



**ТЕХНОЛОГИЧНО УЧИЛИЩЕ ЕЛЕКТРОННИ  
СИСТЕМИ  
към ТЕХНИЧЕСКИ УНИВЕРСИТЕТ - СОФИЯ**

## **ДИПЛОМНА РАБОТА**

**по професия код 481020 „Системен програмист“  
специалност код 4810201 „Системно програмиране“**

Тема: Приложение за анализ на футболни мачове

Дипломант:

*Валентин Веселинов Маринов*

Дипломен ръководител:

*инж. Милен Спасов*

СОФИЯ

2024



**ТЕХНОЛОГИЧНО УЧИЛИЩЕ ЕЛЕКТРОННИ СИСТЕМИ  
към ТЕХНИЧЕСКИ УНИВЕРСИТЕТ - СОФИЯ**

Дата на заданието: 15.11.2023 г.  
Дата на предаване: 15.02.2024 г.

Утвърждавам:.....  
/проф. д-р инж. П. Якимов/

## **ЗАДАНИЕ за дипломна работа**

ДЪРЖАВЕН ИЗПИТ ЗА ПРИДОБИВАНЕ НА ТРЕТА СТЕПЕН НА ПРОФЕСИОНАЛНА КВАЛИФИКАЦИЯ  
по професия код 481020 „Системен програмист“  
специалност код 4810201 „Системно програмиране“

на ученика Валентин Веселинов Маринов от 12А клас

1. Тема: Приложение за анализ на резултати от футболни мачове
2. Изисквания:
  - 2.1. Проектиране на база данни и приложение с клиент/сървър архитектура
  - 2.2. Разработване на модел за трансформация и импорт на исторически данни в базата данни на приложението
  - 2.3. Разработване на уеб приложение, със следните функционалности:
    - Регистрация и логин на потребители
    - Възможност за избор на мач или отбор от Висша лига
    - Търсене на интересни факти или новини, свързани с двата отбора
    - Показване на история на отборите
    - Сравнение на ефективността на играта и успехите на отборите, базирано на данни от минали сезони
3. Съдържание
  - 3.1 Теоретична част
  - 3.2 Практическа част
  - 3.3 Приложение

Дипломант:.....  
/ Валентин Маринов /

Ръководител:.....  
/ инж. Милен Спасов /

ВРИД Директор:.....  
/ ст. пр. д-р Веселка Христова /



ТЕХНОЛОГИЧНО УЧИЛИЩЕ "ЕЛЕКТРОННИ СИСТЕМИ"  
към ТЕХНИЧЕСКИ УНИВЕРСИТЕТ - СОФИЯ

## СТ А Н О В И Щ Е КЪМ ДИПЛОМНА РАБОТА

**Тема на дипломната работа:** Приложение за анализ на резултати от футболни мачове

**Ученик:** Валентин Веселинов Маринов

**Клас:** 12 А

**Професия:** код 481020 „Системен програмист“

**Специалност:** код 4810201 „Системно програмиране“

**Дипломен ръководител:** Милен Спасов

Дипломантът е реализирал напълно поставените в заданието изисквания и функционалности като е разработил уеб приложение на Java Spring Boot, като успя за много кратко време да се запознае с използваните технологии, да проучи какви архитектурни подходи са приложили в случая и да продължи с имплементацията и тестването на всички модули от системата.

В началния етап повече време отне анализа на технологиите, избора на развойни средства и архитектурните решения, които са изцяло документирани и обосновани. Дипломантът не само е изпълнил заданието си, но и е придобил много нови технически познания и е показал желание да продължи да се развива в посока разработка на уеб приложения.

Предлагам за рецензент Георги Николов, бакалавър Софтуерно инженерство от Софийски университет "Св. Климент Охридски", [georgi.nik95@gmail.com](mailto:georgi.nik95@gmail.com), 0887 499 189.

Всичко изложено дотук дава основание ученикът Валентин Маринов да бъде допуснат до защита пред комисията за провеждане на държавен изпит за придобиване на професионална квалификация по теория и практика на специалността и комисията да оцени дипломната работа отлично.

27.02.2024 г.

Дипломен ръководител:.....

/ инж. Милен Спасов /

## **Увод**

Целта на дипломната работа е да се предостави уеб приложение за анализ на резултати от футболни мачове, което да предоставя интересна и подробна информация при въвеждане на двубой от английската Висша лига. То позволява бърза и лесна ориентация в посоченият от потребителят двубой и осигурява достоверна информация.

Идеята за разработване на това приложение идва от голямата дезинформация във футболното пространство и все по - голямата липса на интерес от младите. То включва функционалност за регистрация и автентикация на потребители, възможност за въвеждане на мач от най - популярната футболна лига, предоставяне на най - различни исторически данни за мачовете между двата отбора, тренинзите им, техните успехи и най - актуалните новини около тях. Разработването включва изследване на съществуващи решения за предоставяне на футболна статистика и новини, както и анализ на потребностите на клиентите търсещи подобна информация. В реализацията на уеб приложението се използват съвременни технологии и принципи за проектиране на потребителски интерфейс.

# **ПЪРВА ГЛАВА**

## **МЕТОДИ И ТЕХНОЛОГИИ ЗА**

### **РЕАЛИЗИРАНЕ НА WEB ПРИЛОЖЕНИЕ ЗА**

### **АНАЛИЗ НА ФУТБОЛНИ МАЧОВЕ**

#### **1.1 Основни принципи за реализиране на WEB приложение**

Основният принцип, който използват модерните WEB приложения е Клиент - Сървър. Клиентът е потребителят, който използва приложението от своето устройство, а сървърът е отдалечена машина, която му осигурява достъп до данни и услуги. Процесът на работа е следният: клиентът изпраща заявка за данни към сървъра, който я приема и изпълнява, след което връща данните обратно на потребителя, за да осигури правилното функциониране на приложението.

#### **1.2 Архитектури за изграждане на WEB приложение**

Монолитна архитектура е подход в софтуерното инженерство, при който целият софтуерен продукт се изгражда като едно цяло, неразделно приложение. В такава архитектура всички компоненти и функционалности на системата се разработват, разглеждат се и поддържат съвкупно. Този вид архитектура е тясно свързан със структурата на монолита - едно голямо, неразделно цяло, което може да бъде трудно за разбиване или модификация на отделния му компонент без да се засегне цялостната функционалност.

Предимства на монолитната архитектура:

- Улеснява разбирането и изграждането - Монолитната архитектура е по-лесна за разбиране и разработка, особено за по-малки екипи или

за начинаещи разработчици. Целият софтуер е концентриран в една монолитна апликация, което прави процеса на разработка и отстраняване на проблеми по-прост.

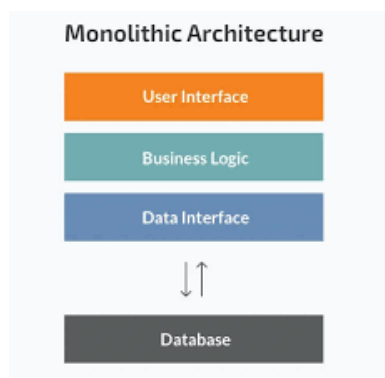
- Лесно мащабиране на малък обем от данни - При приложения, които не изискват голяма мащабируемост, този вид архитектура е добър избор, поради неговата опростеност.
- По - малко сложност в оперативната част - Тъй като монолитната архитектура се изпълнява като една апликация, няма нужда от сложна конфигурация или координация между отделните части. Това прави операциите по-лесни и по-малко склонни към грешки.
- По - добра производителност в някои случаи - Монолитната архитектура може да бъде по-ефективна от микросървисната архитектура при определени случаи на употреба, където разходите за комуникация и координация между отделните сървиси са по-големи от ползите.
- Улеснява тестването и отстраняването на грешки - Поради своята единоличност, монолитните приложения са по-лесни за тестване и отстраняване на грешки. Не е нужно да се координират множество различни сървиси или да се изследват проблеми в тяхната комуникация.

Недостатъци на монолитната архитектура:

- Трудно скалиране - Монолитните приложения обикновено се скалират хоризонтално, което може да бъде ограничаващо за много големи проекти. Понякога скалирането на отделни части от приложението може да бъде трудно или дори невъзможно без пренаписване на големи части от кода.
- Трудна поддръжка - С растежа на проекта, монолитните приложения често стават много сложни и трудни за поддръжка. С времето, нови

функции и корекции на грешки могат да доведат до прекомерна сложност на кода и усложнение на разработката.

- Липса на гъвкавост и иновации - Поради своята структура, монолитните приложения могат да бъдат по-трудни за актуализация и внедряване на нови технологии. Това може да намали скоростта на разработка и възможността за иновации.
- Зависимост между компонентите - В монолитните приложения компонентите обикновено са тясно свързани помежду си. Това може да доведе до проблеми със зависимости, когато се правят промени в единия компонент, които изискват промени и в други компоненти.
- Трудно разгръщане и мащабиране - Монолитните приложения често изискват цялостно разгръщане на приложението, вместо да позволяват изолирано разгръщане на отделни части. Това може да затрудни мащабирането и поддръжката на приложението.
- Ограничения в технологичния избор - Поради тяхната интегрирана среда, монолитните приложения често ограничават свободата на разработчиците при избора на технологии и инструменти за разработка.
- Риск от повреда в цялото приложение, причинена от един компонент - Поради централизирания характер на монолитната архитектура, има по-голям риск повреда на един компонент да засегне цялото приложение.



*фиг. 1 Монолитна архитектура*

Въпреки тези недостатъци и предимства, монолитната архитектура все още може да бъде подходящ избор за някои проекти, особено за по-малки или по-малко сложни приложения, където скалируемостта и гъвкавостта не са от съществено значение. Въпреки това, за по-големи и по-сложни системи често е по-подходяща микросървисната архитектура, която предлага по-голяма гъвкавост и мащабируемост.

Предимства на микросървисната архитектура:

- Гъвкавост и лесна мащабируемост: Поради своята децентрализирана природа микросървисите могат да бъдат мащабирани независимо един от друг. Това позволява гъвкавост при управлението на натоварването и оптимизиране на ресурсите.
- Технологично разнообразие: Всяка услуга може да бъде изградена с различни технологии и езици за програмиране спрямо нейните специфични изисквания. Това дава възможност за използване на най-подходящите инструменти за конкретните задачи.
- Независимост и изолация: Проблеми в един сървис не се разпространяват до други. Това води до по-голяма стабилност и устойчивост на системата като цяло.
- Бърза доставка на софтуер: Поради малките размери и ясно дефиниран обхват на микросървисите, те могат да бъдат разработени, тествани и доставени по-бързо от по-големите, монолитни приложения.
- Лесна поддръжка и обновяване: Възможността за независимо обновяване на всеки сървис прави процеса на поддръжка и актуализация по-лесен, тъй като не се налага да се правят прекъсвания в цялостната система.



Недостатъци на микросървисната архитектура:

- Сложност в управлението: Управлението на голям брой микросървиси може да бъде сложно, тъй като се изисква отделно проследяване, мащабиране и поддръжка на всеки един от тях. Това включва разгръщане и управление на множество инстанции на всеки сървис.
- Мрежови забавяния и латентност: Поради комуникацията между микросървисите през мрежата може да възникнат забавяния и латентност, което може да влоши производителността, особено при обработка на голям обем от трафик.
- Сложност в поддържането на съгласуваност на данните: Поддържането на съгласуваност на данните между микросървисите може да бъде сложно. При необходимост от транзакции или консистентност в базите на сървисите, координирането и управлението на данните може да представлява предизвикателство.



*фиг. 2 Микросървисна архитектура*

### 1.3 Технологии за създаване на WEB приложения

Технологии за разработване на потребителски интерфейс в WEB приложение:

- HTML (Hypertext Markup Language): HTML е основният език за създаване на структура на уеб страници. Той се използва за дефиниране на съдържанието на уеб страниците, включително текст, изображения, видеоклипове и други медийни елементи.
- CSS (Cascading Style Sheets): CSS се използва за стилизиране на уеб страници, позволявайки на разработчиците да контролират външния вид и изглед на уеб приложенията. Това включва форматиране на текста, управление на цветовете, подреждане на елементите на страницата и много други.
- JavaScript: JavaScript е език за програмиране, който се използва за добавяне на интерактивност към уеб приложенията. Той позволява на разработчиците да създадат динамични ефекти, като например анимации, валидация на формуляри, интерактивни менюта и други.

#### 1.3.1 React.js - JavaScript library

React е JavaScript библиотека за създаване на потребителски интерфейси, която е сред най-популярните и широко използвани технологии за уеб разработка. Тя е разработена от Facebook и позволява на разработчиците да създават ефективни и динамични уеб приложения чрез използването на компонентен подход. React е особено популярен заради своите предимства като единствен истински декларативен подход към създаването на уеб интерфейси. Той предоставя проста и интуитивна модела на компонентите за потребителски интерфейс, което прави разработката по-лесна и по-прозрачна. Този компонентен подход позволява разделянето на кода на малки, самостоятелни компоненти, които могат да бъдат лесно поддържани, преизползвани и тествани.

### **1.3.2 SpringBoot - Java framework**

Spring Boot е фреймуърк за разработка на сървър частта на приложения в Java. Той е базиран на платформата Spring и осигурява голяма удобност и бързина при създаване на Java приложения. Една от основните характеристики на Spring Boot е, че той идва с вградени конвенции и настройки, които правят разработката на приложения по-лесна. Това включва автоматично конфигуриране на много от основните компоненти на приложението. Spring Boot е много популярен избор за разработване на микросървисни приложения поради своите удобства за конфигуриране и мащабируемост. Също така има обширна общност от разработчици и много добра документация, което прави процеса на учене и разработка по-лесен.

### **1.3.3 Node.js**

Node.js е среда за изпълнение на JavaScript, базирана на V8 engine на Google Chrome. Той позволява на разработчиците да използват JavaScript за създаване на сървърни страници и мрежови приложения. Масово използван е заради неговият асинхронен и събитиеен модел на програмиране.

### **1.3.4 Express.js**

Това е фреймуърк написан на JavaScript, който улеснява създаването на WEB приложения с Node.js. Той предоставя лесни функции за създаване на сървъри и обработка на заявки. Сред ключовите му характеристики са управлението на сесий, възможността за интеграция на междинен софтуер, поддръжката на различни шаблони за генериране на HTML и удобното маршрутизиране на заявки. Използването на Express.js прави създаването на уеб приложения бързо и ефективно, което го прави избор номер едно сред разработчиците на Node.js.

### **1.3.5 Visual Studio Code(VS Code)**

Visual Studio Code е лек и мощен текстов редактор, разработен от Microsoft. Той е изключително популярен сред уеб разработчиците на софтуер, поради своята гъвкавост и богатите функции. VS Code поддържа множество езици за програмиране и разширения, които правят работата си още по-удобна.

### **1.3.6 IntelliJ IDEA**

IntelliJ IDEA е интегрирана среда за разработка (IDE), предназначена предимно за Java. Известна е със своите мощни инструменти за улесняване на програмистите в техния работен процес, включително интелигентно завършване на кода, рефакторинг, дебъгване и много други.

### **1.3.7 Docker**

Docker е платформа за създаване, разгръщане и управление на контейнери. Контейнерите са стандартизирани единици на софтуер, които включват всичко необходимо за изпълнението на приложение, включително библиотеки, зависимости и настройки. Docker улеснява разработката и разгръщането на приложения, като осигурява консистентност и изолация на приложението от външни ресурси и системи.

### **1.3.8 MySQL**

MySQL е релационна база данни, която е известна със своята надеждност, скорост и лесна употреба. Тя се използва широко в уеб разработката и други области, където се изисква съхранение и управление на данни.

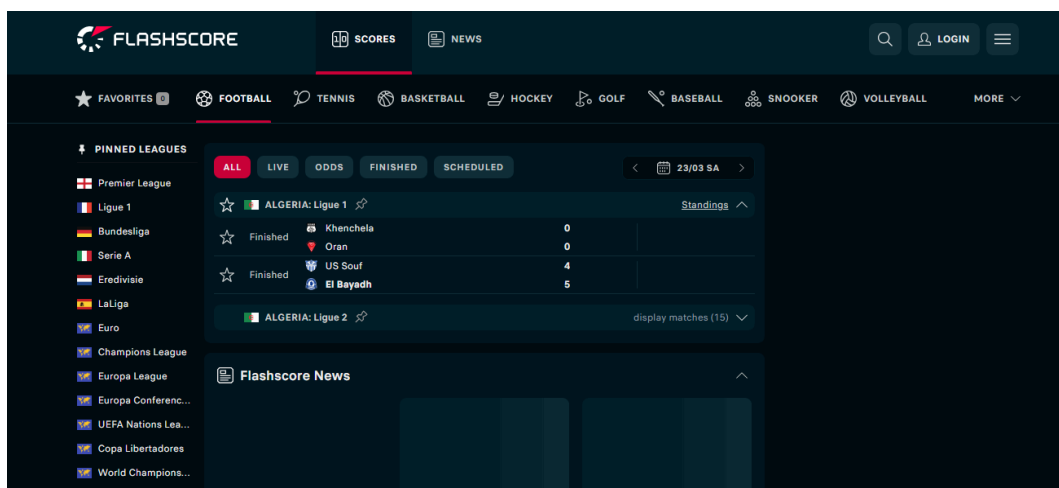
### **1.3.9 Microsoft Azure**

Microsoft Azure е облачна платформа, предлагаща широка гама от облачни услуги, включително изчислителна мощ, съхранение на данни, бази данни, изкуствен интелект и много други. Тя позволява на организациите да разработват, разгръщат и управляват приложения и услуги чрез глобална мрежа от центрове за данни.

## **1.4 Преглед на съществуващи подобни програмни системи и продукти**

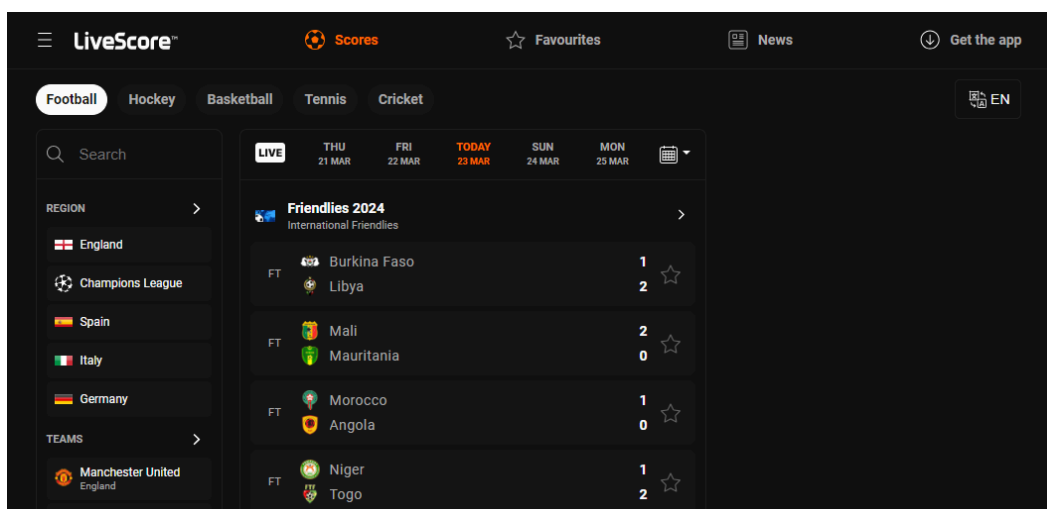
Уебсайтът на Flashscore е един от най - разпространените уебсайтове за спортна информация. Уебсайтът предлага огромен избор от различни спортове, турнири и мачове. Разполага с огромен набор статистика от всички краища на света. Също така има функционалност за избиране на любим отбор и получаване на известия за мачовете на отбора и новини за него. Уебсайтът предоставя най - актуалните новини за повече от двадесет вида спорт. Също предоставя информация и статистика за събития, провеждащи се в реално време. Уебсайтът може да бъде разглеждан както

в тъмна тема, така и в светла, има и голям набор от настройки за часова зона, вид на получаване на известията и начин на подредба на събитията в листът със събития.



фиг. 3 Flashscore

Друг известен уебсайт за спортна информация е liveScore, който предоставя същите функционалности, като се отличава с поле за търсене на турнир или отбор. За сметка на това разполага с по - малък набор от първенства.



фиг. 4 liveScore

## **ВТОРА ГЛАВА**

# **ПРОЕКТИРАНЕ НА СТРУКТУРАТА ЗА WEB ПРИЛОЖЕНИЕ**

### **2.1 Функционални изисквания към WEB приложение за анализ на футболни мачове.**

За Разработването на такъв тип WEB приложение ще се нуждаем от клиент сървър, база данни, в която да съхраняваме данните на потребителите и цялата налична статистика от футболни мачове, тренировки и отбори и трофеи. Също така от облачна технология, която да съхранява всички снимки от които ще имаме нужда, като например логата на отборите, снимките на тренировките и др. Клиент частта е това, което вижда потребителят, така че тя трябва да предоставя удобен потребителски интерфейс, който осигурява необходимата функционалност и показва съхранената информация по красив и подреден начин. Сървър частта трябва да осигурява основната логика на приложението, да изпълнява заявки и да взаимодейства с базата данни и облачното хранилище.

#### **2.1.1 Функционални изисквания към клиента**

- Форма за логин, регистрация.
- Връзка със сървъра.
- Визуализация на резултати с всякаква статистика и новини за търсеният мач.
- Предоставяне на възможност за търсене на избран мач между отбори участващи в тазгодишното издание на английската Висша лига.

### **2.1.2 Функционални изисквания към сървъра**

- Обработка на заявки - трябва при получаване на заявка от клиента, да я обработи и да върне отговор.
- Да поддържа автентикация и имейл верификация - да проверява дали потребител е в акаунта си или посещава приложението без акаунт и в такъв случай да поиска да си направи такъв, изпращане на имейл за верификация на акаунт след регистрация.
- Взаимодействие с базата данни - да комуникира с базата данни, да запази цялата налична информация за Висшата лига и да я предоставя в удобен вариант при поискване, също да запазва потребителите с направена регистрация и тяхната потребителски данни.
- Взаимодействие с облачна технология - да запази наличните снимки, на външно хранилище, които ще се използват в потребителския интерфейс и да ги предоставя при поискване.
- Запълване на облачното хранилище - автоматично да открие и запази нужните снимки.
- Обработка на грешки - за да се предотврати срив на приложението, сървърът трябва да разполага с механизъм за обработка на грешки.
- Търсене на новини - трябва да предоставя актуални новини за отборите участващи в търсеният мач.

## **2.2 Съображения за избор на програмни средства и развойна среда**

### **2.2.1 Технологии за разработване на клиент**

При разработване на клиент частта на WEB приложение задължителните технологии са HTML, CSS и JavaScript, за тях съществуват различни библиотеки и среди за разработка. Например JavaScript библиотеката React. Тя се използва за създаване на потребителски интерфейси, като



осигурява лесно създаване на компоненти, които динамично представят данни на уеб страницата. Известна е с опростеността си, модулността и възможността да се използва JSX - специален синтаксис, който обединява JavaScript и XML за удобно описание на структурата на потребителския интерфейс. В моето приложение, аз използвам основен HTML, CSS и JavaScript, защото не съм използвал React в минали проекти и това ще ми спести време от изучаване на тази библиотека.

### **2.2.2 Технологии за разработване на сървър**

Съществуват редица технологии за разработване на сървър частта на едно WEB приложение, например:

- Spring Boot - Java framework
- Express.js - JavaScript framework
- Flask - Python framework
- Django - Python framework

Flask и Django се използват за малки приложения, защото при тях с разрастването на проекта надграждането става по - трудно и отнема повече време. Spring Boot и Express.js са от най - разпространените технологии за разработка и поддръжка на големи WEB приложения. Те предлагат високо ниво на абстракция и улесняват разработката на backend приложения. Те идват с множество вградени функции, модули и библиотеки, които значително улесняват процеса на създаване на приложения. В моето приложение реших да използвам Spring Boot, защото съм го използвал преди и има дълга описателна документация. Използвам версия 3.2.2, защото към моментът на започване на проекта тя е най - новата стабилна версия на пазара.

### **2.2.3 Автентикация и имейл верификация**

За да може приложението ми да бъде защитено и да не бъдат източвани неговите данни от автоматизиран софтуер използвам автентикация. Автентикацията е една от двете най-важни понятия в областта на сигурността. Използва се за контрол на достъпа до ресурси и услуги в WEB приложенията. Автентикацията е процес на проверка на самоличността на даден потребител. В моето приложение автентикацията ще се състои от имейл и парола. Имейл верификацията се използва, за да потвърди, че потребителят, който се регистрира, е истинският собственик на посочения имейл адрес. Това предпазва от злоупотреби като създаване на фалшиви акаунти или нежелани атаки. В моето приложение след регистрация потребителят ще получава имейл, чрез който да активира своя акаунт в рамките на 15 минути след регистрацията. За да реализирам описаното ще използвам фреймуъркът Spring Security, който е препоръчителен при изграждане на защитата на WEB приложение с помощта на Spring. За изпращането на имейли ще използвам библиотеката JavaMail.

### **2.2.4 Съхранение на данни**

За да съхранявам цялата налична информация за Висшата лига от нейното създаване ще имам нужда от база данни.

Базите данни се делят на два вида:

- Релационни(Relational database) - Данните се съхраняват в таблици и могат да бъдат взаимно свързани. Полезни се за малки и средни проекти, тъй като са по - лесни за управление.
- Нерелационни(Non-relational database) - Данните се съхраняват в йерархична структура, което ги прави по - сложни за използване и

изисква повече ресурси. Затова този тип бази данни се използва от големи компании за разработване на големи проекти.

В моето приложение използвам релационна база, за съхранението на данните. Примери за релационни бази:

- MySQL
- PostgreSQL
- SQLite

Ще използвам MySQL, защото съм използвал нея в предишни проекти. Също, за да съхранявам всички снимки, които ще използвам във клиентската част на приложението ще използвам външно хранилище. Причините са:

- Неограничени ресурси - Така ще мога да съхранявам големи обеми данни, които не могат да бъдат ефективно съхранявани в самото приложение поради ограничената памет на моето устройство.
- Сигурност - При локално съхранение на данните винаги в приложението може да има срив и това да доведе до загуба на данни, докато в облачното пространство данните се съхраняват по - надеждно и с резервно копие.

Най - използваните облачни хранилища са тези на:

- Microsoft Azure
- Amazon Web Service

Двата софтуера са изключително добри за използване и предлагат обширен набор от функционалности. Разработени са и се поддържат от двете най - големи софтуерни корпорации в света, което гарантира тяхната надеждност и качество. Избрах да използвам Microsoft Azure, защото е безплатно и съм го използвал в минали проекти. Ще използвам Blob(Binary

Large Object) хранилището на Azure за съхранението на снимките. За да мога да качвам снимки в него и да ги взимам ще използвам библиотеката за Java SDK - Azure storage blob. В нея са представени всички необходими функции за качване и изтегляне на снимки по различни начини, които са добре документирани от Azure. Разделил съм снимките в няколко контейнера, за да е по подредено и улеснено за достъпване при нужда от тях. Контейнерите са:

- nations - флаговете на държавите, които са имали треньор във Висшата лига
- manages - снимки на треньорите
- teams - логата на отборите
- stadiums - снимки на стадионите
- managerappointments - снимки от момента на подписването на треньорите със даден клуб
- managerdepartures - снимки от последните дни на треньорите в съответните отбори
- premierleaguewinners - снимки на шампионският състав от всеки сезон
- premierleaguerrunnersup - снимки на вице-шампионският състав от всеки сезон

Имената са такива, защото конвенцията за именуване на контейнерите на Azure не позволява camelCase, което е конвенцията за именуване в Java, а с такова именуване преобразуването ще става най лесно.

## 2.3 Проектиране на структурата на базата данни

**В моята база данни ще имам нужда от следните таблици:**

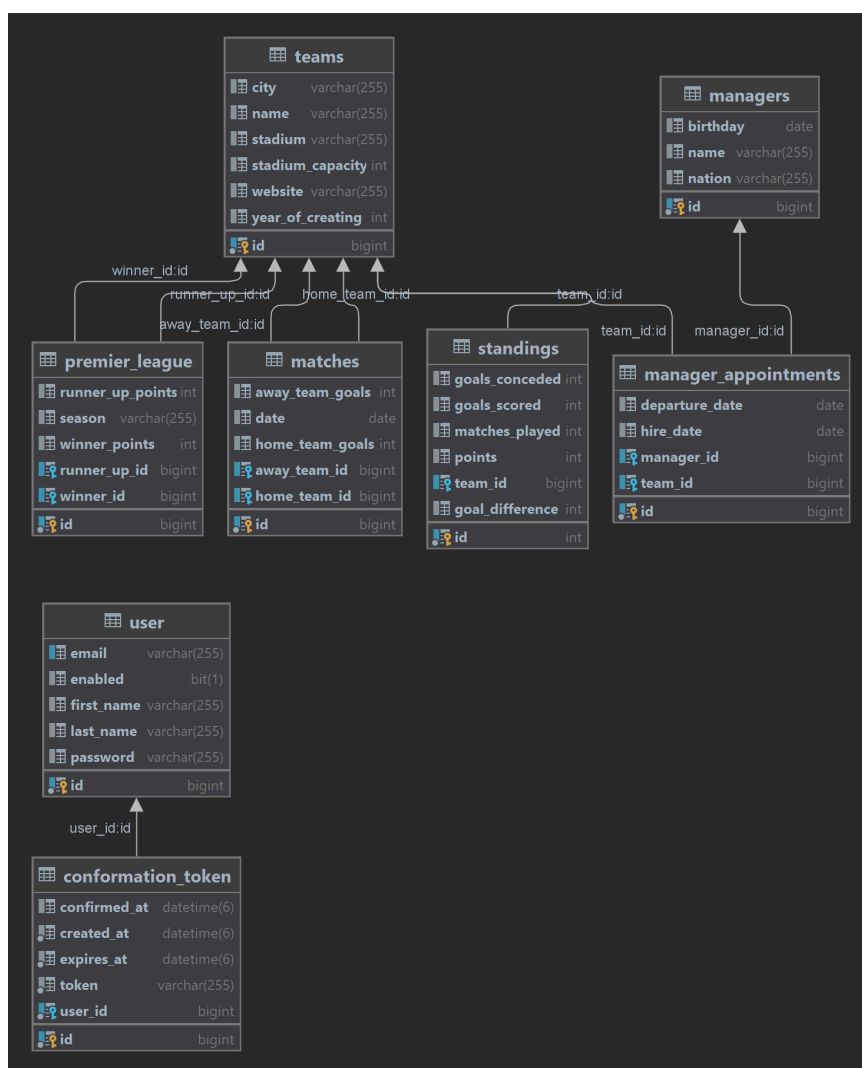
- `confirmation_token` - има връзка с таблицата `user`. Таблицата ще съдържа информация за всички създадени активационни низове - колони съдържащи стойност с момента на създаването им, момента на изтичането им, който ще бъде зададен да бъде 15 минути след този на създаване, момента на потвърждаването им, която ще се попълва при активиране на профилът с който е свързан съответният низ, както и колона съдържаща самият низ. Таблицата ще се използва за да бъде имплементирана имейл верификацията. Много редове от тази таблица могат да бъдат свързани с един ред от таблицата `user`, защото при изтичане на активационният низ, за да влезе в акаунта си потребителят ще трябва да поиска нов.
- `user` - таблицата ще съдържа колони, със стойности потребителските данни предоставени от него самият - неговият имейл, име, фамилия и криптирана версия на неговата парола. Също ще съдържа булева колона индикираща дали потребителят е верифицирал своят профил или не.
- `managers` - таблицата ще съдържа колони със стойности името на съответният треньор, неговата националност и дата на раждане.
- `teams` - таблицата ще съдържа колони със стойности името на отбора, капацитетът, градът и името на неговият стадион, които да бъдат предоставяни на потребителя, защото това ще му е от полза, ако иска да посети конкретно събитие. Също така уебсайтът на отбора и годината на създаването му.
- `manager_appointments` - таблицата ще представлява списък с всички треньорски назначения и уволнения на всеки отбор от създаването на Висшата лига. Ще има колони със стойности датата на

назначаване и на освобождаване на съответният треньор, като ако все още е начело на клубът стойността за дата на освобождаване ще бъде нулева. Таблицата е необходима, защото един треньор може за определен период от време да ръководи един отбор, а след това да поеме друг. За това таблицата ще съдържа връзка Many to One с таблицата с отбори и връзка Many to One с таблицата с треньори.

- matches - таблицата ще съдържа резултатите от мачовете от създаването на Висшата лига до днес. За това ще трябва да има колони за вкарани голове от домакин и гост, за да можем да извлечем резултатът от мача. Дата на провеждане на мача, за да можем да определим от кой сезон е той. Също ще съдържа връзка Many to One към таблицата с отбори, за да можем да запазим двата отбора участващи в мача и да се подсигурим, че няма да бъде въведен отбор, който не участва в първенството. Двата отбора ще се записват като домакински и гостуващ, защото при вземане на данни от базата това ще има значение, представянето на отборите е различно в различна среда.
- standings - таблицата ще се използва да съхранява моментното класиране във Висшата лига. Тя ще има връзка One to One с таблицата с отбори, за да подсигури, че няма да бъде запазен отбор, който не може да е част от първенството. Данните в нея ще се създават преди старта на сезона, като колоните в нея за брой изиграни мачове, вкарани голове, допуснати голове, голова разлика и точки ще имат стойност 0. Данните в нея ще бъдат обновявани само и единствено при добавяне на мач в таблицата с мачове. Избрах този подход, защото така моментното класиране ще бъде извличано по най - бързият начин, а това е важно понеже резултатите от нея ще бъдат показвани в основната страница на приложението и не можем

да си позволим при всяко отваряне на страницата класирането да се калкулира на ново.

- premier\_league - таблицата ще се използва, за да съхранява шампионът и вице-шампионът от всички сезони на Висшата лига от създаването и до днес. За целта ще има Many to One връзка към таблицата с отбори, по този начин се подsigуряваме, че няма да бъде запазен отбор, който дори не е участвал, за да различаваме различните сезони ще имаме и колона съдържаща стойност годините на сезона. Също за пълнота две колони за броя изкарани точки в съответният сезон от шампионът и вице-шампионът.



фиг. 5 Схема на базата данни

## **ТРЕТА ГЛАВА**

# **ПРОГРАМНА РЕАЛИЗАЦИЯ НА WEB ПРИЛОЖЕНИЕ ЗА АНАЛИЗ НА ФУТБОЛНИ МАЧОВЕ**

### **3.1 Архитектура на приложението**

Приложението се състои от клиентска и сървърна част.

### **3.2 Разработване на сървърната част**

За разработването на сървърната част е използвана технологията Spring Boot и езикът Java.

#### **3.2.1 Изграждане на база данни и комуникация с нея**

За да изградя базата, първо трябва да дефинирам моделите. Моделът представлява клас с анотация Entity, като името на класа е името на таблицата, а неговите полета са колоните на таблицата. За всеки такъв клас добавям към него анотация Data, за да получа имплементирани за него Getters - чрез тях ще достъпвам полетата, спазвайки принципите на ООП, Setters - чрез тях ще задавам стойности на полетата спазвайки принципите на ООП, EqualsAndHashCode - чрез тях ще мога да сравнявам обекти от този клас и да създавам hash value на обекти от този клас, ToString - чрез него ще мога да превърна обекта с данните от него в символен низ и конструктор за класа включващ всички негови полета. Всички използвани анотации в проекта са от библиотеката Lombok, без която меко казано Spring Boot приложение не може да съществува. Всички модели се съдържат в папка Entities на проекта.



Дефиниране на клас `User`, намиращ се в подпапка `User`, който имплементира `UserDetails` от фреймуъркът `Spring Security`:

- `Long id` - анотиран с `Id` и оказана стратегия за уникално генериране отново чрез анотация представлява 64-битова стойност, която се генерира автоматично и е уникална за всеки един обект.
- `firstName`, `lastName` - символни низове
- `email` - също символен низ, анотиран да бъде уникален за всеки обект, тъй като имейлът на всеки е уникален сам по себе си.
- `password` - символен низ, хешира се с помощта на класа `BCryptPasswordEncoder`, част от `Spring Security`, който използва функцията на `BCrypt` за хеширане, за да запази неговият ключ, като по този начин ни улеснява при работа с хешираната парола. По този начин паролата е защитена дори и от самите нас.
- `enabled` - 2-битова стойност показваща ни дали даденият обект е активирал профила си. По подразбиране профила се създава неактивиран.
- `fullName` - символен низ съставен от конкатениране на `firstName` и `lastName`, анотацията `Transient` оказва това поле да не бъде включено в таблицата за този клас.

За да може моето приложение да функционира правилно използвайки `Spring Security`, тук трябва да имплементирам интерфейса `UserDetails` и да пренапиша неговите методи, в приложението няма различни роли, за това в метода за вземане на права връщам празен лист. В приложението ми няма изтичане на профили и потребителски данни, за това в метода за тяхното вземане връщам `true`. Същото важи за заключването на акаунта. По подразбиране `Spring Security` използва `username` и `password`, а аз използвам `email` и `password` за автентикация на потребители. Полето за парола е едно и също за това няма нужда да пренаписвам `getPassword()`. В `getUsername()`

давам моят начин, а именно email, същото важи и за пренаписването на isEnabled()).

```
12  @Data
13  @Entity
14  public class User implements UserDetails {
15
16      @Id
17      @GeneratedValue(strategy = GenerationType.IDENTITY)
18      private Long id;
19
20      1 usage
21      private String firstName;
22
23      1 usage
24      private String lastName;
25
26      1 usage
27      @Column(unique = true)
28      private String email;
29
30      private String password;
31
32      1 usage
33      private Boolean enabled = false;
34
35      @Transient
36      private String fullName;
37
38      @Override
39      public Collection<? extends GrantedAuthority> getAuthorities() { return Collections.emptyList(); }
```

*фиг. 6 Клас User*

Дефиниране на клас ConformationToken, намиращ се в подпапка User:

- Long id - представлява 64-битова стойност, която се генерира автоматично и е уникална за всеки един обект.
- token - символен низ за верификация, анотацията му показва, че не може стойността му да е нулева.
- createdAt, expiresAt, confirmedAt - представляват дата и час съответно на създаване на низът за верификация, изтичането му и потвърждаването му.
- user - друг модел, полето е анотирано с ManyToOne, което оказва таблицата да бъде свързана с таблицата на другият модел по

уникалният идентификатор (id), съобразявайки, че много обекти от този модел (Many) могат да бъдат свързани (to) с един и същи обект от другият (One).

```
8  @Data
9  @Entity
10 public class ConformationToken {
11
12     @Id
13     @GeneratedValue(strategy = GenerationType.IDENTITY)
14     private Long id;
15
16     @Column(nullable = false)
17     private String token;
18
19     @Column(nullable = false)
20     private LocalDateTime createdAt;
21
22     @Column(nullable = false)
23     private LocalDateTime expiresAt;
24
25     private LocalDateTime confirmedAt;
26
27     @ManyToOne
28     @JoinColumn(nullable = false)
29     private User user;
```

*фиг. 7 Клас ConformationToken*

Дефиниране на клас Managers:

- Long id - представлява 64-битова стойност, която се генерира автоматично и е уникална за всеки един обект.
- name, nation - символни низове, представляващи името и националността на треньорите.
- birthday - дата на рожденият ден на треньора

```
9  @Entity
10 @Data
11 public class Managers {
12
13     @Id
14     @GeneratedValue(strategy = GenerationType.IDENTITY)
15     private Long id;
16
17     private String name;
18
19     private LocalDate birthday;
20
21     private String nation;
```

*фиг. 8 Клас Managers*

Дефиниране на клас Teams:

- Long id - представлява 64-битова стойност, която се генерира автоматично и е уникална за всеки един обект.
- name, stadium, city, website - символни низове, представляващи името, името на стадиона, града и уебсайтът на даден отбор.
- yearOfCreating, stadiumCapacity - 4-байтови обекти, които съдържат стойността на число съответно за годината на създаване на даден отбор и капацитета на стадиона му.

```
9      @Entity
10     @Data
11     public class Teams {
12         @Id
13         @GeneratedValue(strategy = GenerationType.IDENTITY)
14         private Long id;
15
16         private String name;
17
18         private Integer yearOfCreating;
19
20         private String stadium;
21
22         private Integer stadiumCapacity;
23
24         private String city;
25
26         private String website;
```

фиг. 9 Клас Teams

Дефиниране на клас ManagerAppointments:

- Long id - представлява 64-битова стойност, която се генерира автоматично и е уникална за всеки един обект.
- hireDate, departureDate - дати на назначаване и напускане на треньор от даден клуб. Датата за напускане е аотирана с Nullable, защото

когато треньор все още е на чело на даден отбор в тази таблица стойността за тази колона ще е нулева.

- manager - установена Many to One връзка с модел Managers, чрез уникалният идентификатор (id).
- team - установена Many to One връзка с модел Teams, чрез уникалният идентификатор (id).

```
9  @Entity
10  @Data
11  public class ManagerAppointments {
12      @Id
13      @GeneratedValue(strategy = GenerationType.IDENTITY)
14      private Long id;
15
16      private LocalDate hireDate;
17
18      @Nullable
19      private LocalDate departureDate;
20
21      @ManyToOne
22      private Teams team;
23
24      @ManyToOne
25      private Managers manager;
```

*фиг. 10 Клас ManagerAppointments*

Дефиниране на клас Matches:

- Long id - представлява 64-битова стойност, която се генерира автоматично и е уникална за всеки един обект.
- date - дата, оказваща денят на изиграване на даден мач.
- homeTeamGoals, awayTeamGoals - 4-байтови обекти, които съдържат стойността на число съответно за броят вкарани голове на домакина и госта в даденият мач.

- homeTeam, awayTeam - установена Many to One връзка с модел Teams, чрез уникалният идентификатор (id), полетата представляват съответно домакинският и гостуващият отбор в даденият мач.

```

10  @Entity
11  @Data
12  public class Matches {
13      @Id
14      @GeneratedValue(strategy = GenerationType.IDENTITY)
15      private Long id;
16
17      7 usages
18      private LocalDate date;
19
20      1 usage
21      @ManyToOne
22      private Teams homeTeam;
23
24      1 usage
25      @ManyToOne
26      private Teams awayTeam;
27
28      1 usage
29      private Integer homeTeamGoals;
30
31      1 usage
32      private Integer awayTeamGoals;

```

*фиг. 11 Клас Matches*

Дефиниране на клас PremierLeague:

- Long id - представлява 64-битова стойност, която се генерира автоматично и е уникална за всеки един обект.
- season - символен низ, представляващ началната и крайната година за съответен сезон. Няма смисъл от разграничаване в две отделни полета, тъй като това поле ще се използва като име на сезон.

- winnerPoints, runnerUpPoints - 4-байтови обекти, които съдържат стойността на число за броят натрупани точки през този сезон съответно за шампионът и вице-шампионът, направили ги такива.
- winner, runnerUp - установена Many to One връзка с модел Teams, чрез уникалния идентификатор (id), полетата предоставят съответно отборът шампион и отборът вице-шампион в съответен сезон.

```

6  @Entity
7  @Data
8  public class PremierLeague {
9      @Id
10     @GeneratedValue(strategy = GenerationType.IDENTITY)
11     private Long id;
12
13     private String season;
14
15     @ManyToOne
16     private Teams winner;
17
18     @ManyToOne
19     private Teams runnerUp;
20
21     private Integer winnerPoints;
22
23     private Integer runnerUpPoints;

```

*фиг. 12 Клас PremierLeague*

Дефиниране на клас Standings:

- Long id - представлява 64-битова стойност, която се генерира автоматично и е уникална за всеки един обект.
- matchesPlayed, goalsScored, goalsConceded, goalDifference, points - 4-байтови обекти, които съдържат стойността на число за съответно броят изиграни мачове, вкарани и допуснати голове, голова разлика и точки на отбор за настоящият сезон.



- team - друг модел, полето е анотирано с OneToOne, което оказва таблицата да бъде свързана с таблицата на другият модел по уникалният идентификатор (id), съобразявайки, че само един обект от този модел (One) може да бъде свързан (to) с един и същи обект от другият (One).

```
6  @Data
7  @Entity
8  public class Standings {
9
10     @Id
11     @GeneratedValue(strategy = GenerationType.IDENTITY)
12     private Integer id;
13
14     @OneToOne
15     private Teams team;
16
17     private Integer matchesPlayed;
18
19     private Integer goalsScored;
20
21     private Integer goalsConceded;
22
23     private Integer points;
24
25     private Integer goalDifference;
```

*фиг. 13 Клас Standings*

След като създадем моделите трябва създадем базата данни и да ги интегрираме в нея. Тъй като класовете са анотирани с Entity при стартиране на проекта таблиците автоматично ще бъдат създадени по модела, който сме задали, преди това обаче, за да създам базата данни отварям нов connection в MySQL Workbench и изпълнявам заявката “CREATE DATABASE premierLeague;”. За да се свърже със базата трябва да поставя необходимата информация за връзка с нея като url, username,

password и driver, във файла application.properties, необходимият driver за моето приложение е този за MySQL. За управление на базата използвам Spring Boot JPA, отразяваме и това във файла application.properties, като настройваме, така че при всяко следващо пускане да се отразяват промените по моделите. След като стартираме приложението в базата данни се поставят създадените модели под формата на таблици и връзки между тях. За комуникация с различните таблици създавам repository interface за всяка една от тях, което да имплементира JpaRepository<T, ID>. T представлява моделът на таблицата, която ще искам да достъпя през това repository, а ID се попълва с типа на ключът на съответната таблица, в моето приложение винаги е Long. В папката repositories са всички интерфейси, които се използват за връзка с базата. Методите от всяко repository ще бъдат използвани в имплементацията на сървис за конкретен клас като всяко repository ще бъде инжектирано в конструктора на прилежащият му сървис, всеки сървис си има интерфейс, с всички методи, които трябва да имплементира, интерфейсите на сървисите се намират в папката services на проекта, където се намират и техните имплементации в подпапката impl.

### 3.2.2 Импорт на необходимите данни

Импортът на данните се осъществява чрез конфигурационни класове с ComandLineRunner метод, анотиран като Bean, за да се изпълни веднага след стартирането на проекта. Всички конфигурационни класове се намират в под папката на config - dataImportConfig, там те се разделят на два вида, първо се изпълняват тези в databaseImportConfig папката, където всеки метод е анотиран като Transactional, за да не бъдат въведени никакви данни ако се случи срыв по време на техният импорт, голямата част от импорта се осъществява чрез csv файлове намиращи се в под папката data на папката resources на проекта ми. Повечето от csv файловете са взети от готов сет от данни. За да ги обработвам използвам библиотеката com.opencsv. Редът в който всеки Bean бива извикан при стартиране на проекта е имплементиран използвайки анотацията Order(N), с която Bean-ът с най ниска стойност на N се извиква първи и на същият принцип следващите един след друг, след като веднъж са вкарани данните коментирам всички анотации Bean и Order в тези класове, за да избегна повторение на данни в базата. Избраният ред на извикване не е случаен, първо импортирам данните за таблиците, които ще се ползват от други таблици и след това всичко останало. Редът е следният:

- ManagersImportConfig - Данните се вземат от manager.csv - част от сета от данни. Игнорирам първият ред, защото съдържа заглавия на колоните във файла, след което започва обхождане на всеки ред, необходимите неща от него са данните за името на треньора, намиращи се във втората колона, националността му от третата колона и рождената му дата от шестата колона. Тъй като първият индекс в листа винаги е 0 индексите на взетите елементи са с 1 по малко от посочените. При всяко обхождане се създава нов обект на създаденото от нас Entity - Managers, което се запазва в съответното repository, което създадох преди това чрез метода

repository.save(manager), този сет от данни, обаче не беше актуален и се наложи да си направя още един с данните от последните 2 години, като източник на информация използвах уикипедия.

- TeamsImportConfig - в Bean метода на този клас импорта се осъществява малко по - различно, тъй като в сета от данни нямаше информация за годината на създаването на отборите, за да си я набавя по лесен начин използвам библиотеката Jsoup. Използвам я, за да взема html файла на уебсайта, който предоставя тази информация. Библиотеката предлага вградени функции за търсене в html файл, и по този начин във метода searchYearOfCreation() успях да набавя нужните данни, като резултатът от метода е HashMap<String, Integer> където String - ключът съдържа име на отбор, а Integer - стойността съдържа годината на създаването му. В уебсайта има подобна статистика за отбори от всички краища на света, за да взема тези, които ми трябва проверявам всеки ред от таблицата с отбори дали съдържа текст "England", защото отборите от Висшата лига са основно английски, но първенството е имало участници и от Уелс, за това ми се наложи да взема и техните записи. Също за отбори като Шефилд Юнайтед, които според уебсайта са с неясна година на създаване или други, които те изписват по различен начин от този, който е дефиниран в базата данни се наложи да направя няколко допълнителни проверки. За два от необходимите ми отборите този източник няма информация, за това се наложи да напиша информация за тях на ръка, за източник използвах уикипедия. Останалата част не е по различна от тази за треньорите, като за име на отбора и техният уебсайт използвам club.csv, а за данните свързани със стадиона - stadium.csv, като използвам предоставената ни връзка по id между файловете в сета от данни. За

последните две години има само отбора, които за първи път участват в лигата, за това ги добавих на ръка.

- **ManagerAppointmentsImportConfig** - Следващата таблица е тази с треньорските назначения. В нейният импорт няма нищо по - особено, сета от данни е направен с помощта на уикипедия и Google Sheets, като файлът с треньорски назначения - `managersStats.csv` е sheet изтеглен като csv документ. Особеностите тук са `departureDate` на треньора, когато стойността е "Present\*" в базата да бъде записано като нулева стойност, понеже треньорът все още не е напуснал клуба. Също по - странният формат на датите, специално за който имплементирах `parseDate(String date)`, за да приведа дататите в обикновения за съхранение в базата формат. Правя го като разделям полученият символен низ на 3 части и ги препоредя, като понеже месеца е изписан с букви извиквам `getMonthIndex(StringMonthName)`, където минавам през месеците от годината и връщам кой под ред е подаденият на метода месец по неговото име.
- **MatchesImportConfig** - Тук и двата сета от данни са взети на готово с леки редакции ръчно от мен, това е импорта на мачовете, в този клас има два метода, аотирам `fillOldMatches()` с `Order(1)` и `fillMatches()` с `Order(2)`, когато се стигне до този клас ще бъдат намерени два метода аотирани с Bean и няма да знаем кой ще бъде изпълнен първи, за това освен `Order` на класовете за този клас добавих и `Order` на методите в него. Единственият проблем тук е, че при един от старите мачове датата е неизвестна и в такъв случай поставям нулева стойност в това поле, също да се преобразува символният низ на датата в стойност, готова за импорт в базата. За старите данни се

използват `match.csv` и `club.csv`, а за новите: `premier_league_last_two_seasons_matches.csv`

- `PremierLeagueImportConfig` - тук сета от данни е малък и реших да си го направя изцяло сам, за това и обработването стана по - просто, това е импорта на шампионите и вице-шампионите от всички минали сезони. Използва се `pl.csv`.
- `StandingsImportConfig` - тук импорта не е от `csv`, а е изписан в самият метод, прецених, че за толкова малък обем от данни няма нужда от излишни усложнения, данните от таблицата с моментното класиране се изтриват и се добавят нови веднъж в годината, преди началото на новият сезон, края на Юни като отборите всяка година са почти еднакви, таблицата се ъпдейтва автоматично при импорт на нов мач в базата.

След като съм готов с базата започвам да импортирам снимки във външното хранилище, импортирам ги като използвам току що импортираните данни в базата, взема нужните от тях и правя автоматизиран `Google search images` чрез библиотеката `Selenium WebDriver`. Библиотеката се използва за тестване на потребителски интерфейс, но в случая се оказва доста удобна за взимане на изображения от `Google`, като процеса е имплементиран в другата под папка на `dataImportConfig` - `storageImportConfig`. В нея в класа `Google connect` методът `searchImage(String searchString)` се използва във всеки от класовете за импорт на снимки, като той изисква единствено символният низ, който трябва да бъде изписан в търсачката и връща обратно снимка под формата на масив от байтове.

```

public byte[] searchImage(String searchString){

    int indexOfTheFirstImageFromTheSearch = 0;
    int marginToTheBSFCode = 1;
    int morePagesThanExpected = 2;
    int waitToRenderPageWithSearch = 150;
    int waitToRenderPageWithImages = 500;

    WebDriver driver = new ChromeDriver();

    driver.get("https://www.google.com/imghp");
    driver.findElement(By.id("L2AGLb")).click();
    try {
        Thread.sleep(waitToRenderPageWithSearch);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
    WebElement searchBox = driver.findElement(By.name("q"));
    searchBox.sendKeys(searchString);
    searchBox.sendKeys(Keys.RETURN);

```

```

    try {
        Thread.sleep(waitToRenderPageWithImages);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }

    if (driver.getWindowHandles().size() == morePagesThanExpected){
        driver.findElement(By.id("TYtByb")).click();
    }

    String html = driver.getPageSource();
    Document doc = Jsoup.parse(html);
    Elements imagesString = doc.select(cssQuery: "div#isIrg");

    if(imagesString.isEmpty()){
        imagesString = doc.select(cssQuery: "div#rcnt");
    }

    Elements images = imagesString.select(query: "img");

    String image = images.get(indexOfTheFirstImageFromTheSearch).attr(attributeKey: "src");
    String BSFImage = image.substring(beginIndex: image.indexOf(",") + marginToTheBSFCode);

    driver.quit();

    return Base64.getDecoder().decode(BSFImage);
}

```

*физ. 14 Метод searchImage*

След като методът бъде извикан се отваря нов `ChromeDriver`, на който подавам `GET` заявка за `Google Images`. Веднага се зарежда изскачащ екран за бисквитки, които приемам след изпълнение на 21 ред, там намирам бутона за съгласие по `id` и чрез метода `click()` го натискам. След това изчакам 0.15 секунди, за да съм сигурен, че страницата ще се зареди и отново чрез драйвера откривам търсачката по име - "q" и чрез `sendKeys` подавам символният низ, на който искам да получа снимка и чрез `sendKeys(Keys.RETURN)` зареждам страницата с резултатите. Пак изчакам, този път половин секунда, за да съм сигурен, че пълният резултат се е заредил. Тъй като използвайки `Selenium WebDriver` посещавам `Chrome` без да съм влязъл в профил се случва да излязат неочаквани екрани като например смяна на езика. За това първо проверявам, колко страници са се заредили и ако са две натискам бутонът "keep the same". Вече може да вземем `html` файла на заредената страница чрез метода `getPageSource()` и да направим документ от него чрез `Jsoup`. `Google` има два вида `css` за снимки, за това първо проверявам за резултатите на единият и ако няма такива търся тези на другият, по този начин взимам лист от контейнерите на всяка снимка, а чрез `select("img")`, взимам само параметрите на всяка снимка и чрез `get(0).attr("src")`, вече имам символният низ на първата снимка от резултатът на търсенето. След това взимам частта с картинката в `Base64` формат, декодирам я и я връщам като масив от байтове. Методът за запазване на снимки в облачното хранилище е имплементиран в същата папка в класа `CloudManagment`. В него има предварително зададено поле `blobServiceClient` на което е предоставен линк към хранилището и низ от символи за достъп до него. Чрез това поле ще мога да управлявам моето хранилище. Качването се осъществява чрез метода `upload(String name, byte[] image, String containerName)`, където:



- name е низ от символи представляващ името на новият Blob, който ще създадем в хранилището.
- image е масив от байтове и представлява самата снимка, която ще качим.
- containerName е низ от символи представляващ името на контейнера, в който ще запазим новото изображение.

```

15 public class CloudManagement {
16
17     2 usages
18     private final BlobServiceClient blobServiceClient = new BlobServiceClientBuilder()
19         .endpoint("https://plprovider.blob.core.windows.net")
20         .sasToken("sv=2022-11-02&ss=bfgt&srt=sco&sp=rwdlacupiytfx&se=2024-06-01T16:16:07Z&st=2024-03-21")
21         .buildClient();
22
23     @
24     public void upload(String name, byte[] image, String containerName){
25
26         if (name.contains(" ")){
27             name = name.replace(target: " ", replacement: "_");
28         }
29
30         BlobContainerClient blobContainerClient = blobServiceClient.getBlobContainerClient(containerName);
31
32         BlockBlobClient blockBlobClient = blobContainerClient.getBlobClient(name).getBlockBlobClient();
33
34         try (ByteArrayInputStream dataStream = new ByteArrayInputStream(image)) {
35             blockBlobClient.upload(dataStream, image.length);
36         } catch (IOException e) {
37             e.printStackTrace();
38         }
39     }
40 }

```

*фиг. 15 Метод upload*

Първата стъпка е да подсигурия name, така че да отговаря на стандартите за именуване на Blob, а именно, ако има празните разстояния да бъдат заменени с “\_”. Това също ни гарантира че данните ни вътре ще бъдат форматиран по един и същ начин и когато ни потрябват трябва само да използваме същото име, което сме подали като параметър на този метод, за целта във всеки конфигурационен клас това поле е съставено само от елементи съхранявани в базата данни и винаги можем да видим кои са те и да ги използваме обратно при изтегляне на изображение. Втората стъпка е

да създам `blobContainerClient`, използвайки предварително създаденият `blobServiceClient`, който да ни свърже с контейнера посочен в параметърът `containerName`. Третата стъпка е да се създаде `blockBlobClient`, използвайки създаденият във втора стъпка `blobContainerClient`, който да създаде новият `Blob` с такова име, което сме обработили след подаването като параметър и да запази създаденият `blockBlobClient`. Вече имаме всичко нужно, за да запазим нашата снимка. Създаваме един `try - catch` блок, защото `upload` метода на `blockBlobClient` може да хвърли `IOException`. Като ресурс на блока трябва да заредим нов `ByteArrayInputStream` от подаденият като параметър на метода масив от байтове `image` и в тялото на `try` блока подаваме на метода `upload` създаденият `ByteArrayInputStream` и дължината на масива от байтове подаден като параметър на нашият метод. В случай на грешка при качването `catch` блока ще изпринтира в терминала случилото се. Имайки тези два инструмента качването на снимките е кратко за имплементация. Тук разполагаме с пет конфигурационни файла, всеки от които има инстанция на класовете `CloudManagment` и `GoogleConnect`, за да може да се възползва от техните методи, отново всеки клас има `Bean` и класовете имат съответния `Order` за изпълнение и след импорт на необходимите снимки биват закоментирани, за да се избегне повторно качване. Имплементиране на петте конфигурационни класа по реда им на извикване:

- `NationsStorageImportConfig` - в `ManagersRepository` създавам нов метод с `Query` анотация за вземане на всички уникални елементи в колоната с нации. В сървисът към съответната таблица добавям метод `List<String> getAllDifferentNations()` и връщам резултата от метода със заявката. В самият конфигурационен клас инжектирам в конструктора `ManagersServiceImpl` и извиквам неговият нов метод, итерирам през полученият лист и за всяка нация търся снимка и я

качвам в контейнера nations със уникално име съставено от името на нацията + “\_flag”

- ManagersStorageImportConfig - тук отново инжектирам ManagersServiceImpl и извиквам неговият метод getManagers() той от своя страна използва ManagersRepository, за да вземе всички треньори в него. В конфигурационният файл итерирам през полученият лист от треньори, като за всеки треньор търся снимка по текст: името на треньора, взето от листа + “ football manager”, получената снимка запазвам в контейнерът managers, като името на всяка снимка е името на треньора.
- TeamsStorageImportConfig - стъпките са същите като при треньорите, с разликата, че се използват съответното repository и service за таблицата с отбори. При итерирането през листа с отбори запълваме контейнери teams и stadiums, като от листа взимаме всяко име на отбор и на стадион. teams се запълва от логата на отборите, а stadiums от техните стадиони.
- ManagerAppointmentsStorageConfig - тук отново итерираме през листа със записите на таблицата, създаден от съответното repository и предоставен чрез инжектиране на съответният сървис. Нужната информация ни е името на треньора, на отбора и датата на неговото назначаване, като снимката се запазва с име същото като низа от символи създаден, за да я намери. По същият начин и с уволненията, но преди да се търси първо трябва да се провери дали треньорът не е на поста си още. Така се напълват контейнери: managerappointments и managerdepartures.
- Последната таблица отговаря за импортирането на снимките в premierleaguwinners и premierleaguerunnersup като следва стъпките на предходните конфигурационни файлове.

### 3.2.3 Автентикация и регистрация

За целият процес по автентикация и регистрация в моето приложение използвам Spring Security. Този фреймуърк предоставя вече изградена система за автентикация, след неговото инсталиране към проекта го модифицирах, така че да бъде полезен за моят проект. По-горе обясних за UserDetails интерфейсът, който имплементирам в моя entity user клас. В класа WebSecurityConfig в папка config е имплементирана моята модификация на базовата част на Spring Security. Важно е да бъде анотиран с configuration, за да бъдат инициализиран всеки негов Bean. Също е важно да бъде анотиран с EnableWebSecurity, по този начин изключваме режимът на работа по подразбиране, за да имплементираме наш.

- DaoAuthenticationProvider - по подразбиране при автентикация Spring Security търси UserDetailsService, за да знае как да обработи заявката, в моето приложение обаче, заявката искам да бъде обработена от мен, за това правя мой собствен UserDetailsServiceImpl, където имплементирам UserDetails и пренаписвам метода loadUserByUsername(String email), защото при мен този метод използва имейл, за това в UsersRepository създавам метод findByEmail(String email). Също искам моето приложение да използва BCryptPasswordEncoder. Залагането на приложението да използва моя сървис и моя начин за криптиране на пароли се настройва в този Bean.
- securityFilterChain(HttpSecurity http) - тук можем да заложим нашият authenticationProvider, изключвам csrf, защото ще подсигурия крайните си точки (endpoints) сам. В следващите редове разрешавам достъпа до всички крайни точки започващи с “/registration”. Със следващият ред .anyRequest().authenticated() забранявам достъпа до всички остали крайни точки без автентикация. При желание за достъпване на друга крайна точка потребителят ще бъде пренасочен

в страницата за login. В следващите редове заменям login страницата на spring boot с моя собствена намираща се в папката templates на проекта и с име login.html. Също смених страницата за пренасочване след регистрация с моята “/home” и пренасочващият линк генериран при грешка по време на login. Важно е отново да се отбележи, че в моята форма за username се използва email.

```
@Bean
public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {

    http.authenticationProvider(authenticationProvider());

    http.csrf(AbstractHttpConfigurer::disable);

    http
        .authorizeHttpRequests((requests) -> requests
            .requestMatchers(...patterns: "/registration", "/registration/**").permitAll()
            .anyRequest().authenticated()
        )
        .formLogin((form) -> form
            .loginPage("/login")
            .usernameParameter("email")
            .failureUrl( authenticationFailureUrl: "/login-error")
            .defaultSuccessUrl("/home")
            .permitAll()
        );

    return http.build();
}
```

*фиг. 16 Метод securityFilterChain*

В случай, че потребителят няма акаунт съм създал метод registerUser в класа UserDetailsService.

```

public String registerUser(User user, Model model){

    String tokenString = UUID.randomUUID().toString();

    ConformationToken token = new ConformationToken();

    token.setToken(tokenString);
    token.setCreatedAt(LocalDateTime.now());
    token.setExpiresAt(LocalDateTime.now().plusMinutes(15));

    String link = "http://localhost:8080/registration/confirm?token=" + tokenString;

    if(repository.findByEmail(user.getEmail()).isPresent()){

        User foundedUser = repository.findByEmail(user.getEmail()).get();

        if(foundedUser.getUsername().equals(user.getEmail()) && !foundedUser.isEnabled()){
            token.setUser(foundedUser);
            service.save(token);
            emailService.send(foundedUser.getEmail(), buildEmail(foundedUser.getFullName(), link));

            model.addAttribute(attributeName: "message", attributeValue: "Check your email for verification");
        }else {
            model.addAttribute(attributeName: "message", attributeValue: "Account with this email already exists");
        }
    }
    return "registration";
}

```

```

65     }
66
67     String encodedPassword = encoder.bCryptPasswordEncoder().encode(user.getPassword());
68
69     user.setPassword(encodedPassword);
70
71     repository.save(user);
72
73     token.setUser(user);
74
75     service.save(token);
76
77     emailService.send(user.getEmail(), buildEmail(user.getFullName(), link));
78
79     model.addAttribute(attributeName: "message", attributeValue: "Check your email for verification");
80     return "registration";
81 }

```

### *фиг. 17 Метод registerUser*

В него взимам подаденият User и генерирам уникален символен низ за потвърждаване на профил и правя нов запис в таблицата с тях като за момент на създаване подавам сегашното време и дата, за момент на изтичане същото с добавени 15 минути, след това няма да може да бъде потвърден профилът и ще трябва да бъде поискан нов низ за

потвърждение. Създавам линкът, който трябва да бъде посетен, за да се активира профилът и изпращам имейл с него, като преди това проверявам дали вече няма създаден профил с този имейл. Накрая запазвам профилът в базата. При отваряне на линкът за потвърждение се извиква метода `confirmToken`.

```
@Transactional
public String confirmToken(String token) {

    ConformationToken conformationToken = service.getToken(token);

    if (conformationToken.getConfirmedAt() != null) {
        throw new IllegalStateException("email already confirmed");
    }

    service.setConfirmedAt(token);

    String email = conformationToken.getUser().getEmail();

    if(repository.findByEmail(email).isPresent()){
        User user = repository.findByEmail(email).get();
        user.setEnabled(true);
        repository.save(user);
    }

    return "login";
}
```

*фиг. 18 Метод registerUser*

В него се задава момент на активация на низът - сегашните час и дата след проверка за все още валиден низ в метода `setConfirmedAt` бива заложена стойност 1 в полето `enabled` за този потребител в базата, след което е пренасочен към логин страницата.

### 3.2.4 Показване на новини за отборите след търсене

Извиква се метода `getTeamNews` в който се подава низ от символи с името на отбора, който се търси. Метода е описан в `NewsServiceImpl` в папка `services` на проекта. В него преправям подаденият низ, така че да мога да взема новини за подаденият отбор от BBC news чрез Jsoup. От всяка новина взимам по едно изречение, снимка, заглавие и час на качване, които да бъдат показвани на потребителя.

```
19: public List<News> getTeamNews(String URIWithNews) throws IOException {  
20:  
21:     int indexOfStart = 0;  
22:     int indexOfEndArticlesList = 3;  
23:     int indexOfBeginCreationMoment = 12;  
24:     int lengthOfUsefulPart = 13;  
25:     int duplicatedString = 2;  
26:  
27:     Document document = Jsoup.connect(URIWithNews).get();  
28:  
29:     Elements elements = document.select(cssQuery: "article");  
30:  
31:     List<Element> firstThreeArticles = elements.subList(indexOfStart, indexOfEndArticlesList);  
32:  
33:     List<News> newsList = new ArrayList<>();  
34:     for (Element article : firstThreeArticles) {  
35:  
36:         Element header = article.selectFirst(cssQuery: "header");  
37:         Element firstImage = article.selectFirst(cssQuery: "figure img");  
38:         String firstSentence = article.selectFirst(cssQuery: "p").text();  
39:  
40:         int index = header.text().indexOf("published at");  
41:         String publishMoment = header.text().substring(index);  
42:         String extractFromFirstSentence = publishMoment.substring(indexOfBeginCreationMoment);  
43:         publishMoment = publishMoment.substring  
44:             (indexOfStart, publishMoment.length() - (publishMoment.length() - lengthOfUsefulPart)/duplicatedString);  
45:         String title = header.text().substring(index, index);  
46:  
47:         firstSentence = firstSentence.replace(extractFromFirstSentence, replacement: "");  
48:  
49:         News news = new News(title, firstImage.attr(attributeKey: "src"), firstSentence, publishMoment);  
50:         newsList.add(news);  
51:     }  
52:  
53:     return newsList;  
54: }
```

*фиг. 19 Метод `getTeamNews`*

### 3.2.5 Основни крайни точки(endpoints) на проекта

За да се осъществява комуникация между клиентът и сървърът използвам следните крайни точки:



- POST /registration - След успешна регистрация потребителят получава имейл с връзка за активиране на профила си.
- GET /registration/confirm{token} - След успешно активиране на профила си потребителят бива пренасочен в страницата за логин.
- POST /login - След успешно попълнена логин форма введените данни се предават за проверка
- POST /home - След успешно въведен мач, имената на введените отбори се предават като параметри на страницата с резултатът от търсенето на съответният мач и потребителят бива пренасочен към нея.
- GET /result{homeTeam}{awayTeam} - Предоставя на потребителя страница със сравнения на моментната форма на отборите участващи в търсеният от него мач, минали техни срещи, място на провеждане на мача, новини около тях и сравнение на техните треньори.

Клиентът получава тяхната template имплементация и чрез различни действия генерира различни заявки.

### **3.3 Разработване на клиентската част**

За разработването на клиентската част съм използвал следните технологии:

- HTML
- CSS
- JavaScript

- Thymeleaf

Всяка страница си има собствен темплейт в папката `template` на проекта, за да свържа темплейт с метод от сървърната част използвам Thymeleaf. Чрез нея взимам листове, символни низове, снимки и други от сървърната част на приложението и ги показвам в клиентската. Пример за показване на таблица:

```
341 <table id="awayTable">
342   <caption>Last five [[${awayTeam}]] away matches:</caption>
343   <tbody>
344     <tr th:each="match : ${lastFiveAwayTeamAwayMatches}">
345       <td class="homeTeamColumn" th:text="${match.homeTeam.getName()}"></td>
346       <td class="resultColumn" th:text="${match.result}"></td>
347       <td class="awayTeamColumn" th:text="${match.awayTeam.getName()}"></td>
348       <td th:text="${match.date}"></td>
349       <td th:text="${match.season}"></td>
350     </tr>
351   </tbody>
352 </table>
```

*фиг. 20 Визуализация на таблица*

Използвам `th:each` за да подреда елементите от листа с последните пет гостуващи мача на гостуващият отбор в таблица, като в съответният контролер съм заложил листа чрез поленцето от модела `lastFiveAwayTeamAwayMatches`. По този начин мога да достъпя всякакви полета и методи на подаденият лист от обекти. Поставяне на желания лист в темплейта:

```
89 matches = service.getLastFiveAwayMatchesByTeamName(awayTeam);
90 model.addAttribute( attributeName: "lastFiveAwayTeamAwayMatches", matches);
```

*фиг. 21 Визуализация на таблица*

# ЧЕТВЪРТА ГЛАВА

## РЪКОВОДСТВО ЗА ПОТРЕБИТЕЛЯ

### 4.1 Инсталация на приложението

Приложението е качено в Azure, линк към него има в readme файлът на хранилището, следващите стъпки описват процеса по инсталация на собствена машина.

Необходими технологии:

- Среда за изпълнение на MySQL заявка
- Java 21
- maven 3.8.1

След като сте изтеглили проекта от хранилището или флашката и сте отворили проекта върху избрано от вас IDE, препоръчително IntelliJ, трябва да създадете нова SQL връзка и да създадете база данни на име premierLeague. В проекта трябва да бъде зададена стойност на съответните полета в application.properties файлът, намиращ се в папка resources. В него трябва да бъдат нанесени следните данни:

```
spring.datasource.url=jdbc:mysql://127.0.0.1:3306/premierLeague
spring.datasource.username=|
spring.datasource.password=
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.jpa.hibernate.ddl-auto=update
```

*фиг. 22 Въвеждане на данни нужни за базата данни*

```
10 spring.mail.username=${EMAIL_USERNAME}
11 spring.mail.password=${EMAIL_PASSWORD}
```

*фиг. 23 Въвеждане на имейлът, от който ще се изпращат имейлите за верифициране на профил и неговата парола.*

В полето за парола не се попълва истинската парола, а генерирана парола за достъп през приложение. В секцията с литература има линк с обяснение как се прави това. Тези данни също трябва да поставят на редове 19 и 20 в единственият метод на класа MailSenderConfig в папка config на проекта ето тук:

```
14      public JavaMailSender getJavaMailSender(){
15          JavaMailSenderImpl mailSender = new JavaMailSenderImpl();
16          mailSender.setHost("smtp.gmail.com");
17          mailSender.setPort(587);
18
19          mailSender.setUsername("HERE");
20          mailSender.setPassword("HERE");
21          Properties properties = mailSender.getJavaMailProperties();
22
23          properties.put("mail.transport.protocol", "smtp");
24          properties.put("mail.smtp.auth", "true");
25          properties.put("mail.smtp.starttls.enable", "true");
26          properties.put("mail.debug", "true");
27
28          return mailSender;
29      }
```

*фиг. 24 Допълване на данни в конфигурационния файл за изпращане на имейли*

След това трябва да премахнете коментарите на всички закоментирани Order и Bean анотации от подпапката dataImportConfig.databaseImportConfig и да пуснете приложението. Не забравяйте при повторно пускане да върнете коментарите и да добавите този символен низ:

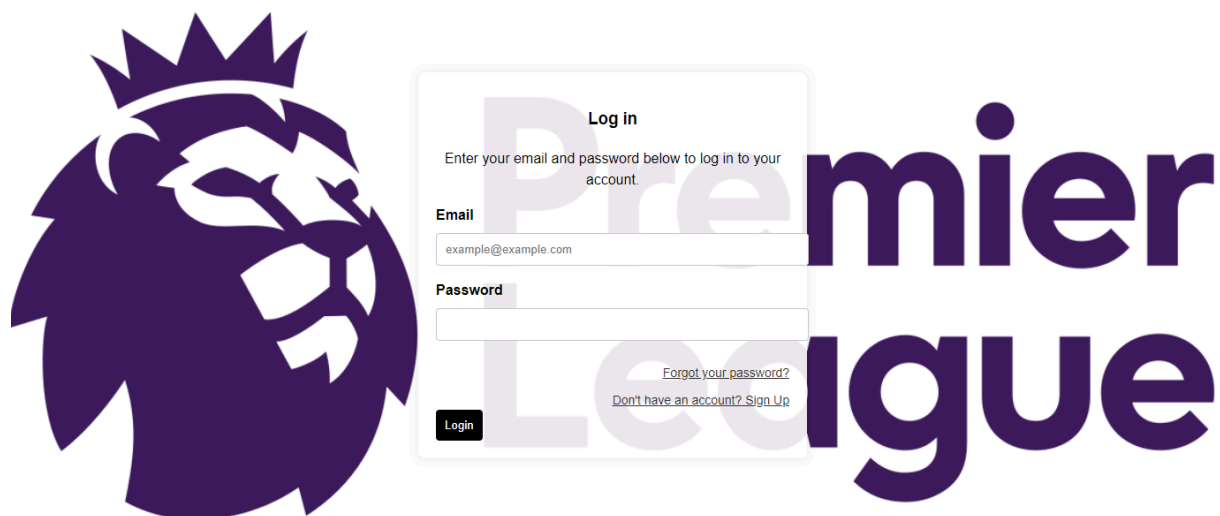
"sv=2022-11-02&ss=bfqt&srt=sco&sp=rwdlacupiytfx&se=2024-06-01T16:16:07Z&st=2024-03-20T09:16:07Z&spr=https,http&sig=BtbivvL7XoWhpRYOtkxY1QQ7iYeOxpiKtBmzWdVqIO4%3D" за достъп до облачното хранилище в конфигурационният клас CloudManagment, отново в папка config, на ред 19 ето тук:

```
17     private final BlobServiceClient blobServiceClient = new BlobServiceClientBuilder()
18         .endpoint("https://plprovider.blob.core.windows.net")
19         .sasToken("HERE")
20         .buildClient();
```

*фиг. 25 Символен низ за достъп до облачното хранилище*

## 4.2 Логин

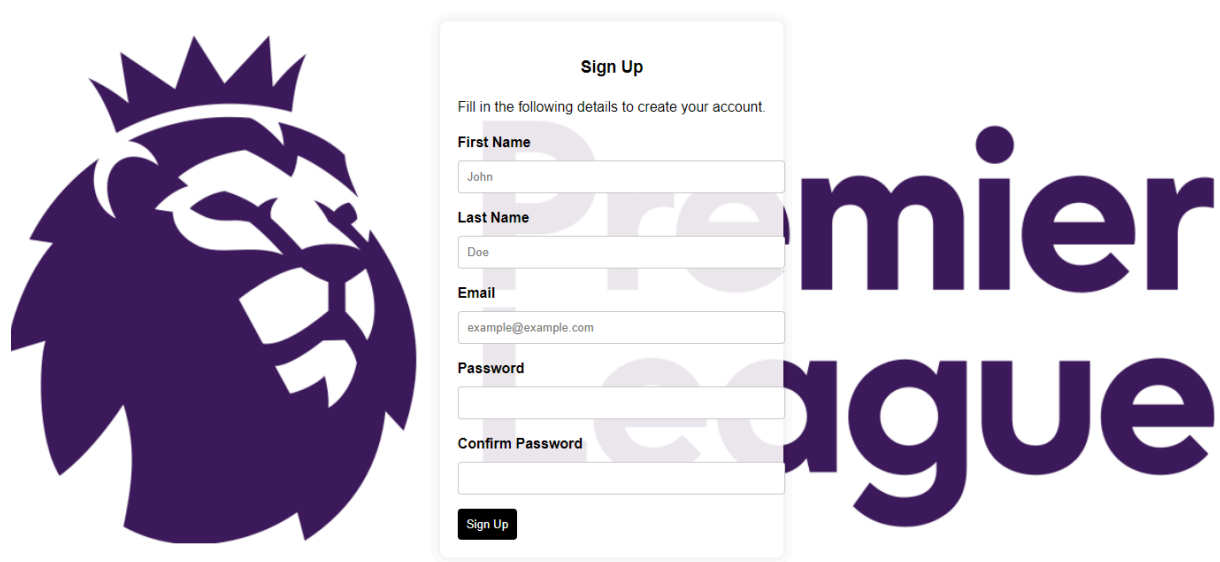
В момента приложението няма да ви позволи да достъпите ресурс различен от логин или регистрация. Ще бъдете пренасочени в логин страницата. Тъй като нямате профил, ще трябва да си направите такъв чрез линка, долу вдясно.



*фиг. 26 Логин страница*

### 4.3 Регистрация

Ще бъдете пренасочени към страница с форма на регистрация, попълнете я и натиснете sign up



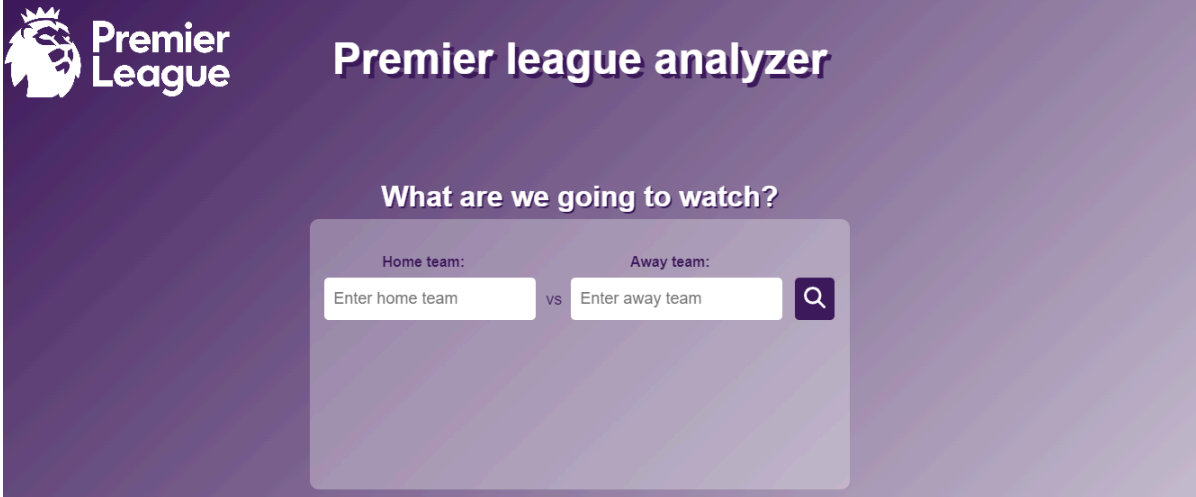
*фиг. 27 Логин страница*

при успешно направена регистрация ще получите верификационен имейл и при натискане на бутона activate now ще бъдете пренасочени към

страницата за login, ако потвърдите имейла от същото устройство от което е пуснато приложението. Въведете вашите данни и ще бъдете пренасочени към основната страница.

#### 4.4 Основна страница

Сега можете да въведете мачът, за който искате да получите статистика


The image shows a web application interface for the Premier League. At the top left is the Premier League logo. To its right, the text "Premier league analyzer" is displayed in a large, bold, white font. Below this, the question "What are we going to watch?" is centered. Underneath the question is a form with two input fields: "Home team:" and "Away team:". The "Home team:" field contains the placeholder text "Enter home team". The "Away team:" field contains the placeholder text "Enter away team". Between the two input fields is the text "vs". To the right of the "Away team:" field is a search icon (a magnifying glass). The entire form is set against a dark purple background with a subtle geometric pattern.

*фиг. 28 Страница за търсене на мач*


след въвеждане ще бъдете пренасочени в страницата съдържаща информация за дадените отбори.

## 4.5 Страница със сравнения


Това е една примерна страница за срещата между Уест хям и Фулъм

**Premier League**

**Premier league analyzer**

**West Ham United**  



**vs**

**Fulham**

Venue: London Stadium, capacity: 60000  
City: London

**Last five West Ham United home matches:**

West Ham United	3 : 1	Leeds United	21.05.2023	2022/23
West Ham United	1 : 0	Manchester United	07.05.2023	2022/23
West Ham United	1 : 2	Liverpool	26.04.2023	2022/23
West Ham United	2 : 2	Arsenal	16.04.2023	2022/23
West Ham United	1 : 5	Newcastle United	05.04.2023	2022/23



**Last five Fulham away matches:**

Manchester United	2 : 1	Fulham	28.05.2023	2022/23
Southampton	0 : 2	Fulham	13.05.2023	2022/23
Liverpool	1 : 0	Fulham	03.05.2023	2022/23
Aston Villa	1 : 0	Fulham	25.04.2023	2022/23
Everton	1 : 3	Fulham	15.04.2023	2022/23

**Current Premier League Standings**

фиг. 29 Страница с информация за търсеният мач



## ЗАКЛЮЧЕНИЕ

При разработката на дипломната работа се постарях да използвам всичко научено от престоя ми в ТУЕС. Темата, която избрах, ми позволи да използвам най - различни иновативни технологии и успях да създам един развлекателен и интригуващ сайт, който може да бъде използван от всеки.

В тази дипломна работа беше разработено WEB приложение за анализ на футболни мачове. Системата е изградена по моделът клиент - сървър. В него клиентът и сървърът си комуникират чрез HTTP заявки. Сървърната част поддържа следните функционалности:

- Автентикация и регистрация
- Потвърждаване на профил чрез имейл
- Съхранение на информация в база данни
- Съхранение на снимки в облачно хранилище

По време на разработването на сървърната част бяха използвани различни софтуери, с помощта на които да се постигнат по - добри функционалности, например използването на Azure като място за съхранение на снимки.

Клиентската част поддържа следните функционалности:

- Удобен за използване потребителски интерфейс
- Показване на грешки и съобщения при нужда
- Визуализиране на сравнения между отбори

По време на разработката на клиентската част бяха използвани най - различни технологии като например thymeleaf.

За в бъдещо развитие на приложението бих подобрил следните неща:

- По - добра визуализация на страницата със статистика
- Архив от минали сезони
- Поддръжка на базата в реално време

## ИЗПОЛЗВАНА ЛИТЕРАТУРА

What is client - server architecture:

<https://www.theknowledgeacademy.com/blog/client-server-architecture/>

Monolithic vs Microservices Architecture:

<https://www.geeksforgeeks.org/monolithic-vs-microservices-architecture/>

Документацията на Sprig:

<https://spring.io/projects/spring-boot>

Spring security:

<https://spring.io/guides/gs/securing-web>

Azure Storage Blob client library for Java:

<https://learn.microsoft.com/en-us/java/api/overview/azure/storage-blob-readme?view=azure-java-stable>

Jsoup:

<https://jsoup.org/>

Selenium:

<https://www.selenium.dev/documentation/>

Thymeleaf:

<https://www.thymeleaf.org/doc/tutorials/3.0/thymeleafspring.html>

What is Object-Oriented Programming (OOP):

<https://www.techtarget.com/searchapparchitecture/definition/object-oriented-programming-OOP>

Relational vs. Non-Relational Database: Pros & Cons:

<https://aloha.co/blog/relational-vs-non-relational-database-pros-cons>

Как се генерира парола за достъп до имейл чрез приложение:

<https://support.google.com/accounts/answer/185833?hl=bg>

Линкове за сетовите от данни:

<https://www.kaggle.com/datasets/evangower/premier-league-matches-19922022>

<https://www.kaggle.com/datasets/narekzamanyan/barclays-premier-league?resource=download&select=club.csv>

# СЪДЪРЖАНИЕ

ТЕХНОЛОГИЧНО УЧИЛИЩЕ ЕЛЕКТРОННИ СИСТЕМИ.....	1
Увод.....	4
ПЪРВА ГЛАВА.....	5
МЕТОДИ И ТЕХНОЛОГИИ ЗА РЕАЛИЗИРАНЕ НА WEB ПРИЛОЖЕНИЕ ЗА АНАЛИЗ НА ФУТБОЛНИ МАЧОВЕ.....	5
1.1 Основни принципи за реализиране на WEB приложение.....	5
1.2 Архитектури за изграждане на WEB приложение.....	5
1.3 Технологии за създаване на WEB приложения.....	10
1.3.1 React.js - JavaScript library.....	10
1.3.2 SpringBoot - Java framework.....	11
1.3.3 Node.js.....	11
1.3.4 Express.js.....	12
1.3.5 Visual Studio Code(VS Code).....	12
1.3.6 IntelliJ IDEA.....	12
1.3.7 Docker.....	13
1.3.8 MySQL.....	13
1.3.9 Microsoft Azure.....	13
1.4 Преглед на съществуващи подобни програмни системи и продукти.....	13
ВТОРА ГЛАВА.....	15
ПРОЕКТИРАНЕ НА СТРУКТУРАТА ЗА WEB ПРИЛОЖЕНИЕ.....	15
2.1 Функционални изисквания към WEB приложение за анализ на футболни мачове.....	15
2.1.1 Функционални изисквания към клиента.....	15
2.1.2 Функционални изисквания към сървъра.....	16
2.2 Съображения за избор на програмни средства и развойна среда.....	16
2.2.1 Технологии за разработване на клиент.....	16
2.2.2 Технологии за разработване на сървър.....	17
2.2.3 Автентикация и имейл верификация.....	18
2.2.4 Съхранение на данни.....	18
2.3 Проектиране на структурата на базата данни В моята база данни ще имам нужда от следните таблици:.....	21
ТРЕТА ГЛАВА.....	24
ПРОГРАМНА РЕАЛИЗАЦИЯ НА WEB ПРИЛОЖЕНИЕ ЗА АНАЛИЗ НА ФУТБОЛНИ МАЧОВЕ.....	24
3.1 Архитектура на приложението.....	24
3.2 Разработване на сървърната част.....	24
3.2.1 Изграждане на база данни и комуникация с нея.....	24
3.2.2 Импорт на необходимите данни.....	35
3.2.3 Автентикация и регистрация.....	44
3.2.4 Показване на новини за отборите след търсене.....	48
3.2.5 Основни крайни точки(endpoints) на проекта.....	48
3.3 Разработване на клиентската част.....	49

<b>ЧЕТВЪРТА ГЛАВА.....</b>	<b>51</b>
<b>РЪКОВОДСТВО ЗА ПОТРЕБИТЕЛЯ.....</b>	<b>51</b>
4.1 Инсталация на приложението.....	51
4.2 Логин.....	53
4.3 Регистрация.....	54
4.4 Основна страница.....	55
4.5 Страница със сравнения.....	56
<b>ЗАКЛЮЧЕНИЕ.....</b>	<b>57</b>