**Содержание**

[Введение 4](#_Toc26919817)

[1 Постановка задачи 5](#_Toc26919818)

[1.1 Обзор прототипов 5](#_Toc26919819)

[1.2 Актуальность задачи 6](#_Toc26919820)

[2 Разработка архитектуры проекта 7](#_Toc26919821)

[2.1 Архитектура системы 7](#_Toc26919822)

[2.2 Структура моделей и классов 7](#_Toc26919823)

[3 Разработка функциональной модели и модели данных ПС 9](#_Toc26919824)

[3.1 Проектирование базы данных 9](#_Toc26919825)

[3.2 Разграничение прав доступа 9](#_Toc26919826)

[3.3 Разработка клиентской части приложения 11](#_Toc26919827)

[4 Руководство пользователя 12](#_Toc26919828)

[4.1 Роль «Гость» 12](#_Toc26919829)

[4.2 Роль «Администратор» 13](#_Toc26919830)

[4.3 Роль «Спортсмен» 14](#_Toc26919831)

[4.4 Роль «Тренер» 15](#_Toc26919832)

[4.5 Роль «Доктор» 15](#_Toc26919833)

[4.6 Выводы 16](#_Toc26919834)

[5 Тестирование 17](#_Toc26919835)

[Заключение 19](#_Toc26919836)

[Список используемых источников 20](#_Toc26919837)

[ПРИЛОЖЕНИЕ А 21](#_Toc26919838)

[ПРИЛОЖЕНИЕ Б 24](#_Toc26919839)

[ПРИЛОЖЕНИЕ В 26](#_Toc26919840)

[ПРИЛОЖЕНИЕ Г 27](#_Toc26919841)

[ПРИЛОЖЕНИЕ Д 29](#_Toc26919842)

# **Введение**

В наше время сложно представить какую-либо организацию без своего программного средства или своей автоматизированной системы или веб-приложения.

При разработке веб-приложения «Футбольный клуб» следует руководствоваться потребностями спортсменов, тренеров и врачей, а именно: удобство, простота, интуитивность. Также одна из задач автоматизированной системы – упрощать работу, а не усложнять.

Необходимо сделать приложение, в котором будут чёткие границы между пользователями, которые находятся в структуре клуба для дальнейшей совместной работы. Спортсмены должны относиться к группе спортсменов и выполнять свои функции: просматривать график тренировок, осмотров у врача в своем футбольном клубе. Тренера должны относиться соответственно к группе тренеров и также выполнять свои функции: устанавливать график тренировок и отмечать отсутствующих. В наше время известно, что у футбольных клубов большой штаб с медициной, поэтому в структура клуба должны относиться и врачи, у которых будет своя группа «Medic», в которой они будут выполнять следующие функции: информировать тренерский штаб о состоянии здоровья спортсменов и составлять удобный график посещений процедур и осмотров.

Для создания простого и удобного веб-приложения, мне нужно реализовать регистрацию, чтобы каждый пользователь мог с лёгкостью запомнить свой логин и пароль. Также требуются такие функции поиска, которые упростят работу в приложении и оставят хорошие впечатления у пользователя. Так как штаб футбольного клуба большой, требуется реализовать сортировку штаба по ролям.

Программное средство должно быть выполнено с использованием асинхронного программирования, взаимодействовать с базой данных, реализовано под разными платформами. При разработке приложения будут использоваться:

* ОС Windows 10;
* Интернет;
* Microsoft SQL Server Management Studio 17;
* Visual Studio 2019 (ASP.NET Core 2.2);
* Microsoft Entity Framework Core v2.2.0;
* Angular 7;
* IIS Express.

# **Постановка задачи**

# **Обзор прототипов**

Как показал анализ, аналогов приложений в свободном доступе нет. Есть сайты различных футбольных клубов, где написана основная информация. Можно воспользоваться этим для изучения персонала, спортсменов и тренеров. Первым для обзора оказался сайт футбольного клуба «Динамо-Минск», показанный на рисунке 1.1.

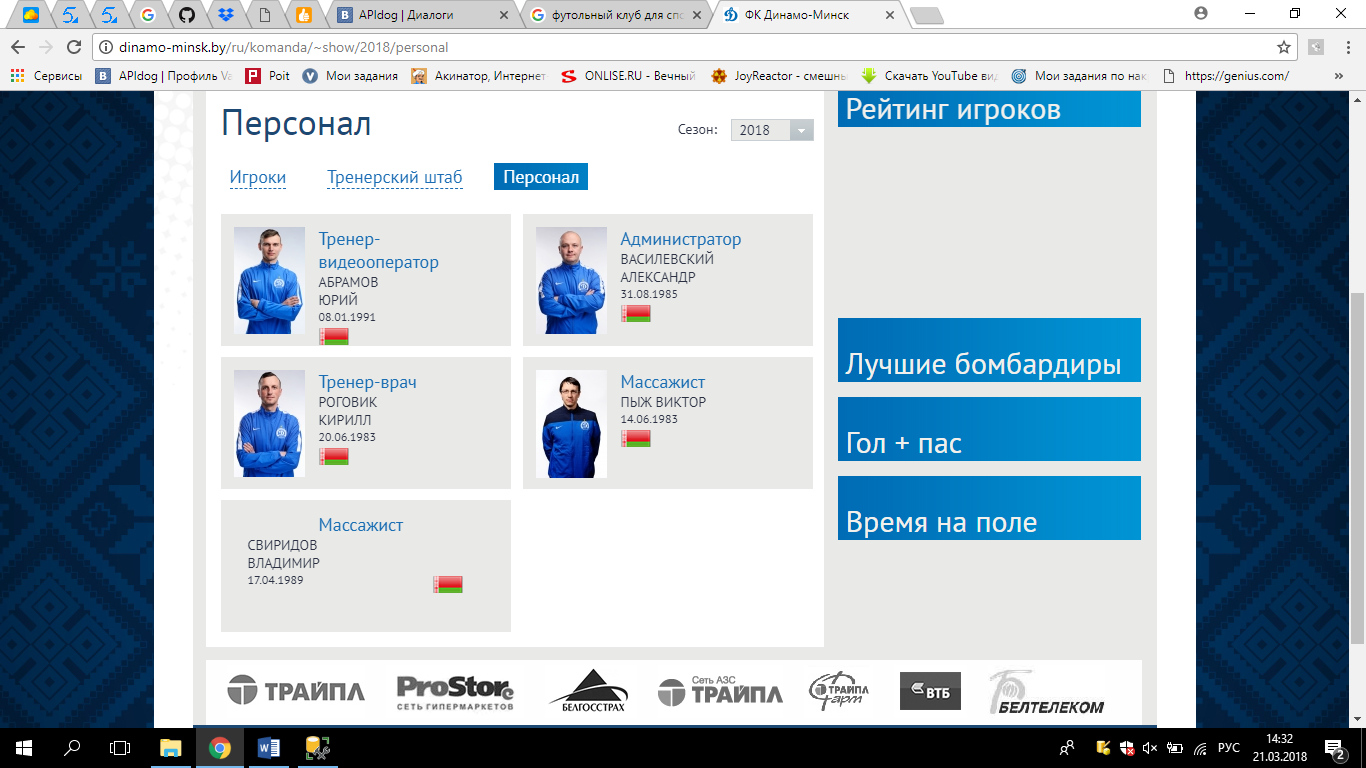


Рисунок 1.1 – Пример «Персонал» сайта ФК «Динамо-Минск»

По сайту футбольного клуба «Динамо-Минск» видно, что не вся информация доступна простому пользователю, то есть человеку, который выполнил авторизацию под ролью «Гость». Гость не может увидеть более подробную информацию по сотруднику или другим членам клуба и поэтому в своем веб-приложении я учту данное предложение, так как моё приложение хоть и будет являться внутриклубным, но всё-таки есть вероятность утечки данных о пользователях, поэтому стоит предотвратить это с помощью одобрения заявок пользователей администратором веб-приложения.

Вторым по счету оказался сайт футбольного клуба «БАТЭ». Дизайн выполнен в приятной цветовой схеме, показанный на рисунке 1.2. В данном случае обычный пользователь не видит вообще никакой информации. Это решение является тоже правильным, но всё-таки моим веб-приложением будут пользоваться игроки, тренера и доктора, а они в свою очередь видятся очень часто и атмосфера в клубе дружеская, то на мой взгляд остальные участники клуба должны знать когда день рождения у одноклубника.

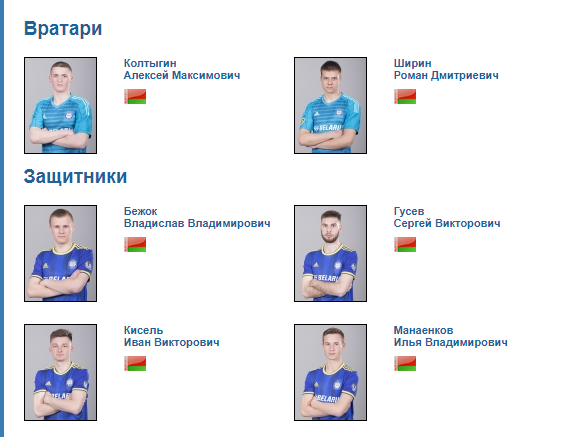


Рисунок 1.2 – Пример «Дублирующий состав» сайта ФК «БАТЭ»

Данной информации мне достаточно для дальнейшей разработки веб-приложения. В каждом из примеров есть фотографии персонала, поэтому целесообразно добавить фотографии и в моё веб-приложение.

# **Актуальность задачи**

Веб-сайты или же веб-приложения набирают всё большую популярность, что соответствует тому, что разработка веб-приложения довольно-таки актуальная задача. Учитывая тот фактор, что уже многие футбольные клубы имеют свои юношеские академии и большой бюджет, это веб-приложение пойдет на пользу всей инфраструктуре футбольного клуба. Для каждой футбольной академии будет выбираться свой путь развития приложения, добавления своих дополнительных функций и решения актуальных проблем внутри клуба. С помощью данного веб-приложения игроки с лёгкостью смогут узнать, когда у них была последняя тренировка, доктора узнают состояние здоровья игрока на последнем обследовании, а тренер воспользуется всеми данными в структурированном виде и сможет оценить готовность игрока к ближайшей игре. Поэтому это приложение даст возможность оперативно принимать решения в различных ситуациях и избегать недопонимания между персоналом футбольного клуба или академии.

# **Разработка архитектуры проекта**

# **Архитектура системы**

Для программного средства в курсовом проекте будет использоваться способ разработки программных модулей, основанный на клиентском приложении. В приложении будут 3 части, представленные на рисунке 2.1.



Рисунок 2.1 – Архитектура системы

Клиентская часть, в которой будет происходить взаимодействие пользователя с порталом футбольного клуба. Она имеет графический интерфейс. В клиентской части пользователь сможет получить ограниченную информацию в зависти от его роли:

* гость (самый ограниченный функционал (просмотр минимальных данных, регистрация) и занимает самую низшую ступень иерархии пользователей);
* спортсмен имеет возможность просмотра расширенных данных, но не имеет возможность редактировать их;
* доктор имеет минимальный функционал, а точнее только изменение типа и даты осмотра спортсменов, а также изменения состояния здоровья спортсмена.
* тренер имеет доступ к изменению типа и даты тренировки спортсменов;
* администратор имеет полный контроль над всем веб-приложением. Все функции добавления, удаления, редактирования. Занимает высшую ступень иерархии пользователей.

База данных, в которой будут храниться информация о пользователях.

API. Взаимодействует с базой данных. Далее отдает данные на UI [1].

# **Структура моделей и классов**

Структура WebAPI представлена на рисунке 2.2.

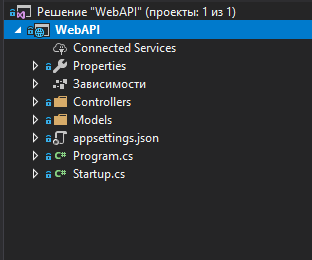


Рисунок 2.2 – Структура проекта WebAPI

В папке Properties в файле launchSettings.json можно установить среду разработки локальной машины. В папке зависимости указаны все установленные NuGet пакеты, которые используются в приложении. В папке Controllers хранятся все контроллеры, используемые в приложении. В папке Models хранятся все модели соответственно. В файле appsettings.json хранится путь подключения к базе данных. В файле Startup.cs описаны основные настройки WebAPI и устанавливается соединение с UI и с базой данных. В приложении А и в приложении Б указаны примеры контроллеров.

Структура проекта – «скелет» любой программы. Он состоит из различных элементов: ресурсы, статические классы, формы и прочее [2]. «Скелет» моего приложения представлен на рисунке 2.3.

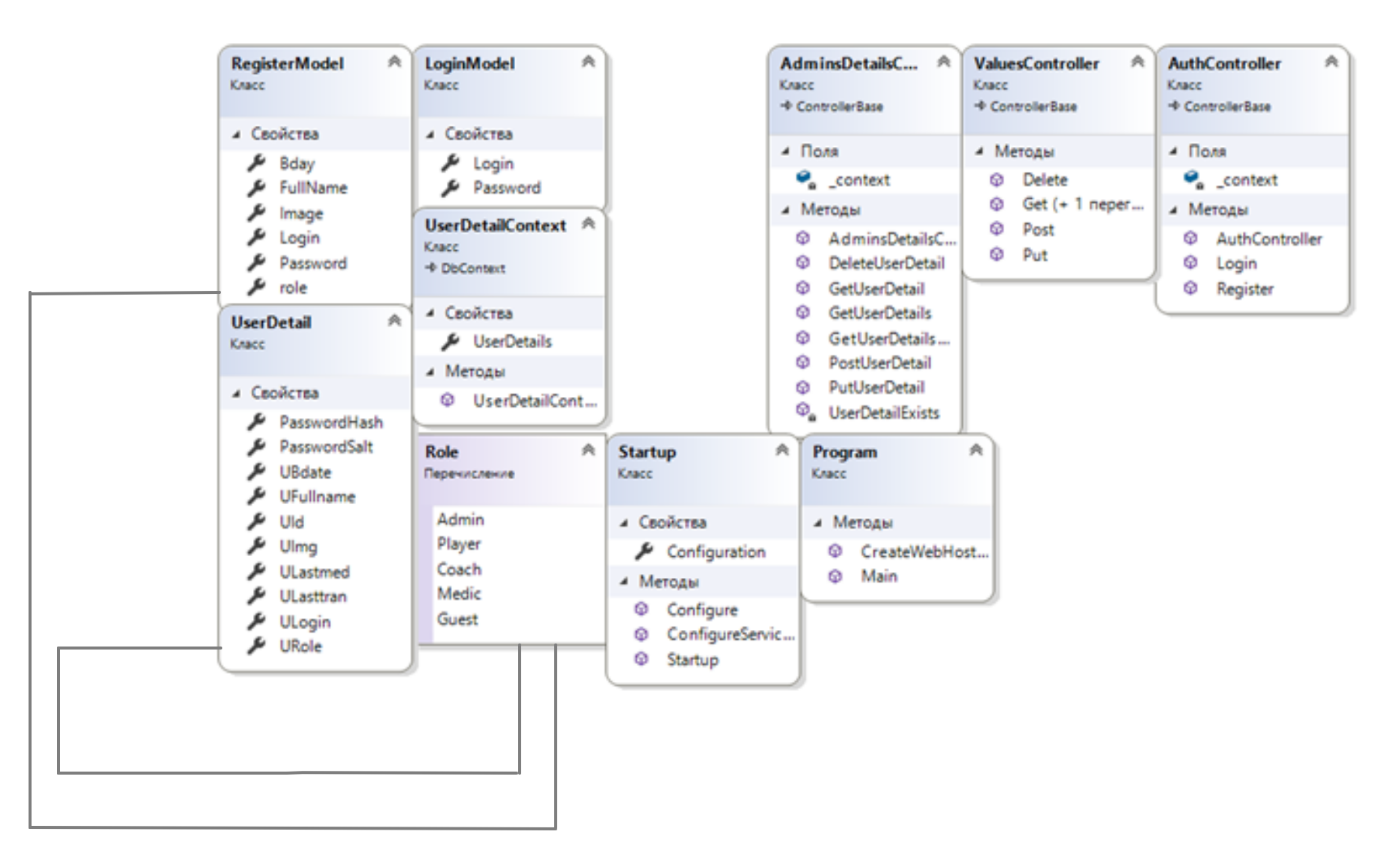


Рисунок 2.3 – Диаграмма классов WebAPI

На рисунке 2.3 представлены все классы, которые используются в WebAPI. Описаны все свойства, методы и поля. В моделе RegisterModel показаны все свойства, которые нужны для регистрации пользователя. LoginModel содержит все данные для авторизации пользователя в систему UserDetail содержит все свойства, которые относятся к авторизованному пользователю и другому персоналу клуба. Класс Role содержит перечисление все ролей для авторизации пользователей. AdminsDetailsController содержит методы, которые нужны для отображения информации на UI. AuthController содержит все методы, которые нужны для авторизации пользователя.

Каждый из компонентов имеет важную роль для работы веб-приложения в целом.

# **Разработка функциональной модели и модели данных ПС**

# **Проектирование базы данных**

Для реализации поставленной задачи была создана база данных CourseW с помощью Entity Framework Core, что значительно облегчило процесс работы с базой данных: не нужно вписывать SQL команды напрямую, а работать непосредственно с LINQ и сущностями (entities).

База даных – совокупность данных, организованных по определенным правилам, предусматривающим общие принципы описания, хранения и манипулоривания данными, независимая от прикладных программ.]

Система управления базами данных – совокупность программ и языковых средств, предназначенных для управления данными в базе данных, ведения базы данных и обеспечения взаимодействия её с прикладными программами [3].

База данных моей автоматизированной системы состоит из 1 таблицы. Схема базы данных изображена на рисунке 2.1.

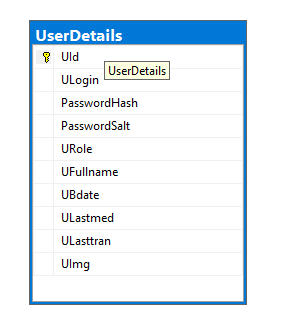


Рисунок 3.1 – Диаграмма базы данных

Поле UId является ключевым полем и уникальным идентификатором пользователя. ULogin соответственно уникальный символьный набор, которые служит отличительной чертой пользователя. PasswordHash и PasswordSalt представляют пароль пользователя в зашифрованном виде для более безопасного хранения данных. URole показывает уровень доступа каждого пользователя в системе. Поле UFullname показывает полное имя пользователя зарегистированного в системе. UBdate показывает дату рождения пользователя. Поля ULastmed и ULasttran специальные поля для редактирования докторами и тренерами. Доступ к редактированию будет определен на UI части. UImg хранит ссылку на изображение пользователя.

# **Разграничение прав доступа**

В моей программной подсистеме существует несколько разновидностей ролей пользователей. На диаграмме вариантов использования, показанной на рисунке 3.2, представлены следующие роли пользователей:

* администратор;
* спортсмен;
* тренер;
* доктор;
* гость.

Администратору предоставляется доступ к странице, где он может удалить аккаунты, которые не относятся к футбольному клуба. Также администратору предоставляется полный доступ к приложению, возможность редактировать или удалять информацию.

Спортсмену предоставляется доступ к просмотру всех полей, но не имеет возможность их редактировать, то есть спортсмен работает с приложением в режиме для чтения и отличается от гостя только тем, что имеет возможность просматривать информацию о последней тренировке и медицинском обследовании.

Тренеру предоставляется доступ к просмотру всех полей, а также редактирование поля последняя тренировка.

Доктору предоставляется доступ к просмотру всех полей, а также редактирование поля последнее медицинское обследование.

Гостю предоставляется доступ только к регистрации и авторизации и просмотру информации об игроках, как в аналогах.

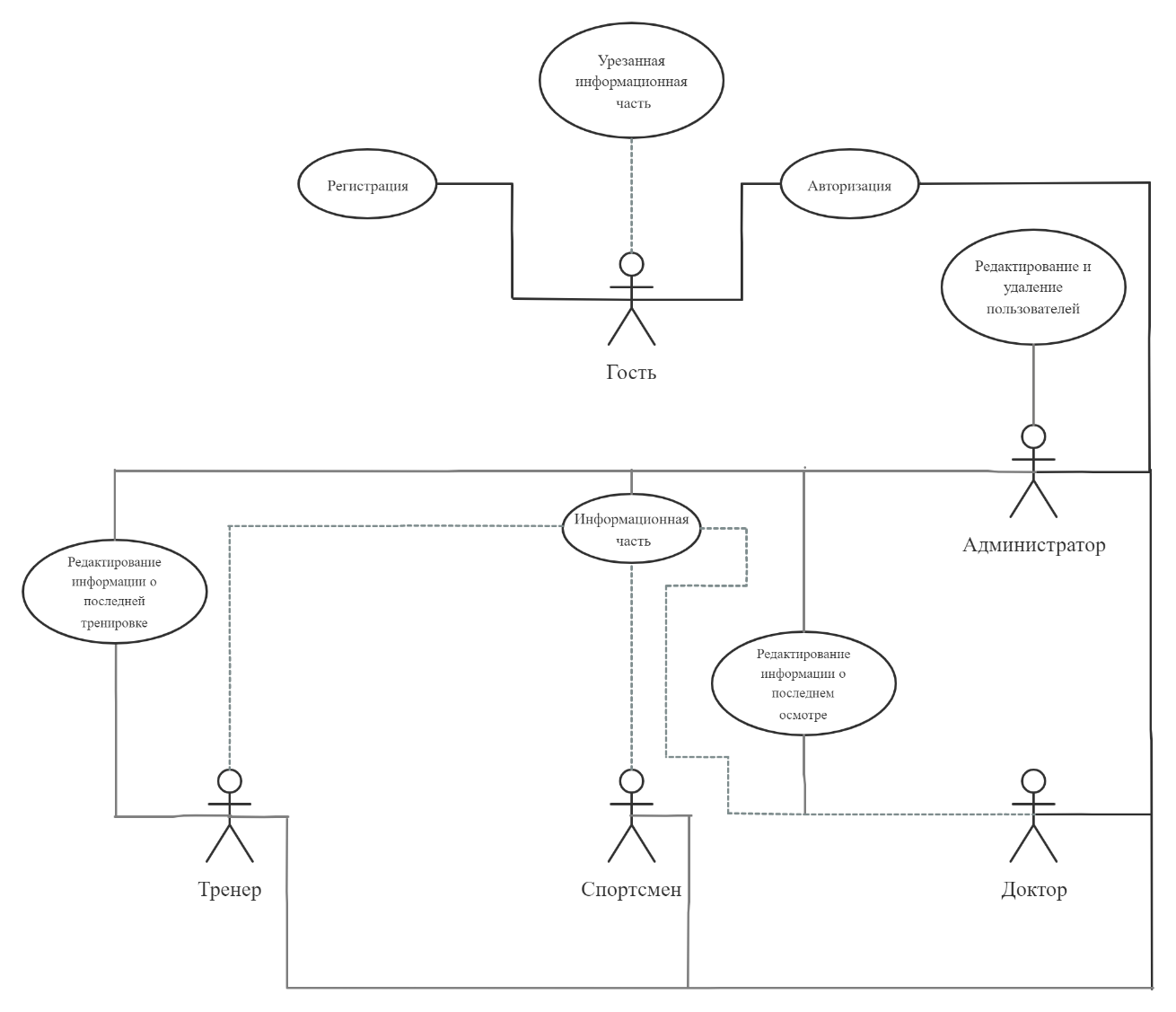


Рисунок 3.2 – Диаграмма взаимосвязи ролей с ресурсами приложения

Так же эта диаграмма отображает различные взаимодействия между ролями и элементами приложения. Система не позволяет без аутентификации пользователя просматривать данные футбольного клуба.

# **Разработка клиентской части приложения**

Клиентский UI разработан на Angular 7.

Структура UI части включает в себя множество модулей. На рисунке 3.3 показаны основные папки клиентской UI.

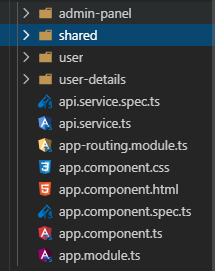


Рисунок 3.3 – Структура проекта в Angular 7

В папке admin-panel описаны стартовая страница после авторизации пользователя. В папке shared описана модель данных, с которой будет последующая работа, а также сервисы для связи с WebAPI. В папки user описаны формы регистрации и авторизации. В папке user-details описаны поля для редактирования, а также поля для просмотра.

Стартовой страницей веб-приложения является страница регистрации и авторизации, так как приложения является только для персонала футбольного клуба и никого более. После регистрации будет перенаправление на главную страницу, в которой будет приветствие пользователя и доступна кнопка в меню навигации «Info». После перехода пользователь получит полный доступ к просмотру данных, а некоторые и к редактированию.

При разработке UI в основном работа проходила с разметкой, в файле .html, ознакомление с основой в приложении Д, а также в файле .ts, о нем информация в приложении В и Г. В файлах формата .html код был реализован на языке гипертекстовой разметки. В файлах .ts код был реализован на языке TypeScript. TypeScript – язык программирования, представленный Microsoft в 2012 году и позиционируемый как средство разработки веб-приложений, расширяющее возможности JavaScript. TypeScript отличается от JavaScript возможностью явного статического назначения типов, поддержкой использования полноценных классов (как в традиционных объектно-ориентированных языках), а также поддержкой подключения модулей, что призвано повысить скорость разработки, облегчить читаемость, рефакторинг и повторное использование кода, помочь осуществлять поиск ошибок на этапе разработки и компиляции, и, возможно, ускорить выполнение программ.

# **Руководство пользователя**

# **Роль «Гость»**

При запуске веб-приложения пользователь попадает на окно авторизации. Если пользователь уже является зарегистрированным, ему необходимо пройти процедуру аутентификации. Процедура аутентификации включает в себя форму, где необходимо указать свой логин и пароль, всё это показано на рисунке 4.1.

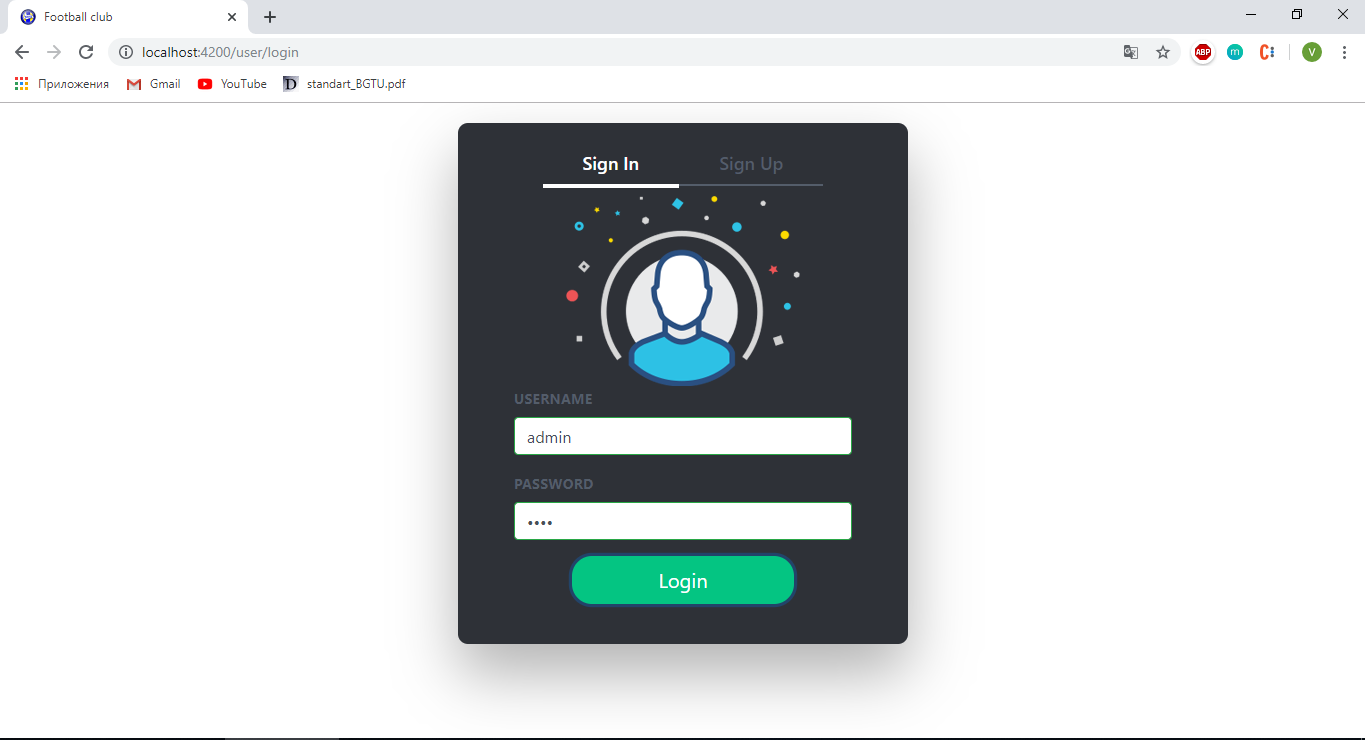


Рисунок 4.1 – Страница авторизации

Если пользователь не зарегистрирован в системе, он может пройти процедуру регистрации. Для того, чтобы зарегистрироваться, пользователю необходимо корректно ввести все необходимые данные в поле регистрации, показано на рисунке 4.2. Зарегистрировавшись, пользователю необходимо дождаться подтверждения администратора.

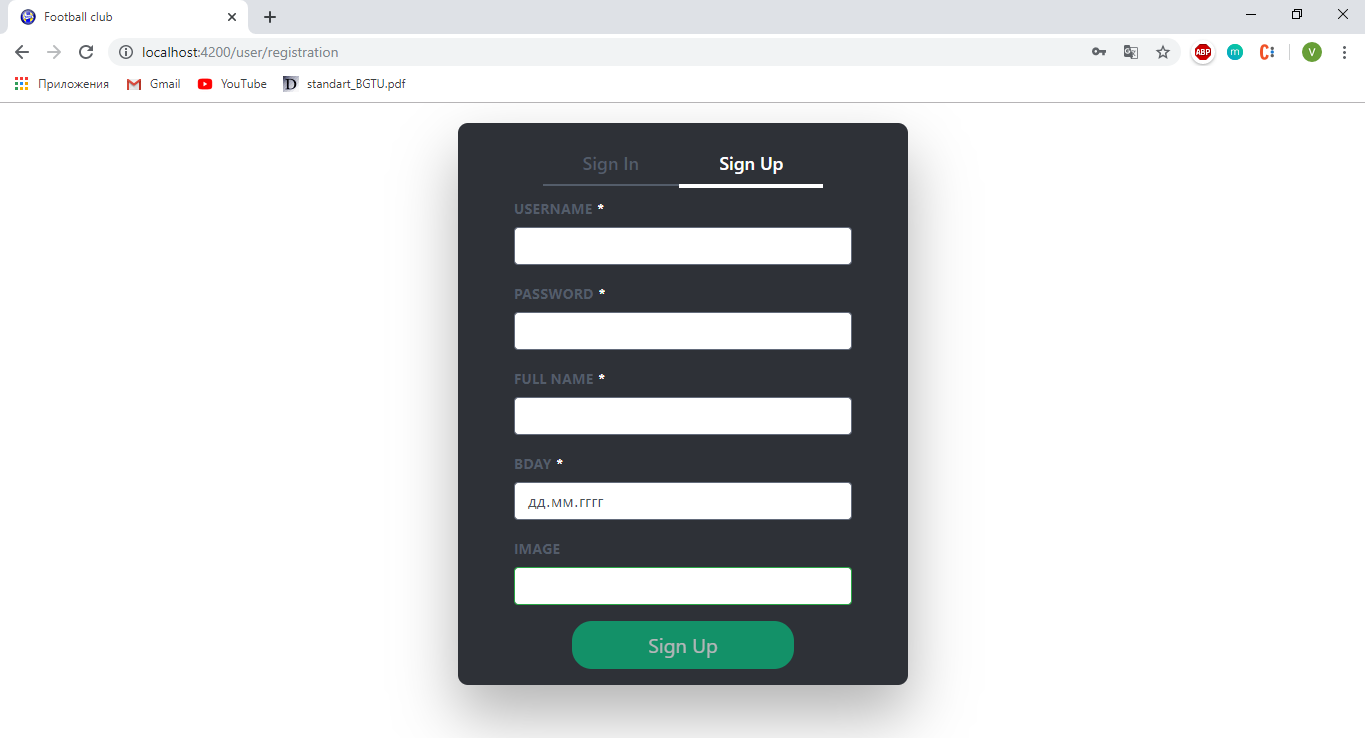


Рисунок 4.2 – Страница регистрации

После успешной авторизации, пользователь попадает на главную страницу. Главная страница показана на рисунке 4.3.

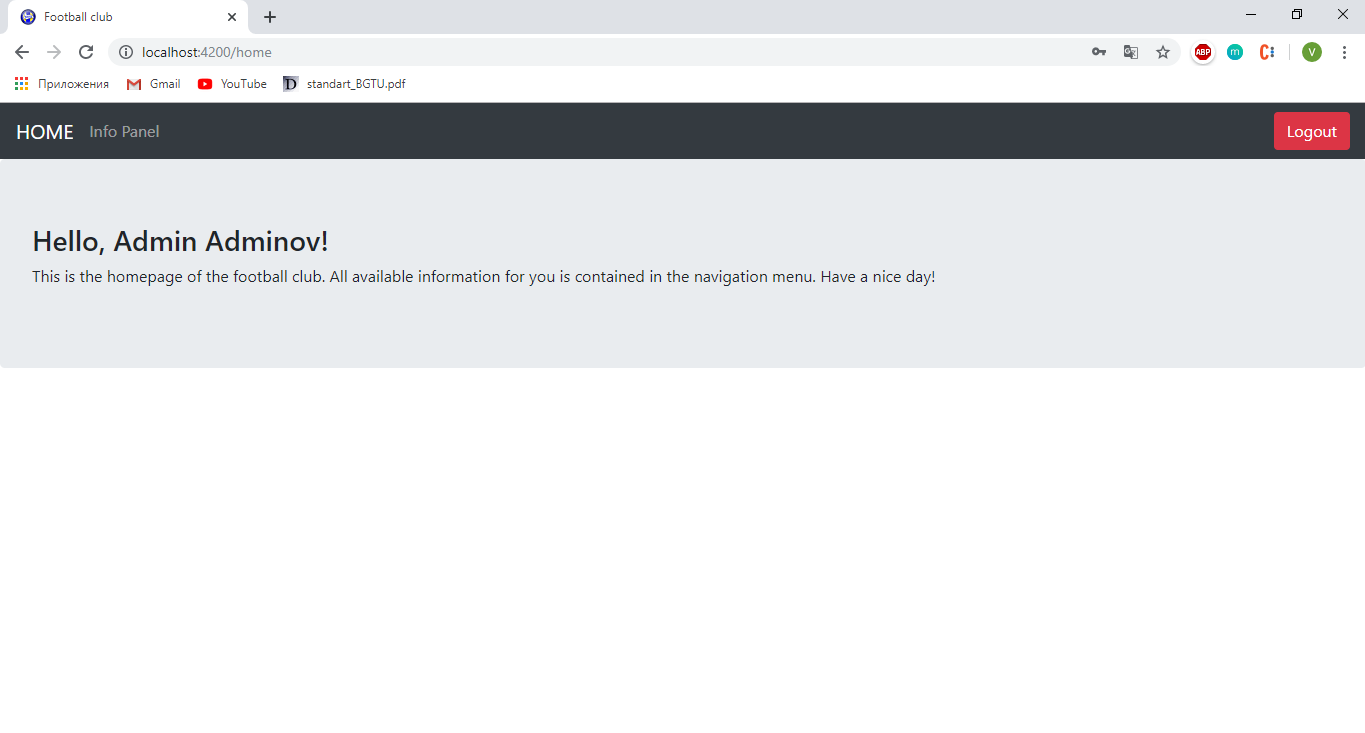


Рисунок 4.3 – Главная страница веб-приложения

На рисунке 4.4 представлена Info Panel для пользователя с ролью «Гость», которая присваивается автоматически при регистрации.

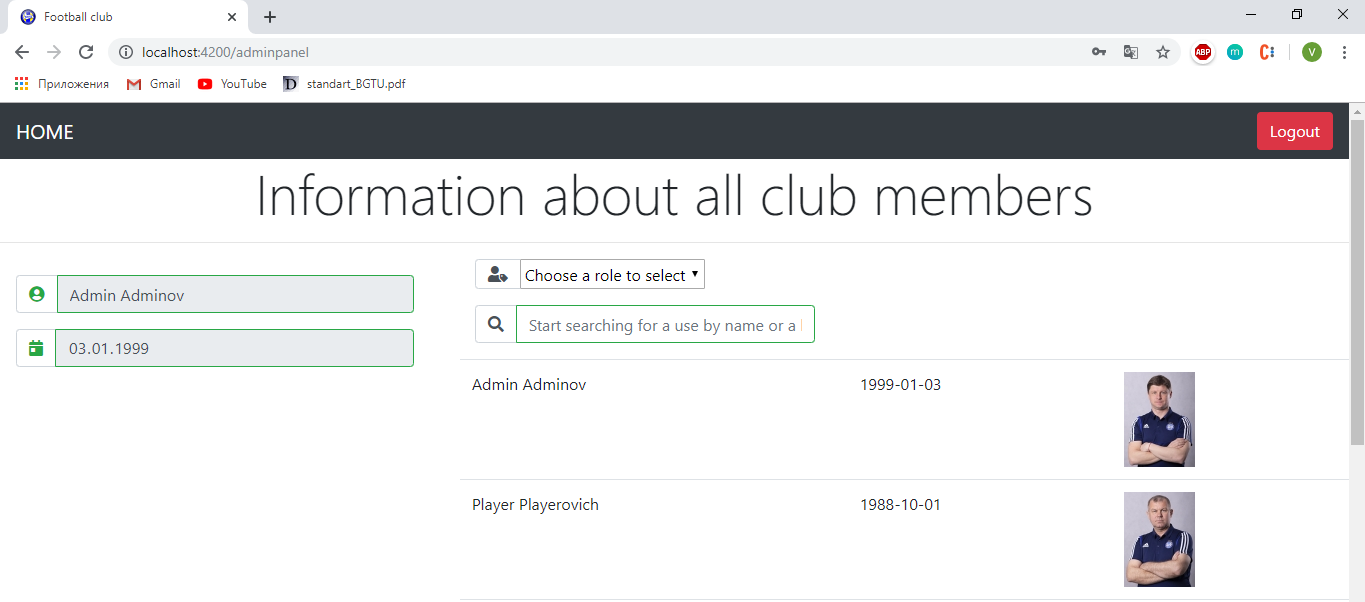


Рисунок 4.4 – Info Panel веб-приложения

Далее мы будем рассматривать веб-приложения под разными ролями и просматривать возможности каждой роли.

# **Роль «Администратор»**

После входа администратору предоставляется доступ к Info Panel, показанным на рисунке 4.5. Администратору доступны такие возможности как:

* поиск по ФИО;
* сортировка по ролям;
* изменение и просмотр информации о каждом пользователе.

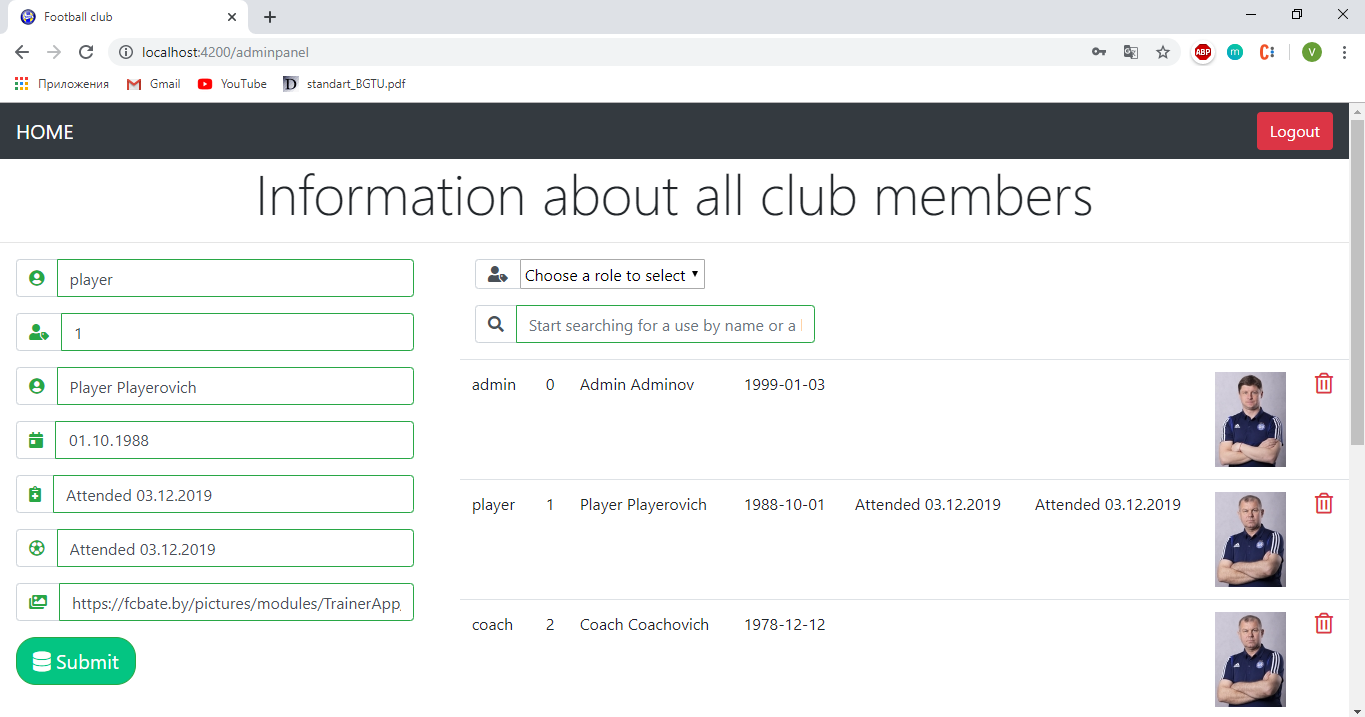


Рисунок 4.5 – Info Panel для администратора веб-приложения

Далее рассмотрим возможности спортсмена.

# **Роль «Спортсмен»**

После входа спортсмену предоставляется доступ к Info Panel, показанным на рисунке 4.6. Спортсмену доступны такие возможности как:

* поиск по ФИО;
* сортировка по ролям;
* расширенная информация о каждом пользователе.

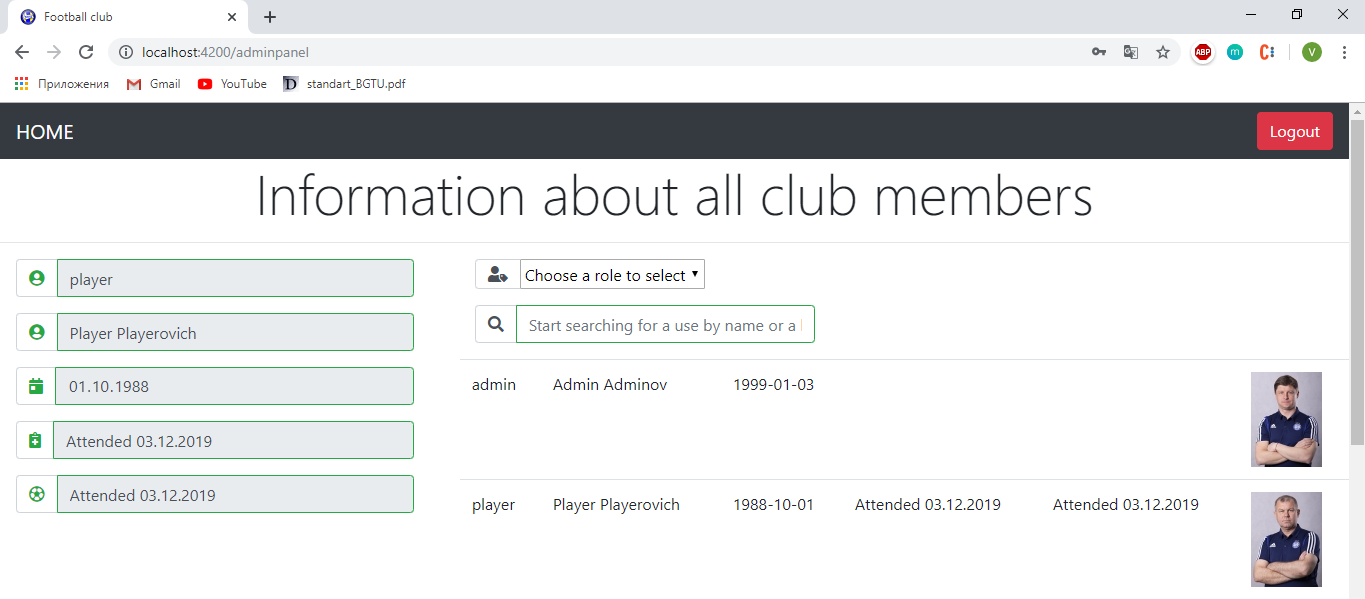


Рисунок 4.6 – Info Panel для спортсмена веб-приложения

Далее рассмотрим возможности тренера.

# **Роль «Тренер»**

После входа администратору предоставляется доступ к Info Panel, показанным на рисунке 4.7. Администратору доступны такие возможности как:

* поиск по ФИО;
* сортировка по ролям;
* возможность редактировать поле с тренировками.

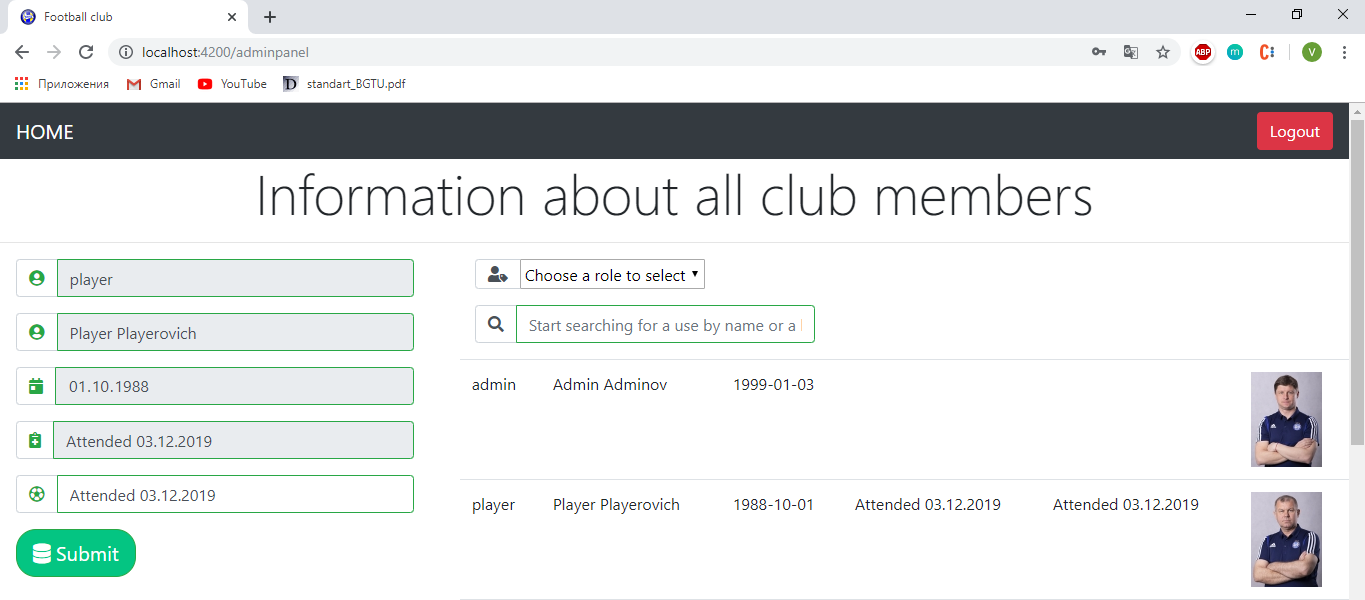


Рисунок 4.7 – Info Panel для тренера веб-приложения

Далее рассмотрим возможности доктора футбольного клуба.

# **Роль «Доктор»**

После входа доктору предоставляется доступ к Info Panel, показанным на рисунке 4.8. Доктору доступны такие возможности как:

* поиск по ФИО и сортировка по ролям;
* расширенная информация о каждом пользователе;
* возможность редактировать поле с медицинскими осмотрами.

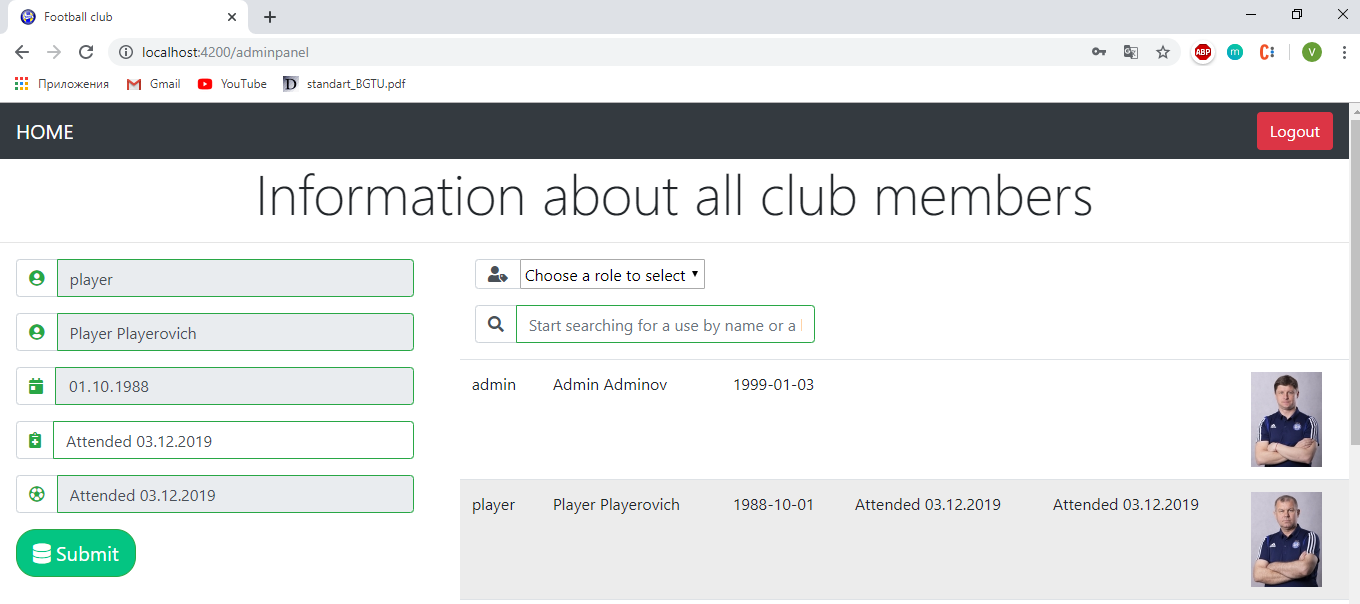


Рисунок 4.8 – Info Panel для доктора веб-приложения

На этом обзор возможностей каждого пользователя окончена.

# **Выводы**

В данной главе был подробно рассмотрен процесс использования разработанного веб-приложения разными типами пользователями, показан функционал системы.

Доступ к страницам приложения определяется назначенными ролями, что не вызывает пересечение.

Администратор отвечает за назначаемые роли зарегистрированными пользователями, а также за состояниями их аккаунтов. Так же администратор может удалять аккаунты, что может быть необходимо, например, при увольнении или смене персонала. Система призвана помочь в передаче срочной информации быстрым путём и всем сразу.

# **Тестирование**

Тестирование производилось на основе ручного тестирования и автоматизации. Для автоматизации был использован Selenium Webdriver.

В самом начале были написаны тест-кейсы. Тест-кейс – это документ, представляющий последовательность определенных действий для достижения ожидаемого результата. Тест-кейс состоит из заголовка, предусловия, шагов, приложения. Заголовок должен отвечать на три вопроса: «что? где? когда?».

На основе написанных тест-кейсов были написано тесты с помощью Selenium Webdriver. Данные тесты писались на языке программирования Java.

Основной набор инструментов и библиотек для автоматизации использовал Selenium Webdriver, Maven, TestNG.

Написанные тесты запускались локально с моего устройства. Обычно тесты запускают с удаленных машин. Для этого используют CI. Наиболее часто встречающаяся CI – это Jenkins.

Использование автоматических тестов экономит огромное количество времени в дальнейшем, потому что, например, регрессионное тестирование может быть покрыто данными тестами, и человек просто запускает тесты и выполняет другие задачи.

Над приложением проводилось три типа ручного тестирования: функциональное тестирование, регрессионное тестирование и тестирование GUI. Функциональное тестирование – это тестирование ПО в целях проверки реализуемости функциональных требований, то есть способности ПО в определённых условиях решать задачи, нужные пользователям.

Регрессионное тестирование – это набор тестов, направленных на обнаружение дефектов в уже протестированных участках приложения. Делается это совсем не для того, чтобы окончательно убедиться в отсутствии багов, а для поиска и исправления регрессионных ошибок. Функциональное тестирование проводилось на всех этапах разработки приложения [4]. Тестирование GUI – тестирование графического интерфейса пользователя [5].

Также были предусмотрены ошибки при регистрации пользователя с существующем логином, показано на рисунке 5.1.

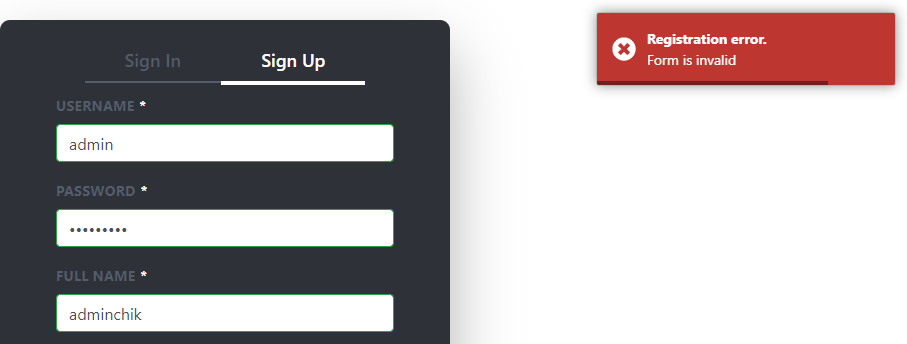


Рисунок 5.1 – Обработка регистрации пользователя с одинаковым логином

Также присутствует обработка ошибки при попытке входа в систему с неправильным логином или паролем. Обработка ошибки представлена на рисунке 5.2.

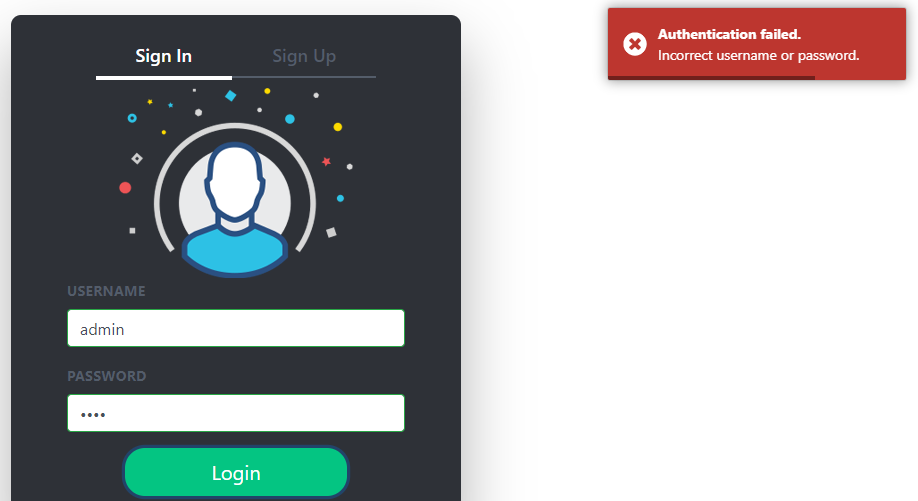


Рисунок 5.2 – Обработка входа пользователя с неправильным паролем

Остальное тестирование проводилось во время разработки приложения и замечания, найденные во время предварительного тестирования были устранены.

На рисунке 5.3 показан график тестирование производительности программного средства.

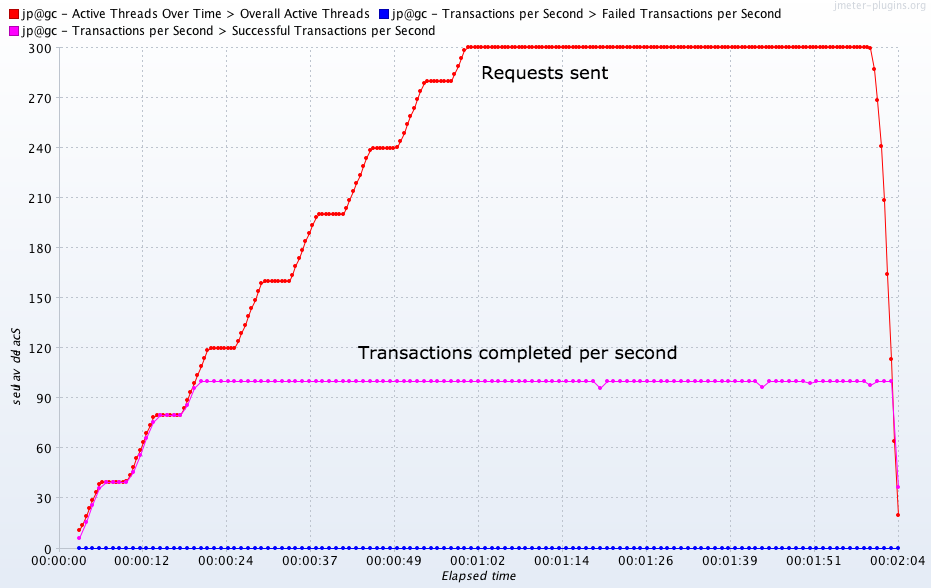


Рисунок 5.3 – Тестирование веб-приложения на производительность

График нагрузочного тестирование показывает, что система способна обработать не более 100 запросов в секунду. Добавление 300 запросов создаст длинную очередь из 200 ожидающих запросов. Данный показатель является приемлемым для моего веб-приложения.

# **Заключение**

Данный проект был разработан для футбольного клуба с максимально простой системой управления.

Главной задачей было сделать простое и удобное приложение. Для этого был использован API-интерфейс и база данных, основанная на принципах Entity Framework Core. Чтобы сделать веб-приложение легко изменяемым и доступным, были использованы максимально удобные функции наведения пользователя. Также был проработан ряд исключительных ситуаций с выводом сообщений, в случае некорректной работы пользователя.

Данное приложение является мои первым опытом создания масштабных проектов, нацеленных на большое количество записей в БД.

В целом, программа получила максимально удобный интерфейс и понятна даже для человека, который мало знаком с компьютером.

Опыт, полученный в ходе создания данного курсового проекта, пригодятся мне и в дальнейшей разработке проектов, созданных на ASP.NET Core в связке с Entity Framework Core и Angular 7 UI.

Считаю, что данное приложение имеет место быть, так как это веб-приложение можно расширить в зависимости от потребностей конкретного футбольного клуба или футбольной академии, а также преобразовать для других спортивных сфер. Приложение не нуждается в дополнительной поддержке, поэтому его могут себе позволить спортивные организации и ограниченным бюджетом.

# **Список используемых источников**

1. MSDN сеть разработчиков в Microsoft [Электронный ресурс] / Microsoft – Режим доступа: <http://msdn.microsoft.com/library/rus/> – Дата доступа: 15.11.2019.
2. METANIT.COM Сайт о программировании [Электронный ресурс] / <http://metanit.com> – Режим доступа: <https://metanit.com>. – Дата доступа: 10.11.2019.
3. Службы SQL Server Reporting Services (SSRS) [Электронный ресурс] / Microsoft – Режим доступа: <https://docs.microsoft.com/ru-ru/sql/reporting-services/create-deploy-and-manage-mobile-and-paginated-reports?view=sql-server-2017>. – Дата доступа: 15.11.2019.
4. Примеры массового импорта и экспорта XML-документов (SQL Server) [Электронный ресурс] / Microsoft – Режим доступа: [https://docs.microsoft.com/ru-ru/sql/relational-databases/import-export/examples-of-bulk-import-and-export-of-xml-documents-sql-server. – Дата доступа 25.11.2019](https://docs.microsoft.com/ru-ru/sql/relational-databases/import-export/examples-of-bulk-import-and-export-of-xml-documents-sql-server.%20–%20Дата%20доступа%2025.11.2019)
5. Professorweb.ru Платформа .NET Framework + Angular 7 [Электронный ресурс] / <http://professorweb.ru> – Режим доступа: <https://professorweb.ru> . – Дата доступа: 10.11.2019.

# **ПРИЛОЖЕНИЕ А**

**Листинг контроллер с данными о пользователях**

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using Microsoft.AspNetCore.Mvc;

using Microsoft.EntityFrameworkCore;

using Microsoft.EntityFrameworkCore.Internal;

using WebAPI.Models;

namespace WebAPI.Controllers

{

[Route("api/[controller]")]

[ApiController]

public class AdminsDetailsController : ControllerBase

{

private readonly UserDetailContext \_context;

public AdminsDetailsController(UserDetailContext context)

{

\_context = context;

}

// GET: api/AdminsDetails

[HttpGet]

public async Task<ActionResult<IEnumerable<UserDetail>>> GetUserDetails()

{

return await \_context.UserDetails.ToListAsync();

}

// GET: api/AdminsDetails/5

[HttpGet("{id}")]

public async Task<ActionResult<UserDetail>> GetUserDetail(int id)

{

var userDetail = await \_context.UserDetails.FindAsync(id);

if (userDetail == null)

{

return NotFound();

}

return userDetail;

}

// GET: api/AdminsDetails/GetUserByRole?=

[HttpGet("GetUserByRole")]

public async Task<ActionResult<UserDetail>> GetUserDetailsByRole(Role role)

{

var roles = await \_context.UserDetails

.Where(p => p.URole == role)

.ToListAsync();

if (roles == null)

{

return NotFound();

}

return Ok(roles);

}

// PUT: api/AdminsDetails/5

[HttpPut("{id}")]

public async Task<IActionResult> PutUserDetail(int id, UserDetail userDetail)

{

if (id != userDetail.UId)

{

return BadRequest();

}

\_context.Entry(userDetail).State = EntityState.Modified;

try

{

await \_context.SaveChangesAsync();

}

catch (DbUpdateConcurrencyException)

{

if (!UserDetailExists(id))

{

return NotFound();

}

else

{

throw;

}

}

return NoContent();

}

// POST: api/AdminsDetails

[HttpPost]

public async Task<ActionResult<UserDetail>> PostUserDetail(UserDetail userDetail)

{

\_context.UserDetails.Add(userDetail);

await \_context.SaveChangesAsync();

return CreatedAtAction("GetUserDetail", new { id = userDetail.UId }, userDetail);

}

// DELETE: api/AdminsDetails/5

[HttpDelete("{id}")]

public async Task<ActionResult<UserDetail>> DeleteUserDetail(int id)

{

var userDetail = await \_context.UserDetails.FindAsync(id);

if (userDetail == null)

{

return NotFound();

}

\_context.UserDetails.Remove(userDetail);

await \_context.SaveChangesAsync();

return userDetail;

}

private bool UserDetailExists(int id)

{

return \_context.UserDetails.Any(e => e.UId == id);

}

}

# **ПРИЛОЖЕНИЕ Б**

**Листинг контроллер для авторизации**

using System;

using System.Linq;

using Microsoft.AspNetCore.Mvc;

using WebAPI.Models;

namespace WebAPI.Controllers

{

[Route("api/[controller]")]

[ApiController]

public class AuthController : ControllerBase

{

private readonly UserDetailContext \_context;

public AuthController(UserDetailContext context)

{

\_context = context;

}

[HttpPost("Register")]

public ActionResult Register(RegisterModel registerModel)

{

if (registerModel == null)

{

return BadRequest("Register model is empty");

}

if (string.IsNullOrWhiteSpace(registerModel.Password) || string.IsNullOrWhiteSpace(registerModel.Login))

{

return BadRequest("wtf");

}

var user = new UserDetail

{

UBdate = registerModel.Bday,

ULogin = registerModel.Login,

URole = Role.Guest,

UFullname = registerModel.FullName,

UImg = registerModel.Image

};

var password\_salt = Guid.NewGuid().ToString();

var password\_hash = PN.Crypt.AES.SHA256Hash(registerModel.Password + password\_salt);

user.PasswordSalt = password\_salt;

user.PasswordHash = password\_hash;

var checklogin = \_context.UserDetails.ToList().FirstOrDefault(p => p.ULogin == registerModel.Login);

if (checklogin != null)

{

return BadRequest("This login is zanyat");

}

\_context.UserDetails.Add(user);

\_context.SaveChanges();

return Ok();

}

[HttpPost("Login")]

public ActionResult Login(LoginModel loginModel)

{

if (loginModel == null)

{

return BadRequest("wtf");

}

if (string.IsNullOrWhiteSpace(loginModel.Login) || string.IsNullOrWhiteSpace(loginModel.Password))

{

return BadRequest("wtf");

}

var user = \_context.UserDetails.ToList().FirstOrDefault(p => p.ULogin == loginModel.Login);

if (user == null)

{

return BadRequest("wtf");

}

if (user.PasswordHash != PN.Crypt.AES.SHA256Hash(loginModel.Password + user.PasswordSalt))

{

return BadRequest("incorrect password");

}

var fullname = user.UFullname;

var role = user.URole;

return Ok(new { role, fullname } );

}

}

}

# **ПРИЛОЖЕНИЕ В**

**Листинг сервиса в Angular 7**

import { UserDetail } from './user-detail.model';

import { Injectable } from '@angular/core';

import { HttpClient, HttpHeaders } from "@angular/common/http";

@Injectable({

providedIn: 'root'

})

export class UserDetailService {

formData: UserDetail;

readonly rootURL = 'http://localhost:52250/api';

list : UserDetail[];

constructor(private http: HttpClient) { }

updateAdminsDetail() {

return this.http.put(this.rootURL + '/AdminsDetails/'+ this.formData.UId, this.formData);

}

deleteUserDetail(id) {

return this.http.delete(this.rootURL + '/AdminsDetails/'+ id);

}

refreshList(){

this.http.get(this.rootURL + '/AdminsDetails')

.toPromise()

.then(res => this.list = res as UserDetail[]);;

}

getUserRoles(role) {

this.http.get(this.rootURL + '/AdminsDetails/GetUserByRole?=' + role)

.toPromise()

.then(res => this.list = res as UserDetail[]);

}

register(form) {

var body = {

Login: form.Login,

Bday: form.Bday,

Image: form.Image,

FullName: form.FullName,

Password: form.Password

};

return this.http.post(this.rootURL + '/auth/register', body);

}

login(formData) {

return this.http.post(this.rootURL + '/auth/login', formData);

}

}

# **ПРИЛОЖЕНИЕ Г**

**Листинг логика для отображения данных для в Angular**

import { Component, OnInit, Directive } from '@angular/core';

import { UserDetailService } from 'src/app/shared/user-detail.service';

import { NgForm } from '@angular/forms';

import { ToastrService } from 'ngx-toastr';

import { Router } from '@angular/router';

@Component({

selector: 'app-user-detail',

templateUrl: './user-detail.component.html',

styles: []

})

export class UserDetailComponent implements OnInit {

userRole;

isReadonly: boolean;

isCoach: boolean;

isMedic: boolean;

isGuest: boolean;

constructor(private service: UserDetailService,

private toastr: ToastrService,

private router: Router) { }

ngOnInit() {

this.resetForm();

this.userRole = localStorage.getItem('Role');

switch(this.userRole){

case "0":

this.isReadonly = false;

this.isMedic = false;

this.isCoach = false;

this.isGuest = false;

break;

case "1":

this.isReadonly = true;

this.isCoach = true;

this.isMedic = true;

this.isGuest = false;

break;

case "2":

this.isReadonly = true;

this.isCoach = false;

this.isMedic = true;

this.isGuest = false;

break;

case "3":

this.isReadonly = true;

this.isCoach = true;

this.isMedic = false;

this.isGuest = false;

break;

case "4":

this.isReadonly = true;

this.isCoach = true;

this.isMedic = true;

this.isGuest = true;

break;

}

if (this.userRole == null) {

this.router.navigate(['/user/login']);

}

}

resetForm(form?: NgForm) {

if (form != null)

form.form.reset();

this.service.formData = {

UId: null,

ULogin: '',

PasswordHash: '',

PasswordSalt: '',

URole: null,

UFullname: '',

UBdate: '',

ULastmed: '',

ULasttran: '',

UImg: ''

}

}

onSubmit(form: NgForm) {

if (this.service.formData.UId == null)

this.toastr.info('You cannot create a new user', 'User Detail Register');

else

this.updateRecord(form);

}

updateRecord(form: NgForm) {

this.service.updateAdminsDetail().subscribe(

res => {

this.resetForm(form);

this.toastr.info('Submitted successfully', 'User Detail Register');

this.service.refreshList();

},

err => {

console.log(err);

}

)

}

}

# **ПРИЛОЖЕНИЕ Д**

**Листинг разметки страницы в Angular 7**

<form #form="ngForm" autocomplete="off" (submit)="onSubmit(form)">

<input type="hidden" name="UId" [value]="service.formData.UId">

<div class="form-group">

<div class="input-group" \*ngIf="!isGuest">

<div class="input-group-prepend">

<div class="input-group-text bg-white">

<i class="fas fa-user-circle" [class.green-icon]="ULogin.valid" [class.red-icon]="ULogin.invalid && ULogin.touched"></i>

</div>

</div>

<input [readonly]="isReadonly" name="ULogin" #ULogin="ngModel" [(ngModel)]="service.formData.ULogin" class="form-control" placeholder="User login" required maxlength="20" minlength="3">

</div>

</div>

<!-- -->

<div class="form-group" \*ngIf="!isReadonly">

<div class="input-group">

<div class="input-group-prepend">

<div class="input-group-text bg-white">

<i class="fas fa-user-tag" [class.green-icon]="URole.valid" [class.red-icon]="URole.invalid && URole.touched"></i>

</div>

</div>

<input [readonly]="isReadonly" name="URole" #URole="ngModel" [(ngModel)]="service.formData.URole" class="form-control" placeholder="User role" required maxlength="1" minlength="1">

</div>

</div>

<!-- -->

<div class="form-group">

<div class="input-group">

<div class="input-group-prepend">

<div class="input-group-text bg-white">

<i class="fas fa-user-circle" [class.green-icon]="UFullname.valid" [class.red-icon]="UFullname.invalid && UFullname.touched"></i>

</div>

</div>

<input [readonly]="isReadonly" name="UFullname" #UFullname="ngModel" [(ngModel)]="service.formData.UFullname" class="form-control" placeholder="Fullname" required maxlength="20" minlength="5">

</div>

</div>

<!-- -->

<div class="form-group">

<div class="input-group">

<div class="input-group-prepend">

<div class="input-group-text bg-white">

<i class="fas fa-calendar-day" [class.green-icon]="UBdate.valid" [class.red-icon]="UBdate.invalid && UBdate.touched"></i>

</div>

</div>

<input type="date" [readonly]="isReadonly" name="UBdate" #UBdate="ngModel" [(ngModel)]="service.formData.UBdate" class="form-control" >

</div>

</div>

<!-- -->

<div class="form-group" \*ngIf="!isGuest">

<div class="input-group">

<div class="input-group-prepend">

<div class="input-group-text bg-white">

<i class="fas fa-notes-medical" [class.green-icon]="ULastmed.valid" [class.red-icon]="ULastmed.invalid && ULastmed.touched"></i>

</div>

</div>

<input [readonly]="isMedic" name="ULastmed" #ULastmed="ngModel" [(ngModel)]="service.formData.ULastmed" class="form-control" placeholder="Attended/Be away && Date">

</div>

</div>

<!-- -->

<div class="form-group" \*ngIf="!isGuest">

<div class="input-group">

<div class="input-group-prepend">

<div class="input-group-text bg-white">

<i class="far fa-futbol" [class.green-icon]="ULasttran.valid" [class.red-icon]="ULasttran.invalid && ULasttran.touched"></i>

</div>

</div>

<input [readonly]="isCoach" name="ULasttran" #ULasttran="ngModel" [(ngModel)]="service.formData.ULasttran" class="form-control" placeholder="Attended/Be away && Date" >

</div>

</div>

<!-- -->

<div class="form-group" \*ngIf="!isReadonly">

<div class="input-group">

<div class="input-group-prepend">

<div class="input-group-text bg-white">

<i class="fas fa-images" [class.green-icon]="UImg.valid" [class.red-icon]="UImg.invalid && UImg.touched"></i>

</div>

</div>

<input [readonly]="isReadonly" name="UImg" #UImg="ngModel" [(ngModel)]="service.formData.UImg" class="form-control" placeholder="You should copy URL image and input text to this box" required minlength="5">

</div>

</div>

<!-- -->

<div class="form-group" \*ngIf="!isReadonly || !isMedic || !isCoach">

<button class="btn btn-success btn-lg bnt-block" type="submit" [disabled]="form.invalid">

<i class="fas fa-database"></i> Submit

</button>

</div>

</form>