

**SAPIENTIA ERDÉLYI MAGYAR TUDOMÁNYEGYETEM  
MAROSVÁSÁRHELYI KAR,  
INFORMATIKA SZAK**



**SAPIENTIA  
ERDÉLYI MAGYAR  
TUDOMÁNYEGYETEM**

Érettségi vizsga eredményt begyűjtő és vizualizáló automatizált  
rendszer

**DIPLOMADOLGOZAT**

Témavezető:

Dr. Jánosi-Rancz Katalin Tünde,  
Egyetemi adjunktus

Végzős hallgató:

Nyitrai Orsolya Éva

**2023**

**UNIVERSITATEA SAPIENTIA DIN CLUJ-NAPOCA  
FACULTATEA DE ȘTIINȚE TEHNICE ȘI UMANISTE,  
SPECIALIZAREA INFORMATICĂ**



**UNIVERSITATEA  
SAPIENTIA**

Sistem automatizat de colectare și vizualizare a rezultatelor de la  
examenul de bacalaureat

**LUCRARE DE DIPLOMĂ**

Coordonator științific:

Dr. Jánosi-Rancz Katalin Tünde,  
Lector universitar sau Șef de lucrări  
**2023**

Absolvent:

Nyitrai Orsolya Éva

**SAPIENTIA HUNGARIAN UNIVERSITY OF  
TRANSYLVANIA**  
**FACULTY OF TECHNICAL AND HUMAN SCIENCES**  
**COMPUTER SCIENCE SPECIALIZATION**



**SAPIENTIA**  
HUNGARIAN UNIVERSITY  
OF TRANSYLVANIA

Automated baccalaureate exam result gatherer and visualizer  
system

**BACHELOR THESIS**

Scientific advisor: Dr. Jánosi-Rancz Katalin Tünde,  
Student: Nyitrai Orsolya Éva  
Lecturer  
**2023**

## LUCRARE DE DIPLOMĂ

Coordonator științific:  
Dr. Jánosi-Rancz Katalin Tünde

Candidat: Nyitrai Orsolya-Éva  
Anul absolvirii: 2023

- a) Tema lucrării de licență:** Tema lucrării este dezvoltarea unui sistem automatizat de colectare și vizualizare a rezultatelor de la examenul de bacalaureat.
- b) Problemele principale tratate:** Lucrarea de licență tratează următoarele probleme: adunarea datelor despre intuițiile de învățământ și adunarea rezultatelor de la examenul de bacalaureat din 2014 până în 2023; normalizarea și extinderea datelor adunate, respectiv convertirea acestora într-o bază de date optimizată; implementarea unui API bazat pe aceste date; crearea unei aplicații web care să ofere utilizatorilor o interfață grafică ușor de folosit, prin care aceștia pot analiza și căuta în aceste date.
- c) Desene obligatorii:** Desenele obligatorii sunt diagrama de cazuri de utilizare, diagrama arhitecturii sistemului, diferențele vizualizări și statistici bazate pe datele colectate.
- d) Softuri obligatorii:** Partea de colectare de date a sistemului este bazată pe tehnologia GoLang, care navighează pe un website și descarcă conținutul HTML al acestuia, folosind librăria Geyizor, urmând ca acesta să extragă datele necesare din HTML și le salvează în format JSON. Scriptul de prelucrare a datelor este realizat în tehnologia Python, în interiorul unui Jupyter Notebook. Aceasta încarcă datele JSON mai apoi le prelucrează folosind librăriile pandas și numpy, ulterior le salvează în format CSV. Sistemul de convertire a datelor din format CSV în SQL este bazat pe tehnologii JavaScript și TypeScript, folosind framework-ul Next.js, în care sunt utilizate următoarele librării: React, PrismaORM, React-Query, React-Table, DanfoJS. API-ul este implementat cu ajutorul tehnologiilor Node.js și Nest.js. Pentru comunicare cu baza de date se folosește PrismaORM, iar pentru autentificarea utilizatorilor se folosește Clerk. Aplicația web principală este realizat cu ajutorul acelorași tehnologii ca și sistemul de convertirea datelor din CSV în SQL, și încă câteva librării extra: Clerk, React-PDF, TailwindCS, i18next, D3js, Recharts, Three.js, react-share.
- e) Bibliografia recomandată:** Bibliografia recomandată este alcătuită din documentațiile fiecărei tehnologii utilizate pentru dezvoltarea sistemului, respectiv articole web despre webscraping într-un mod etic și despre data science în Python.  
<http://myweb.sabanciuniv.edu/rdehkharghani/files/2016/02/The-Morgan-Kaufmann-Series-in-Data-Management-Systems-Jiawei-Han-Micheline-Kamber-Jian-Pei-Data-Mining.-Concepts-and-Techniques-3rd-Edition-Morgan-Kaufmann-2011.pdf>
- f) Termene obligatorii de consultații:** Studentul a început dezvoltarea aplicației în mai 2022 și ne am întâlnit la fiecare 2-3 săptămâni
- g) Locul și durata practiciei:** Universitatea „Sapientia” din Cluj-Napoca,  
Facultatea de Științe Tehnice și Umaniste din Târgu Mureș, sala / laboratorul 327A  
Primit tema la data de: mai 2022  
Termen de predare: 2 iulie 2023

Semnătura Director Departament



Semnătura responsabilului  
programului de studiu



Semnătura coordonatorului



Semnătura candidatului



## Declarație

Subsemnatul/a NYITRAI ORSOLYA - EVA absolvent(ă) al/a specializării .....INFORMATICA....., promoția 2023.. cunoscând prevederile Legii Educației Naționale 1/2011 și a Codului de etică și deontologie profesională a Universității Sapientia cu privire la furt intelectual declar pe propria răspundere că prezenta lucrare de licență/proiect de diplomă/disertație se bazează pe activitatea personală, cercetarea/proiectarea este efectuată de mine, informațiile și datele preluate din literatura de specialitate sunt citate în mod corespunzător.

Localitatea, CORUNCA

Data: 2023. 06 14.

Absolvent

Semnătura.....Wyo.....

# Kivonat

Dolgozatom témája egy olyan automatizált rendszer megvalósítása, amely a romániai diákok érettségi eredményeit hivatott begyűjteni, tisztítani majd vizualizálni a végfelhasználók részére.

Az érettségi eredmények kiemelkedő fontossággal bírnak mind a diákok és tanárok, mind pedig az oktatási intézmények számára. Ezek az adatok nagyban befolyásolják az előbbiektől jövőbeli lehetőségeit, akár karrierlehetőségekről, akár továbbtanulásról lenne szó, illetve tükrözik az oktatás minőségét az iskolákban. Az eredmények vizualizálása alkalmat ad az oktatás hatékony értékelésére is, hiszen az adatok alapján mindenki tudásáról és fejlődéséről, amely segítheti a tanulási folyamat jobbá tételeit és a tanítási program kidolgozását.

Jelen dolgozatomban azt szerettem volna elérni, hogy a célcsoport számára egy felhasználóbarát felületet biztosítsak, amely hatékonyan és gyorsan képes az adathalmazt feldolgozni. Azért éreztem szükségét egy ilyen rendszer kifejlesztésének, mert a jelenlegi lehetőségek az adatok áttekintésére korlátosak, többségük hosszadalmas és manuális folyamatokon alapszik, amely gyakran vezet hibákhoz és késedelmekhez. Emellett kiemelt figyelmet fordítottam az adatok minőségére is, hogy ezeket most már konzisztens és normalizált formában használjuk a továbbiakban.

A felhasználói felület Next.js keretrendszer segítségével került megvalósításra, amelyen belül D3 és az erre épülő Recharts adatvizualizációs eszközök nyújtottak segítséget a vizualizációk elkészítéséhez.

A szerver oldali alkalmazás Nest.js keretrendszerre épült TypeScript nyelvet használva, ez felelős az adatok kezeléséért és kiszolgálásáért.

Az architektúra alapját a begyűjtő és tisztító rendszerek képezik. A begyűjtő rendszert GoLang nyelvben fejlesztettem ki, amely alkalmas nagy mennyiségű adathalmaz hatékony kezelésére és feldolgozására. Az adatok tisztítására és előfeldolgozására pedig Python nyelvet használtam, ezen belül is a Pandas könyvtárat, amely egy népszerű adatmanipulációs eszköz. Ezekben kívül még számos könyvtárral és technológiával dolgoztam, amelyekről a későbbiekben részletesesen is szó esik majd.

A fent említett technológiák kiválasztása során a fő cél nem csak az volt, hogy a végtermék lehetővé tegye mindenki számára az adatok könnyű és gyors hozzáférését, pontosabb következtetések levonását és az eredmények jobb megértését, hanem hogy a rendszer jövőbiztos legyen és alkalmas a későbbi továbbfejlesztésekre is.

# Rezumat

Tema tezei este implementarea unui sistem automatizat care are ca scop colectarea, prelucrarea și vizualizarea rezultatelor de la examenul de bacalaureat al elevilor din România, în vederea folosirii acestora de către utilizatorii finali.

Rezultatele examenului de bacalaureat au o importanță deosebită atât pentru elevi și profesori, cât și pentru instituțiile de învățământ. Aceste date influențează în mod semnificativ viitoarele opțiuni ale elevilor, fie că este vorba despre cariere sau continuare studiilor, și reflectă calitatea educației în școli. Vizualizarea acestor rezultate oferă oportunitatea evaluării eficiente a sistemului educațional, deoarece pe baza datelor colectate, atât profesorii, cât și părinții pot obține o imagine completă asupra cunoștințelor și progresului elevilor, ceea ce poate contribui la îmbunătățirea procesului de învățare și elaborare a programelor de studiu.

Pentru implementarea interfeței utilizatorului, am folosit framework-ul Next.js, împreună cu D3 și librăria Recharts pentru realizarea vizualizărilor.

API-ul a fost realizat cu ajutorul tehnologiei Nest.js și al limbajului TypeScript, având responsabilitatea gestionării și furnizării datelor.

Arhitectura platformei se bazează pe sistemele de colectare și curățare. Componenta de colectare a fost dezvoltată în limbajul de programare GoLang, cunoscut pentru eficiență să în gestionarea și procesarea unui volum mare de date. Pentru curățarea și pre-procesarea datelor am utilizat limbajul Python, în special librăria Pandas, care este o unealtă populară de manipulare a datelor. În afară de cele menționate, am lucrat cu numeroase alte librării și tehnologii, despre care voi vorbi mai detaliat în continuare.

La selectarea acestor tehnologii, obiectivul principal nu a fost doar asigurarea unui acces facil și rapid la date pentru toți utilizatorii, obținerea de concluzii mai precise sau înțelegerea mai bună a rezultatelor, ci și crearea unui sistem pregătit pentru viitor și ușor de extins ulterior.

# Abstract

My thesis topic is implementing an automated system that aims to collect, clean, and visualize the baccalaureate exam results of Romanian students for the end users.

The baccalaureate exam results are of paramount importance not only for students and teachers but also for educational institutions alike. These data greatly influence the future possibilities of students, whether it's about career opportunities or further education, and they also reflect the quality of education in schools. Visualizing these results provides an opportunity for effective evaluation of education since parents and teachers can gain a comprehensive understanding of students' knowledge and progress based on the data. This can help improve the learning process and the development of teaching programs.

In my thesis, I aimed to provide a user-friendly interface to the target audience, which can efficiently and quickly process the given data set. I felt the need to develop such a system because the current options for reviewing the data are limited, mostly relying on time-consuming and manual processes, which often lead to errors or delays. Additionally, I paid special attention to the quality of the data to ensure its consistency and normalization for future use.

The user interface was implemented using the Next.js framework, with the assistance of D3 and the related Recharts data visualization tools for creating visualizations.

The server-side application was built with the Nest.js framework, utilizing the TypeScript language, and it is responsible for managing and serving the data.

The foundation of the architecture consists of the scraper and cleaning systems. The web scraper was developed in GoLang, which is suitable for the efficient handling and processing of large data sets. For data cleaning and pre-processing, I used the Python language, specifically the Pandas library, which is a popular data manipulation tool. Additionally, I worked with various other libraries and technologies, which will be discussed in more detail later on.

The main goal in selecting the aforementioned technologies was not only to enable easy and fast access to the data, more accurate inference drawing, and better understanding of the result for everyone but also to ensure that the system is future-proof and allows further development.

# Tartalomjegyzék

<b>1. Bevezető</b>	<b>10</b>
<b>2. Követelmények</b>	<b>12</b>
2.1. Felhasználói követelmények . . . . .	12
2.2. Rendszer követelmények . . . . .	13
2.3. Használati esetek . . . . .	14
2.3.1. Adatbegyűjtő . . . . .	14
2.3.2. Adattisztító . . . . .	15
2.3.3. Adatátalakító . . . . .	16
2.3.4. Fő webalkalmazás . . . . .	17
<b>3. Rendszer tervezés és architektúra</b>	<b>20</b>
3.1. Rendszer architektúra . . . . .	20
3.2. Adatbázis . . . . .	21
<b>4. Webalkalmazás bemutatása</b>	<b>23</b>
4.1. Alkalmazás áttekintése . . . . .	23
4.2. Alkalmazás megvalósítása . . . . .	29
<b>5. Szerver</b>	<b>32</b>
<b>6. Adatfeldolgozó rendszer</b>	<b>35</b>
6.1. Adatbegyűjtő rendszer implementálása . . . . .	35
6.2. Adattisztító rendszer implementálása . . . . .	37
6.3. Adatátalakító rendszer implementálása . . . . .	39
<b>7. Alkalmazás összehasonlítása előző verziójával</b>	<b>41</b>
<b>8. Továbbfejlesztési lehetőségek</b>	<b>44</b>
<b>Összefoglaló</b>	<b>45</b>
8.1. Verziókövetés . . . . .	45
<b>Ábrák jegyzéke</b>	<b>47</b>
<b>Táblázatok jegyzéke</b>	<b>48</b>
<b>Irodalomjegyzék</b>	<b>50</b>

# 1. fejezet

## Bevezető

Egyszer minden líceumi diák elérkezik arra a pontra tanulmányai során amikor itt az ideje megmutatni, hogy mit is gyűjtött a kosarába az úton. Ez a próbatétel a sokat emlegetett érettségi vizsga. Azonban sikerüljön ez kitűnően, vagy akár nem is olyan fényesen, egy doleg közös volt mindenki között ezen a ponton: érdekelt minket más véleménye és eredménye. Természetesen nem azt szeretném ezzel mondani, hogy ne legyünk megelégedve a saját teljesítményünkkel. Sőt, ellenkezőleg, minden ember célja az kell legyen, hogy saját magának megfeleljen, képességeihez mérten. De ugyanakkor biztosan mindenki között ott van a mások iránti kíváncsiság, és tudat alatt is hasonlítani kezdjük a saját eredményeink az övékkel, legyen szó az élet bármely területéről.

A Sapviz nevű projekt ezért is született meg. Amikor az egyetemre érkeztem már létezett egy verziója, amely a középiskolai képességvizsga eredményeit dolgozta fel. Megfogott a téma, ezért csatlakoztam én is a kis csapathoz, aki ezt fejlesztette. Útközben pedig megszületett az ötlet, hogy miért ne készíthetnénk egy hasonló felületet, csakhogy ezúttal diáklelkünk egy másik nagy mérföldköve álljon majd a középpontban. Félreértés ne essék, célunk nem az, hogy a felhasználóink egymás között versengjenek és magukat másokhoz viszonyítsák, nem ezt képviseljük. Célunk, hogy lehetőséget nyújtsunk esetleges problémák vagy éppen sikerek jobb észrevételére és felhívjuk a figyelmet olyan dolgokra, amelyek néha nem egyértelműek.

A projektem célja tehát a romániai diákok érettségi eredményeinek begyűjtése, feldolgozása és megjelenítése a felhasználóknak. Az eredmények kulcsfontosságú szerepet játszanak mind a tanulók és tanárai, mind pedig az oktatási intézmények számára. Az itt kapott értékelések lehetővé teszik számunkra, hogy átfogó képet kapunk a diákok tudásáról, képességeiről és fejlődésükről. Ez segíthet az oktatási folyamat hatékonyabbá tételeben, mivel az adatok alapján az oktatók észrevehetik a gyengeségeket és erősségeket, majd eszerint személyre szabott módszereket alkalmazhatnak a diákok további támogatására.

Az eredmények elemzése továbbá alkalmat ad az oktatási intézmények számára, hogy értékelhessék a tanítási programok hatékonyságát, az adatok alapján javíthatnak a tanterveken, erőforrások eloszlásán és egyéb módszereiken, annak érdekében, hogy a diákok a lehető legjobb körülmények között dolgozhassanak.

A platform lehetőséget kínál arra, hogy a felhasználók megismerjék más diákok és iskolák eredményeit, helyezéseit megyei vagy akár országos ranglistákon. Emellett to-

vábbi statisztikákat és vizualizációkat is elérhetnek, bongéshetnek a tantárgyak között, valamint akár egy érettségit szimuláló környezetben is gyakorolhatnak.

Végső soron, a Sapviz projekt célja, hogy a felhasználóknak egy átfogó és informatív eszközt biztosítson, egy olyan kezdeményezés, amely abban segít, hogy az eredményekből meríthessünk felismeréseket és inspirációt a további fejlődés érdekében.

## **2. fejezet**

### **Követelmények**

#### **2.1. Felhasználói követelmények**

Annak érdekében, hogy egy felhasználó regisztrálni vagy bejelentkezni tudjon a következő adatokat szükséges biztosítania:

1. E-mail cím
2. Jelszó

Egy másik lehetőség a fent említett műveletekre, ha az azokat elvégezni kívánó személy rendelkezik az alábbi fiókok egyikével:

1. Dicord
2. Facebook
3. Google
4. Microsoft

Egy adott diák eredményeinek a megtekintéséhez a következő információkat kell megadni:

1. Diák egyedi azonosítója
2. Megye, amelyben keresünk (opcionális)

Egy adott iskola eredményeinek a megtekintéséhez a következő információkat kell biztosítani:

1. Iskola egyedi azonosítója
2. Megye, amelyben keresünk (opcionális)

Adott megyében levő összes diák eredményeinek megtekintéséhez szükséges információk:

1. Megye, amelyben keresünk

Adott megyében levő összes iskola eredményeinek megtekintéséhez szükséges információk:

1. Megye, amelyben keresünk

Az érettségi vizsga tételeinek értékeléséhez a felhasználónak a következő adatok megadására van lehetősége:

1. Tétel minőségének értékelése (1-től 5-ig terjedő skálán)
2. Tétel nehézségi fokozata (könnnyű, közepes, nehéz)

A Felhasználónak lehetőségében áll a következő műveleteket elvégezni a profilján:

1. Profilkép feltöltése vagy törlése
2. Új e-mail cím hozzáadása
3. Meglevő e-mail cím törlése
4. Új fiók hozzáadása
5. Meglevő fiók törlése
6. Új jelszó beállítása

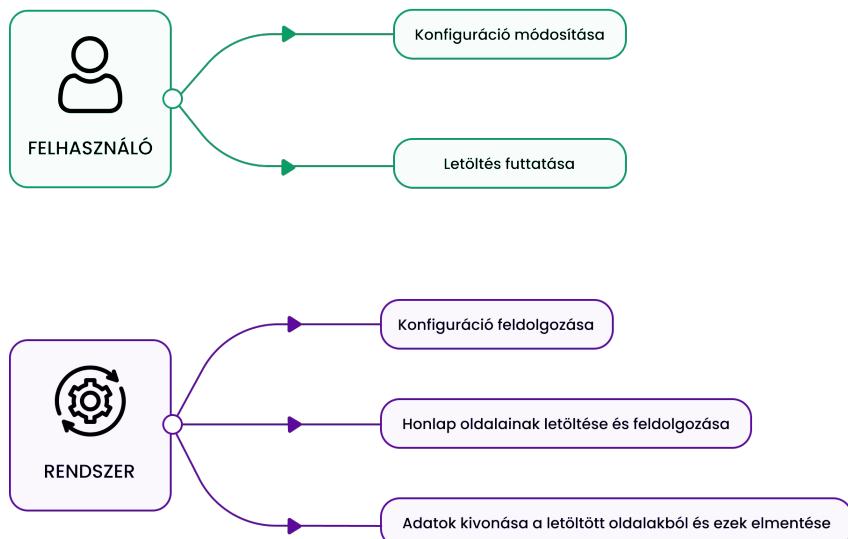
## **2.2. Rendszer követelmények**

A rendszer működéséhez szükséges egy aktív és stabil internetkapcsolat, amely lehetővé teszi a felhasználó számára a webes alkalmazás elérését és az adatok betöltését a szerverről. Használata igényel egy modern böngészőt, amely támogatja a legújabb webes technológiákat és biztonsági protokollokat. A rendszer minimális memória használatot igényel, az alkalmazás teljes mérete kevesebb, mint 10 MB, a pontos memória használat azonban függ a böngésző típusától és verziójától.

## 2.3. Használati esetek

### 2.3.1. Adatbegyűjtő

A 2.1. ábrán látható a projekt adatbegyűjtő rendszerének folyamata. Az alábbiakban részletesebben felsorolom és leírom az ehhez kapcsolódó használati eseteket.



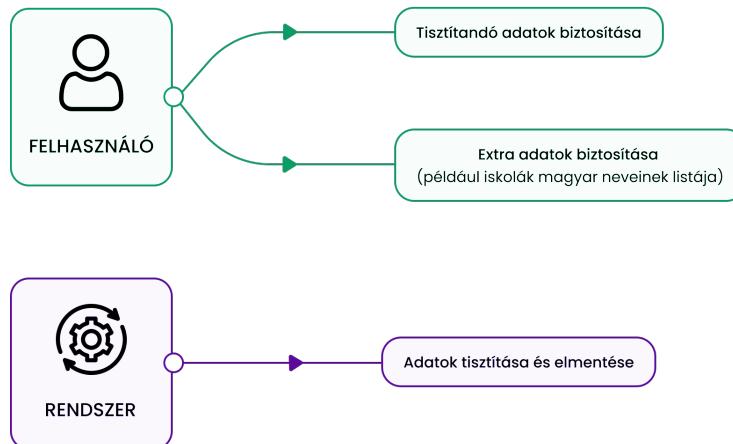
2.1. ábra. Adatbegyűjtő rendszer használati folyamatábrája

UC-0001	Érettségi eredmények letöltése
Aktorok	Felhasználó
Leírás	A Felhasználó beállítja a rendszer konfigurációját, majd elindíthatja az adatletöltést.
Előfeltételek	<ul style="list-style-type: none"><li>A Felhasználó érvényes konfigurációs beállításokat adott meg.</li><li>A Felhasználónak telepítve vannak a futtatóhoz szükséges Go moduljai.</li></ul>
Utófeltételek	<ul style="list-style-type: none"><li>A Felhasználó fájlrendszerében megjelenik egy állomány JSON formátumban, amely tartalmazza a letöltött adatokat.</li></ul>
Normál lefolyás	<ul style="list-style-type: none"><li>A Felhasználó módosítja a konfigurációs fájlt.</li><li>A Felhasználó elindítja a letöltést.</li><li>A Felhasználó megkapja az adatokat JSON formátumban.</li></ul>

2.1. táblázat. Érettségi eredmények letöltése használati eset

### 2.3.2. Adattisztító

A 2.2. ábrán látható a projekt adattisztító rendszerének folyamata. Az alábbiakban részletesebben felsorolom és leírom az ehhez kapcsolódó használati eseteket.



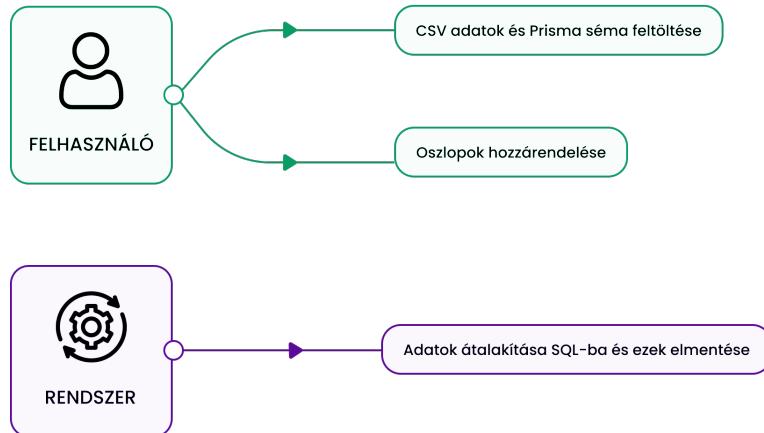
**2.2. ábra.** Adattisztító rendszer használati folyamatábrája

UC-0001	Érettségi adatok tisztítása
Aktorok	Felhasználó, Rendszer
Leírás	A Felhasználó biztosítja a tisztítandó adatokat illetve kiegészítő információkat amelyek szükségesek az eredmények kibővítéséhez. Ezt követően a rendszer feldolgozza az adatokat.
Előfeltételek	<ul style="list-style-type: none"> <li>A Felhasználó helyes kiegészítő információkat biztosított.</li> <li>A megadott adatok helyes formátumban vannak.</li> <li>A Felhasználónak telepítve vannak a futtatáshoz szükséges Python könyvtárak.</li> </ul>
Utófeltételek	<ul style="list-style-type: none"> <li>A felhasználó fájlrendszerében megjelenik egy állomány CSV formátumban, amely tartalmazza a tisztított adatokat.</li> </ul>
Normál lefolyás	<ul style="list-style-type: none"> <li>A Felhasználó feltölti a tisztítandó adatokat a helyes kiegészítő információkkal (például magyar iskolák listája).</li> <li>A Felhasználó elindítja a tisztítást.</li> <li>A Rendszer feldolgozza és normalizálja a kapott adathalmazt.</li> <li>A Felhasználó megkapja a már tisztított adatokat CSV formátumban.</li> </ul>

**2.2. táblázat.** Érettségi adatok tisztítása használati eset

### 2.3.3. Adatátalakító

A 2.3. ábrán látható a projekt adatátalakító rendszerének folyamata. Az alábbiakban részletesebben felsorolom és leírom az ehhez kapcsolódó használati eseteket.



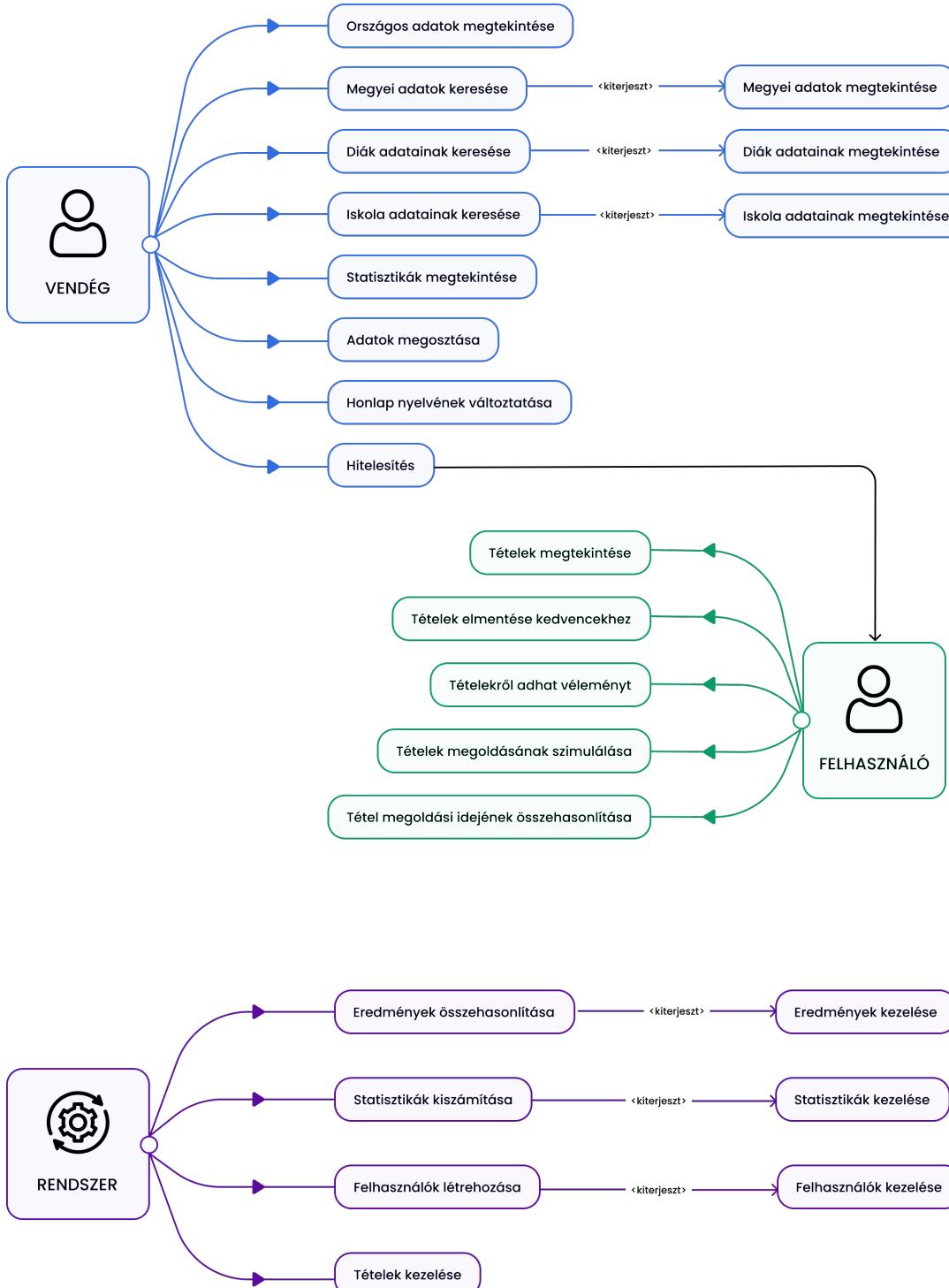
**2.3. ábra.** Adatátalakító rendszer használati folyamatábrája

UC-0001	Adatok átalakítása SQL-ba
Aktorok	Felhasználó
Leírás	A Felhasználó feltölti az átalakítandó adatokat, majd megadja a kívánt Prisma sémát. A Rendszer átalakítja ezeket SQL-ba és letölti.
Előfeltételek	<ul style="list-style-type: none"> <li>• A Felhasználó helyes adatokat biztosított.</li> <li>• A megadott séma helyes formátumban van.</li> <li>• A Felhasználó helyesen végzi el a megfeleltetést.</li> </ul>
Utófeltételek	<ul style="list-style-type: none"> <li>• Letöltésre kerül a helyes SQL fájl.</li> </ul>
Normál lefolyás	<ul style="list-style-type: none"> <li>• A Felhasználó feltölti az átalakítandó adatokat.</li> <li>• A Felhasználó feltölti vagy manuálisan kiválasztja a megfelelő Prisma sémát.</li> <li>• A Felhasználó elvégzi a helyes SQL oszlop név - CSV oszlop név megfeleltetéseket.</li> <li>• A Felhasználó elindítja az átalakítást.</li> <li>• A Felhasználó letöltheti az érettségi adatokat tartalmazó SQL fájlt.</li> </ul>

**2.3. táblázat.** Adatok átalakítása SQL-ba használati eset

#### 2.3.4. Fő webalkalmazás

A 2.3. ábrán láthatóak a projekt fő webalkalmazásának folyamatai. Az alábbiakban részletesebben felsorolom és leírom az ehhez kapcsolódó használati eseteket.



2.4. ábra. Fő webalkalmazás használati folyamatábrája

<b>UC-0001</b>	<b>Bejelentkezés és Regisztráció</b>
Aktorok	Vendég, Felhasználó
Leírás	A Vendég az oldalon új fiókot hozhat létre vagy bejelentkezhet, ha rendelkezik a következő fiókok egyikével: Discord, Google, Microsoft vagy Facebook.
Előfeltételek	<ul style="list-style-type: none"> <li>• A Vendég érvényes, helyes adatokat biztosított.</li> <li>• A Vendég rendelkezik a fenti fiókok egyikével vagy már regisztrált az oldalra.</li> </ul>
Utófeltételek	<ul style="list-style-type: none"> <li>• A Vendég Felhasználó lesz.</li> </ul>
Normál lefolyás	<ul style="list-style-type: none"> <li>• A Vendég kitölti és elküldi a regisztrációs adatokat vagy regisztrál a felsorolt fiókok valamelyikével.</li> <li>• A Vendégből Felhasználó lesz.</li> <li>• A Felhasználót átirányítjuk a főoldalra.</li> </ul>

#### **2.4. táblázat.** Felhasználó hitelesítése használati eset

<b>UC-0002</b>	<b>Adatok keresése és megtekintése</b>
Aktorok	Vendég
Leírás	A Vendég rákereshet a diákok, iskolák, megyék adataira vagy éppen országos eredményekre. A diák és iskola esetében módjában áll kezni ezek egyedi azonosítója szerint.
Előfeltételek	<ul style="list-style-type: none"> <li>• Ha azonosító szerint keres, akkor érvényes kód megadása.</li> <li>• Ha megye szerint keres, akkor létező megye kiválasztása.</li> </ul>
Utófeltételek	<ul style="list-style-type: none"> <li>• A Vendég megtekintheti az adatokat.</li> </ul>
Normál lefolyás	<ul style="list-style-type: none"> <li>• A Vendég rákeres azonosító szerint a kívánt adatra vagy kiválasztja, hogy mely megyében szeretné ezeket látni.</li> <li>• A Vendég megtekintheti a kereséshez tartozó eredményeket, statisztikákat és vizualizációkat.</li> </ul>

#### **2.5. táblázat.** Adatok keresése és megtekintése használati eset

<b>UC-0003</b>	<b>Adatok megosztása</b>
Aktorok	Vendég
Leírás	A Vendég az 'Adatok megosztása' gombra kattintva megoszthatja a diákok megyei vagy akár országos eredményeit.
Előfeltételek	<ul style="list-style-type: none"> <li>• A Vendég a főoldalon van.</li> <li>• Rendelkezik a megosztható felületek valamelyikén felhasználói fiókkal.</li> </ul>
Utófeltételek	<ul style="list-style-type: none"> <li>• A Vendég megosztotta az adatokat.</li> </ul>
Normál lefolyás	<ul style="list-style-type: none"> <li>• A Vendég a főoldalon kiválaszthatja hogy melyik megye statisztikát szeretné megosztani.</li> <li>• A Vendég rákattint az 'Adatok megosztása' gombra.</li> <li>• A Vendég kiválasztja, hogy mely szociális felületen szeretné megosztani az adatokat.</li> <li>• A Vendég megosztja az eredményeket.</li> </ul>

#### **2.6. táblázat.** Adatok megosztása használati eset

<b>UC-0004</b>	<b>Tételek kezelése</b>
Aktorok	Felhasználó
Leírás	A Felhasználónak lehetőségében áll böngészni az érettségi tételek illetve a javítókulcsok között, értékelni őket vagy akár egy érettségit szimuláló környezetben is gyakorolhat.
Előfeltételek	<ul style="list-style-type: none"> <li>• A Felhasználó be van jelentkezve az értékeléshez.</li> <li>• A Felhasználó a 'Vizsgatételek összehasonlítása' oldalon van.</li> </ul>
Utófeltételek	<ul style="list-style-type: none"> <li>• A Felhasználó megtekintheti az érettségi tételeket és javítókulcsokat, különböző műveleteket végezhet ezekkel kapcsolatosan.</li> </ul>
Normál lefolyás	<ul style="list-style-type: none"> <li>• A Felhasználó a 'Vizsgatételek összehasonlítása' oldalon van.</li> <li>• A Felhasználó megtekintheti a tételeket és javítókulcsokat azon a nyelven amelyet kiválasztott.</li> <li>• A Felhasználónak lehetőségében áll értékelni a téTEL minőségét egy 1-től 5-ig terjedő skálán. <ul style="list-style-type: none"> <li>• A Felhasználónak lehetőségében áll értékelni a téTEL nehézségét három fokozat közül választva: könnyű, közepes, nehéz.</li> <li>• A Felhasználó elindíthat egy érettségit szimuláló környezetet, amelyben elkezdődik egy 3 órás időmérő. Eközben a javítókulcs végig rendelkezésre áll.</li> <li>• A Felhasználó láthatja, hogy hányan oldották már meg az adott téTELt.</li> <li>• A Felhasználónak lehetőségében áll elmenteni az adott téTELt a kedvencekhez illetve letölteni azt.</li> </ul> </li> </ul>

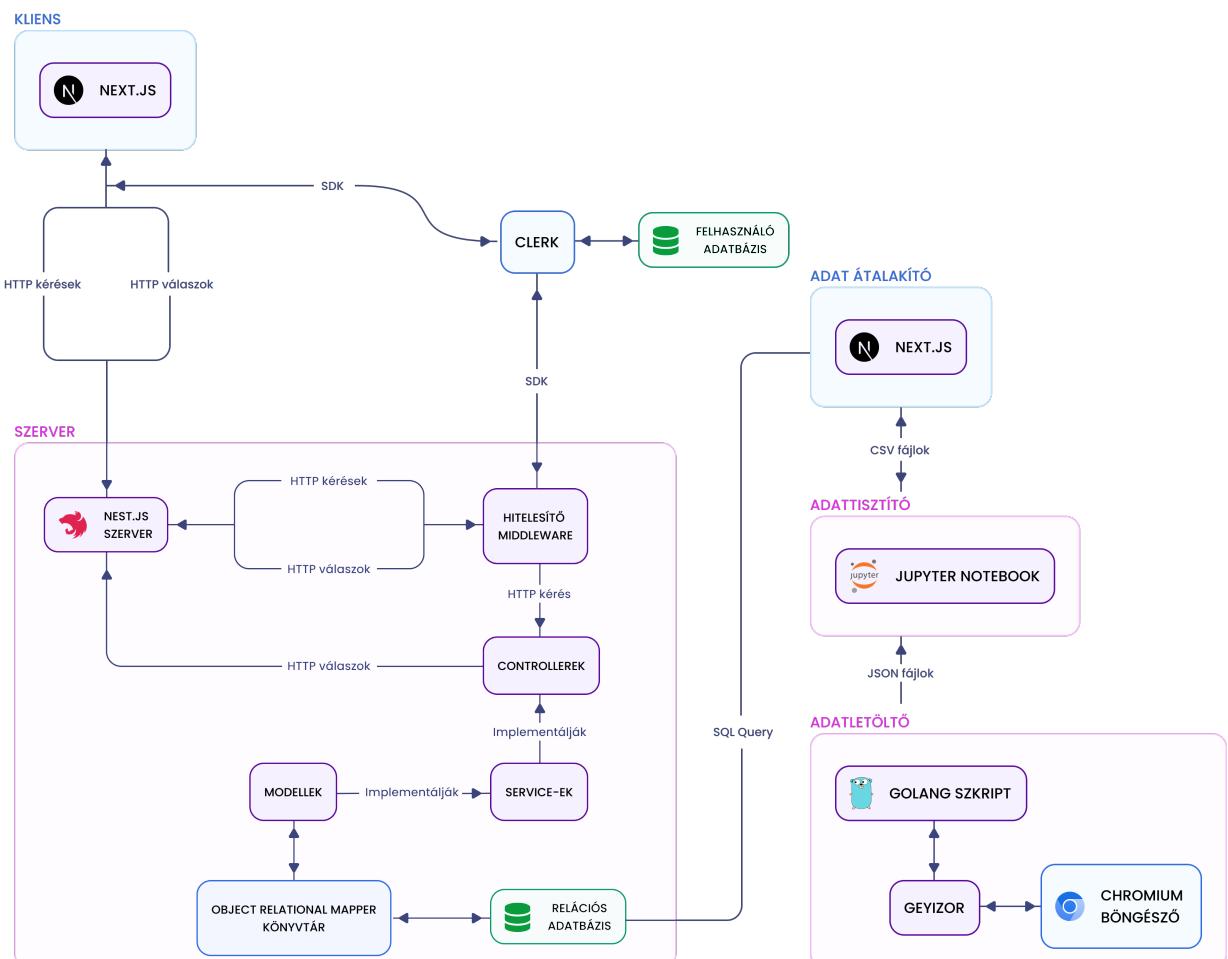
**2.7. táblázat.** Tételek kezelése használati eset

### 3. fejezet

## Rendszer tervezés és architektúra

### 3.1. Rendszer architektúra

A 3.1. ábrán látható a rendszerarchitektúra, amely az alkalmazás felépítését és működését részletezi.



3.1. ábra. Rendszer architektúrája

A rendszer két fő részre bontható le, az adat begyűjtő és feldolgozó csővezeték, illetve a végső felhasználók számára elérhető alkalmazás, amely szerepe az adatok kiszolgálása és vizualizálása. A csővezeték három komponensből épül fel, az első az adatletöltő rendszer, amely egy GoLang szkriptben került implementálásra a Geyizor[\[Gul23\]](#) webkaparó könyvtárral. A második komponens az adat tisztító, normalizáló és kiegészítő rendszer, amelyet egy Jupyter Notebook-ban sikerült megvalósítani a Pandas[\[Num23a\]](#) és a NumPy[\[Num23b\]](#) könyvtárak segítségével. Az utolsó lépés a csővezetékben a CSV-ből SQL-be átalakító web alkalmazás. Ez a Next.js keretrendszer és a Danfo.js könyvtár felhasználásával került megvalósításra.

A rendszer második fele szintén két fő komponensből áll, egy Node.js[\[Fc23c\]](#) alapú API és egy web alkalmazás. A két szoftver a REST (Representational State Transfer)[\[Wik23\]](#) architektúrára alapszik, ahol a szerver szerepét az API tölti be, a kliensét pedig a honlap. A REST tervezési elv jelenleg a legelterjedtebb a web fejlesztés köreben. Lehetővé teszi az adatok megjelenítését különböző reprezentációkban, illetve az erőforrásokkal történő interakcióhoz a HTTP protokoll műveleteit használja. Ugyanakkor a szerver nem tárol semmilyen kliens oldali állapotot a kérések között, ami csökkenti a terhelést és megkönnyíti a skálázhatóságot.

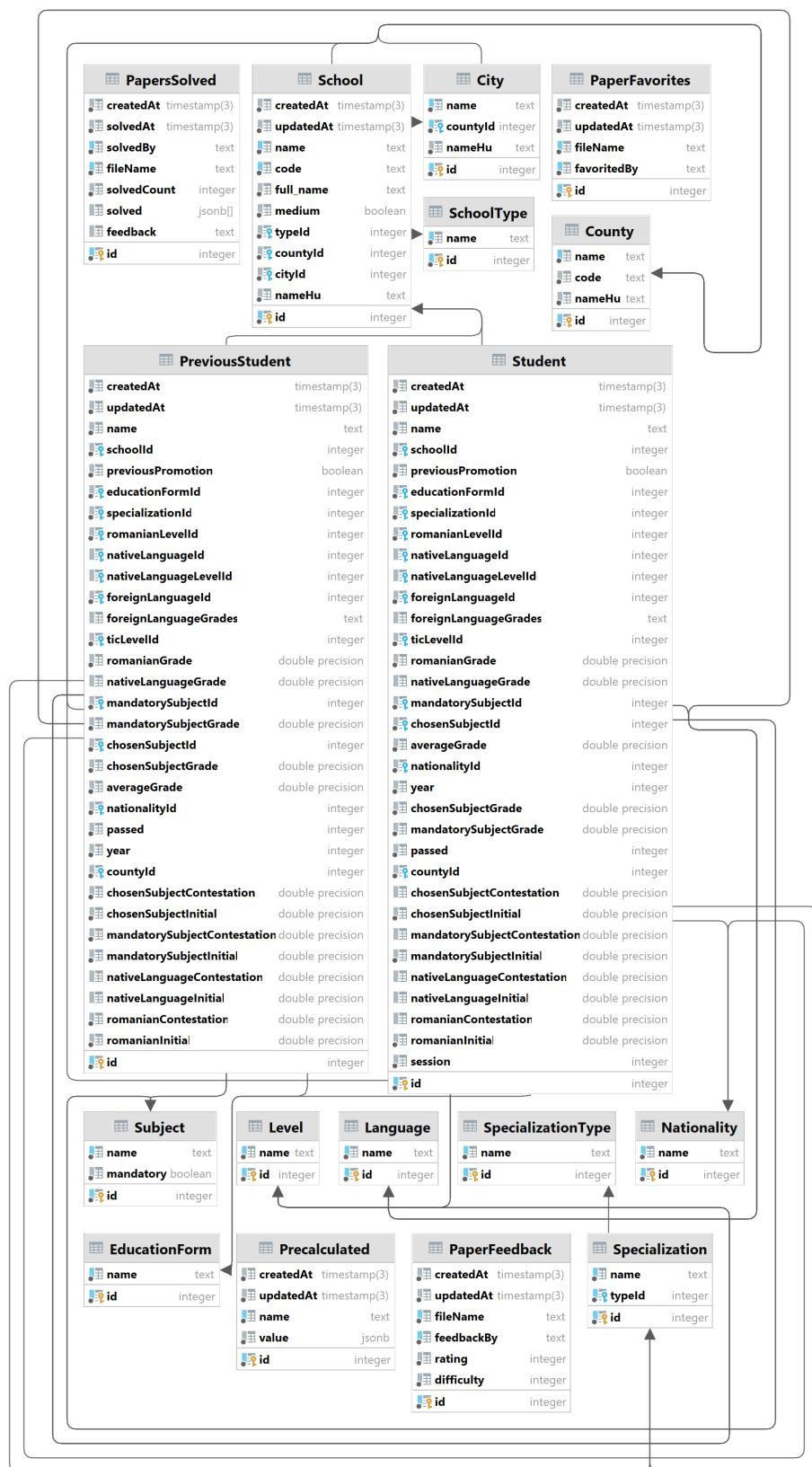
A kliens oldala a React.js-t magába foglaló Next.js keretrendszerben került kivitelezésre. Egy komponens alapú architektúra vehető észre az alkalmazásban, amely a felhasználói felületet újra hasznosítható elemekre bontja fel, ezáltal támogatja a kód modularitását és csökkenti ennek redundáns jellegét. A komponensek magukba foglalják a saját logikájukat, állapotaikat, illetve a különböző tulajdonságokat, melyeket a szülő komponensek adhatnak át nekik. Ennek eredménye egy olyan rendszer, ahol az adatok egy irányban folynak, az apáktól a gyerekhez és nagyon könnyen összeállíthatóak komplex felhasználói felület struktúrák.

## 3.2. Adatbázis

A rendszer nagy mennyiségi adatokat kell feldolgozzon és emiatt a tervezés egyik kulcsfontosságú része egy megfelelő adatbázis kezelő rendszer kiválasztása volt. Mivel az eltárolásra kerülő adatok egy szigorúan megszabott struktúrának felelnek meg, nagyon sok az ismétlődő információ és számos kapcsolat létezik az adatok között, egy relációs adatbázist választottam. A PostGreSQL mellett döntöttem, mivel ennek erőssége a gyors végrehajtása a komplex műveleteknek nagy adathalmazokon, illetve kiválóan van integrálva számos más technológiával. Ez egy fontos tényező volt számomra mert egy jövőbiztos és könnyen kibővíthető rendszert próbáltam tervezni, amelynek bármely része könnyen frissíthető vagy kicserélhető.

Ugyancsak a rendszer modularitásának elősegítése érdekéből építettem be egy absztraktiós réteget az API és az adatbázis közé, ami megengedi azt, hogy az adatbázis kezelő rendszert bármikor ki tudjam cserélni anélkül, hogy befolyásoljam az alkalmazás helyes működését. Ezt a réteget a PrismaORM[\[Dat23\]](#) könyvtár segítségével valósítottam meg, ami lehetővé tette az API implementálását úgy, hogy ez ne kelljen direkt kapcsolatban álljon az adatbázis kezelő rendszerrel. A könyvtár ugyanakkor lehetővé teszi, hogy objektum-orientált módon dolgozzunk az adatokkal, anélkül, hogy SQL lekérdezéseket írnánk a kódba, ezáltal sokkal olvashatóbbá és újra felhasználhatóbbá téve ezt.

Az adatbázis jelenlegi felépítése látható a 3.2. diagrammon.



3.2. ábra. Adatbázis felépítése

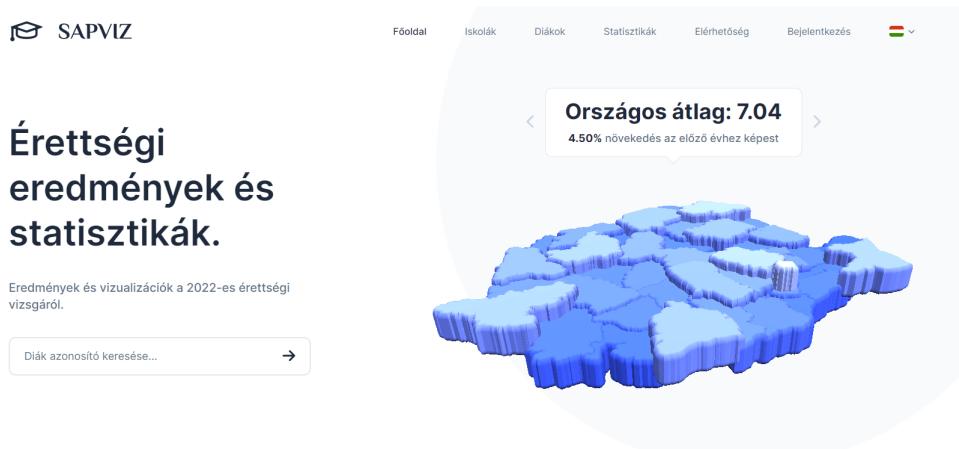
## 4. fejezet

# Webalkalmazás bemutatása

### 4.1. Alkalmazás áttekintése

A fejezet célja, hogy egy átfogó képet nyújtson a webalkalmazásról és annak főbb elemeiről. Törekedtem az implementáció során egy olyan felület létrehozására, ami felhasználóbarát, egyszerű és átlátható, lehetővé téve a megtekintőknek, hogy könnyedén megtalálják a számukra fontos információkat és funkcionálitásokat.

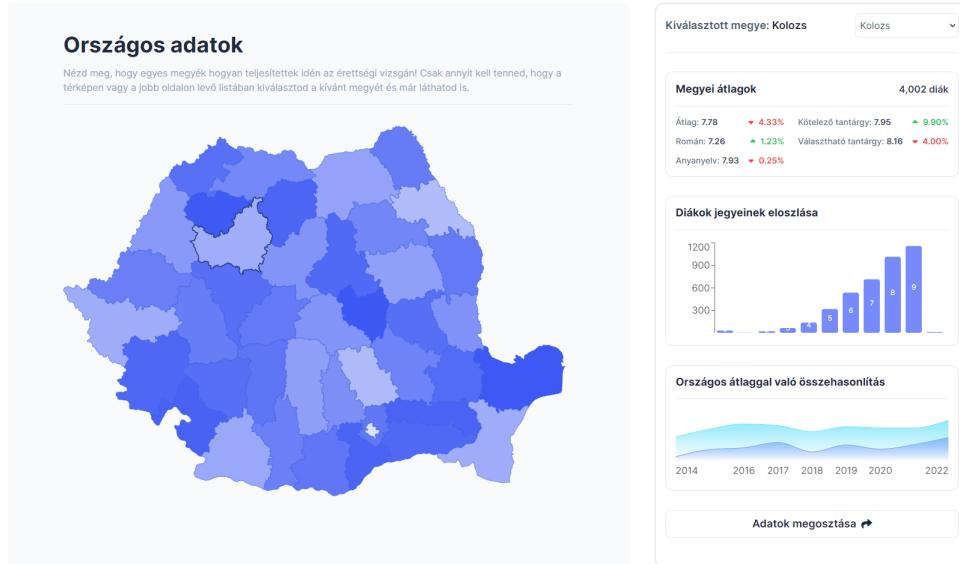
Az alkalmazás meglátogatásakor a főoldal az első nézet, amely a vendég előtt tárul, ezt láthatjuk a 4.1. ábrán. Szokták mondani, hogy az első benyomás mérvadó, éppen ezért ennek tervezésekor szem előtt tartottam, hogy a felület letisztult és könnyen kezelhető legyen. Az alkalmazást meglátogatóknak lehetősége van rákeresni adott diákok teljesítményére ezek egyedi azonosítója alapján vagy éppen különböző információkat megtekinteni az utolsó érettségi vizsgáról a térkép feletti lebegő szövegdobozkában lépegetve. A platform két nyelven érhető el: magyarul és románul.



4.1. ábra. Webalkalmazás főoldala

A kezdőoldalon lefelé görgetve, a 4.2 ábrán, láthatóvá válnak különböző statisztikák az aktuális érettségi eredményekről, a megyei átlagok, a diákok jegyeinek eloszlása, illetve az országos átlaggal való összehasonlítás. A felhasználó egyszerűen, a térkép valamely

pontjára kattintva vagy a jobb oldalon levő legördülő listából kiválaszthatja, hogy éppen mely megye adatait szeretné látni. Ugyanitt esélye van az adott megye eredményeit megosztani ismerőseivel a közösségi média platformokon, mint például Facebook, LinkedIn, Twitter, WhatsApp vagy akár e-mailen keresztül.



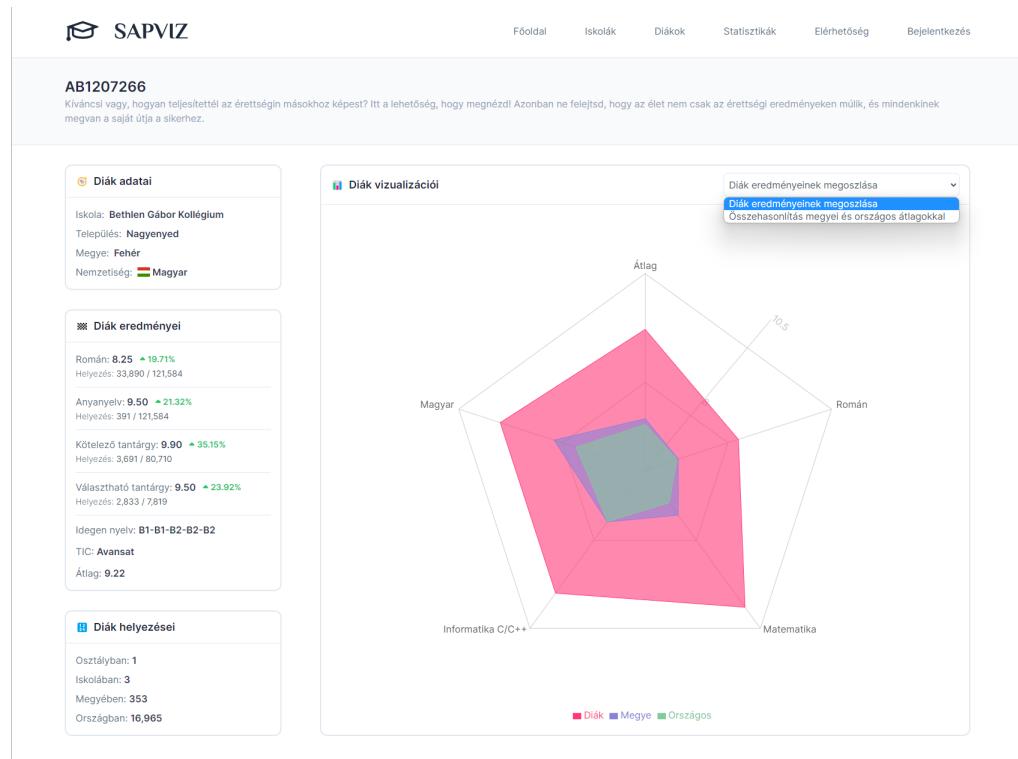
**4.2. ábra.** Megyei eredmények megtekintése a főoldalon

A 4.3. ábra szemlélteti az érettségiző diákok listáját. Ebben a nézetben a felhasználóknak lehetőségeiben áll keresni a diákok között vagy a megadott oszlopok alapján rendezni az eredményeket. A keresés funkció lehetővé teszi a tanulók gyors szűrését a keresett diák azonosítójának megadásával vagy a megye kiválasztásával.

Diákok										Fehér	2289 eredmény
#	Nemzetiségi	Tanuló kódja	Iskola	Szak	Település	Román	Anyanyelv	Kötélező tantárgy	Választható tantárgy	Átlag	
1	Román	AB1196922	Colegiul National "Inochentie Micu Clain" Blaj	Természettudományok	Balázsfalva	8.75	-	9.40 Matematika	8.25 Emberi Anatómia És Fiziológia, Genetika És Emberi Ökológia	8.80	
2	Román	AB1292937	Colegiul Economic "Dionisie Pop Martin" Alba Iulia	Bankett szervező	Gyulafehérvár	4.35	-	5.45 Matematika	7.65 Földrajz	5.82	
3	Román	AB1251510	Colegiul National "Ilu Măiorescu" Alad	Matematika-Informatika	Nagyenyed	9.30	-	9.00 Matematika	10.00 Emberi Anatómia És Fiziológia, Genetika És Emberi Ökológia	9.43	
4	Román	AB1272198	Liceul Tehnologic Sebes	Gazstronomiai technikus	Sebes	7.80	-	8.25 Matematika	8.25 Földrajz	8.10	
5	Román	AB1243742	Liceul Tehnologic Alad	Gazdasági tevékenységek technikusa	Nagyenyed	6.65	-	5.70 Matematika	6.55 Logika, Ervezés És Kommunikáció	6.30	
6	Román	AB1264735	Liceul De Arte "Regina Maria" Alba Iulia	Képzőművészeti	Gyulafehérvár	6.40	-	5.50 Történelem	7.55 Földrajz	6.48	
7	Román	AB1223222	Colegiul Tehnic "Apulum" Alba Iulia	Fordász-stillista	Gyulafehérvár	6.50	-	6.00 Matematika	8.60 Logika, Ervezés És Kommunikáció	7.03	
8	Román	AB1283954	Liceul Tehnologic Silvic Cimpeni	Gazdasági tevékenységek technikusa	Carpanesti	5.10	-	7.10 Matematika	5.75 Földrajz	5.98	
9	Román	AB1191195	Seminariul Teologic Ortodox "Sfântul Simion Stefan" Alba Iulia	Orthodox teológia	Gyulafehérvár	5.40	-	5.70 Történelem	7.80 Földrajz	6.30	
10	Román	AB1225013	Colegiul Tehnic "Apulum" Alba Iulia	Fordász-stillista	Gyulafehérvár	5.95	-	7.40 Matematika	7.40 Földrajz	6.92	

**4.3. ábra.** Diákok listája

Amennyiben a felhasználó kiválaszt egy diákot, a rendszer átirányítja őt a diák „egyedi profiljához”. A 4.4. ábrán található oldalon megjelennek a diák adatai, mint például az iskolája, a nemzetisége, az érettségin elért jegyei, illetve az, hogy a tantárgyakból országos szinten milyen helyezést ért el eredményeivel. Emellett a diák a helyezéseit megtekintheti osztályszinten, iskolai szinten, megyei és országos ranglistákon. Kétféle vizualizáció is elérhető, amelyek bemutatják a tanuló teljesítményének eloszlását, valamint a jegyeinek összehasonlítását a megyei és országos átlagokkal.



**4.4. ábra.** Diák egyedi oldala

Az iskolák esetében egy, a diákokhoz hasonló listában böngészhetünk, majd az egyik intézmény kiválasztásával megjelenik az iskola egyedi oldala, amint az a 4.5. ábrán is megfigyelhető. Ezen az oldalon számos fontos információ található a kiválasztott iskoláról. Először is, megismerhetjük az iskolából érettségiző diákok számát, az átmentek és a megbukottak arányát. Megjelennek az iskola átlagai is, tantárgyak szerinti lebontásban, illetve az oktatási intézmény helyezései megyei és országos ranglistákon. Három vizualizáció áll a felhasználók rendelkezésére. Az első az osztályok teljesítményének összehasonlítása, amely segít megérteni az osztályok közötti különbségeket. A második az osztályok jegyeinek eloszlását mutatja be, az utolsó pedig az iskola eredményeit veti össze a megyei és országos átlagokkal.

**Bethlen Gábor Kollégium**

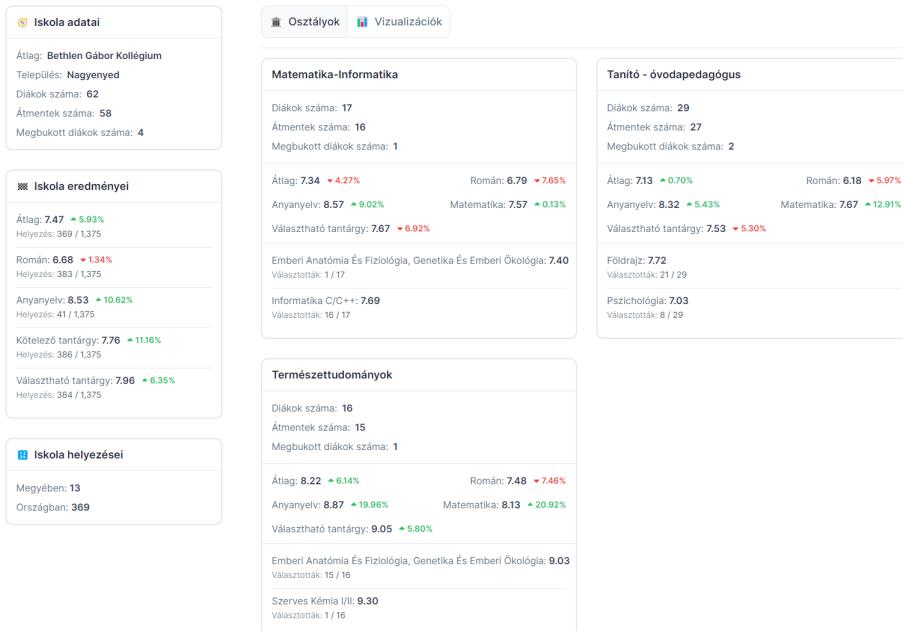
Kíváncsi vagy hogyan teljesített az iskolád idén az érettségin? Nos, jó hírünk van. Az alábbiakban ezt mind megnézheted, hiszen különböző ranglistákkal és statisztikákkal vártunk már Téged!


**4.5. ábra.** Iskola egyedi oldala

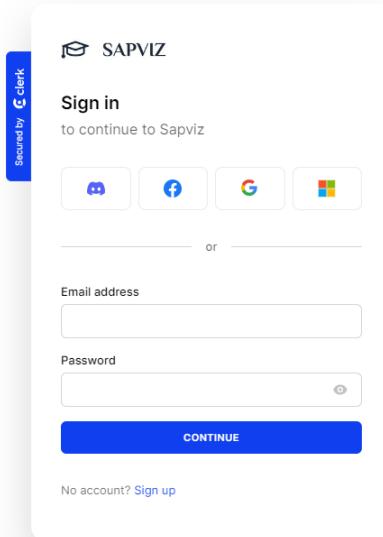
A 4.6. ábrán látható, hogy az oldalon található egy fül, amely listázza az iskola különböző szakjait és ezek teljesítményét, összehasonlítva az előző év eredményeivel.

**Bethlen Gábor Kollégium**

Kíváncsi vagy hogyan teljesített az iskolád idén az érettségin? Nos, jó hírünk van. Az alábbiakban ezt mind megnézheted, hiszen különböző ranglistákkal és statisztikákkal vártunk már Téged!


**4.6. ábra.** Szakok teljesítménye adott iskolában

A felhasználó hitelesítése, tehát mind a regisztráció, mind pedig a bejelentkezés többféleképpen történhet, amint azt a 4.7. ábrán is megfigyelhetjük. A felhasználó azonosíthatja magát e-mail cím és jelszó párosával, illetve lehetősége van az alábbi fiókok valamelyikével történő bejelentkezésre: Discord, Facebook, Google vagy Microsoft.



**4.7. ábra.** Felhasználó hitelesítése

A tételek megtekintését biztosító oldal a 4.8. ábrán figyelhető meg. Itt a felhasználónak lehetősége van megnézni egy kiválasztott év tételeit, tantárgyakra és ezen belül szakokra lebontva. minden téTEL az összes, a Romániai Tanügymenisztérium által biztosított nyelven elérhető. Ugyanitt látható, hogy a többi diák véleményével együtt a téTEL milyen értékelést kapott a tanulóktól, illetve az is, hogy hányan oldották már meg az adott feladatsort.

A gyakorlatsor kiválasztását követően megjelenik a 4.9. ábrán látható nézet, amely a téTEL jeleníti meg PDF formátumban. Az itt található eszközök lehetővé teszik a felhasználó számára a téTEL méretének megváltoztatását, a fájl letöltését vagy más egyszerű műveleteket. A gyakorlatok mellett elérhetők a hozzájuk tartozó javítókulcsok is.

A bejelentkezett felhasználóknak lehetőségeük van a téTEL minőségének értékelésére egy 1-től 5-ig terjedő skálán, amelyet a nézet jobb alsó részén levő csillagok reprezentálnak. Hasonlóan visszajelzés adható a feladatok nehézségére vonatkozóan, amely során 3 lehetőség közül lehet választani: könnyű, közepes vagy nehéz. A felhasználó számára alkalom nyílik a kedvenc fájlok elmentésére is.

Továbbá, biztosítunk egy, az érettségit szimuláló gyakorló környezetet is. A „Szimuláció indítása” gombra kattintva elkezdődik a háromórás időmérő, amelynek segítségével a felhasználó felmérheti az érettségire való felkészültségét. A szimuláció teljes ideje alatt bármikor elérhető a javítókulcs.

**2022 Tételek**

Matematika	Történelem	Román
<b>Matematika-Informatika szakok</b> ★★★★☆ 3 Elérhető nyelvek: 	<b>Humán szakok</b> ★★★★☆ 0 Elérhető nyelvek: 	<b>Reál szakok</b> ★★★★☆ 0 Elérhető nyelvek: 
<b>Pedagógiai szakok</b> ★★★★★ 5 Elérhető nyelvek: 	<b>Megoldotta: 0</b> <b>Könnyű</b>	<b>Megoldotta: 0</b> <b>Könnyű</b>
<b>Természettudományi szakok</b> ★★★★☆ 4 Elérhető nyelvek: 	<b>Megoldotta: 0</b> <b>Nehéz</b>	<b>Megoldotta: 0</b> <b>Könnyű</b>
<b>Technikai szakok</b> ★★★★★ 0 Elérhető nyelvek: 	<b>Megoldotta: 0</b> <b>Könnyű</b>	<b>Megoldotta: 0</b> <b>Könnyű</b>

**4.8. ábra.** Tételek listája

**2022 Tételek**

**Matematika-Informatika szakok - 2022**

**Examenul național de bacalaureat 2022**

**Proba E. c)**

**Matematică M\_mate-info**

**Varianta 1**

**Feladatok, proiecte și rezolvări matematică-informatică**

**Proiecte matematică, proiecte matematice, specializarea matematică-informatică**

**Toate subiectele sunt obligatorii. Se acordă trei puncte din oficiu.**

- Timpul de lucru efectiv este de trei ore.

**I. FELADATSOR** (30 pont)

**Sp** 1. Igazolja, hogy  $8 - 6\sqrt{6} + 6(\sqrt{6} - 1) = 2$ .

**Sp** 2. Adot az  $f: \mathbb{R} \rightarrow \mathbb{R}$ ,  $f(x) = 3x + m$  függvény, ahol  $m$  valós szám. Határozza meg azt az  $m$  valós számot, amelyre  $|f'(x)| = 0$ .

**Sp** 3. Oldja meg a valós számok halmazain a  $3 \cdot 2^{3x} + 4^x = 4$  egyenletet!

**Sp** 4. Számítsa ki annak a valószínűséget, hogy a körtegyű természetes számok halmazából véletlenszerűen kiválasztott számban a tizedik számjegye a 6 osztja legyen!

**Sp** 5. Az  $\alpha\vartheta$  derékszögű koordináta-rendszerben adott  $y = 3x - 2$  egyenletű  $d$  egyenes és az  $A(a, a)$  pont, ahol  $a$  valós szám. Határozza meg az  $a$  valós számot úgy, hogy a  $A$  pont rajta legyen a  $d$  egyenccsen!

**Sp** 6. Adot az  $ABC$  egyenlő szárú háromszög, amelyben  $AB = 10$  és  $\cos A = 0$ . Igazolja, hogy az  $ABC$  háromszög területe 50.

**II. FELADATSOR** (30 pont)

**Sp** 1. Adott az  $A(x) = \begin{pmatrix} 1 & -x & x^2 \\ 0 & 1 & -2x \\ 0 & 0 & 1 \end{pmatrix}$  mátrix, ahol  $x$  valós szám.

**Sp** a) Igazolja, hogy  $\det(A(1)) = 1$ .

**Sp** b) Igazolja, hogy  $A(x) \cdot A(y) = A(x+y)$ , bármely  $x$  és  $y$  valós szám esetén!

**Sp** c) Határozza meg azt az  $n$  természetes számot, amelyre  $A(n) \cdot A(n+1) \cdot A(n+2) \cdot A(n+3) = A(2n^2)$ .

**Sp** 2. Az  $M = [0, +\infty)$  halmazon értelmezett az  $x \times y = \frac{2x}{y+2} + \frac{2y}{x+2}$  művelet.

**Sp** a) Igazolja, hogy  $1 \times 0 = 1$ .

**Sp** b) Igazolja, hogy az  $x = 0$  a  $\times$ -ról művelet semleges eleme!

**Sp** c) Határozza meg az  $x$  nullától különböző,  $x \in M$  számot, amelyre  $x \times \frac{4}{x} = x$ .

**Szimuláció indítása** **Letöltes** **www.sapviz.ro**

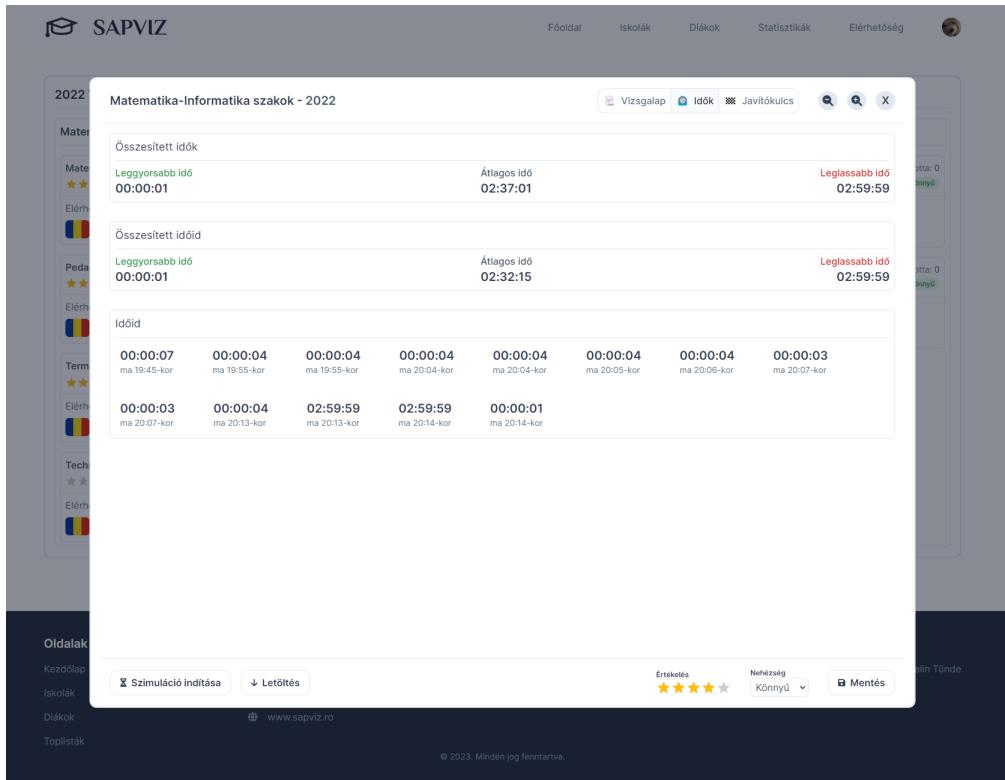
**Értékelés** ★★★★☆ **Nehézség** Könnyű **Mentés**

© 2023. minden jog fenntartva.

**4.9. ábra.** Tétel nézet részletes megtekintése

A szimuláció befejezetével a rendszer rögzíti a felhasználó időbeli eredményeit, amelyek a későbbiekben bármikor visszatekinthetőek, ahogy az a 4.10. ábrán is látható. Ugyanitt megtekinthető, hogy a többi diák hogyan teljesített, illetve a személyes mé-

réseink is három kategóriába sorolva: leggyorsabb idő, átlagosan eltöltött idő a téTEL megoldásával és leglassabb idő.



**4.10. ábra.** Időmérések téTEL megoldására

## 4.2. Alkalmazás megvalósítása

A rendszer fő webalkalmazása a React.js-t [Met23] magába foglaló Next.js [Ver23] keretrendszerrel került implementálásra. A fejlesztési folyamatban a TypeScript [Mic23] programozási nyelvet használtam, mivel ez segít a kód olvashatóságán és egy sokkal jobb figyelmeztetés és hibakezelést biztosít. A React jelenleg a legelterjedtebb SPA (Single Page Application) könyvtár a web fejlesztés világában, mivel lehetővé teszi a nagyon pontos irányítását a komponensek rendering folyamatának. A komponensek felelősök a saját állapotaik kezeléséért, ez azt jelenti, hogy meg lehet bennük határozni azt, hogy mi történjen, amikor bármelyik állapot értéke változik. Az alap funkcionálitás az, hogy minden érték változáskor újra rajzolódik a komponens, ami nem egy hatékony megoldás, emiatt hasznát vett a React Hook-oknak, ezen belül is a useState, useEffect és a useMemo-t használtam leggyakrabban. A useState segítségével hoztam létre egy állapotot, a useEffect-el írtam meg a logikát, ami reagál egy állapot változásaira, a useMemo-val pedig megadtam, hogy specifikus értékek változása ne befolyásolja a renderinget, csak az értékük változzon meg a DOM-ban de ne rajzolódjon újra a teljes komponens.

A honlap egy kliens oldalon renderelt alkalmazás, ami azt jelenti, hogy a szerver nem térti vissza a teljes tartalmát a honlapnak, hanem küld egy kis HTML kódot, azon kívül pedig JavaScript-et, amely felelős a honlap kitöltéséért. A Next.js a React.js

funkcionalitásán kívül lehetővé teszi a szerver oldali renderinget is, viszont ez nem volt szükséges a mi esetünkben. Az egyik extra funkcionálisat amit felhasználtam az a Next.js útvonal kezelése, ami nagyon egyszerűvé teszi egy érték kimentését változóba.

Az információk lekérdezéséhez a szervertől az axios és a react-query[[Lin23a](#)] könyvtárakat használtam fel. Az axios kibővíti a JavaScript-be beépített XHR-t, nagyon komplex hibakezelést biztosítva, a beérkező és kiküldendő adatok típusait automatikusan átalakítja, illetve képes könnyen kezelni a hitelesítéshez szükséges paramétereket is. A react-query lehetővé teszi a HTTP válaszokból kapott adatok eltárolását egy globális állapotba, illetve gyorsítótárazza az értékeket egy kulcs-érték formátumban. Ennek köszönhető az, hogy amikor több komponens ugyanazt az információt akarja lekérni az API-tól akkor csak egyszer kerül kiküldésre HTTP kérés a szervernek, a többi komponens pedig a queryClient állapotából kapja meg a válaszokat. A könyvtár úgy van megvalósítva, hogy segítségével egy zökkenőmentes élményt tudunk biztosítani. Ha a weboldalon módosul valamilyen adat akkor egy egyszerű függvényhívással érvénytelenítem egy adott kulcs alatt tárolt adatot, ezáltal a react-query újra lekéri a szervertől ezt és minden komponensben automatikusan frissül az adott érték.

Az alkalmazás hitelesítési logikáját a szerver oldalhoz hasonlóan a Clerk[[Cle23](#)] kezeli. Az egyik nagy előnye a könyvtárnak az, hogy előre megírt React komponenseket biztosít, amelyek alapjáraton modern design-al vannak ellátva, illetve garantálják a bejelentkezési és regisztrációs folyamat biztonságát. Az alkalmazásban a következő Clerk komponenseket használtam fel:

1. **SignedIn, SignedOut** - ezek burkoló komponensek, amelyek a tartalmukat a felhasználó bejelentkezési állapota függvényében rajzolják ki
2. **UserButton afterSignOutUrl="/"** - egy komponens, ami kimutatja a felhasználó profilképét ha ez be van jelentkezve, és amikor rákattint megjelenít egy menüt ahonnan ki tud jelentkezni, vagy módosíthatja az adatait
3. **SignIn, SignUp** - a komponensek, amelyek megjelenítik a vendégnek a bejelentkezési vagy a regisztrációs űrlapot

Az egyik nagyon fontos dolog az alkalmazás tervezésekor az volt, hogy a honlap legyen elérhető több nyelven is és legyen nagyon könnyű új fordítások hozzáadása a weboldalhoz. Ennek érdekében az i18next könyvtárat használtam, amely segítségével a kódban bárhol betölthető egy szöveg lefordított verziója a következő formátumban: t('kulcs'), a "t" egy metódus amit a useTranslation react hook biztosít. A függvény a kulcs alapján betölti a fordítást egy JSON fájlból, ugyanakkor leegyszerűsíti az új nyelvekkel való bővítését az oldalnak, mivel csak át kell másoljuk a JSON fájlt egy más ország kódjával elnevezett mappába és módosítjuk benne a fordításokat.

Az alkalmazás célja az, hogy egy felhasználóbarát felületet biztosítson, emiatt nagyon fontos volt egy minőségi design elkészítése. A TailwindCSS[[Lab22](#)] könyvtár segítségével került megvalósításra a honlap stílusa. Ez egy token alapú CSS könyvtár, ami a HTML elemek className-jéban levő értékei alapján kigenerál egy jól optimalizált és kicsinyített CSS fájlt. A felhasználói élmény másik jelentős része, hogy a nagy mennyiségi adatok közötti keresés egy könnyen rendezhető és szűrhető struktúrában történjen a gyors eredmény találás érdekében. Az adatokat kimutató táblázatok létrehozására a react-table[[Lin23b](#)]

könyvtárat használtam fel, mivel az állapotkezelési logikája nagyon fejlett, illetve lehetséges tőle teszi a tartalom oldalakra osztását, az oszlopok könnyű rendezését és ezek könnyű szűrését.

A honlap jelentős része vizualizációkból épül fel, ezért erre egy olyan könyvtárat használtam, ami képes gyorsan feldolgozni nagy mennyiségű adatokat és ugyanakkor jól átlátható kódot lehet írni vele. A Recharts könyvtár jelenleg az egyik leghíresebb vizualizációs csomag, mivel lehetővé teszi az ábrák implementálását react komponensekkel, illetve ezeknek a könnyű személyre szabását minimális kód felhasználásával. A Recharts a D3.js-re épül, ami a legkomplexebb JavaScript csomag ezen a téren, viszont az egyik nagy hátránya, amely miatt nem használtam csak ezt a könyvtárat az, hogy a kód kevésbé olvasható. A D3-at alap formában felhasználtam együtt a Three.js-el a 3D térkép ábrázolásához, aminek a szerepe a 3D környezet létrehozása és az SVG vizualizáció átalakítása 3 dimenziós objektumba, a térkép megrajzolásáért pedig a D3 könyvtár felelős.

A 4.11. ábrán egy olyan kódrészlet látható, amelyben a Recharts könyvtár felhasználásával készítettem egy reszponzív és személyre szabható vizualizációt ami az iskolán belüli szakok eredményeinek eloszlását mutatja be.

```
<ResponsiveContainer width="100%" height="100%">
  <RadarChart data={data}>
    <PolarGrid />
    <PolarAngleAxis dataKey="name" tick={<CustomizedAxisTick />} />
    <PolarRadiusAxis angle={50} domain={[0, 10]} />
    <Tooltip />

    {classes.map((entry, index) => (
      <Radar
        key={`bar-${entry}-${index}`}
        dataKey={`${entry}_grade`}
        fill={stringToHslColor(entry, 70, 60)}
        fillOpacity={0.6}
        name={t(`variables:specializations.${entry}`)}
      ></Radar>
    )))
    <Legend />
  </RadarChart>
</ResponsiveContainer>
```

**4.11. ábra.** Vizualizáció a Recharts könyvtár felhasználásával

A szerver oldalhoz hasonlóan nagyon fontos volt az, hogy egy tiszta és átlátható kódállomány legyen, ezért döntöttem úgy, hogy a webalkalmazásnál is felhasználom az ESLint[Fc23a] és a Prettier[Pre23] könyvtárakat.

## 5. fejezet

# Szerver

A szerver a NestJS[Mys23] keretrendszer segítségével valósult meg, mivel ez egy modern technológia, amely lehetővé teszi a Node.js-ben való létrehozását egy komplex és ugyanakkor jól optimalizált REST API-nak. A csomag a TypeScript programozási nyelvben van írva, ami a JavaScript szupersetje, ez azt jelenti, hogy számos funkcióval bővíti az alap programozási nyelvet, mint például a típusok használata, illetve egy sokkal minőségibb figyelmeztetés és hibakezelést biztosít. A NestJS keretrendszer nagyon megkönnyíti a fejlesztést azáltal, hogy számos funkcionalitása egy-egy decorator-ba van beépítve. Az útvonal választási folyamat annyiból áll, hogy minden controller osztály föle beírunk egy "@Controller('név')"-decorator-t ami megállapítja, hogy milyen cím alatt legyenek elérhetők az osztály függvényei. A végpontok logikáját implementáló metódusok előtt egy decorator-al meghatározzuk a HTTP műveletet és a nevet, ami alatt érhető el az adott végpont. A függvény fejlécében ugyancsak a decoratorok segítségével könnyen megadhatjuk azt, hogy a HTTP kérés mely részében (például: query-ben, params-ban vagy body-ban) található meg a számunkra szükség információ, pontosítjuk a nevet, amely alatt szerepel és hogy milyen típusba alakítsa a keretrendszer a megkapott értéket.

```
@Controller('users')
export class UsersController {
  constructor() {}

  @Get('hello/:name')
  public async getUserById(@Param('name') name: string): Promise<string> {
    return `Hello, ${name}`;
  }
}
```

5.1. ábra. Példa UserController

Az 5.1. ábrán levő kódrészletben látható, hogy beállítjuk a controller útvonalát "users"-ra, ez azt jelenti, hogy ha az API a "www.example.com" címen található, akkor az adott osztály elérhető lesz a "www.example.com/users/" útvonalon. A controlleren belül van egy függvényünk, ami egy string értéket vár. A @Get decorator jelzi, hogy ez kerüljön meghívásra amikor egy HTTP GET kérés a "www.example.com/users/hello/" címre érkezik. Ugyancsak a Get decoratorban megállapítottuk azt, hogy a címben a hello/ utáni rész legyen elmentve egy "name" elnevezésű paraméterben. A fejlécben a Param decorator segítségével pedig beállítjuk, hogy a változó értékét a "name" paraméterből vegye át a

keretrendszer. A függvény egy sima return művelettel visszatérít egy értéket, amit majd a háttérben a NestJS átalakít egy HTTP válaszba, amelynek a státusza 200(OK) és a body-ba beilleszti a visszatérített értéket. Hiba esetén a típusa alapján automatikusan egy HTTP választ küld a hiba üzenettel és egy adott státusz kóddal.

A NestJS egy nagyon fejlett gyorsítótárazó megoldást biztosít, ez egy CacheManager osztályból áll, amelyet a controllerekben használunk fel. A rendszer a háttérben kezeli helyettünk az adatok érvényességi idejét és ezeket a helyi memóriában, vagy egy Redis adatbázisban tárolja. A használata nagyon egyszerű, három függvényt biztosít, amellyel a gyorsítótárazást kezeljük:

1. **cacheManager.get(key: string)** - ez a függvény megnézi ha létezik-e valamilyen érték tárolva a CacheManager-ben egy adott kulcs alatt és ha igen akkor visszatéríti azt
2. **cacheManager.set(key: string, value: any, ttl: number)** - ezzel a metódussal egy értéket tárolhatunk el egy adott kulcs alatt a CacheManagerbe, illetve megadjuk hogy hány másodpercig legyen érvényes ez az eltárolt információ
3. **cacheManager.del(key: string)** - lehetővé teszi egy kulcs-érték párt törlését a CacheManager-ből

A gyorsítótárazás használata segítségével sikerült az API válasz idejét adott esetekben több mint 100x felgyorsítani, erre egy jó példa az előre kiszámított évi átlagokat és statisztikákat visszatérítő végpont. Eredetileg az átlag válasz idő 1 másodperc volt, viszont a CacheManager segítségével ezt sikerült csökkenteni 4-5 milliszekundumra.

A szerver oldalt a NestJS segítségével több modulból építettem fel, mindegyik magába foglal egy controller-t, service-t és a modelleket, amelyek egy specifikus erőforrással kapcsolatos kérések feldolgozásáért felelősek. Az API-t a következő modulokból állítottam össze:

1. **AppModule** - az API fő modulja, amely összefoglalja az összes többi modult, ezeket feldolgozza, majd pedig átadja a hátterében futó Express.js[Fc23b] HTTP szervernek
2. **StudentModule, SchoolModule, ExamModule és CountyModule** - a diákokkal, iskolákkal, vizsgatételekkel és megyékkel kapcsolatos adatok és statisztikák feldolgozásáért felelősek
3. **PrecalculatedModule** - az összetettebb számítások és statisztikák kiszámításáért és feldolgozásáért felelős
4. **UserModule** - a felhasználok által végezhető műveletek végrehajtásáért és a felhasználok adatinak visszatérítéséért felelős

Mivel az alkalmazás egy felhasználó funkcióval is rendelkezik, szükséges volt ezen kliensek adatait megvédeni és úgy implementálni az API-t, hogy csak az ügyfél tudja elérni a saját adatait. A Clerk hitelesítő rendszer felelős ezért. Ez egy különálló platform, amely garantálja a biztonságos hitelesítést és a kliens információk tárolását. A rendszer biztosít egy Node.JS SDK-t, amely segítségével könnyen implementálható az azonosítás

az API-ba, én egy middleware-t készítettem, amelyet a UserController előre helyezem. Ez azt jelenti, hogy minden HTTP kérés ami a UserController felé van irányítva először átmegy a middleware logikáján, ami a Clerk segítségével ellenőrzi ha a kérés header-jében érkező Authentication token helyes, ha igen akkor továbbítja a felhasználó adataival bővített HTTP kérést a controller-nek, máskülönben 401-es státusz kódval küld vissza hibaüzenetet.

A NestJS nem biztosít egy alapértelmezett megoldást a relációs adatbázissal való kommunikáláshoz, emiatt úgy döntöttem, hogy a PrismaORM könyvtár segítségével fogom megoldani ezt. Prisma-ban egy sémát kell definiálunk, amely modellekből épül fel és amiben egy sokkal olvashatóbb formátumban adhatjuk meg az adatbázis modelljét. Az alábbiakban látható egy tábla definíció a Prisma sémából:

```
model City {
    id      Int      @id @default(autoincrement())
    name   String
    nameHu String   @default("")
    schools School[]
    county  County   @relation(fields: [countyId], references: [id])
    countyId Int

    @@unique([name, countyId])
}
```

5.2. ábra. Példa Prisma séma

Amint látható az 5.2. ábrán, nagyon egyszerű a relációk, az alapértelmezett értékek és a kulcsok definiálása ebben a környezetben. Ezt követően egy npm parancs lefuttatásával kigeneráljuk a Prisma TypeScript könyvtárat, amely minden modellből TS típusokat hoz létre, illetve az összes táblának létrehoz egy objektumot, amely tartalmazza az SQL lekérdezések megfelelőit. Ezen függvények segítségével nagyon könnyen végre tudunk hajtani nagyon komplex lekérdezéseket is.

Az egyik fontos célom fejlesztés közben az volt, hogy egy olvasható kódot írjak, ennek érdekében a Prettier és az ESLint könyvtárakat használtam fel, amelyek beállításait egy-egy konfigurációs fájlban biztosítottam.

# 6. fejezet

## Adatfeldolgozó rendszer

### 6.1. Adatbegríjtő rendszer implementálása

Az adatletöltő rendszer egy szkript, amely a GoLang[Goo23] programozási nyelvben lett megvalósítva. A Geyizor könyvtár segítségével történik a weboldalakról való letöltése az adatoknak. A szkriptnek a felhasználó egy JSON formátumban levő konfigurációt kell biztosítson, illetve ugyancsak JSON formátumban kerülnek elmentésre a letöltés eredményei. A GoLang által biztosított Marshal és Unmarshal műveletek által történik a JSON fájlok átalakítása.

A GoLang egy Google által kifejlesztett nyílt forráskódú programozási nyelv, amely ismert a nagyon gyors kompilálási és futási idejéről, illetve az olvasható és tömör szintaxisáról. A kód jellege hasonlítható a Python[Fou23] programozási nyelvhez, viszont a teljesítménye sokkal jobb. A Sipviz előző változatában a letöltő Pythonban volt megvalósítva, amelyben egy megközelítőleg azonos adatmennyiség letöltése körülbelül 4 órát igényelt, mindenkorban a GoLang-ben implementált verzió ezt 1 óra alatt képes volt megvalósítani.

A weboldalak tartalmának a letöltése a Geyizor könyvtár segítségével lett implementálva. A könyvtár biztosít egy osztályt számunkra, amelyből létrehozunk egy új példányt úgy, hogy átadjuk neki a konfigurációkat, amellyel egy etikus letöltési folyamatot biztosítunk, illetve megadjuk a kezdeti weboldal URL címét és a metódust, amely majd feldolgozza ennek a tartalmát. Annak érdekében, hogy a letöltés etikusan történjen a következő konfigurációt adjuk át a letöltő példánynak:

#### 1. **RobotTxtDisabled**: true

Itt megmondjuk, ha ignorálja-e a letöltő a honlap által biztosított robots.txt fájlt, amiben megadják hogy mely botok töltethetnek le tőlük adatokat és melyek nem. Ez a beállítás rendesen a "false" értéket kellene kapja, mert szeretnénk ezt betartani, viszont az edu.ro nem biztosít egy robots.txt-t és akkor a letöltő egy "404 - not found" hibát térít vissza amikor ezt próbálja kiolvasni.

#### 2. **UserAgent**: "Mozilla/5.0(X11; Linuxx86\_64) AppleWebKit/537.36(KHTML, likeGecko)"

A "UserAgent" beállítás megmondja a honlapnak, ahonnan adatokat töltünk le azt, hogy kitől jön a HTTP kérés. Itt két megközelítés gyakori a webkaparás világában, az első az, hogy egy egyedi bot nevet adunk át itt, a Google például a "googlebot" nevet használja. Ez akkor hasznos amikor gyakran töltünk le adatokat az adott

oldalról és azt szeretnénk, hogy a honlap feltudja ismerni a mi letöltőnkét, a másik pedig az hogy egy böngésző adatait adjuk meg és akkor a honlap úgy érzékeli, hogy egy sima felhasználó lépett be a böngészőből az oldalra. Én a második megközelítést választottam, mivel évente csak egyszer töltök le adatokat amikor új eredmények jelennek meg.

3. **RequestDelay:** *time.Duration(configuration.RequestDelay)\*time.Millisecond*  
Itt megadjuk azt, hogy mennyi időt várjon a letöltő az oldalak közötti navigálás között. Én a felhasználó által biztosított konfigurációban levő időt adom át ezen opciónak. Az ideális várakozási idő a HTTP kérések között 5-10 másodperc.

4. **RequestDelayRandomize:** *configuration.RequestDelayRandomize*

Ezzel az opciónnal megadjuk, ha szeretnénk-e, hogy véletlenszerűen generálja-e a várakozási időt a HTTP kérések között. Ha ezt "true"-ra állítjuk akkor a fentebb megadott időt megszorozza a rendszer egy random számmal .5 és 1.5 között. Itt is a felhasználó által biztosított konfigurációt használjuk fel értéknek.

5. **RequestsPerSecond:** *configuration.RequestPerSecond*

Ez a beállítás limitálja azt, hogy maximum hány oldalra navigálhat a letöltő egy másodperc alatt, itt is ugyancsak a felhasználótól kapott értéket adjuk át.

6. **ConcurrentRequests:** *configuration.ConcurrentRequests*

Itt beállítjuk azt, hogy hány oldalt kérhet le a letöltő egyszerre, ez a döntés is ugyancsak a felhasználó kezében van.

Miután a Geyizor példányosítva van, belép a kezdeti URL címre és letölti az oldal HTML tartalmát, majd pedig ezt háttérben felbontja és létrehoz belőle egy "Response" típusú objektumot, ezt pedig átadja az általunk biztosított feldolgozó metódusnak. Az adott metódusban felhasználjuk a Response osztály függvényeit és azok segítségével szűrjük ki a számunkra fontos adatokat az oldal tartalmából. Az általam leggyakrabban használt függvénye a Response-nak a következő:

1. **r.HTMLDoc.Find(selector: string):** Ez a függvény az oldal tartalmában mekeres minden HTML elemet, amelyre érvényes a paraméterként megkapott szelektor és visszatérít egy "Selection" objektum tömböt, amelyben minden elem egy-egy találat.

```
func parseTable(g *geziyor.Geziyor, r *client.Response) {
    r.HTMLDoc.Find(selectors.TableSelector).Each(func(i int, s *goquery.Selection) {
        s.Find("script").Each(func(i int, el *goquery.Selection) {
            el.Remove()
        })
        parseFirstRows(s)
        parseSecondRows(s)
        students = append(students, currentPageStudents...)
    })
}
```

**6.1. ábra.** Táblázatok feldolgozása Geyizor-al

Amint az a 6.1. ábrán levő kódrészletben is látható, a Geyizor megtalálja a HTML oldalban a keresendő táblázatokat, átiterál rajtuk, majd feldolgozza ezek sorait és elmenti a kinyert adatokat egy Student nevű struktúrába.

A felhasználó által biztosított konfigurációban meg van adva minden számunkra releváns adat szelektora, és akkor ezeken átiterálók, majd pedig kiveszem a szöveg tartalmát a HTML elemnek a `.Text()` metódussal és azt átalakítom `float64`-ba vagy `boolean`-ba ha szükséges. Miután egy oldalról letöltöttem minden adatot megnézem ha van-e következő gomb oldal és ha igen, akkor megmondom a Geyizor-nak hogy kattintson rá és navigáljon a következő oldalra. Ha pedig nincs akkor átmegyek a következő megyének az oldalaira.

A Geyizor könyvtár egyik legfontosabb tulajdonsága az, hogy képes adatokat letölteni JavaScript-et igénylő oldalakról is. Egy weboldal lehet akár SSR-es (Server Side Rendering), ami azt jelenti, hogy az oldal teljes tartalma a szerver oldalon generálódik ki és a kliens a teljes oldalt kapja meg válaszként a HTTP kérésre. A másik lehetőség az a CSR (Client Side Rendering), ami abban különbözik az SSR-től, hogy a tartalomnak csak egy adott része van kigenerálva a szerver oldalon, és ezzel a tartalommal együtt a HTTP válaszban kapunk JavaScript kódot is, ami majd "hidratálja" az oldalt a többi tartalommal a kliens oldalon. A legtöbb webkaparó technológia nem képes a CSR-es oldalakat feldolgozni, mivel ezek csak letöltik a HTTP választ és nem képesek lefuttatni a JavaScript kódot, ami kitölti az oldalt. A Geyizor úgy tudja ezt megoldani, hogy a háttérében egy Chromium WebDriver-t (egy Chromium alapú böngészőt) futtat. Ezen belül tölti be az oldalt, majd pedig miután érzékeli, hogy nem módosult semmi ennek struktúrájában egy adott ideig, tehát ha hidratálva van az oldal akkor visszatéríti azt a HTML-t, ami a böngészőben van.

## 6.2. Adattisztító rendszer implementálása

Az adatokat tisztító, normalizáló és kibővítő rendszer a Python programozási nyelv segítségével került megvalósításra egy Jupyter Notebook keretén belül. Az adatfeldolgozásért felelős könyvtárak a Pandas és a NumPy, a két leghasználatabb könyvtár az adattudomány világában[JH12]. A rendszer használata egy JSON fájlt igényel, amely a letöltött érettségi eredményeket tartalmazza, illetve két kiegészítő információt biztosító CSV fájlt: egy lista az összes romániai iskolával, illetve egy lista az iskolák és a települések magyar elnevezéseivel.

A Jupyter Notebook használatát fontosnak tartottam, mivel ez megengedi, hogy az adat tisztítási folyamatot könnyen módosítható és átrendezhető cellákba építsem fel, amelyek egyenként futtathatóak, illetve lehetővé tette a folyamat látványos dokumentálását is a kódban. Ezáltal a tisztítás minden lépéseinél eredményei könnyen ellenőrizhetőek voltak.

Az adatok kezeléséért a Pandas könyvtár felelős, ez egy CSV fájlból betölti az érettségi eredményeket majd pedig átalakítja őket egy Pandas Adatkeretbe, amely segítségével nagyon gyorsan tudunk komplex műveleteket végezni az adathalmazon. Az adatkeret sorokba és oszlopokba rendeződik, illetve számos metódust biztosít, amelyek könnyítik a halmazban való iterálást, illetve az adatok változtatását. Az adathalmaz betöltését követően, elindul a tisztítási, normalizálási és kibővítési folyamat, amelyben a leggyakrabban végrehajtott műveletek a következők:

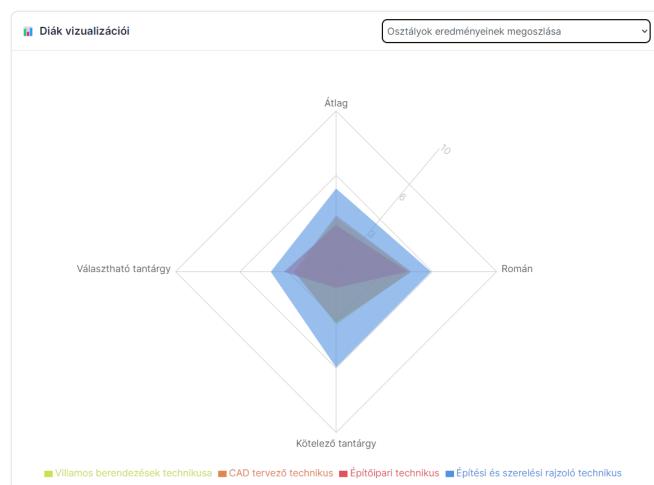
1. Egy adott oszlop tartalmából dekódolom az unicode karaktereket az unidecode könyvtárral. minden szóköz karaktert átalakítok egy standard szóköz karakterre, a "space"-re, majd pedig kitörlök minden szóközt a szöveg elejéről és végéről, illetve, ha valahol több mint egy szóköz van egymás mellett akkor azokat átalakítom egy szóközzé. Ezt követően pedig normalizálom a szöveget egy olyan formátumra, ahol minden szó első betűje nagybetű, azon kívül pedig minden betű kisbetű.
2. Egy adott számokat tartalmazó oszlop értékeit két decimális pontosságra kerekítem, ha a cellában van egy szám, ami nagyobb, mint nulla, ha a szám kisebb nullánál akkor az értéket nullára állítottam, mivel nem lehet negatív szám az adathalmazban, ha pedig a cella üres ugyancsak nullára lesz az értéke.
3. Egy oszloban a hiányos adatok helyettesítése egy alapértelmezett értékkel, vagy az oszlop adat típusának megfelelő "üres" értékkel, például számok esetén nulla, stringek esetén egy üres string.

Az adattisztító működésének bemutatására szolgáló példának vehetjük a 6.2. ábrán található programrészletet. Ennek szerepe, hogy a számítógépkezelés tantárgy eredményeit tartalmazó oszloból kitöröljük a felesleges információkat jelentő szavakat, ezt normalizáljuk, majd az üres vagy hibás adatot tartalmazó cellákat egy standard értékre állítjuk, amely a mi esetünkben 'Neprezentat'.

```
df['tic'] = df['tic'].str.replace('Utilizator ', '').str.replace('Nivel ', '').str.strip()
df['tic'].fillna("Neprezentat", inplace=True)
df['tic'] = df['tic'].apply(lambda x: "Neprezentat" if len(x) < 1 else x)
```

**6.2. ábra.** Adattisztító működését bemutató kódrészlet

A fenti kódsorok által eredményezett grafika a 6.3. ábrán tekinthető meg.



**6.3. ábra.** Recharts vizualizáció példa

Az adatok kiegészítése úgy zajlik, hogy a felhasználó által megadott információkat ugyancsak tisztítom és normalizálom, majd pedig egy közös oszlop alapján egybevonom az érettségi eredményes és az kibővítő adatkeretet.

### 6.3. Adatátalakító rendszer implementálása

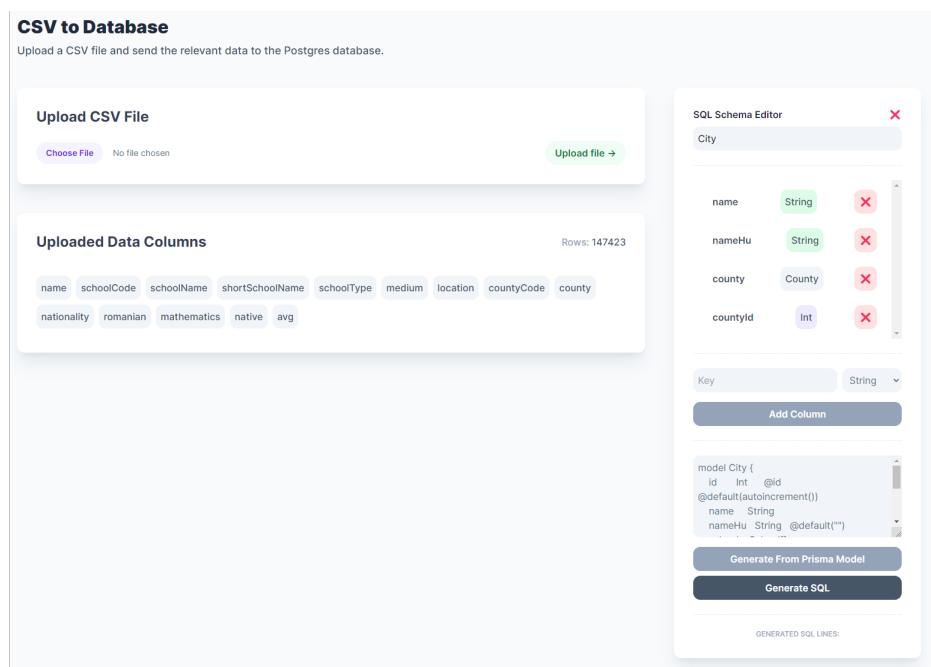
A Sapviz egyik kulcs fontosságú része az adatátalakító rendszer, amely keretén belül a felhasználó feltölthet egy CSV fájlt és egy adatbázis sémát, azt követően pedig átalakíthatja az adatokat SQL formátumba. Ezt egy web alkalmazásként valósítottam meg felhasználva a Next.js, PrismaORM és a Danfo.js technológiákat.

A weboldalt Next.js-ben fejlesztettem, mivel a keretrendszer megkönnyíti egy kliens és egy szerver oldal létrehozását ugyanabban a kódmezőben.

A felhasználó fel kell töltön az oldalra egy CSV fájlt, ami az átalakítandó információkat tartalmazza. A feltöltést követően a Danfo.js könyvtárral feldolgozom az adatokat. Ez a Pandas csomagnak a JavaScript megfelelője, ezáltal a CSV-ből könnyen létrehozhatunk egy adatkeretet, amely egy jól optimalizált feldolgozást tesz lehetővé.

Az adatkeret létrehozása után a honlap kimutatja ennek az összes oszlopát. A felhasználó akár manuálisan, akár egy Prisma séma feltöltésével meghatározza a SQL táblának a struktúráját. A rendszer a háttérben találtatja az oszlopokat az adatkeretből a SQL-ban levő információkkal és átalakítja ezek típusát, ahol ez szükséges. Az alkalmazás képes kezeln a relációkat is a táblából, ez úgy történik, hogy a megadott CSV oszlop alapján megkeresi az idegen kulcs értékét. A kulcs megszerzésért a honlap küld egy HTTP kérést a szervernek, ami ugyancsak Next.js-ben van implementálva Node.js segítségével. Az API a PrismaORM-el csatlakoztatva van egy adatbázishoz, és ennek segítségével betölti az idegen kulcs értékeit, majd pedig visszatérít egy HTTP választ ezekkel. A webalkalmazás a megkapott kulcsokat az adatkerethez illeszti és ennek alapján kigenerál egy SQL fájlt, amely INSERT parancsokat tartalmaz.

A felhasználói felület, amelyen a fentiekben felsorolt műveleteket el lehet végezni, a 6.4. ábrán látható.



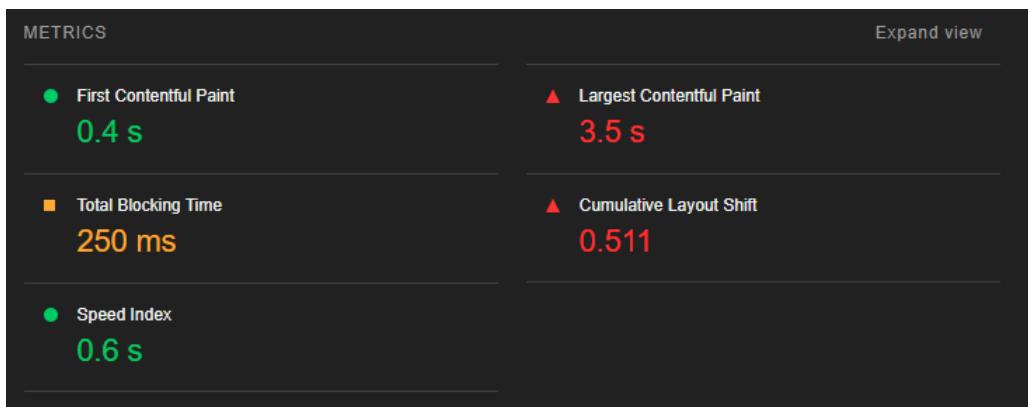
6.4. ábra. Adatátalakító rendszer grafikus felülete

Az eredeti megoldással, ahol minden sornak egy külön beszúrás volt létrehozva az eredmény fájl több mint 100Mb volt egy 120,000 elemet tartalmazó adathalmaz átalakítása után. Ezt a megközelítést optimalizáltam tömeges beszúrások segítségével, ahol minden INSERT művelet ezer sor információt tartalmaz, így a generált fájlok méretei körülbelül 20Mb-ra csökkentek.

## 7. fejezet

# Alkalmazás összehasonlítása előző verziójával

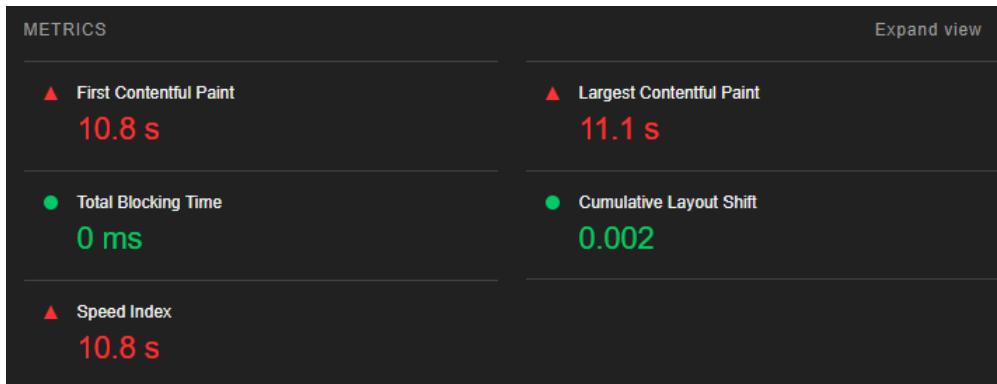
A projekt az eredeti Sapviz ötletből született, amely a nyolcadikos képességvizsga eredményeit vizualizálta és hasonlította össze. A célom az volt, hogy az érettségiző diákoknak is biztosítsak egy hasonló platformot, ami egy jobb felhasználói élményt, helyesebb adatokat és látványos vizualizációkat állít rendelkezésükre. Az eredeti honlap Angular technológiával került megvalósításra, ami egy SPA keretrendszer a Next.js-hez hasonlóan, viszont hiányosságai voltak a teljesítmény és a kód átláthatóság oldalán.



7.1. ábra. Az új webalkalmazás betöltési ideje

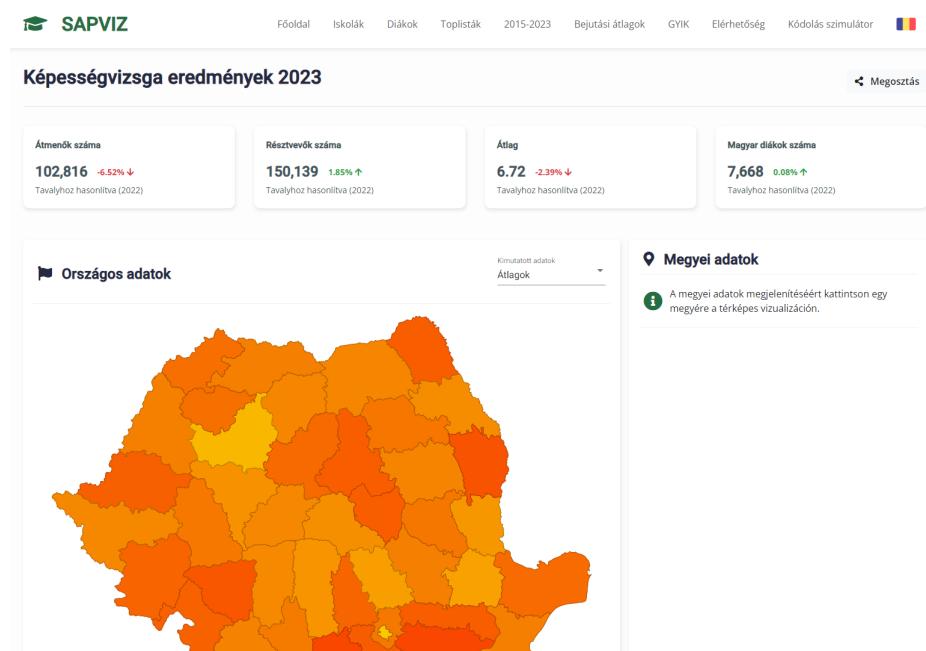
A régi weboldal az összes eredményt JSON fájlok segítségével töltötte be, majd pedig a kliens oldalon számítja ki a statisztikákat és az összehasonlításokat ezek alapján. Az új verzió az adatokat egy adatbázisból tölti be egy REST API-n keresztül és csak a felhasználó számára releváns előre kiszámított statisztikákat tériti vissza. Ezáltal sikerült jelentősen optimalizálni az információ betöltési folyamatot, átlagban kétszer gyorsabban töltődnek be a honlap oldalai a nyolcadikos weblaphoz képest, amint látható a 7.1. és a 7.2. ábrákban is.

A másik fontos cél az volt, hogy egy modern és egyszerű felhasználói felületet biztosítsak, ami a hangsúlyt az adatokra fekteti és amelynek kezelhetősége egyszerű. Ez látható a diákok és az iskolák listáját ábrázoló oldalakon is, ahol eddig a felhasználók csak akkor látták a fontos adatokat miután egy keresést végeztek és kiválasztottak egy



**7.2. ábra.** Képességvizsgát vizualizáló honlap betöltési ideje

eredményt. Az új alkalmazásban pedig a fontos adatok egyből megjelennek egy jól átlátható és navigálható táblázatban. A weblap modernebb kinézete egy fontos részlet, mivel az átlagos felhasználó pár másodperc alatt eldönti marad-e tovább vagy kilép róla, így egy minőségi design növeli a felhasználó megőrzési időt. Amint látható a 7.3. ábrán, a képességvizsga eredményeket vizualizáló honlap egy relatív régi design stílusra épült.



**7.3. ábra.** Középiskolai képességvizsga főoldala

Az adatfeldolgozási oldalon is nagy hiányosságai voltak a régi rendszernek, mivel az adatletöltés sima Python-ban volt implementálva, oly módon, hogy manuálisan kellett indítani minden megye eredményeinek a letöltését. Tisztítás előtt pedig ezeket össze kellett csatolni egy fájlba külön. A manuális lépések és a Python könyvtár optimalizálása miatt átlagban egy év adatainak a letöltése körülbelül 4 órát igényelt. Mindeközben a GoLang-ben készített adatbegyűjtő automatikusan átiterál minden megyén és az eredményeket egyben menti el egy JSON fájlba, mindezt körülbelül 1 óra alatt.

Az adattisztítási folyamat is nagyon korlátozott volt a nyolcadikos eredményeknél, sok esetben manuális normalizálást igényelt, vagy ha emberi hiba miatt egy cellában nem megfelelő információ volt akkor a folyamat kiakadt és ezt újra kellett indítani a manuális javítást követően. A Pandas-al implementált tisztítót úgy terveztem első lépéstől, hogy emberi hibától függetlenül minden fusson le és normalizálja az adatokat, ha pedig valahol nem megfelelő cellát talál azt egy standardizált értékre állítsa.

A végső adathalmaz alakja egy CSV fájl volt a régi rendszerekben, ami azt jelentette, hogy az adatokat csak Google Sheets vagy Excel segítségével tudtuk ellenőrizni, ha nem szerettünk volna pluszba kódot írni, ami feldolgozza a CSV fájlokat. Számos ismétlődő információ volt az eredmények között, emiatt csökkentve ezek átláthatóságát. A mostani rendszer utolsó lépését követően az adatok SQL formátumban vannak elmentve úgy, hogy a relációkat felhasználva csökkenti az információ ismétlődést minimálisra. Ezeket egy PostGreSQL adatbázisba lehet feltölteni, ennek köszönhetően nagyon egyszerű az eredmények helyességének ellenőrzése akár SQL lekérdezésekkel, akár egy asztali alkalmazással, mint a Microsoft PowerBI.

## 8. fejezet

# Továbbfejlesztési lehetőségek

Az alkalmazás továbbfejlesztése során számos lehetőség nyílik a funkcionalitások ki-bővítésére és egyben a felhasználói élmény növelésére. A továbbiakban felsorolok néhány ötletet, amely ezen lehetőségek listáját bővíti:

- Osztályok hozzáadása:** ez funkció lehetővé tenné a tanárok számára, hogy virtuális osztálytermeket hozzanak létre a diákokkal, ezáltal követni tudnák a tanulók elért eredményeit, illetve a megoldásokhoz szükséges időt a tételek szimulációja során.
- Vizualizáció generátor:** dinamikus vizualizáció készítő rendszer, amely megengedi a felhasználóknak, hogy a mi adataink segítségével különböző vizualizációkat építsenek pár kattintásból, amelyek a mi honlapunkon fognak elni és ők megosztják ezeket egy beépített formában vagy képként.
- Tételek kijavítása:** olyan rendszer implementálása, amelyben a felhasználónak lehetőségeiben áll a felületen megoldani a tételeket vagy ezek megoldását manuálisan feltölteni. A feltöltést követően a háttérben ezeket tanárok ellenőriznek és értékelnék, majd visszajelzést küldenének a diáknak, amely tartalmazza az elért minősítést.
- Online tanórák:** különböző önkéntesen jelentkező tanárok közreműködésével, olyan felület biztosítása, ahova videóanyagokat töltenénk fel a tantárgyakból vagy online korrepetálásokat, tanórákat ütemezhetünk, ahova a diákok becsatlakozhatnak.
- Magánóra rendszer:** magántanárokat és diákokat kötnénk össze. Ez azt jelenti, hogy lenne egy megyére vagy városra bontott lista magántanárokkal minden tantárgyból, a tanárok beállíthatnak maguknak egy órarendet, hogy mikor vannak szabad időpontjaik és hogy mennyibe kerül az adott tanóra. A diákok a tanáronak adhatnak értékeléseket és úgy választhatnak tanárt, hogy láthatják más diákok véleményét, illetve azt, ha az adott tanár programja megfelel a diákok programjával.
- Tétel megoldás mesterséges intelligenciával:** olyan mesterséges intelligencia, amely segít a diákoknak a téTEL megoldásában, úgy, hogy felel azok kérdéseire. Ez sokat növelné a felhasználói élmény fokozásában, hiszen interaktívabb, gyorsabb és könnyelmesebb, mintha csak a javítókulcs állna rendelkezésükre.

# Összefoglaló

A projekt célja egy olyan webalkalmazás létrehozása volt, amely összegyűjti, feldolgozza és megjeleníti a romániai diákok érettségi eredményeit. A Sapviz azért jött létre, hogy lehetőséget biztosítsunk minden diáknak, szülőnek és oktatónak egyaránt, hogy az egyéni és kollektív teljesítményeket értékelhessék és azonosíthassák a még fejlődést igénylő területeket.

Az oldal sikere után, amely a középiskolai képességvizsga eredményeit dolgozza fel, szükségét éreztük egy olyan platformnak is, amely az érettségi köré épül, hiszen diákok életünknek ez a másik nagy mérföldköve.

Igyekeztünk kiemelt figyelmet fordítani arra, hogy egy felhasználóbarát, letisztult és könnyen kezelhető felületet alakítsunk ki, annak érdekében, hogy egy pozitív felhasználói élményt nyújtsunk az oldal látogatóinak. Szerettük volna, ha az alkalmazásunk átláthatóbbá teszi az érettségin nyújtott teljesítményeket, miközben fontos összefüggésekre hívjuk fel a figyelmet.

A platform átfogó információkat nyújt az érettségi vizsgákra vonatkozóan mindenki számára, ezáltal a diákok és a tanárok egyaránt könnyen hozzáférhetnek majd a vizsgák részleteihez. Alkalmat kínálunk a diákok teljesítményeinek elemzésére és értékelésére, az érettségi eredményeiket összehasonlíthatják más diákokéval, így láthatják, hol is helyezkednek el különböző rangsorokban.

Az oldalon szereplő vizualizációk és statisztikák segítségével könnyen láthatóvá válnak az eredményeket befolyásoló tényezők, továbbá megfigyelhetőek az időbeli változások.

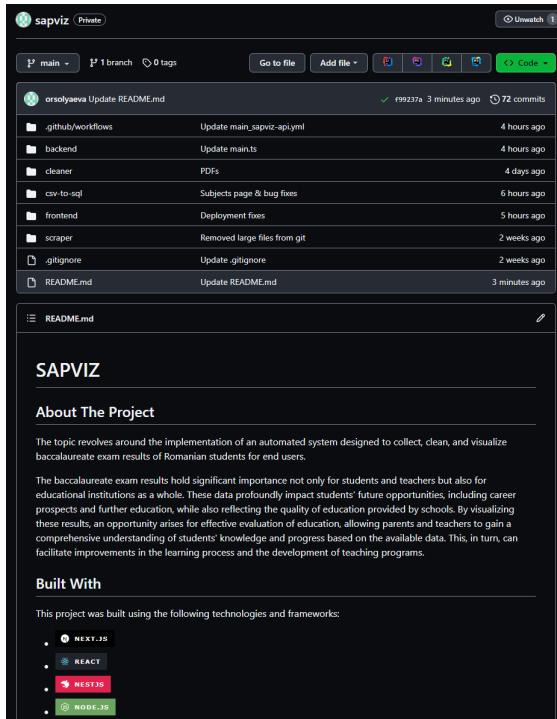
Bízunk benne, hogy a projektünk segíti majd a diákokat az érettségi vizsgára való hatékonyabb felkészülésben, valamint a teljesítményeik és eredményeik pontosabb nyomon követésében.

Teret nyitunk felhasználói visszajelzések küldésére, amelyeket figyelembe veszünk majd a jövőbeli verziók során, elősegítve ezáltal az interaktív közösségepítést és a tapasztalatok megosztását.

Természetesen a projekt fejlesztése tovább folytatódik, új vizualizációk és számos funkcionális implementálásával várjuk majd a minket látogatókat.

## 8.1. Verziókövetés

Az alkalmazás fejlesztése alatt verziókövetést használtam, amelyet a Github segítségével valósítottam meg. A 8.1. ábrán látható a repository felépítése.



**8.1. ábra.** Projekt struktúrája Githubon

A 8.2. ábrán megfigyelhető egy statisztika a repository commitjairól.



**8.2. ábra.** Commit előzmények Githubon

A repository nem publikus, mivel olyan rendszereket és érzékeny adatokat tartalmaz, amelyeket nem akarunk elérhetővé tenni hasonló platformok számára. Előző évek vissza-jelzései alapján, több konkurencia is az általunk beszerzett és tisztított adatokat szeretné felhasználni. Jelentős erőforrásokat fektettünk a saját algoritmusok és rendszerek fejlesztésébe, amik versenyelőnyt biztosítanak számunkra. A kód nyilvánosságának korlátozása segít megvédeni ezeket és megakadályozni az engedély nélküli használatot vagy másolást.

# Ábrák jegyzéke

2.1. Adatbegyűjtő rendszer használati folyamatábrája . . . . .	14
2.2. Adattisztító rendszer használati folyamatábrája . . . . .	15
2.3. Adatátalakító rendszer használati folyamatábrája . . . . .	16
2.4. Fő webalkalmazás használati folyamatábrája . . . . .	17
3.1. Rendszer architektúrája . . . . .	20
3.2. Adatbázis felépítése . . . . .	22
4.1. Webalkalmazás főoldala . . . . .	23
4.2. Megyei eredmények megtekintése a főoldalon . . . . .	24
4.3. Diákok listája . . . . .	24
4.4. Diák egyedi oldala . . . . .	25
4.5. Iskola egyedi oldala . . . . .	26
4.6. Szakok teljesítménye adott iskolában . . . . .	26
4.7. Felhasználó hitelesítése . . . . .	27
4.8. Tételek listája . . . . .	28
4.9. Tétel nézet részletes megtekintése . . . . .	28
4.10. Időmérések tétel megoldására . . . . .	29
4.11. Vizualizáció a Recharts könyvtár felhasználásával . . . . .	31
5.1. Példa UserController . . . . .	32
5.2. Példa Prisma séma . . . . .	34
6.1. Táblázatok feldolgozása Geyizor-al . . . . .	36
6.2. Adattisztító működését bemutató kódrészlet . . . . .	38
6.3. Recharts vizualizáció példa . . . . .	38
6.4. Adatátalakító rendszer grafikus felülete . . . . .	39
7.1. Az új webalkalmazás betöltési ideje . . . . .	41
7.2. Képességvizsgát vizualizáló honlap betöltési ideje . . . . .	42
7.3. Középiskolai képességvizsga főoldala . . . . .	42
8.1. Projekt struktúrája Githubon . . . . .	46
8.2. Commit előzmények Githubon . . . . .	46

# Táblázatok jegyzéke

2.1.	Érettségi eredmények letöltése használati eset	14
2.2.	Érettségi adatok tisztítása használati eset	15
2.3.	Adatok átalakítása SQL-ba használati eset	16
2.4.	Felhasználó hitelesítése használati eset	18
2.5.	Adatok keresése és megtekintése használati eset	18
2.6.	Adatok megosztása használati eset	18
2.7.	Tételek kezelése használati eset	19

# Irodalomjegyzék

- [Cle23] Clerk. Welcome to clerk docs | clerk. <https://clerk.com/docs>, 2023. [Online; visited at 30-June-2023].
- [Dat23] Prisma Data. Prisma documentation. <https://www.prisma.io/docs/>, 2023. [Online; visited at 30-June-2023].
- [Fc23a] OpenJS Foundation and ESLint contributors. Documentation - eslint - pluggable javascript linter. <https://eslint.org/docs/latest/>, 2023. [Online; visited at 30-June-2023].
- [Fc23b] OpenJS Foundation and Express.js contributors. Express routing. <https://expressjs.com/en/guide/routing.html>, 2023. [Online; visited at 22-May-2023].
- [Fc23c] OpenJS Foundation and Node.js contributors. Documentation | node.js. <https://nodejs.org/en/docs/>, 2023. [Online; visited at 30-June-2023].
- [Fou23] Python Software Foundation. Our documentation | python.org. <https://www.python.org/doc/>, 2023. [Online; visited at 30-June-2023].
- [Goo23] Google. Documentation - the go programming language. <https://go.dev/doc/>, 2023. [Online; visited at 30-June-2023].
- [Gul23] Musab Gultekin. geziyor package - github.com/geziyor/geziyor - go packages. <https://pkg.go.dev/github.com/geziyor/geziyor#section-documentation>, 2023. [Online; visited at 30-June-2023].
- [JH12] Jian Pei Jiawei Han, Micheline Kamber. *Data Mining. Concepts and Techniques, 3rd Edition*. Morgan Kaufmann, 3rd edition, 2012.
- [Lab22] Tailwind Labs. Tailwind documentation. <https://tailwindcss.com/docs/>, 2022. [Online; visited at 30-June-2023].
- [Lin23a] Tanner Linsley. Tanstack query | react query, solid query, svelte query, vue query. <https://tanstack.com/query/latest/>, 2023. [Online; visited at 30-June-2023].
- [Lin23b] Tanner Linsley. Tanstack table | react table, solid query, svelte query, vue query. <https://tanstack.com/table/v8>, 2023. [Online; visited at 30-June-2023].
- [Met23] Meta. React. <https://react.dev/>, 2023. [Online; visited at 26-June-2023].

- [Mic23] Microsoft. Typescript: The starting point of learning typescript. <https://www.typescriptlang.org/docs/>, 2023. [Online; visited at 30-June-2023].
- [Mys23] Kamil Mysliwiec. Nestjs documentation. <https://docs.nestjs.com/>, 2023. [Online; accessed 30-June-2023].
- [Num23a] Inc. NumFOCUS. Pandas documentation. <https://pandas.pydata.org/docs/>, 2023. [Online; visited at 30-June-2023].
- [Num23b] NumPy. Numpy documentation. <https://numpy.org/doc/stable/>, 2023. [Online; visited at 30-June-2023].
- [Pre23] Prettier. What is prettier? - prettier. <https://prettier.io/docs/en/index.html>, 2023. [Online; accessed 28-June-2023].
- [Ver23] Vercel. Getting started | next.js. <https://nextjs.org/docs>, 2023. [Online; visited at 30-June-2023].
- [Wik23] Wikipedia contributors. Representational state transfer — Wikipedia, the free encyclopedia. [https://en.wikipedia.org/w/index.php?title=Representational\\_state\\_transfer&oldid=1095218299](https://en.wikipedia.org/w/index.php?title=Representational_state_transfer&oldid=1095218299), 2023. [Online; visited at 30-June-2023].