ПРИЛОЖЕНИЕ А

**Конфигурационный класс сервера**

namespace Amadeus

{

public class Startup

{

public Startup(IConfiguration configuration)

{

Configuration = configuration;

}

public IConfiguration Configuration { get; }

// This method gets called by the runtime. Use this method to add services to the container.

public void ConfigureServices(IServiceCollection services)

{

services.AddDbContext<AmadeusContext>();

services.AddHttpsRedirection(options =>

{

options.RedirectStatusCode = StatusCodes.Status307TemporaryRedirect;

options.HttpsPort = 44336;

});

services.AddAuthentication(CookieAuthenticationDefaults.AuthenticationScheme)

.AddCookie(options =>

{

options.LoginPath = new Microsoft.AspNetCore.Http.PathString("/Account/Login");

options.AccessDeniedPath = new Microsoft.AspNetCore.Http.PathString("/Account/Login");

});

services.AddControllers().AddNewtonsoftJson(options => options.UseMemberCasing());

services.AddControllersWithViews();

services.AddCors(); // добавляем сервисы CORS

// In production, the React files will be served from this directory

services.AddSpaStaticFiles(configuration =>

{

configuration.RootPath = "ClientApp/build";

});

}

// This method gets called by the runtime. Use this method to configure the HTTP request pipeline.

public void Configure(IApplicationBuilder app, IWebHostEnvironment env)

{

if (env.IsDevelopment())

{

app.UseDeveloperExceptionPage();

}

else

{

app.UseExceptionHandler("/Error");

// The default HSTS value is 30 days. You may want to change this for production scenarios, see https://aka.ms/aspnetcore-hsts.

app.UseHsts();

}

app.UseDeveloperExceptionPage();

app.UseHttpsRedirection();

app.UseStaticFiles();

app.UseSpaStaticFiles();

app.UseRouting();

app.UseCors(builder => builder.AllowAnyOrigin());

app.UseAuthentication();

app.UseAuthorization();

app.UseEndpoints(endpoints =>

{

endpoints.MapControllerRoute(

name: "default",

pattern: "{controller}/{action=Index}/{id?}");

});

app.UseSpa(spa =>

{

spa.Options.SourcePath = "ClientApp";

if (env.IsDevelopment())

{

spa.UseReactDevelopmentServer(npmScript: "start");

}

});

}

}

}

# ПРИЛОЖЕНИЕ Б

**Метод, где представлены доступные маршруты в зависимости от роли пользователя**

insertRoutes() {

let { userInfo } = this.state;

if (userInfo != null) {

switch (userInfo.role) {

case 1: {

return (

<>

<Route path='/trainings'>

<AddTrainingPage userInfo={userInfo}/>

</Route>

</>

)

break;

}

case 2: {

return (

<>

<Route path='/trainings'>

<AddTrainingPage userInfo={userInfo}/>

</Route>

</>

)

break;

}

case 3: {

return (

<>

<Route path='/admin'>

<Admin />

</Route>

<Route path='/admin\_trainers'>

<AdminTrainers />

</Route>

<Route path='/admin\_users'>

<AdminUsers />

</Route>

<Route path='/admin\_calls'>

<AdminCalls />

</Route>

<Route path='/admin\_schedule'>

<AddSchedule />

</Route>

<Route path='/admin\_news'>

<AdminNews />

</Route>

</>

)

break;

}

default: { return null; }

}

}

return null;

}

# ПРИЛОЖЕНИЕ В

**Метод для отображение списка тренировок в зависимости от роли пользователя**

[HttpGet]

[Route("getTrainings")]

public async Task<IActionResult> GetTrainings()

{

try

{

string tokenData = this.HttpContext.Request.Headers["Authorization"];

if (tokenData == null)

{

return Unauthorized();

}

string token = tokenData.Split(" ")[1];

var login = AccountController.UncodeJwt(token);

int? id = \_context.Users.Where(x => x.Login == login).Select(x => x.Id).FirstOrDefault();

int? role = \_context.Users.Where(x => x.Id == id).Select(x => x.IdRole).FirstOrDefault();

if (role.HasValue)

{

List<Shedule> schedules = new List<Shedule>();

if (role.Value == 3) //Admin

{

schedules = \_context.Shedules.ToList();

}

else if (role.Value == 2) //Trainer

{

schedules = \_context.Shedules.Where(sch => sch.IdTrainer == id.Value).ToList();

}

else if (role.Value == 1) //User

{

List<training> trainings\_elems = \_context.training.Where(t => t.IdUser == id.Value).ToList();

foreach (training training\_el in trainings\_elems)

{

var sch = \_context.Shedules.Where(sch => sch.Id == training\_el.IdShedule).FirstOrDefault();

if (sch != null)

{

schedules.Add(sch);

}

}

}

List<FrontTraining> front\_trainings = new List<FrontTraining>();

foreach (Shedule shedule\_note in schedules)

{

User trainer\_info = \_context.Users.Where(u => u.Id == shedule\_note.IdTrainer).FirstOrDefault();

FrontTraining front\_training\_data = new FrontTraining(shedule\_note.Id,

shedule\_note.Data,

shedule\_note.HoursStart,

shedule\_note.HoursEnd,

trainer\_info.Surname + " " + trainer\_info.Name,

trainer\_info.Phone);

List<training> temp\_trainings = \_context.training.Where(tr => tr.IdShedule == shedule\_note.Id).ToList();

foreach (training training in temp\_trainings)

{

User participant\_info = \_context.Users.Where(u => u.Id == training.IdUser).FirstOrDefault();

FrontTrainingParticipant participant = new FrontTrainingParticipant(participant\_info.Id,

participant\_info.Surname + " " + participant\_info.Name,

participant\_info.Phone,

training.Status,

training.NeedEquipment);

front\_training\_data.Participants.Add(participant);

}

front\_trainings.Add(front\_training\_data);

}

return Json(front\_trainings);

}

return Json(BadRequest(new { errorMsg = "Некорректные данные" }));

}

catch(Exception ex)

{

return Json(BadRequest(new { errorMsg = "Некорректные данные" }));

}

}

# ПРИЛОЖЕНИЕ Г

**Метод для поиска и отображения подходящих тренировок клиенту в зависимости от его уровня и указанных параметров**

[HttpPost]

[Route("searchTrainings")]

public IActionResult SearchTrainings([FromBody] TrainingSearchModel searchModel)

{

if (searchModel == null)

{

return Json(new BadResponse("Некорректные данные"));

}

List<Shedule> schedules = \_context.Shedules.Where(s => s.Data.Value == searchModel.Data &&

s.IdTrainer == searchModel.TrainerId).ToList();

List<FrontTraining> front\_trainings = new List<FrontTraining>();

foreach (Shedule sc in schedules)

{

if (DoesTrainingSuitForUser(sc, searchModel.SearcherId))

{

User trainer\_info = \_context.Users.Where(u => u.Id == sc.IdTrainer).FirstOrDefault();

FrontTraining front\_training\_data = new FrontTraining(sc.Id,

sc.Data,

sc.HoursStart,

sc.HoursEnd,

trainer\_info.Surname + " " + trainer\_info.Name,

trainer\_info.Phone);

List<training> temp\_trainings = \_context.training.Where(tr => tr.IdShedule == sc.Id).ToList();

foreach (training training in temp\_trainings)

{

User participant\_info = \_context.Users.Where(u => u.Id == training.IdUser).FirstOrDefault();

FrontTrainingParticipant participant = new FrontTrainingParticipant(participant\_info.Id,

participant\_info.Surname + " " + participant\_info.Name,

participant\_info.Phone,

training.Status,

training.NeedEquipment);

front\_training\_data.Participants.Add(participant);

}

front\_trainings.Add(front\_training\_data);

}

}

if (front\_trainings.Count == 0)

{

return Json(new BadResponse("Тренировки по заданными параметрами не найдены"));

}

else

{

return Json(front\_trainings);

}

}

bool DoesTrainingSuitForUser(Shedule shedule, int userId)

{

List<training> trainings\_participants = \_context.training.Where(t => t.IdShedule == shedule.Id).ToList();

if (trainings\_participants.Count == 0)

return true;

UsersInformation users\_information = \_context.UsersInformations.Where(u => u.IdUser == userId).FirstOrDefault();

if (users\_information == null)

return false;

string new\_user\_level\_status = users\_information.LevelStatus;

Template current\_template = new Template();

foreach (training training\_participant in trainings\_participants)

{

UsersInformation user\_information = \_context.UsersInformations.Where(u => u.IdUser == training\_participant.IdUser).FirstOrDefault();

AddUserLvlToTemplate(current\_template, user\_information.LevelStatus);

}

AddUserLvlToTemplate(current\_template, new\_user\_level\_status);

Template result\_template = \_context.Templates.Where(t => t.Ozn >= current\_template.Ozn &&

t.Z >= current\_template.Z &&

t.FirstLevel >= current\_template.FirstLevel &&

t.SecondLevel >= current\_template.SecondLevel &&

t.Poni >= current\_template.Poni).FirstOrDefault();

return result\_template != null;

}

void AddUserLvlToTemplate(Template template, string userLvl)

{

switch (userLvl)

{

case "Ozn": template.Ozn++; break;

case "Z": template.Z++; break;

case "1": template.FirstLevel++; break;

case "2": template.SecondLevel++; break;

case "Poni": template.Poni++; break;

default: break;

}

}