Debreceni Egyetem

Informatikai Kar

Szakdolgozat

# Fullstack labdarúgó tippjáték készítés Spring Boot és React keretrendszerekkel

## Témavezető: Készítette:

## Major Sándor Roland Szarvas Péter

## Tanársegéd Programtervező informatikus

Debrecen

2025

# Bevezetés

Az informatika és a játéktervezés rohamos fejlődése az elmúlt évtizedekben alapjaiban változtatta meg a szórakoztatóipart, lehetőséget teremtve a klasszikus játékok digitalizálására és kreatív újragondolására. A sportfogadás és a tippelős játékok sem maradtak ki ebből a folyamatból, hiszen a modern technológiák – például adatbázisok, backend és frontend keretrendszerek – segítségével új szintre emelhetők ezek az élmények. Számos weboldal létezik ma már, amely sporteseményekhez kapcsolódó tippelési lehetőségeket kínál (pl. TippmixPRO [1]), ahol a felhasználók valós időben követhetik az eredményeket és versenghetnek egymással.

Szakdolgozatom témájaként egy focimeccs-eredmény tippelős weboldal megvalósítását választottam, amely nem csupán a hagyományos tippjátékok digitális adaptációja, hanem egyedi funkciókkal bővített, interaktív élményt nyújtó platform. A projektem egy olyan online játékot hoz létre, amely ötvözi a focimeccsek kiszámíthatatlan izgalmát és a stratégiai gondolkodást, miközben a modern webfejlesztési technológiák erejét kihasználva közösségi és kompetitív elemekkel gazdagítja a játékot.

A „GoalRush” nevű játékom legfontosabb újítása a pontozási és ranglista-rendszer. A felhasználók a valós focimeccsek eredményeire tippelhetnek, és pontokat szerezhetnek az alábbiak szerint: egy tökéletesen eltalált eredmény (gólszám és nyertes csapat) 3 pontot ér, a nyertes csapat helyes megtippelése (vagy döntetlennél a döntetlen ténye, de eltérő gólszámmal) 1 pontot ad, minden egyéb esetben pedig 0 pont jár. A játék ingyenes, a pontok csupán a versengést és a ranglétrákon való előrehaladást szolgálják. A rendszer heti és havi ranglétrákat kínál, így a játékosok különböző időtávokon mérhetik össze tudásukat és szerencséjüket. A felhasználók regisztrációval saját profilt hozhatnak létre, amelyen követhetik teljesítményüket és eredményeiket.

A projekt relevanciája több szempontból is kiemelkedő. Egyrészt a focirajongók széles közönségét célozza meg, miközben a tippelés stratégiai mélysége gondolkodásra és elemzésre ösztönzi a játékosokat – mindezt ingyen, kockázat nélkül tehetik meg, szemben a legtöbb sportfogadási oldallal, ahol pénz vagy digitális kredit szükséges egy tipp leadásához. A fiatalabb generáció számára, akik a videójátékok dinamizmusához szoktak, a ranglétrák és a közösségi versengés izgalmasabbá teszi az élményt. Másrészt fejlesztési szempontból is jelentős, hiszen bemutatja, hogyan lehet valós idejű adatokat integrálni egy interaktív játékba modern technológiák – például a MongoDB adatbázis [2], a Spring Boot backend [3] és a React frontend [4] – felhasználásával. Harmadrészt a projekt demonstrálja ezeknek az eszközöknek az alkalmasságát skálázható, felhasználóbarát webalkalmazások készítésére.

A témaválasztásom személyes érdeklődésemből fakad: gyermekkorom óta rajongok a fociért, és mindig is kíváncsi voltam, hogyan lehet digitális platformokon keresztül új élményeket teremteni. A konkrét ötletet a témavezetőmmel közösen pontosítottuk, így döntöttem a Java alapú Spring Boot keretrendszer, a MongoDB NoSQL adatbázis és a React frontend kombinációja mellett. A MongoDB rugalmassága lehetővé tette a felhasználói adatok és meccseredmények hatékony kezelését, a Spring Boot stabil backendet biztosított, a React pedig modern, reszponzív felületet kínált a játékosok számára.

A jövőben szeretném továbbfejleszteni a projektet. Terveim között szerepel egy valós idejű értesítési rendszer bevezetése, amely figyelmezteti a játékosokat a közelgő meccsekre, valamint egy baráti liga funkció, ahol a felhasználók saját csoportokat hozhatnak létre. Emellett a pontozási rendszert is bővíteném, például bónuszpontokkal a különösen nehéz meccsek helyes megtippeléséért. A felhasználói felületet is tovább csiszolnám, mivel a fejlesztés során a backendre helyeztem a nagyobb hangsúlyt, így a frontend terén látok még lehetőséget a finomításra. Hosszabb távon az adatszerzést is optimalizálnám, például a külső API-k helyett saját webscraper megoldással, hogy még nagyobb kontrollom legyen az adatok felett. A fejlesztési folyamat során felmerült kihívások megoldása sok tanulást és kreativitást igényelt, amit a jövőben tovább kamatoztatnék a projekt tökéletesítésére.

# Tárgyalás

## Felhasznált eszközök

### Git és GitHub

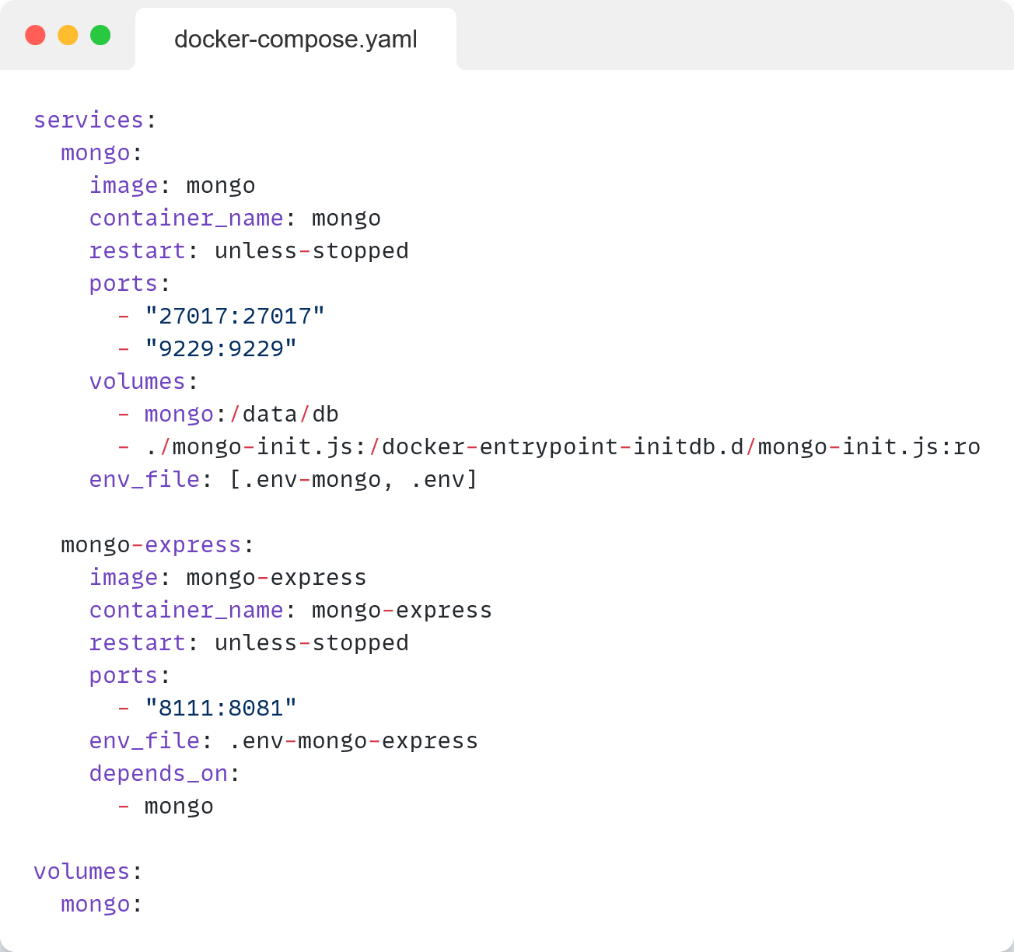
A projekt verziókezeléséhez a Git [5] rendszert választottam, amely egy nyílt forráskódú, Linus Torvalds által 2005-ben létrehozott elosztott verziókezelő rendszer (Version Control System, VCS). A Git eredetileg a Linux kernel fejlesztéséhez készült, hogy hatékonyan támogassa a kódváltozások nyomon követését és a verziók kezelését. A rendszer lehetővé teszi a fejlesztők számára, hogy rugalmasan és megbízhatóan kezeljék a projekt különböző állapotait, akár egyedül, akár csapatban dolgoznak.

A verziókezelő rendszerek használata mára elengedhetetlenné vált a szoftverfejlesztési projektekben. A Git számos előnnyel jár: lehetővé teszi a korábbi verziókhoz való visszatérést hibák esetén, támogatja a párhuzamos fejlesztési ágak (branch-ek) kezelését, amelyeket később egyszerűen össze lehet fésülni (merge). Emellett alkalmas kiadások (release-ek) kezelésére, legyen szó alfa-, béta- vagy végleges verziókról, illetve főbb verziószámú frissítésekről vagy kisebb hibajavító csomagokról.

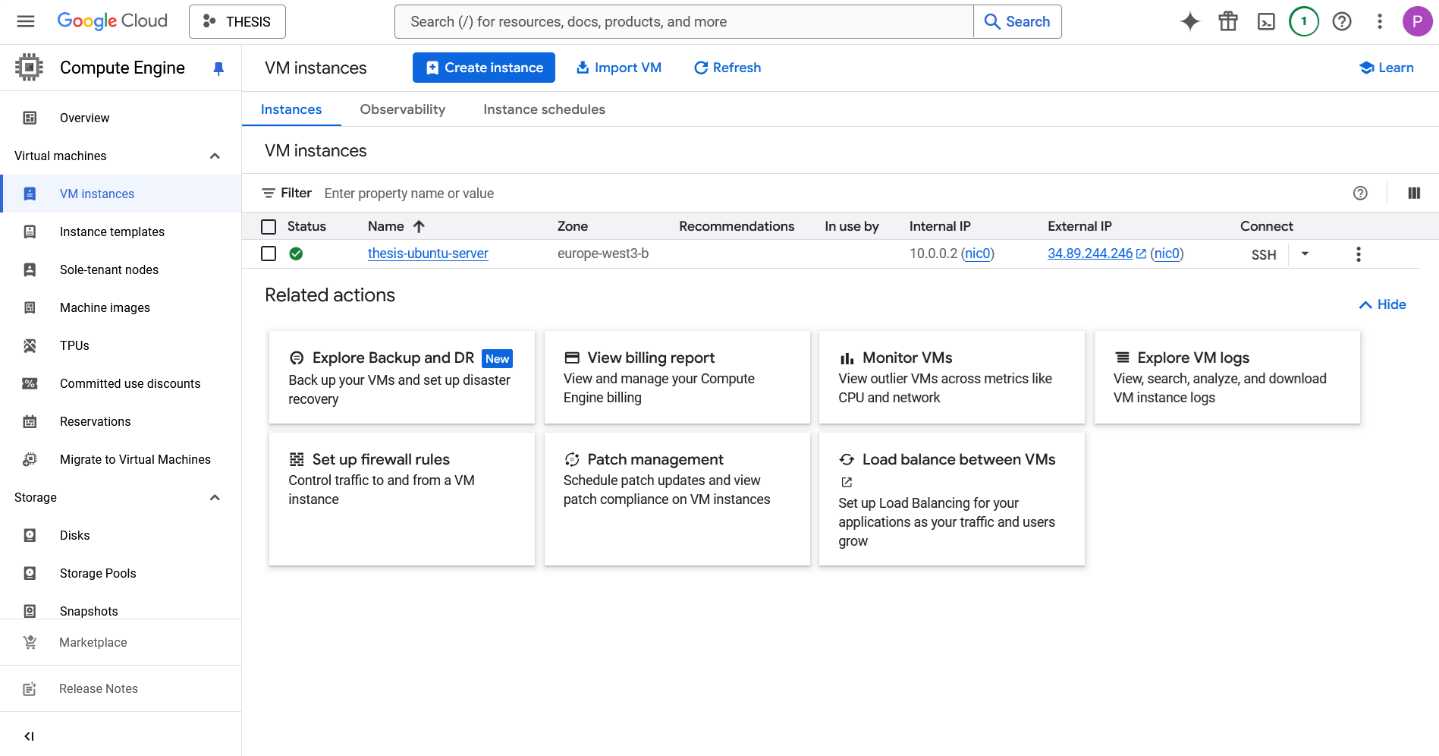
A projekt forráskódja a GitHub [6] platformon érhető el (<https://github.com/Sz4rv4s/SZAKDOLGOZAT>). A GitHub egy Microsoft által üzemeltetett, felhőalapú szolgáltatás, amely nemcsak a Git-tárhelyet biztosítja, hanem számos további funkciót is kínál, például hibajegyek (issue-k) kezelését, kollaborációs eszközöket, valamint CI/CD (Continuous Integration/Continuous Deployment) folyamatok támogatását. Ezek az eszközök jelentősen megkönnyítették a projekt fejlesztési és karbantartási folyamatait.

### Docker és Google Cloud Platform

A fejlesztés során a Docker [7] konténerizációs platformot használtam a MongoDB adatbázis és az ahhoz kapcsolódó, Express.js alapú webes adminisztrációs felület futtatására. A Docker választása azért volt előnyös, mert a projektet több különböző számítógépen fejlesztettem, így szükségem volt egy konzisztens, könnyen telepíthető és módosítható környezetre mind a fejlesztési, mind a tesztelési fázisokban. A konténerizáció lehetővé tette, hogy az alkalmazás függőségei és konfigurációi egységesek maradjanak, függetlenül a futtató környezettől.

A Docker Compose eszközt alkalmaztam a konténerek kezelésére. Ez egy YAML formátumú leírófájl segítségével definiálja a futtatandó szervizeket, azok portjait, indulási sorrendjét, függőségeit, valamint támogatja az egészségügyi ellenőrzések (health check-ek) beállítását. A Docker Compose használata jelentősen leegyszerűsítette a többkonténeres környezet kezelését és a fejlesztési folyamatok automatizálását.

1. ábra – docker-compose.yaml

A projekt éles környezetben való futtatásához a Google Cloud Platform (GCP) [8] szolgáltatásait vettem igénybe. A GCP ingyenes próbaidőszakának kihasználásával egy Ubuntu 24.04 LTS operációs rendszert futtató virtuális gépet hoztam létre, amelyen a backend és a frontend alkalmazások futnak. Az éles környezetet egy NGINX fordított proxyval, NO-IP dinamikus DNS-sel és Let’s Encrypt SSL-tanúsítvánnyal egészítettem ki, biztosítva a biztonságos és stabil hozzáférést. A GCP rugalmassága és skálázhatósága lehetővé tette, hogy a projektet hatékonyan és költségkímélő módon üzemeltessem. 

2. ábra – A Google Cloud konzol ablaka, azon belül a virtuális gépek menüpont

### MongoDB NoSQL adatbázisrendszer

A projekt adatkezeléséhez a MongoDB NoSQL adatbázisrendszert [2] választottam. A MongoDB egy nyílt forráskódú, dokumentumorientált adatbázis, amely JSON-szerű dokumentumokban tárolja az adatokat, lehetővé téve a séma nélküli, rugalmas adatmodellezést. Ez a rugalmasság előnyös volt a projekt során, mivel a felhasználói profilok, tippek és meccseredmények eltérő struktúrájúak lehettek. Annak ellenére, hogy a MongoDB nem igényel előre definiált sémát, készítettem egy init scriptet, amely sémadeklarációkat és validációkat hozott létre a konzisztencia érdekében. Ezáltal biztosítottam, hogy az adatok nagyjából azonos formátumúak legyenek, megkönnyítve az adatfeldolgozást és a lekérdezéseket.

3. ábra – Az adatbázis diagramja

A MongoDB-t egy Express.js alapú adminisztrációs felülettel is kiegészítettem, amely a Node.js környezetben futott. Az Express.js [9] egy könnyű és rugalmas webes keretrendszer, amelyet az adatbázis kezelésére és egyszerűbb műveletek (pl. adatok manuális bevitele, ellenőrzése) elvégzésére használtam a fejlesztési fázisban. Ez a felület lehetővé tette, hogy gyorsan teszteljem az adatbázis működését és az adatstruktúrákat, mielőtt a Spring Boot backend teljes mértékben átvette volna az adatkezelést.

A MongoDB nagy teljesítménye és horizontális skálázhatósága ideálissá tette a projekt számára. A felhasználók adatait (pl. név, pontszámok), a tippeket és a mérkőzések eredményeit tároltam benne, amelyeket a backend hatékonyan kezelt. A MongoDB-t Docker konténerben futtattam, így a telepítés és a konfiguráció egységes maradt a különböző környezetekben.

### Java és Spring Boot keretrendszer

4. ábra - A Mongo Express felülete

A backend fejlesztéséhez a Java programozási nyelvet [10] és a Spring Boot keretrendszert használtam. A projekt a Java 21-es verziójával készült, amely 2023-ban jelent meg mint hosszú távú támogatású (LTS) kiadás [11]. A Java 21 számos újdonságot hozott, például a virtuális szálak (Virtual Threads) teljes körű támogatását a Project Loom részeként, ami jelentősen javítja a párhuzamos műveletek hatékonyságát. Emellett a rekordok (Records) és a mintaegyeztetés (Pattern Matching) továbbfejlesztései egyszerűbbé és olvashatóbbá tették a kódot, különösen az adatkezelő osztályok esetében. A Java 21 választása a projekt jövőbiztosságát és a modern funkciók kihasználását célozta.

A Java futtatókörnyezetként az Eclipse Foundation által fejlesztett Temurin disztribúciót [12] preferáltam, amelyet az Adoptium projekt részeként terjesztenek. A Temurin egy nyílt forráskódú, ingyenesen elérhető OpenJDK implementáció, amely kiváló teljesítményt és stabilitást kínál. Előnyben részesítettem más disztribúciókkal (pl. Oracle JDK) szemben, mert közösségi támogatottsága és licencezési rugalmassága jobban illeszkedik a projekt nyílt forráskódú szellemiségéhez. A Temurin használata biztosította, hogy a fejlesztés és az éles környezet konzisztens legyen, különösen a Docker és a Google Cloud Platform integrációja során.

A Spring Bootot a REST API implementálására használtam, amely a frontend és a MongoDB közötti kommunikációt biztosította. A Spring WebClient segítségével külső API-ból gyűjtöttem be a meccseredményeket, amelyeket a MongoDB-ben tároltam el. Az így feldolgozott adatokat a REST API-n keresztül „konyhakészen” szolgáltam ki a frontend számára. A Spring Data MongoDB könyvtár zökkenőmentes integrációt biztosított az adatbázissal, míg a keretrendszer függőséginjektálása és biztonsági moduljai (Spring Security) növelték a kód hatékonyságát és olvashatóságát.

### Spring Dotenv és Lombok

A környezeti változók kezelésére a Spring Dotenv függőséget [13] alkalmaztam. Ez a könyvtár lehetővé tette, hogy az érzékeny adatokat (pl. adatbázis hitelesítő adatok) egy .env fájlban tároljam, különválasztva a forráskódtól, ami javította a biztonságot és a konfiguráció rugalmasságát.

5. ábra - A lombok annotációinak használata a Competition modellnél

A kód olvashatóságának növelésére a Lombok könyvtárat [14] használtam, amely annotációkkal automatikusan generál boilerplate kódot. A projektben kizárólag a @Builder, @Data és konstruktor annotációkat alkalmaztam, például a felhasználói és meccseredmény entitások definiálásához. Ez csökkentette a felesleges kódmennyiséget, így a logikai implementációkra fókuszálhattam.

### TypeScript és React

A frontend fejlesztéséhez a TypeScript nyelvet [15] és a React könyvtárat [4] választottam. A TypeScript statikus típusellenőrzése növelte a kód megbízhatóságát, különösen az API-hívások és az állapotkezelés során. A React komponensalapú felépítése és virtuális DOM-ja gyors és reszponzív felületet biztosított a tippelési felület, a ranglisták és a felhasználói profilok megjelenítéséhez. A TypeScript és a React kombinációja modern és skálázható frontend fejlesztést tett lehetővé.

### Tailwind CSS, Axios, Zustand, Formik, Yup és React Router

A felhasználói felület stílusának kialakításához a Tailwind CSS-t [16] használtam, amely egy utility-first CSS keretrendszer. A Tailwind CSS lehetővé tette, hogy előre definiált segédosztályok (pl. flex, p-4, bg-blue-500) használatával gyorsan és hatékonyan építsem fel a dizájnt anélkül, hogy egyedi CSS fájlokat kellett volna írnom. A projekt során az alapvető stílusozásra fókuszáltam, például a gombok, táblázatok és űrlapok megjelenésére, így a Tailwind alapértelmezett stílusai jelentősen felgyorsították a fejlesztést, és egységes vizuális megjelenést biztosítottak az alkalmazásnak.

Az API-hívások kezelésére az Axios könyvtárat [17] alkalmaztam, amely egy Promise-alapú HTTP kliens, és leegyszerűsítette a backenddel való kommunikációt. Az Axios segítségével valósítottam meg a tippek beküldését és a meccseredmények lekérdezését a Spring Boot REST API-jától. Az Axios interceptorok használatával automatizáltam a hitelesítési tokenek kezelését: a kérések előtt ellenőriztem a token érvényességét, és ha szükséges volt, automatikusan megújítottam azt egy külön API-hívással, így biztosítva a folyamatos hitelesítést a felhasználók számára. Ez a megoldás javította az alkalmazás stabilitását, és csökkentette a manuális hibakezelés szükségességét.

Az állapotkezeléshez a Zustandot [18] választottam, amely egy könnyű, Redux-szerű globális állapotkezelő könyvtár. A Zustand egyszerűsége miatt ideális volt a projekthez, mivel lehetővé tette a globális állapotok hatékony kezelését anélkül, hogy bonyolult boilerplate kódra lett volna szükség. A projektben egy dedikált store-ban tároltam a bejelentkezett felhasználó adatait, például a nevét és a hitelesítési állapotát, így a különböző komponensek könnyen hozzáférhettek ezekhez az információkhoz, például a navigáció során a bejelentkezési státusz ellenőrzéséhez.

Az űrlapok kezelésére a Formikot [19] és a Yup [20] validációs sémákat használtam. A Formik leegyszerűsítette az űrlapok állapotkezelését és validációját React környezetben, míg a Yup segítségével definiáltam a validációs szabályokat. Például a bejelentkezési és regisztrációs űrlapoknál a Yup biztosította, hogy az email mező formátuma helyes legyen, és a jelszó minimum 8 karakter hosszú legyen, így a felhasználók számára egyértelmű visszajelzést adhattam a hibákról (pl. „Érvénytelen email cím”). A Formik és a Yup együttes használata hozzájárult ahhoz, hogy az űrlapok kezelése pontos és felhasználóbarát legyen.

A navigáció kezelésére a React Routert [21] integráltam a projektbe. A React Router egy népszerű könyvtár a React alkalmazásokban történő útvonalkezeléshez, amely lehetővé tette, hogy az alkalmazás különböző oldalai (pl. főoldal, ranglista, bejelentkezés) között a böngésző újratöltése nélkül navigálhassak. A React Router BrowserRouter komponensét használtam az útvonalak definiálására, például a /login útvonal a bejelentkezési oldalt, a /leaderboard pedig a ranglistát jelenítette meg. A navigáció során a useNavigate hookot alkalmaztam, amely lehetővé tette, hogy programozottan irányítsam át a felhasználókat, például sikeres bejelentkezés után a főoldalra. A React Router biztosította, hogy a navigáció zökkenőmentes legyen, és a felhasználók könnyen váltogathassanak az oldalak között, még ha a frontend fejlesztése során az egyszerűbb megoldásokra koncentráltam is.

Ezek az eszközök – a Tailwind CSS, Axios, Zustand, Formik, Yup és React Router – együttesen biztosították, hogy a frontend működőképes és a projekt céljainak megfelelő legyen. A Tailwind CSS az egyszerű stílusozást, az Axios a backend kommunikációt, a Zustand a felhasználói kontextus kezelését, a Formik és a Yup az űrlapok kezelését, míg a React Router a navigációt támogatta, így egy koherens frontend rendszert hozva létre.

## Kivitelezési folyamatok

### Az adatbázis és az adatok tárolásának kivitelezése külső API-ból

A meccsadatok begyűjtésére a football-data.org [22] ingyenes csomagját használtam, amely egy REST API szolgáltatás, és URL paraméterekkel lehetővé teszi a kívánt adatok lekérdezését a csomag által biztosított kereteken belül. Az API használata során azonban több kihívással is szembesültem. Eredetileg egy másik, teljesen ingyenes API-t terveztem használni, amely azonban a fejlesztés közben megszűnt – sem az API, sem a hozzá tartozó weboldal nem volt már elérhető, és még a Wayback Machine [23] segítségével sem tudtam nyomát találni. Ez a váratlan helyzet gyors döntést igényelt, így a football-data.org API mellett tettem le a voksomat, bár ez kompromisszumokkal járt. Az ingyenes csomag korlátozott funkcionalitása miatt például el kellett vetnem egy eredetileg tervezett csapatösszerakó funkciót, mivel annak megvalósításához túl sok API-hívásra lett volna szükség, amit az ingyenes csomag percalapú limitjei (10 hívás percenként) nem tettek lehetővé, még időzítési megoldások alkalmazásával sem.

A felmerült kihívás új lehetőségeket is nyitott, és több alternatívát is megvizsgáltam az adatok begyűjtésére. Az egyik opció egy saját webscraper fejlesztése lett volna, amely például az eredmenyek.com [24] weboldalról gyűjti be az adatokat, és azokat az általam kívánt formátumban tárolja. A webscraping egy olyan technika, amelynek során egy program automatizáltan kinyeri az adatokat egy weboldal HTML struktúrájából, például meccsadatokat vagy eredményeket. Ehhez gyakran használnak olyan eszközöket, mint a Pythonban népszerű BeautifulSoup [28] vagy Scrapy [29], amelyek lehetővé teszik a HTML dokumentumok elemzését és az adatok strukturált formában történő kinyerését. A webscraping előnye, hogy API hiányában is lehetővé teszi az adatok begyűjtését, de számos technikai és etikai problémát vet fel.

Technikai szempontból a weboldalak gyakran alkalmaznak bot-védelmi mechanizmusokat, például a Cloudflare [30] szolgáltatásait, amelyek megnehezítik az automatizált adatgyűjtést. A Cloudflare olyan módszereket használ, mint az IP-címek ellenőrzése és a HTTP-kérések fejléceinek elemzése (pl. User-Agent string), hogy kiszűrje a gyanús forgalmat [31]. Emellett JavaScript-alapú kihívásokat is alkalmazhat, amelyek a böngésző környezetét vizsgálják, például canvas fingerprinting technikával, hogy megállapítsa, valódi felhasználóról vagy botról van-e szó [32]. A webscraperek körében bevett gyakorlatok közé tartozik a proxyk használata az IP-címek rotálására, valamint a fejlécek módosítása, hogy a kérések valódi böngészőből érkezőnek tűnjenek. Fejlett esetekben headless böngészőket, például a Puppeteer, használnak a JavaScript kihívások megkerüléséhez, de a Cloudflare rendszerei gyakran még ezeket is észlelik [33].

A projektemben azonban több okból is elvetettem a webscrapinget. Egyrészt időigényes fejlesztést igényelt volna, mivel a célzott weboldalak HTML struktúrája bármikor megváltozhat, ami a scraper meghibásodásához vezethetett volna. Másrészt a webscraping etikai kérdéseket is felvet, hiszen extra terhelést jelent a célzott weboldal számára, különösen akkor, ha az nem biztosít hivatalos API-t az adatok elérésére. Emellett fennáll annak a kockázata, hogy az oldal IP-alapú tiltással vagy más szűrőkkel, például a Cloudflare-hez hasonló rendszerekkel blokkolja a hozzáférést. A webscraping jogi szempontból is kockázatos lehet, mivel sok weboldal használati feltételei tiltják az automatizált adatgyűjtést. Ezeket a szempontokat mérlegelve végül a football-data.org API mellett maradtam, amely bár korlátozott, de stabil és dokumentált megoldást kínált, és nem vetett fel etikai vagy jogi problémákat.

Az API ingyenes csomagjának percalapú limitje (10 hívás percenként) nem tette lehetővé a valós idejű adatfrissítést, ezért egy optimalizált adatkezelési stratégiát dolgoztam ki. A Spring WebClient [25] segítségével a backend oldalon kértem le az adatokat, amelyeket aztán a MongoDB adatbázisban tároltam el. Az adatokat két kategóriába soroltam: a statikus és kevésbé dinamikus adatokat (pl. csapatok, ligák, szezonok adatai) havonta egyszer frissítettem, míg a dinamikusabb meccsadatokat (pl. eredmények, státuszok) 15 percenként. A frissítési folyamatot a Spring Scheduling [26] cron jobjaival automatizáltam, amelyek a DataFetchService osztály metódusait hívják meg ütemezett időközönként. A DataFetchService felelős az adatok lekérdezéséért a WebClient segítségével, majd azok megfelelő formátumban történő eltárolásáért a MongoDB-ben. Az így tárolt adatokat Data Transfer Object-ek (DTO-k) segítségével alakítottam át, hogy a frontend számára könnyen feldolgozható formában adhassam át őket, például a meccsek listáját vagy a ranglistákhoz szükséges adatokat.

Az adatbázis használata nem korlátozódott a külső API-ból származó adatok tárolására. A felhasználók, fogadások és pontok kezelésére is szükségem volt a MongoDB-re, így az adatbázis kapacitását teljes mértékben kihasználtam. A MongoDB választása azonban újabb technikai kihívást jelentett, mivel ez az adatbázisrendszer alapértelmezésként UUID (Universally Unique Identifier) [27] típusú azonosítókat használ, amelyek alfanumerikus karakterekből álló, kötőjellel elválasztott stringek (pl. 123e4567-e89b-12d3-a456-426614174000). Ezzel szemben a football-data.org API által biztosított azonosítók klasszikus integer alapú számok voltak, amelyeket célszerűbb volt megtartani, hiszen ezekre hivatkoztak a különböző API-végpontok. Az UUID-k használata ebben az esetben felesleges bonyolultságot jelentett volna, például egy további azonosító oszlop bevezetésével, ami redundanciát okozott volna az adatbázisban.

Bár felmerült a lehetőség, hogy másik adatbázisrendszerre váltsak – például a MariaDB-re [28], amely hagyományos relációs adatbázisrendszerként egyszerűbben kezelte volna az integer alapú azonosítókat –, a MongoDB mellett döntöttem, mert szerettem volna kipróbálni és megismerni ezt a NoSQL technológiát, amely a projekt rugalmas adatstruktúráihoz egyébként is jól illeszkedett. A problémát végül úgy oldottam meg, hogy az egész adatbázist integer alapú azonosítókra állítottam be. A külső API-ból származó adatoknál ez nem jelentett nehézséget, mivel azok már eleve integer azonosítókat tartalmaztak. A saját kollekciókhoz – például a felhasználókhoz (users) és fogadásokhoz (match\_score\_bets) – azonban egy egyedi megoldást kellett implementálnom. Ehhez létrehoztam egy segédkollekciót, amelyet DatabaseSequence-nek neveztem el, és egy SequenceGeneratorService osztályt, amely a szekvenciák kezelését végzi. Amikor új felhasználó vagy fogadás jött létre, a SequenceGeneratorService növelte a megfelelő szekvenciát (pl. user\_sequence, match\_score\_bet\_sequence), és ennek alapján generált egy új integer alapú azonosítót. Ez a megoldás biztosította az azonosítók konzisztenciáját az egész adatbázisban, és elkerültem az olyan hibákat, amelyek az azonosító típusok (pl. Integer vs. String) eltéréséből adódhattak volna.

A MongoDB-ben a kapcsolatok kezelése is eltér a hagyományos relációs adatbázisoktól. Bár a MongoDB támogatja a relációs jellegű kapcsolatokat, ezek használata jelentősen lassabb lehet, és a NoSQL adatbázisok esetében nem is bevett gyakorlat. Ehelyett a kapcsolatokat úgy valósítottam meg, hogy a kapcsolódó kollekciók azonosítóit tároltam a megfelelő dokumentumokban. Például a match\_score\_bets kollekcióban a userId és matchId mezők hivatkoznak a users és matches kollekciók \_id mezőire. A Spring Data MongoDB repository-k segítségével ezek az azonosítók alapján könnyen lekérdezhettem a szükséges adatokat, legyen szó meccsekről, szezonokról, ligákról vagy csapatokról. Ez a megközelítés hatékony és a MongoDB filozófiájához illeszkedő adatkezelést tett lehetővé.

A fejlesztési folyamat során felmerült kihívások – például az API megszűnése, a limitált hívásszám és az azonosítók kezelése – értékes tanulási lehetőséget biztosítottak. A választott megoldások, mint a Spring Scheduling használata az ütemezett frissítésekhez, a WebClient integrálása az API-hívásokhoz, és a DatabaseSequence segédkollekció alkalmazása az azonosítók kezelésére, lehetővé tették, hogy a projekt adatkezelési rendszere stabil és hatékony legyen, miközben a MongoDB nyújtotta rugalmasságot is ki tudtam használni.

### A felhasználókezelés és időzítések kivitelezése

### A REST szervizek és végpontok létrehozása

### A frontend elkészítése

### A tesztek és naplózások hozzáadása

### Az elkészült projekt publikálása Google Cloud Platformon keresztül

## Használati útmutató

# Összefoglalás

# Irodalomjegyzék

A projektben használt képi anyagok

Hivatkozások

[1] TippmixPRO, <https://www.tippmixpro.hu/>

[2] MongoDB hivatalos oldala, <https://www.mongodb.com/>

[3] Spring Boot hivatalos oldala, <https://spring.io/projects/spring-boot>

[4] React hivatalos oldala, <https://reactjs.org/>

[5] Git hivatalos oldala, <https://git-scm.com/>

[6] GitHub, <https://github.com/>

[7] Docker hivatalos oldala, <https://www.docker.com/>

[8] Google Cloud Platform, <https://cloud.google.com/>

[9] Express.js hivatalos oldala, <https://expressjs.com/>

[10] Java hivatalos oldala, <https://www.java.com/en/>

[11] Java 21 dokumentáció, <https://openjdk.java.net/projects/jdk/21/>

[12] Eclipse Temurin, <https://adoptium.net/temurin/>

[13] Spring Dotenv, <https://github.com/paulschwarz/spring-dotenv>

[14] Lombok hivatalos oldala, <https://projectlombok.org/>

[15] TypeScript hivatalos oldala, <https://www.typescriptlang.org/>

[16] Tailwind CSS hivatalos oldala, <https://tailwindcss.com/>

[17] Axios hivatalos oldala, <https://axios-http.com/>

[18] Zustand GitHub, <https://github.com/pmndrs/zustand>

[19] Formik hivatalos oldala, <https://formik.org/>

[20] Yup GitHub, <https://github.com/jquense/yup>

[21] React Router hivatalos oldala, <https://reactrouter.com/>

[22] A football-data.org hivatalos oldala, <https://www.football-data.org>

[23] Az eredmények magyar nyelvű oldala, <https://www.eredmenyek.com/>

[21] Football-data.org API, <https://www.football-data.org/>

[22] Wayback Machine, <https://archive.org/web/>

[23] Eredmenyek.com, <https://www.eredmenyek.com/>

[24] Spring WebClient dokumentáció, <https://docs.spring.io/spring-framework/docs/current/reference/html/web-reactive.html#webflux-client>

[25] Spring Scheduling dokumentáció, <https://docs.spring.io/spring-framework/docs/current/reference/html/integration.html#scheduling>

[26] MariaDB hivatalos oldala, <https://mariadb.org/>

[27] UUID specifikáció, <https://www.ietf.org/rfc/rfc4122.txt>

[28] BeautifulSoup dokumentáció, <https://www.crummy.com/software/BeautifulSoup/bs4/doc/>

[29] Scrapy hivatalos oldala, <https://scrapy.org/>

[30] Cloudflare hivatalos oldala, <https://www.cloudflare.com/>

[31] Cloudflare Bot Management, <https://www.cloudflare.com/products/bot-management/>

[32] Cloudflare Canvas Fingerprinting magyarázat, <https://blog.cloudflare.com/canvas-fingerprinting-protection/>

# Függelék

# Köszönetnyilvánítás