МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение высшего образования «БЕЛОРУССКИЙ

ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных технологий

Кафедра Информационных систем и технологий

Специальность 1-40 05 01 Информационные системы и технологии

Направление специальности 1-40 05 01-03 Информационные системы и

технологии (издательско-полиграфический комплекс)

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**КУРСОВОЙ РАБОТЫ**

по дисциплине «Администрирование информационных систем и веб-порталов».

Тема: «Администрирование CRM системы сервисного центра»

Исполнитель

студент 4 курса 1 группы Е. А. Шедько

(подпись) (инициалы, фамилия)

Руководитель

ассистент А .Д. Томко

(подпись) (инициалы, фамилия)

Курсовая работа защищена с оценкой

**Оглавление**

[Введение 3](#_Toc121527652)

[1 Аналитический обзор технологий проекта 4](#_Toc121527653)

[1.1 Этап разработки веб-приложения 5](#_Toc121527654)

[1.2 Администрирование проекта 11](#_Toc121527655)

[1.3 Начало работы 13](#_Toc121527656)

[2 Проверка работоспособности приложения 15](#_Toc121527657)

[Заключение 17](#_Toc121527658)

[Список литературы 18](#_Toc121527659)

# Введение

Развитие человечества в XXI веке разительно отличается от предыдущего. Сегодня в руках человека находятся такие технологии, о которых еще пару лет назад никто и не мог подумать. В настоящий момент в информационном обществе ни один человек не может обойтись без компьютера и сопутствующих информационных технологий. Все современные технологии направлены на то, чтобы облегчить нашу жизнь, обеспечить комфортную работу в любой сфере, дать возможность выбирать, творить и экономить наше время и силы.

Футбол - самая популярная и любимая игра в большинстве стран мира. Сотни тысяч детей, юношей, взрослых играют в футбол. Развитие сети Интернет предоставило человечеству отличную возможность в режиме онлайн получать любую интересующую их информацию из мира футбола. Объем данных в сфере футбола огромный и зачастую не структурирован. И если для обычного фаната этого вида спорта, не представляется сложным найти структурированную ответ на любой интересующих его вопрос из мира футбола, то для «простого человека» получение интересующей его информации может вызвать ряд сложностей.

И в то время пока всю страну захватила футбольная лихорадка, даже люди далекие от футбола стали следить за спортивной жизнью и участвовать в различных спортивных мероприятиях.

Целью данного курсового проекта является создание веб-приложения менеджера футбольных матчей с последующим его администрированием. Приложение имеет следующие функции: обеспечивать возможность регистрации и авторизации; обеспечить пользователям возможность управлять футбольными турнирами/матчами: добавление/редактирование/удаление; позволять пользователю осуществлять поиск и фильтрацию пользователей матчей, турниров; позволять пользователю оставлять/отменять заявку на участие в футбольном турнире/матче;

Основными принципами, положенными в разработку моего программного средства является простота и эффективность. Приложение не должно быть перегружено функциями, информация должна подаваться пользователям структурированной, а интерфейс должен быть интуитивно понятным любому пользователю.

# 1 Аналитический обзор технологий проекта

Немаловажным этапом в разработке программного продукта является аналитический обзор прототипов уже готовых решений в данной области и литературных источников.

После рассмотрения аналогов, можно сделать вывод, что веб-приложение по управлению футбольными матчами должно иметь следующий основной функционал:

* регистрация и авторизация;
* управление футбольными турнирами/матчами:
* поиск и фильтрацию пользователей матчей, турниров;
* позволять пользователю оставлять/отменять заявку на участие в футбольном турнире/матче;
* управление профилем пользователя: изменение личной информации;
* администрирование футбольных матчей/турниров: блокировка/разблокировка/удаление;
* администрирование учетных записей пользователей: блокировка/разблокировка/удаление;

Что бы корректно произвести обзор веб-приложения, как со стороны администрирования, так и разработки, будет реализована следующая структура пояснительной записки:

* этап разработки приложения;
* администрирование проекта;
* установка и начало работы с приложением.

## **Этап разработки веб-приложения**

Для сохранения данных, с которыми работает веб-приложение, необходимо определиться с хранилищем. Для этого идеально подойдут реляционные СУБД, которое позволяют оперативно сохранять и обрабатывать данные разных типов. Мой взгляд упал на СУБД MSSQL. Она является кроссплатформенной и довольно быстрой, а также простой в использовании. Общая структура базы данных представлена на рисунке 1.1 ниже.

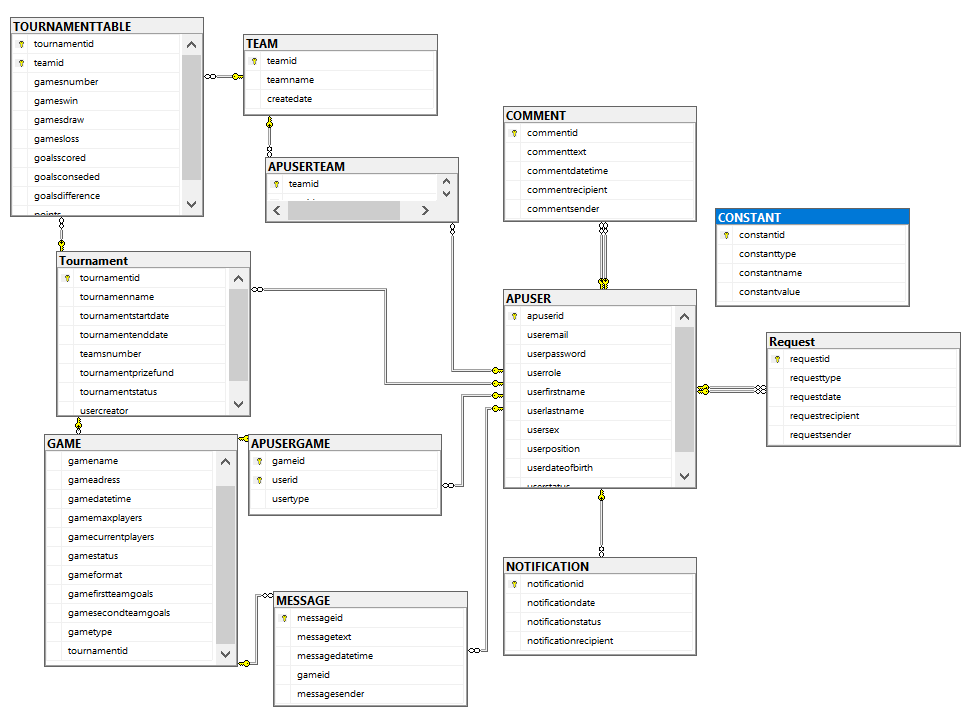


Рисунок 1.1 – Структура базы данных

Можно выделить следующие связи:

* «APUSER – COMMENT»: один ко многим;
* «APUSER – REQUEST»: один ко многим;
* «APUSER – NOTIFICATION»: один ко многим;
* «APUSER – APUSERTEAM»: один ко многим;
* «APUSER –TOURNAMENT: один ко многим;
* «APUSER – APUSERGAME»: один ко многим.
* «APUSER –MESSAGE»: один ко многим.
* «TEAM –APUSERTEAM»: один ко многим.
* «TEAM – TOURNAMENTTABLE»: один ко многим.
* «TOURNAMENT – TOURNAMENTTABLE»: один ко многим.
* «TOURNAMENT – GAME»: один ко многим.
* «GAME – APUSERGAME»: один ко многим.
* «GAME – MESSAGE»: один ко многим.

Следующим шагом является разработка серверной части. Для реализации серверной части использовалась кросс-платформенная технология ASP.NET Core. Язык программирования С#. Первым делом был создан проект типа «Web Api», который предназначен для создания серверной части.

Структура проекта состоит из папок «Properties», «wwwroot», «Controllers», «Model», «Utils» и класса «Program». Всё это можно увидеть на рисунке 1.2.

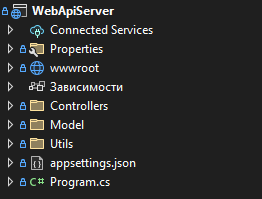


Рисунок 1.2 – Структура проекта WebApiServer

Папка «Model» хранит модели сущностей таблиц из базы данных. Именно с помощью их сервер может взаимодействовать с базой данных. Каждая сущность полностью соответствует таблице. Пример сущности пользователя размещен на рисунке 1.3.

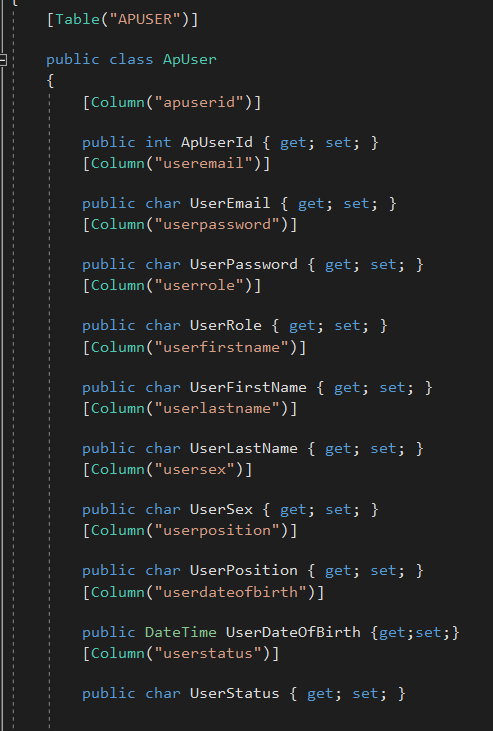


Рисунок 1.3 – Структура класса ApUser

Далее для работы с базой данных был создан класс контекста, а также реализованы паттерны UnitOfWork и Repository. Пример класса контекста изображен на рисунке 1.4:



Рисунок 1.4 – Структура класса AppDBContext

Папка «Controllers» предназначена для обработки запросов от клиента и работы с базой данных. В ней находятся контроллеры для работы с каждой сущностью. Одним из условий проекта было построения приложения, который работает в стиле REST.

REST-архитектура предполагает применение следующих методов или типов запросов HTTP для взаимодействия с сервером:

* get;
* post;
* put;
* delete.

Пример контроллера с методом регистрации предоставлен на рисунке 1.5:

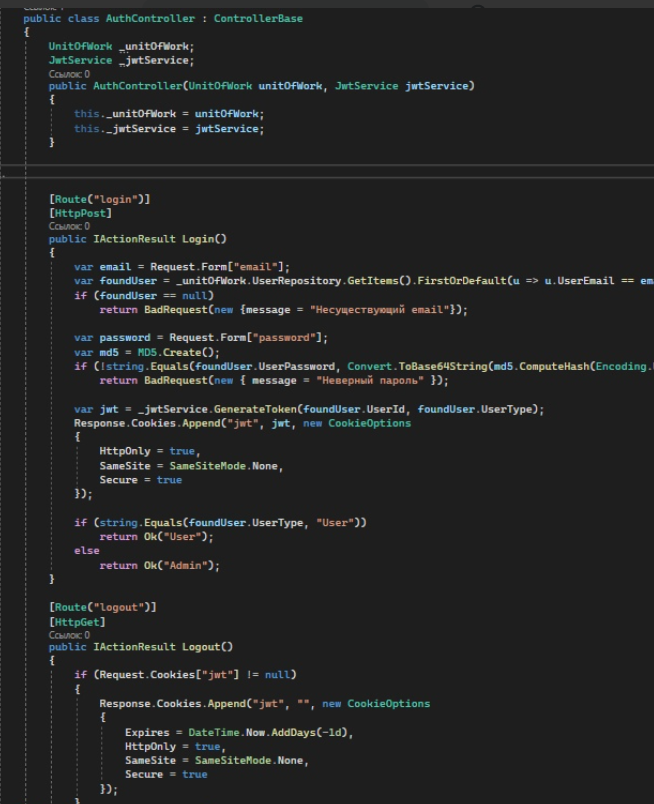


Рисунок 1.5 – Структура контроллера с методом регистрации

Класс «Program» является входной точкой в приложение ASP.NET Core. По сути это уровень конфигурации приложения, определяющий промежуточный слой обработки запросов, регистрирующий зависимости сторонних сервисов. Проект был разработан с помощью среды разработки Microsoft Visual Studio 2022.

Для разработки клиентской части веб-приложения был выбрал прогрессивный фреймворк ReactJS, который написан на языке JavaScript. Проектирование с помощью ReactJS подразумевает создание компонентов, которые взаимодействуют друг с другом. Компоненты ReactJS имеют расширение файла \*.jsx или \*.js. В папке «components» находятся компоненты приложения. Главным компонентом, с которого начинается работа приложения, является index.jsx. В файле package.json подключаются разные библиотеки. Структуру проекта можно увидеть на рисунке 1.6.

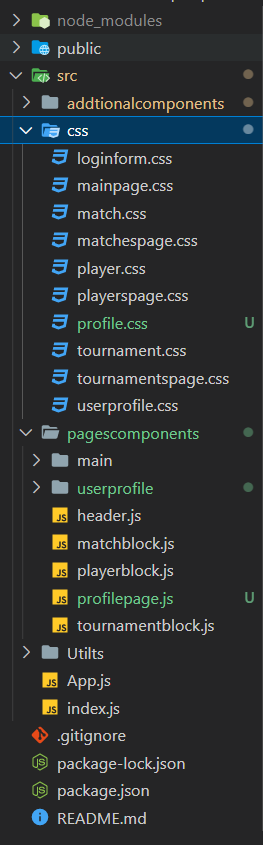


Рисунок 1.6 – Структура проекта пользовательского интерфейса

Так как проектирование с помощью ReactJS подразумевает создание компонентов, то в структуре проекта будут такие директории как:

* public;
* src.

Директория «src» содержит в себе содержит в себе основные поддиректории и файлы:

* additioanlcomponents;
* css;
* pagescomponents;
* App.js
* Index.js

Сам по себе компонент состоит из секций:

* imports;
* render.

Секция «render», которая содержит в себе html разметку, которую видит пользователь, можно увидеть на рисунке 1.7.



Рисунок 1.7 – Секция «render» компонента MatchBlock.js

Компоненты необходимо импортировать из папки и объявить в секции «components».

В секции «imports» подключаются все необходимые зависимости, для создания интерфейса взаимодействия. Структуру секции «imports» можно увидеть на рисунке 1.8.

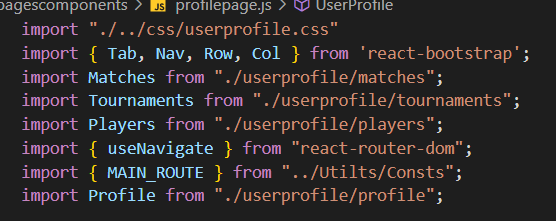


Рисунок 1.8 – Секция «imports»

Чтобы обращаться к серверу, была установлена зависимость «axios», с помощью команды «npm install axios».

Далее, запросы к серверу будут выполняться с помощью JavaScript-библиотеки «axios». Что бы сделать запрос, необходимо указать тип запроса (get, post, put, delete), в параметрах указать url на сервер, заголовки и при необходимости, можно передать данные на сам сервер. Когда приходит ответ от сервера, в переменную response записывается вся необходимая информация. Пример использования можно увидеть на рисунке 1.9



Рисунок 1.9 – Выполнение запроса и получение ответа от сервера

В случае ошибки, используются обработчик «catch». В качестве параметра находится переменная, которая сохраняет в себе ответ сервера в результате ошибки.

1.2 Администрирование проекта

В современном мире существует большое количество сред и оперативных систем, которые имею свою инфраструктуру. Приложения, созданные на одной ОС, вряд ли заработают на другой ОС. Так же, чтобы запустить приложение на той же ОС, на какой создавалось приложение, требуется установить вспомогательные среды разработки, базы данных. Это не эффективно. На мой взгляд, наиболее быстрый и практичный способ является использование Docker.

Docker в настоящее время является одним из самых популярных средств для автоматизации развертывания и управления приложениями в различных средах, например, в операционных системах MacOS, Windows и Linux. Docker одинаково работает независимо от окружающей его среды. Что бы корректно развернуть приложение в контейнере Docker, необходимо написать скрипты развертывания.

Само веб-приложение разработано на базе операционной системы Windows 10. На ней же и будет продолжен процесс настройки администрирования веб-приложения.

Самым удобным способом, благодаря которому можно осуществить поддержку «контейнеризации» приложения, является возможность создания докер-файлов, содержащих в себе базовую конфигурацию. В случае серверной части приложения, Visual Studio имеет встроенную поддержку Docker. Для создания Dockerfile, нужно выбрать «Решение», затем «Добавить» и «Поддержка Docker». По окончанию будет создан Dockerfile со следующим содержимым, приведенным на рисунке 1.10.

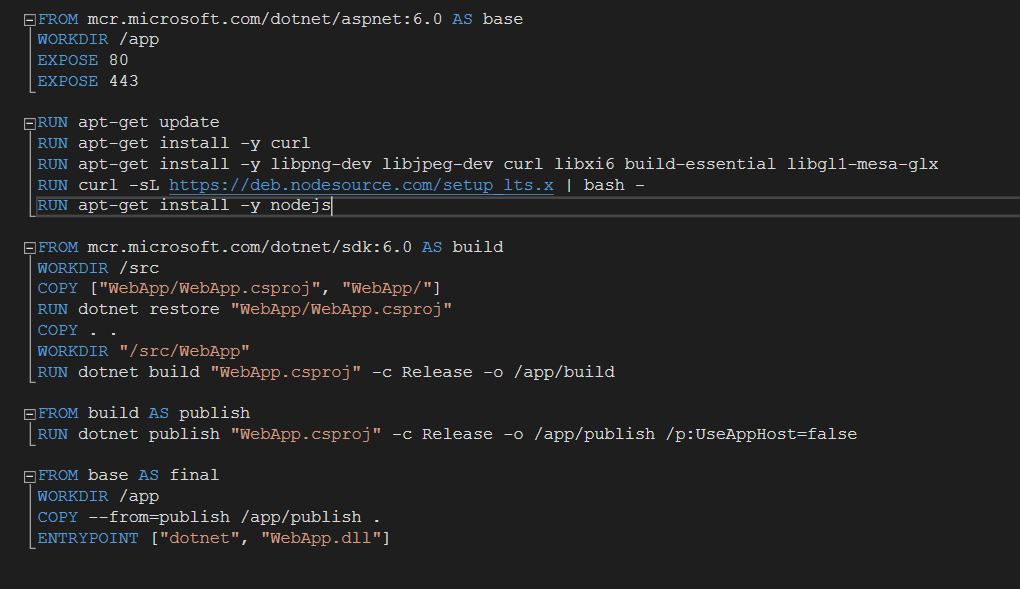


Рисунок 1.10 – Dockerfile серверной части проекта

Здесь можно выделить следующий порядок действий:

* в начале мы качаем образ (image) из DockerHub самой платформы ASP.Net и делаем публикацию папки app на порт 80 и 443 соответственно (для поддержки протоколов HTTP и HTTPS);
* затем мы берем все содержимое из наших исходных файлов, копируем csproj-файл для восстановления зависимостей в папку src/WebApp. После чего запускается сам процесс восстановления зависимостей командой RUN dotnet restore…;
* после того, как были загружены все пакеты из Nuget-репозитория, происходит публикация приложения в путь /app/publish;
* перед самим этапом запуска приложения мы копируем содержимое в папку app для публикации, работа приложения будет осуществляться на раннее описанных портах;
* и последним этапом остается запуск корневой DLL-библиотеки при помощи .Net CLI.

Следующим шагом будет создание файла Dockerfile для развертки клиентской части приложения. Разработчики фреймворка ReactJS позаботились о своих пользователях и разместили на своем сайте стриктуру данного файла, которая отображена на рисунке 1.11.

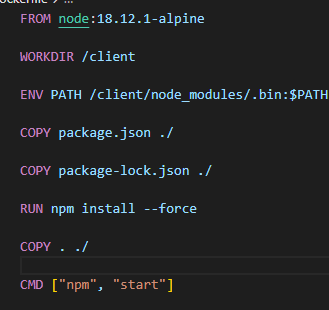


Рисунок 1.11 – Dockerfile клиентской части проекта

Порядок действий следующий:

* в начале мы загружаем образ из DockerHub платформы NodeJS типа Alpine. lts-alpine является более легковесным, примерно 5 Мегабайт, и быстрым образом для создания контейнеров
* далее устанавливаем простой HTTP-сервер для статистики;
* делаем каталог «client» текущим рабочим каталогом;
* копируем оба файла «package.json» и «package-lock.json» если он есть;
* устанавливаем зависимости проекта;
* копируем файлы и каталоги проекта в текущий рабочий каталог;
* собираем приложение.

1.3 Начало работы

Для работы с контейнером Docker нужна сама среда Docker. Для этого нужно с официального сайта скачать установщик Docker Desktop. Процесс установки показан на рисунке 1.12.

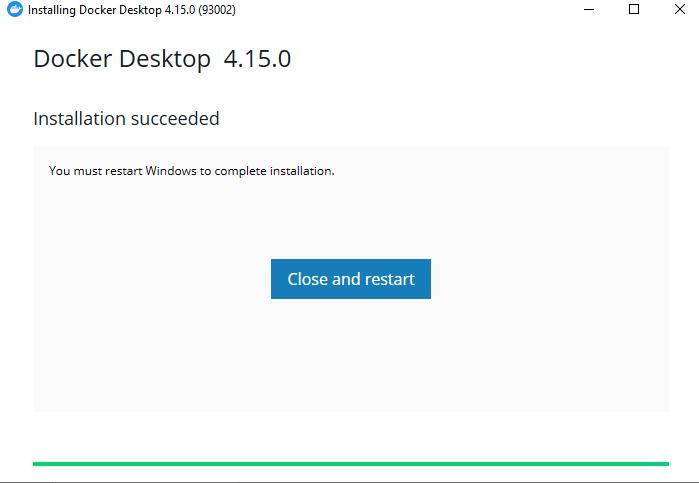


Рисунок 1.12 – Установка Docker Desktop

После установки запускаем приложение. Что бы приложение работало корректно, необходимо убедится в свободном месте на жестком диске и исправном подключении интернета. По умолчанию свободного места должно быть не менее 10гб. Следующим шагом будет открыть консоль или PoweShell в этой папке и прописать команду «docker build -t webapp .». Эта команда создаст образ докер приложения. Результат развертывания можно увидеть на рисунке 1.13.

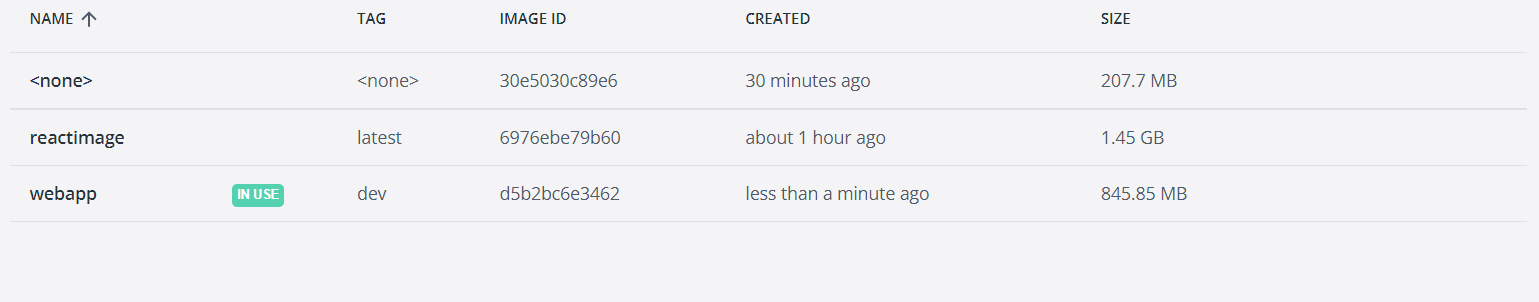


Рисунок 1.13 – образ в Docker Desktor

Далее выполним команду «docker run -d -p 8080:80 --name myapp   
WebApp».

Через некоторое время приложение развернется на докере. После чего можно начинать работу с приложением.

Что бы работать с таблицами базы данных, необходимо создать таблицы. Делается это очень просто. Необходимо установить и запустить приложение sql Management studio, подключиться к серверу, который указан в файле развертки, вызвать «запросник» и вставить скрип создания таблиц, далее выполнить запрос. После данных действий можно полноценно работать с приложением.

Если же понадобиться развертывать приложение не в среде Docker, а на реальной операционной системе, то придется установить следующее программное обеспечение для корректной работы самого приложения:

* база данных – сервер MSSQL не ниже sql server 2019 версии, для более удобно администрирования стоит использовать бесплатное приложение sql Management studio;
* серверная часть – размещение данного приложения на таких серверах, как IIS, Nginx не составит проблем за счет кроссплатформенности и гибкости самой ASP.Net Core плафтормы;
* клиентская часть – размещение не составит также никаких трудностей за счет кроссплатформенности NodeJS. В качестве базового сервера для размещения можно использовать тот же самый Nginx.

# 2 Проверка работоспособности приложения

Первым делом необходимо перейти на локальную страницу на 3000 порту. Видим поля для авторизации. При успешной авторизации и отображении списка пользователей можно сказать что приложение в докер-контейнере работает корректно. На рисунке 2.1 показана страница авторизации пользователя:

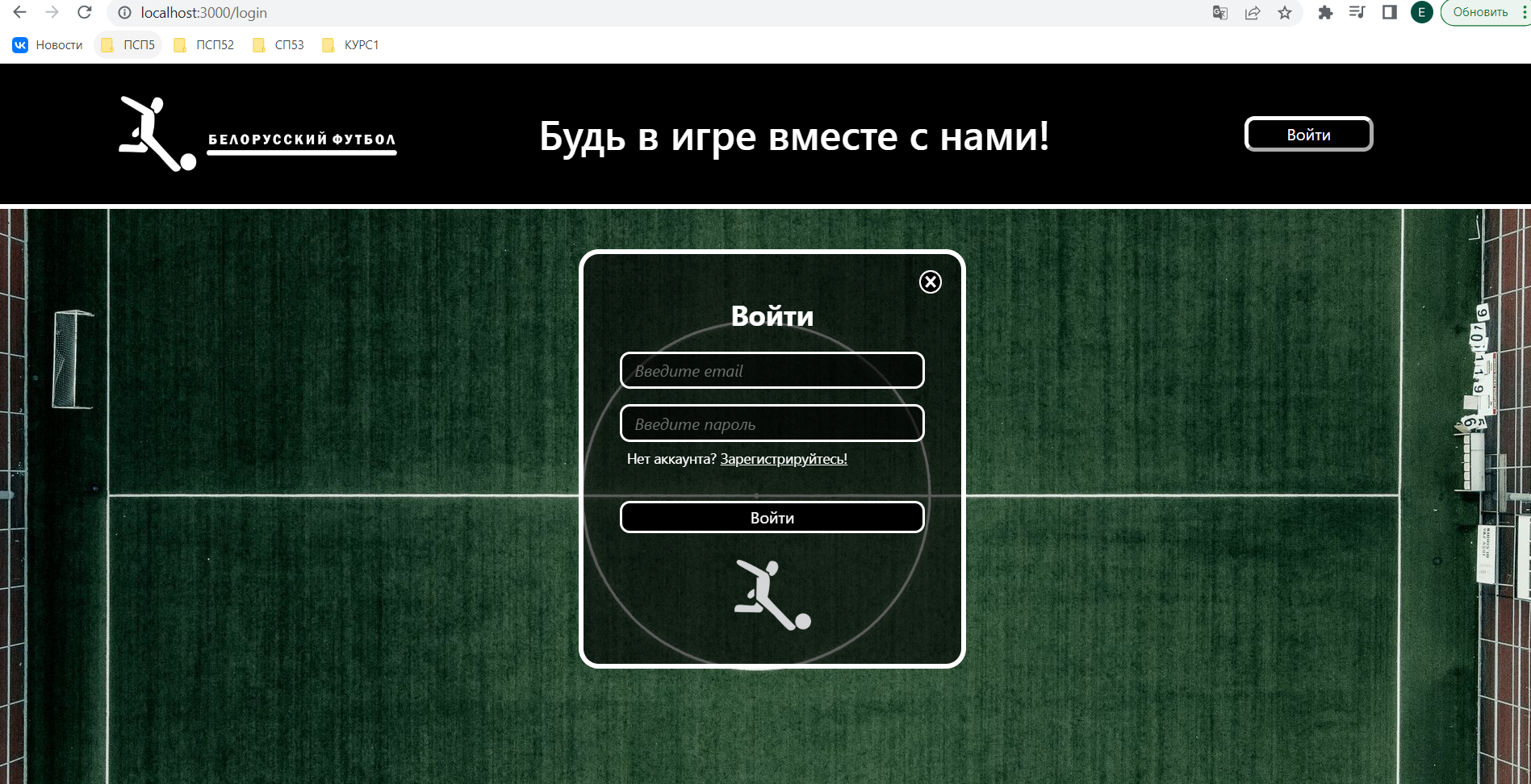


Рисунок 2.1 – Страница авторизации пользователя

При правильных логине и пароле, введённых пользователем, откроется страница профиля пользователя. На рисунке 2.2 отображается результат.



Рисунок 2.2 – Страница матчей в профиле пользователя

Заключение

В результате выполнения работы, первым делом были проанализированы аналоги веб-приложений. При разработке веб-приложения были учтены как положительные, так и отрицательные черты аналогов.

Для реализации серверной части использовалась платформа ASP.NET Core. ASP.NET Core может работать поверх кросс-платформенной среды .NET Core, которая может быть развернута на основных популярных операционных системах: Windows, Mac OS, Linux. В результате была разработан веб-сервис по архитектуре REST, который обрабатывает клиентские запросы и делает запросы к базе данных.

Была разработана реляционная база данных для веб-приложения, использующая экземпляр MSSQL, имеющая в своем наборе 6 таблиц.

С помощью фреймворка React.js была разработана клиентская часть приложения. Запросы к серверу выполняются с помощью JavaScript-библиотеки «axios».

Были созданы дополнительные конфигурационные файлы, позволяющие быстро и легко администрировать, расширять функционал приложения.

В результате проведенной работы, был создан проект, который удовлетворяет потребностям огромного круга людей.

# Список литературы

1. Свободная энциклопедия Википедия [Электронный ресурс] /CRUD – Режим доступа: https://ru.wikipedia.org/wiki/CRUD ­– Дата доступа: 05.12.2022.
2. ASP .Net Core [Электронный ресурс] – Режим доступа: https://metanit.com/sharp/aspnet5/ ­– Дата доступа: 28.11.2022.
3. MSSQL [Электронный ресурс] / MSSQL – Режим доступа: https://www.microsoft.com/ru-ru/sql-server/sql-server-2019 ­– Дата доступа: 08.12.2020.
4. React [Электронный ресурс] / React JS – Режим доступа: https://habr.com/ru/company/ruvds/blog/428077/ – Дата доступа: 18.11.2022.
5. React [Электронный ресурс] / Руководство по React JS – Режим доступа: https://reactjs.org/ – Дата доступа: 23.11.2022.