

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



**SAPIENTIA  
ERDÉLYI MAGYAR  
TUDOMÁNYEGYETEM**

F1 Ticket Manager - Online jegyek kezelése

**DIPLOMADOLGOZAT**

Témavezető:

Dr. Márton Gyöngyvér,  
Egyetemi adjunktus  
Györfi Ágnes,  
Egyetemi tanársegéd

Végzős hallgató:

Kovács Bence

**2023**

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



**UNIVERSITATEA  
SAPIENTIA**

F1 Ticket Manager - Gestiunea tichetelor

**LUCRARE DE DIPLOMĂ**

Coordonator științific:  
Dr. Márton Gyöngyvér,  
Lector universitar sau Șef de lucrări  
Györfi Ágnes,  
Asistent universitar

Absolvent:  
Kovács Bence

**2023**

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



**SAPIENTIA**  
HUNGARIAN UNIVERSITY  
OF TRANSYLVANIA

F1 Ticket Manager - Online ticket management

**BACHELOR THESIS**

Scientific advisor: Student:  
Dr. Márton Gyöngyvér, Kovács Bence  
Lecturer  
Györfi Ágnes,  
Assistant professor

**2023**

## **Declarație**

Subsemnata/ul ..... , absolvant(ă) al/a specializării ..... , promoția..... 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,

Data:

Absolvant

Semnătura.....

# Kivonat

Napjainkban az online jegyvásárlás rohamos elterjedése figyelhető meg a technológia fejlődésével és az internet elérhetőségének növekedésével. Egy online platformon keresztül az emberek ma már kényelmesen és gyorsan tudnak jegyeket vásárolni különféle eseményekre, mint például koncertekre, színházi előadásokra vagy sporteseményekre, mint például a Formula-1.

Az online jegyvásárlás számos előnnyel jár. A vásárlók egyszerűen és kényelmesen böngészhetnek és választhatnak a széles körű jegyválaszték közül. Az online platformok részletes információkat nyújtanak az eseményekről, beleértve a dátumokat, helyszíneket és leírásokat. Emellett az online jegyvásárlás lehetővé teszi a jegyek összehasonlítását, árak és típusok kiválasztását, ami segít a vásárlóknak a legjobb lehetőség megtalálásában.

A technológiai fejlesztések, mint például a biztonságos online fizetési rendszerek és az elektronikus jegyek, hozzájárultak az online jegyvásárlás elterjedéséhez. A vásárlók könnyedén és biztonságosan fizethetnek az online platformokon (webshop) keresztül, és elektronikus jegyet kapnak, amelyet mobil eszközükön vagy nyomtatható formában mutathatnak fel az eseményen. Az online jegyvásárlás megkönnyíti az eseményekre való részvételt, hiszen a vásárlóknak nem kell hosszú sorokban állniuk a jegypénztárnál.

# Rezumat

În prezent, se observă o răspândire rapidă a achiziționării de bilete online, odată cu dezvoltarea tehnologică și creșterea accesului la internet. Prin intermediul unei platforme online, oamenii pot cumpăra bilete confortabil și rapid pentru diverse evenimente, cum ar fi concerte, spectacole de teatru sau evenimente sportive, precum Formula 1.

Achiziționarea de bilete online vine cu numeroase avantaje. Cumpărătorii pot naviga și alege cu ușurință dintr-o gamă largă de opțiuni de bilete. Platformele online oferă informații detaliate despre evenimente, inclusiv date, locații și descrieri. De asemenea, achiziționarea de bilete online permite compararea prețurilor și selecționarea diferitelor tipuri de bilete, ceea ce ajută cumpărătorii să găsească cea mai bună opțiune.

Dezvoltările tehnologice, cum ar fi sistemele de plată online sigure și biletele electronice, au contribuit la răspândirea achiziționării de bilete online. Cumpărătorii pot plăti ușor și în siguranță prin intermediul platformelor online (webshop) și primesc biletele electronice, pe care le pot prezenta pe dispozitivele lor mobile sau sub formă printată la eveniment. Achiziționarea de bilete online facilitează participarea la evenimente, deoarece cumpărătorii nu mai trebuie să stea în rânduri lungi la casele de bilete.

# Abstract

Currently, the rapid spread of online ticket purchasing can be observed due to technological advancements and the increased accessibility of the internet. Through an online platform, people can now conveniently and quickly purchase tickets for various events such as concerts, theater performances, or sports events like Formula 1.

Online ticket purchasing comes with numerous advantages. Customers can easily browse and choose from a wide range of ticket options. Online platforms provide detailed information about events, including dates, venues, and descriptions. Additionally, online ticket purchasing allows for ticket comparisons, price and type selections, which help customers find the best options available.

Technological developments, such as secure online payment systems and electronic tickets, have contributed to the widespread adoption of online ticket purchasing. Customers can easily and safely make payments through online platforms (webshop) and receive electronic tickets that can be presented on their mobile devices or in printable form at the event. Online ticket purchasing simplifies event attendance, as customers no longer have to wait in long queues at ticket counters.

# Tartalomjegyzék

<b>1. Bevezető</b>	<b>11</b>
1.1. Témaválasztás indoklása . . . . .	11
<b>2. Elméleti megalapozás és szakirodalmi áttekintő</b>	<b>14</b>
2.1. Webes alkalmazás felépítése . . . . .	14
2.2. A React JavaScript keretrendszer . . . . .	15
2.3. A Google Firebase platform . . . . .	16
2.4. Az AES titkosítás . . . . .	17
2.4.1. Bevezető . . . . .	17
2.4.2. Az algoritmus elmélete . . . . .	18
2.4.3. Az algoritmus gyakorlatban . . . . .	18
2.4.4. Alkalmazások és érdekességek . . . . .	21
2.5. RSA (TBD) . . . . .	21
2.6. Kutatási kérdések . . . . .	21
2.7. Célkitűzések . . . . .	22
<b>3. Rendszerspecifikáció</b>	<b>23</b>
3.1. Rendszer követelmények . . . . .	23
3.1.1. Funkcionális követelmények . . . . .	23
3.1.2. Nem funkcionális követelmények . . . . .	27
3.2. Felhasználói követelmények . . . . .	27
<b>4. A rendszer architektúrája</b>	<b>30</b>
4.1. GitHub frontend architektúra . . . . .	31
4.2. Google Firebase backend architektúra . . . . .	32
4.2.1. Firebase Authentication . . . . .	33
4.2.2. Cloud Firestore . . . . .	33
4.2.3. Firebase Storage . . . . .	34
4.3. Vercel backend architektúra . . . . .	35
<b>5. Tervezés és megvalósítás</b>	<b>37</b>
5.1. Az alkalmazás frontendje . . . . .	37
5.1.1. Kezdőoldal (Homepage) . . . . .	37
5.1.2. Versenynaptár (Schedule) . . . . .	39
5.1.3. Profil (Profile) . . . . .	40
5.2. Saját szoftverek esetén . . . . .	44
5.3. Összességében . . . . .	44

<b>6. Szoftver</b>	<b>46</b>
6.1. A szoftver bemutatása . . . . .	48
6.2. A szoftver megírásához használt könyvtárak . . . . .	48
6.3. Diagramok . . . . .	50
6.3.1. Use Case diagram . . . . .	50
6.3.2. Osztálydiagram . . . . .	50
6.3.3. Szekvencia diagram . . . . .	52
<b>Összefoglaló</b>	<b>54</b>
<b>Köszönetnyilvánítás</b>	<b>55</b>
<b>Jelölések</b>	<b>56</b>
<b>Ábrák jegyzéke</b>	<b>57</b>
<b>Táblázatok jegyzéke</b>	<b>58</b>
<b>Irodalomjegyzék</b>	<b>59</b>
<b>Függelék</b>	<b>60</b>
F.1. A TeXstudio felülete . . . . .	60
F.2. Válasz az „Élet, a világmindenség, meg minden” kérdésére . . . . .	61

# **1. fejezet**

## **Bevezető**

### **1.1. Témaválasztás indoklása**

Napjainkban az online jegyvásárlás nagy előretörést ért el a technológia fejlődésével az egyre szélesebb körben történő bankkártyás internetes vásárlások következtében. Egy online platformon keresztül az emberek ma már kényelmesen és gyorsan tudnak jegyeket vásárolni különféle eseményekre tekintettel arra, hogy percek alatt el tudjuk végezni a világ bármely pontjából a nap bármely időpontjában. Az online jegyvásárlás számos előnyivel jár, amelyeknek köszönhetően egyre népszerűbbé válik.

Az egyik legfontosabb előny az online jegyvásárlás esetén az, hogy a vásárlók egyszerűen és kényelmesen böngészhetnek és választhatnak a széles körű jegyválaszték közül. Az online platformok részletes információkat biztosítanak az eseménykről, így a potenciális vásárlók teljes körű tájékoztatást kapnak az eseményről, mielőtt eldöntenék, hogy vásárolnak-e jegyet. Tehát a vásárlók magabiztosan dönthetnek arról, hogy melyik eseményre szeretnének jegyet vásárolni, anélkül, hogy bármilyen korlátozásba ütköznek.

A technológiai fejlesztések, mint például a biztonságos online fizetési rendszerek ([1.1](#)), amelyek az EDI rendszerek alkalmazásának ismert, és az elektronikus jegyek, hozzájárultak az online jegyvásárlás népszerűségéhez. A vásárlók könnyedén és biztonságosan fizethetnek az online platformokon keresztül, és elektronikus jegyet kapnak, amelyet mobil eszközükön vagy nyomtatható formában mutathatnak fel az eseményen. Az elektronikus jegyek további előnye, hogy nehezen veszíthetők el vagy semmisülhetnek meg, így a vásárlók biztonságban tudhatják az értékes jegyeiket. Itt fontos megemlíteni, hogy ezen jegyek tárolása és biztonságban tartása további adatbiztonsági kérdéseket vet fel, amellyel foglalkozunk a dolgozat keretében. A platformokon keresztül a vásárlók egyszerűen választhatják ki a kívánt eseményt, a megfelelő ülőhelyet vagy jegytípust, és azonnal megvásárolhatják a jegyüket néhány kattintással. Ez időt és energiát takarít meg a vásárlók számára, ami napjainkban egy lényeges tényező.



**1.1. ábra.** Online fizetési rendszerek

További előnyeként egy ilyen platformnak meglehetősen, hogy a szervezők számára is jelentősen hatékonyabbá teszi a rendelések nyomon követését, statisztikák készítését, amelyeket felhasználhatnak értékesítési jelentések és a marketing javításához. Emellett az online jegyvásárlás lehetőséget nyújt a szervezőknek arra is, hogy célzottan reklámozzák az eseményüket, így nagyobb látogatottságot érhetnek el.

A jelenleg is működő hivatalos platform, ahol direkt módon juthatunk hozzá jegyekhez az F1-es versenyhétvégékre, az F1 Experiences. Itt gyorsan és kényelmesen vásárolhatjuk meg a kívánt jegyünket, amelyet a sikeres rendelés és kifizetés után emailben kapjuk meg PDF formátumban, amely tartalmazza a megvásárolt jegy(ek)et, egyedi azonosítókat és QR kódokat. Az email továbbá tartalmazza a számlázási adatokat. Az eseményre érve, a belépő kapuknál, egy erre a célra kihelyezett okos eszközzel megtörténik a QR kód olvasása és hitelesítése. Pozitív eredmény esetén beléphetünk az esemény helyszínére.

A fent említett folyamat megengedi, hogy ezek az elektronikus jegyek átruházhatóak a tulajdonos által bárki számára. Ezzel persze önmagában nincs probléma, mivel ezt a szabadságot meg kell lehessen adni a felhasználóknak, hogy bizonyos esetekben más személy tudjon részt venni a vásárló helyett, így nem veszik kárba a vásárlás. Ez a rendszer viszont teret ad egy olyan biztonsági kérdésnek, amelyet jelenleg a felhasználó felelősségére van bízva, miszerint ezt a kódot akár hetekkel, hónapokkal a használatuk előtt kapnak meg a felhasználók elektronikus levél formájában és azt bárki megszerezheti, aki-nek hozzáférése van a fiókhoz. Rosszabb esetekben, egy szándékos kibernetikai támadás esetén is eltulajdoníthatják és felhasználhatják. Ennek persze kisebb a valószínűsége, viszont ami egy aggasztó tény, hogy a felhasználók nagy része nem megfelelő módon kezeli az adatainak a biztonságos tárolását és számos esetben fellelhetők olyan emberi hibák, amelyeket kihasználnak az adathalászok, hogy hozzáférjenek a megvásárolt jegyekhez és saját célokra használják fel, többnyire illegális módon kereskedni velük.

Gyakran megtörténik, hogy egy felhasználó több oldalra is ugyan azokat a bejelentkezési adatokat adja meg a regisztráció során. Ez többségében a személyes email fiók felhasználó nevével és jelszójával megegyezik és ezt az adathalászok is figyelembe veszik. Egy másik sebezhetőség, hogy számos esetben egy fiókhoz több személy is hozzáfér, így már nem beszélhetünk biztonságos adattárolásról. Megemlíteni kell, hogy az email szolgáltatók biztosítanak E2EE-t az elektronikus levelek küldésekor és fogadásakor.

Az F1 Ticket Manager webes alkalmazás célja az alapvető jegyvásárlási funkcionálitások biztosítása, valamint a teljes vásárlási folyamat biztonságosabbá tétele. Ennek megvalósítására számos technológiai fejlesztés és programozói technika létezik. Az alkalmazás fejlesztésénél beépítésre került PIN kódok használata, amely egy emelt szintű biztonságot nyújt a felhasználó számára, valamint EGY? titkosítási algoritmus, amely az eredeti adatok azonnali visszafejtését hivatott megnehezíteni.

## 2. fejezet

# Elméleti megalapozás és szakirodalmi áttekintő

### 2.1. Webes alkalmazás felépítése

Az F1TM a React JavaScript programozási nyelv keretrendszerével valósult meg a Microsoft Visual Studio Code IDE-ben. Az alkalmazás használ harmadik féltől származó könyvtárakat is, amelyek felgyorsítják a fejlesztési folyamatot, mivel előre le van implementálva számos funkcionálitás. Ezek általában több fejlesztő által használtak és vannak tesztelve, ezért többségében gyorsabbak és biztonságosabbak.

Lévén, hogy az alkalmazás egy webes platform, a struktúrája két fő részből áll: frontend és backend. Ezen projekt keretében első sorban a frontend implementációján volt a hangsúly.

A frontend az a része a webes alkalmazásnak, amellyel a felhasználók közvetlenül interakcióba lépnek. Ez a rész felelős a UI megjelenítéséért és a felhasználói interakciók kezeléséért. A frontend általában a böngészőben fut, és a felhasználó által látott elemeket jeleníti meg, például az oldalak, űrlapok, gombok, navigációs elemek stb. A frontend tervezésekor figyelembe kell venni a UX aspektusait is. Ezek azért felelnek, hogy a felhasználói élmény a lehető legjobb legyen a weboldal böngészése során. Itt első sorban a letisztultság, átláthatóság és az összezavaró elemek elkerülése a legfőbb cél.

A frontend technológiák közé tartozhatnak:

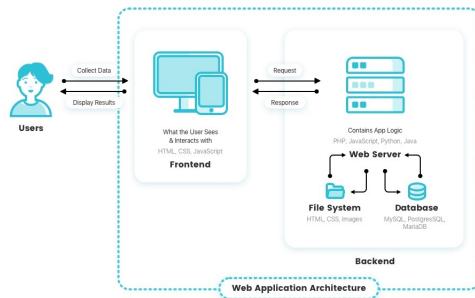
- HTML: Az alapvető struktúrát és tartalmat határozza meg a weboldalakhoz.
- CSS: A megjelenítést és a stílust adja a weboldalaknak, mint például a színek, betűtípusok, elrendezés stb.
- JavaScript: A dinamikus és interaktív funkciókért felelős, például animációk, eseménykezelés, adatmanipuláció. Gyakran használnak frontend keretrendszeret, például a React JS-t, Next.js-t vagy Angular-t, amelyek segítenek az alkalmazás fejlesztésében és szervezésében.

A backend a szerveroldali logikát és adatkezelést végzi. Ez a rész felelős az adatbáziskezelésért, a logika végrehajtásáért, a felhasználói kérések feldolgozásáért és a válaszok generálásáért. A backend nem közvetlenül látható vagy interaktív a felhasználók számára, viszont folyamatosan kommunikál a frontend-el az API-kon keresztül.

A backend technológiák közé tartozhatnak:

- Szerveroldali programozási nyelvek: Python, Ruby, Java, PHP stb.
- Keretrendszerök: Node.js, Django, Ruby on Rails, Laravel stb., amelyek segítenek az alkalmazás fejlesztésében és a szerveroldali logika megvalósításában.
- Adatbázis-kezelő rendszerek: MySQL, PostgreSQL, MongoDB stb., amelyek tárolják az alkalmazás adatait és lehetővé teszik ezek lekérdezését és manipulálását.

A frontend és backend között kommunikáció történik HTTP kérések és válaszok segítségével. A frontend kéréseket küld a backendnek, például adatlekérdezések vagy műveletek végrehajtása érdekében. Ezek a backend(ek) API végpontjain keresztül történnek. A backend feldolgozza ezeket a kéréseket, és visszaküldi a válaszokat a frontendnek (2.1). Nagyon alkalmazások esetében megtörténhet, hogy a frontend több backend szerverről kéri le az információkat. Ez az felhasználó számára nem feltűnő, mivel egy megfelelő UX-szel rendelkező frontend minden esetben egy státuszjelző elemet helyez a betöltés idejére (animáció), függetlenül attól, hogy éppen melyik szerverrel történik a kommunikáció.



**2.1. ábra.** Webes alkalmazás architekúrája

## 2.2. A React JavaScript keretrendszer

Az elmúlt években a React JS jelentős népszerűségre tett szert a webfejlesztés területén. A React egy nyílt forráskódú JavaScript keretrendszer, amelyet a Facebook (Meta) fejlesztett ki, és célja a felhasználói felületek könnyű és hatékony megvalósítása. A React alapvetően egy komponens alapú megközelítést kínál, amely lehetővé teszi a fejlesztők számára, hogy újra felhasználható, önálló építőelemeket hozzanak létre, amelyeket könnyedén kombinálhatnak egymással a komplexebb felhasználói felületek elkészítése érdekében.

A React (2.2) rendkívül népszerűvé vált a fejlesztők körében számos előnye miatt. Elsőként említhetjük a hatékony Virtual DOM implementációját, amely lehetővé teszi az alkalmazások gyors és hatékony frissítését. A Virtual DOM a weboldal megjelenítéséhez használt valós DOM virtuális reprezentációja. Amikor változás történik az adatokban, a React a Virtual DOM-on keresztül kiszámítja az optimális frissítéseket, majd ezeket a változtatásokat csak a valós DOM-ra alkalmazza. Ez a megközelítés jelentős sebességjavulást eredményez a webalkalmazásokban.

A második fontos előny a komponens alapú megközelítés, amely lehetővé teszi a fejlesztők számára a komponensek újra felhasználását és a kód modularizációját. A React komponensek önmagukban zárt egységek, amelyek különböző feladatokat elláthatnak a felhasználói felületeken, például gombok, űrlapok, listák stb. Az egyedi komponensek könnyedén kombinálhatók egymással, így a fejlesztőknek nem kell újra megírniuk a kódot, hanem egyszerűen felhasználhatják a meglévő komponenseket.

Emellett más technológiai óriások is felfedezték a React előnyeit. Például a Netflix, a PayPal, az Airbnb és a Dropbox is a React-et használja az alkalmazásai fejlesztéséhez. Ezek a vállalatok magas forgalommal és komplex felhasználói felületekkel rendelkeznek, és a React segítségével könnyedén kezelhetik ezeket a kihívásokat.

Az open-source software (OSS) közösség is hozzájárult a React népszerűségének növekedéséhez. Számos harmadik által (Third party) fél készített könyvtár és eszköz érhető el a React-hez, amelyek további lehetőségeket kínálnak a fejlesztőknek. Ilyen példa a Redux, amely egy állapotkezelő könyvtár, vagy a React Router, amely segít az alkalmazások útvonalainak kezelésében.

A JavaScript önmagában egy típusfüggetlen programozási nyelv, de egyes keretrendszerök, mint a Next.js, a Microsoft által kifejlesztett TypeScript nyelvet használják, amely már megengedi a beépített és személyre szabott típusok használatát.



**2.2. ábra.** React logó

### 2.3. A Google Firebase platform

Mivel a React a webes alkalmazások frontend-jének fejlesztésére szolgál, szükséges volt egy backend szerverre és egy adatbázis-kezelő rendszerre, amely kiszolgálja a frontend-et, a Google által fejlesztett Firebase platformot és API-kat építettem be az alkalmazásba. A Google Firebase (2.3) egy teljes körű fejlesztői platform, amely különféle eszközöket és szolgáltatásokat kínál a fejlesztőknek, hogy könnyedén hozzanak létre, teszteljenek és üzemeltessenek webes és mobilalkalmazásokat.

A Firebase Realtime Database egy NoSQL alapú adatbázis, amely valós idejű adatszinkronizációt tesz lehetővé az alkalmazások között. Ez azt jelenti, hogy az adatok automatikusan frissülnek minden csatlakozott eszközön, így a felhasználók valós idejű élményt élvezhetnek. Ez különösen hasznos például csevegőalkalmazások vagy valós idejű játékok fejlesztésekor.

A Firebase Authentication lehetővé teszi a felhasználók egyszerű és biztonságos hitelesítését. Támogatja az e-mail és jelszó, a szociális média hitelesítés (pl. Google, Facebook, Twitter) és más autentikációs módokat is. A Firebase emellett lehetőséget biztosít a felhasználói fiókok-, profilok kezelésére és jogosultságkezelésre is.

A Firebase Cloud Storage segítségével könnyedén tárolhatóak és kezelhetőek az alkalmazásban használt fájlok, például képek, hangfájlok vagy videók. Az egyszerű API-k lehetővé teszik a fájlok feltöltését, letöltését és megosztását. Emellett a Firebase Hosting segítségével egyszerűen és gyorsan kiszolgálhatod az alkalmazásod statikus fájljait a világ minden tájáról.

A Storage Bucket-ek egy nagy tárolóhelyet jelentenek a fájlok (például képek, hangfájlok, videók stb.) biztonságos tárolására a felhőben. A Firebase Storage lehetővé teszi a fájlok feltöltését, letöltését, törlését és megosztását egyszerű API-k segítségével. Emellett automatikusan kezeli a fájlok hozzáférési jogosultságait és a CDN révén biztosítja a gyors és megbízható fájlletöltést a felhasználóknak.



**2.3. ábra.** Firebase logó

## 2.4. Az AES titkosítás

### 2.4.1. Bevezető

Az AES egy blokk titkosítási algoritmus, amelyet két belga kriptográfus tervezett 1997-ben. Az algoritmus az 1990-ben meghirdetett nyilvános pályázat nyertese lett, és 2001-ben a NIST (National Institute of Standards and Technology) elfogadta az Egyesült Államok kormányzati szervezetei számára ajánlott titkosítási szabványnak. Mivel az említett pályázat elbírálása publikusan történt, ezért valószínűleg nem történt befolyás a díj megítélését illetően. Az algoritmus egy változata a Rijndael algoritmusnak, amely eredetileg több blokkmérettel is dolgozott és ebből választották ki az AES-t, amelynek blokkmérete 128 bit.

Az AES blokk mérete 128 bit, és a kulcs mérete lehet 128, 192 vagy 256 bit. Az AES hatékonyisége körülbelül 109 Mb/s, amely természetesen függ az adott hardver tulajdon-ságaitól, és szakértők szerint a 256 bites kulcsméret biztonsága "török" időkre szól.

Az AES nem használja a Feistel-sémát, mint a DES (Data Encryption Standard), hanem iteratív szerkezetű. Az algoritmus a kulcsméret alapján különböző számú körökben végzi a titkosítást, amelyekhez korkulcsokat generál. A 128 bites kulcs esetén a körök száma 10, a 192 bites kulcs esetén a körök száma 12, míg a 256 bites kulcs esetén a körök száma 14.

Az AES helyettesítést és permutációt alkalmaz a blokkon belül, és véges testek felett végez aritmetikai és műveleteket. Az algoritmus tényerésére az szolgált, hogy a művelet végzés egész számokkal történik, ezért a szerzők egy olyan algoritmust fejlesztettek, ahol az összeadás, kivonás, szorzás, osztás után is halmaz béli elemet kapunk. Az AES véges testként használja a bináris együtthatós polinomokat a  $GF(2^n)$  felett, ahol  $n = 8$  fokszámnál kisebb.

Az AES minden műveletet 8 biten végez, és az összes 30 irreducibilis polinom közül a következőt használja:  $x^8 + x^4 + x^3 + x + 1$ . Az AES minden bájtot a  $GF(2^8) = GF(2)[x]/(x^8 + x^4 + x^3 + x + 1)$  testelemeként kezel.

Az AES-t általában három algoritmus alkotja: a kulcsgenerálás, a titkosítás és a visszafejtés. Az AES a bemeneti 128 bites állapotot (state) egy 4x4-es mátrix formájában kezeli, és az algoritmus minden iterációja során helyettesítő és permutációs műveleteket végez el a blokkon belül.

### 2.4.2. Az algoritmus elmélete

Az AES a bemeneti adatokat 128 bites blokkokra bontja. A bemeneti adatokat több körön keresztül módosítja míg el nem jutunk a titkosított adatig. A módosításokat az algoritmus több lépésben hajtja végre. Ezeket a lépéseket altranszformációknak nevezzük.

Az altranszformációk három típusúak lehetnek: AddRoundKey, SubBytes és Shift-Rows. Az AddRoundKey lépés során az aktuális kör kulcsát XOR művelettel adja hozzá adatblokkhoz. A SubBytes lépés a bemeneti blokk minden elemén végigmegy egy előre meghatározott nemlineáris függvényel. A ShiftRows lépés során az adatblokk minden sorát egy bizonyos számmal rotáljuk.

Összességében az AES titkosítási algoritmus magában foglalja a bemeneti adatok blokkokra bontását, majd a bemeneti blokkokon végre hajtja az altranszformációkat több körön keresztül, amíg meg nem kapja a titkosított adatot. Az algoritmus célja az adatok biztonságos és hatékony titkosítása a megfelelő védelem érdekében.

### 2.4.3. Az algoritmus gyakorlatban

Az alábbi kódrészlet az AES algoritmus egyik használati módja JavaScript programozási nyelven a CryptoJS könyvtárcsomag segítségével (2.1).

A kód első lépése a *salt (só)* generálása, ami egy 128/8 bájt szekvencia, amely segítségével meghatározásra kerül majd a key (kulcs). A *CryptoJS.lib.WordArray.random(128 / 8)* kódsor a **CryptoJS** könyvtárban található *random()* függvény hívásával egy véletlenszerű 128-bites szekvenciát generál, majd beilleszti az adatokat a WordArray osztályba.

A következő lépésekben a kulcs előállítása történik meg a *secretPass (jelszó)* és a só használatával a kulcsderiváló függvény segítségével. A kulcs előállítása a *CryptoJS.PBKDF2()* függvényel történik, amely egy kulcstervező függvény. Az első paraméter a titkosításhoz használt jelszó, a második argumentum a só, a harmadik pedig a kulcs hosszát és az iterációk számát határozza meg. Ennél az algoritmusnál nagyon fontos odafigyelni, hogy a jelszó szigorúan titkos információ, vagyis ezt biztonságosan ajánlott eltárolni szerver oldalon. A többi paraméter publikus, mert önmagukban nem elegendők a titkosított szöveg visszafejtéséhez.

Az inicializáló *vektor (iv)* generálása következik. Az IV egy véletlenszerű bájt szekvencia, amelynek hossza megegyezik a blokk méretével (128 bit), és a titkosítás során használják, hogy azonos adatok esetén is véletlenszerű kimenetet generáljon.

Az adat titkosítása a *CryptoJS.AES.encrypt()* függvénytel függvénytel történik. Az adatot először JSON formátumba alakítjuk, majd az AES algoritmust a kulcs, az IV és további paraméterek (mód, padding és tag) megadásával alkalmazzuk. A *mode* a blokktitkosítási mód (BCM) kiválasztására szolgál, ezek között található a ECB, CBC, OFB vagy CFB. A mi esetünkben a CBC (Cipher Block Chaining) mód van használva.

```
export const encryptData = text => {
  const salt = CryptoJS.lib.WordArray.random(128 / 8);
  const key = CryptoJS.PBKDF2(secretPass, salt, {
    keySize: 256 / 32,
    iterations: 1000,
  });
  const iv = CryptoJS.lib.WordArray.random(128 / 8);

  const encrypted = CryptoJS.AES.encrypt(JSON.stringify(text), key, {
    iv: iv,
    mode: CryptoJS.mode.CBC,
    padding: CryptoJS.pad.Pkcs7,
    tag: true,
  });

  const data = {
    ciphertext: encrypted.ciphertext.toString(CryptoJS.enc.Base64),
    iv: iv.toString(CryptoJS.enc.Base64),
    salt: salt.toString(CryptoJS.enc.Base64),
    tag: true,
  };
  return JSON.stringify(data);
};
```

## 2.1. kódrészlet. Titkosítás példakód.

A *padding* a blokkok kitöltési módját határozza meg, itt a **Pkcs7** (Public Key Cryptography Standards 7-es szabványa) van használva. A Pkcs7 padding szabvány szerint, ha az üzenet hossza nem egész blokk méretű, akkor azt szükséges kiegészíteni. minden hiányzó bájt felveszi a hiányzó bájtok számának megfelelő értéket. Ha például az utolsó blokkban 3 hiányzó bájl van, akkor a blokk utolsó bájtai a 0x03 értéket vesz fel.

Az textitiv az inicializáló vektort tartalmazza, amit korábban generáltunk. A textit-tag beállítása igazra van állítva, hogy az üzenet hitelesítése (integritásának ellenőrzése) is megtörténjen a titkosítás során. A tag az üzenet elejére kerül beillesztésre a titkosítás során, és végül az eredeti üzenet végén kerül ellenőrzésre a helyessége.

Az utolsó lépés a textittitkosított adat (ciphertext), az IV, a só és az üzenet hitelesítésének értéke (tag) összekapcsolása és JSON formátumba rendezése és visszatérítése.

Az alábbi példakód a visszafejtést valósítja meg (2.2):

```
export const encryptData = text => {
  const salt = CryptoJS.lib.WordArray.random(128 / 8);
  const key = CryptoJS.PBKDF2(secretPass, salt, {
    keySize: 256 / 32,
    iterations: 1000,
  });
  const iv = CryptoJS.lib.WordArray.random(128 / 8);

  const encrypted = CryptoJS.AES.encrypt(JSON.stringify(text), key, {
    iv: iv,
    mode: CryptoJS.mode.CBC,
    padding: CryptoJS.pad.Pkcs7,
    tag: true,
  });

  const data = {
    ciphertext: encrypted.ciphertext.toString(CryptoJS.enc.Base64),
    iv: iv.toString(CryptoJS.enc.Base64),
    salt: salt.toString(CryptoJS.enc.Base64),
    tag: true,
  };
  return JSON.stringify(data);
};
```

## 2.2. kódrészlet. Visszafejtés példakód.

A **decrypt** függvény az **encrypt** függvénnyel ellentétes sorrendben dolgozza fel az adatokat. A paraméterként kapott titkosított szövegből JSON objektumot állít elő. Ezután ebből az objektumból megkapja a só és az iv Base64 formátumból az értékeket. A kulcs előállítása a *PBKDF2* függvénnyel történik a jelszó és a só segítségével. A titkosított adat, az iv, a só és a hitelesítési tag felhasználásával sikeresen vissza tudjuk fejteni az eredeti üzenetet.

#### **2.4.4. Alkalmazások és érdekességek**

- Az AES algoritmusnak 128, 192 és 256 bites változatai vannak, amelyek mindegyike különböző szintű biztonságot nyújt.
- Az algoritmus rendkívül hatékony, amely lehetővé teszi a titkosított adatok nagy sebességű feldolgozását.
- Nyilvánosan elérhető és ingyenesen használható, ami azt jelenti, hogy bárki használhatja és integrálhatja az alkalmazásába.
- Ennek az algoritmusnak számos könyvtárcsomagban van megvalósítva. Léteznek egyaránt publikus és privát forráskódok, amelyek közül néhány optimalizálva van a hardveres és szoftveres rendszerekhez.
- Az alkalmazása különböző területeken népszerű, például az online banki- és pénzügyi tranzakciókban, a VPN (Virtual Private Network) rendszerekben, a Wi-Fi hálózatokban, az adattároló eszközökön.
- Az AES használata akkor biztosít teljes biztonságot, ha ezt együtt használják egyéb kriptográfiai primitívekkel. Ilyen az AES-en kívül az RSA vagy a Hashmap. A kulcsfontosságú védelmi rendszerekben, mint például a HTTPS, SSH vagy TLS, használnak több rétegű védelmi mechanizmusokat az AES kiegészítéseként.
- A 256 bites titkosítási kulcsokat sokkal nehezebb brute-force módon támadni, mint egy 128 bites kulcsot. Azonban az utóbbi is olyan hosszú időbe telik kitalálni, még hatalmas számítási kapacitás mellett is, hogy az előrelátható jövőben nem lesz probléma, mert egy támadónak is hatalmas számítási kapacitásra lenne szüksége a szükséges brute-force (nyers-erő módszere) generáláshoz.
- Azonban a 256 bites kulcsokhoz is több feldolgozási teljesítmény szükséges és hosszabb ideig tarthat a generálásuk. Amikor az energiafogyasztás problémát jelent, különösen kis eszközök esetében, vagy a késletetés valószínű, a 128 bites kulcsok jobb választásnak számítanak.

### **2.5. RSA (TBD)**

### **2.6. Kutatási kérdések**

Az alkalmazás fejlesztése során a következő kérdésekre próbáltam válaszokat keresni:

- Hogyan lehet biztonságosabbá tenni az elektronikus jegyvásárlást és azok felhasználását?
- Mely technológiák segítségével lehet gyors és hatékony online üzletet tervezni és megvalósítani?
- Hogyan lehet a megvalósított rendszert hosszú távon karban tartani és felügyelni?
- Hogyan lehetséges a rendszer automatizálása valós időben?

## 2.7. Célkitűzések

A kutatási kérdésekre keresett válaszok során felmerültek további kérdések, amelyek hozzájárulnak a céljaim pontosabb megfogalmazásában:

- Hol és hogyan fogom tárolni a felhasználói- és alkalmazás adatokat?
- Hogyan fogom kelelni a felhasználói jogosultságokat, hogy minden felhasználó csak a saját adataihoz férjen hozzá?
- Hogyan fogom megvalósítani a többlépcsős azonosítást a jegyek felhasználásánál?
- Hogyan fogom eljuttatni a felhasználóknak a megvásárolt termék(ek)ről és azok azonosításához szükséges információkat?
- Hogyan tudom a felületet felhasználó baráttá tenni?

A megfogalmazott kérdések azt a legfontosabb célt hivatottak szolgálni, hogy egy kényelmes és megbízható alkalmazás jöjjön létre a felhasználók számára. Ennek megvalósításához szükséges szempont, hogy megbizonyosodjanak a felhasználók, hogy a felület használata nem féleérhető és nincsenek olyan működési aspektusok, amelyek zavarhatják a felhasználói élményt.

Az előbbiekt elérésére a kisebb célok egy még pontosabb képet tudnak biztosítani számonra, hogy megfelelő ütemben haladjon a fejlesztés anélkül, hogy kimaradjanak fontos részletek.

A fent felsorolt kérdések felhívják a figyelmet arra a fontos tényezőre, hogy az egyre elterjedtebb körű internet használatának korszakában elengedhetetlen a felhasználók személyes adatainak a védelme. Ennek megvalósítására is érdemes számos megoldást kutatni és majd kiválasztani az alkalmazás szempontjából megfelelőt.

Továbbá a előzőek mellett megoldást kell kapjak arra a problémára, hogy a jelenleg is működő oldalak nagy része nem foglalkozik azzal a tényezővel, hogy egy elektronikus jegy nagyon sok veszélynek van kitéve és ennek ellenére nem alkalmaznak többlépcsős azonosítást a jegyek hitelesítésekor. Ennek egyszerűen az az oka, hogy bizonyos százalékban lassítaná a hitelesítési folyamatot és az eladók nem vállalnák felelősséget a jegyek esetleges elveszítése vagy ellopása esetén.

Végül de nem utolsó sorban fontos, hogy lehetőséget biztosítsak az alkalmazás viselkedésének monitorizálására, valamint az alkalmazás továbbfejleszthetőségére. Az alkalmazás működésének megfigyelése során olyan hibákról és ötletekre kaphatók visszajelzést, amivel tudom biztosítani az alkalmazás hosszútávú karbantartása és továbbfejlesztése érdekében.

## 3. fejezet

# Rendszerspecifikáció

### 3.1. Rendszer követelmények

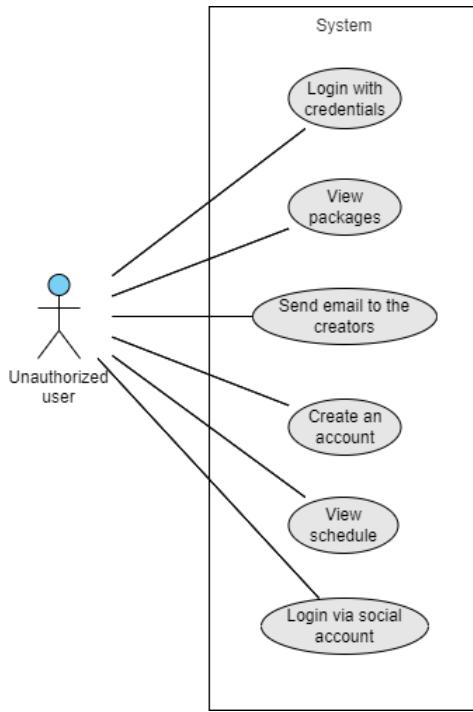
A rendszer specifikációt négy nagy részre oszthatjuk fel, amelyek közé tartoznak a felhasználói felület-, a jegyvásárlás-, a felhasználók- és a biztonság specifikációja. Elsőként a felhasználói felület biztosítja a felhasználó és a rendszer közötti kommunikációt. A jegyvásárlás a rendszer központi funkcionálitása, ezáltal a legkomplexebb is. Az alkalmazás lehetővé teszi a több típusú felhasználók hozzáférését a különböző funkcionálitásokhoz. Végül de nem utolsó sorban a rendszer a bizalmas adatok biztonságát is hivatott kiszolgálni. A specifikációk prezentálására a Visual Paradigm Online felület segítségével hoztam létre a szükséges diagramokat. A rendszer működését vizsgálni tudjuk a funkcionális- és nem funkcionális követelmény alapján.

#### 3.1.1. Funkcionális követelmények

A funkcionális követelmények a rendszer funkcionálitásaira vonatkoznak, azaz meghatározzák, hogy milyen feladatokat és műveleteket kell elvégeznie a rendszernek, milyen funkciókat kell biztosítania a felhasználók számára.

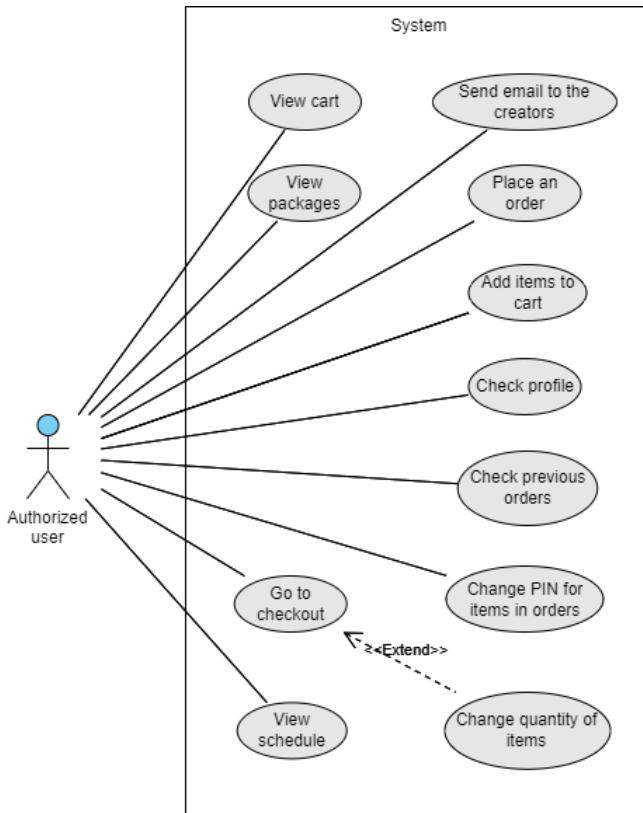
##### a, Felhasználói szerepkörök:

Az oldal böngészésének tekintetében érdemes különboző jogosultságokkal rendelkező szerepköröket kialakítani. Erre azért van szükség, mivel a rendszer több típusú felhasználó által látogatható. Ezek közül a legelső és egyben a legkevesebb funkcionálitással rendelkező az **Anonymous**, amelyek *be nem jelentkezett* felhasználók, esetében ilyen funkcionális követelmények például a jegyek közötti böngészés lehetősége, a jegyek részleteinek megtekintése, valamint az email-küldés a *Support* számára. Emellett fontos, hogy az Anonymous felhasználók be tudjanak jelentkezni az oldalra vagy új felhasználói fiókot tudjanak regisztrálni, és ez a bejelentkezési folyamat lehetőséget biztosít az oldalon regisztrált fiókok vagy akár más közösségi fiókok használatára is (3.1).



**3.1. ábra.** Use case diagram - Nem bejelentkezett felhasználó

A **Regular** felhasználók esetében elvárt, hogy *be legyenek jelentkezve* a rendszerbe. Lehetőségük van a jegyek kosárba helyezése, visszajelzések írása és jegyek értékelése. Emellett a felhasználók képesek megtekinteni a profiljukat, ahol lehetőségük van felhasználói név és profil kép változtatására, valamint a vásárlási előzményeik megtekintésére. Az előzmények részleteinél a felhasználóknak lehetőségük van az adott jegy PIN kódjának megváltoztatására, amennyiben szükséges. A vásárlási folyamat során a felhasználók a rendelési oldalra (*Checkout*) jutnak el, ahol lehetőségük van a listaelemek mennyiségének módosítására vagy azok törlésére. Itt történik meg a fizetés és a rendelés leadása (3.2).

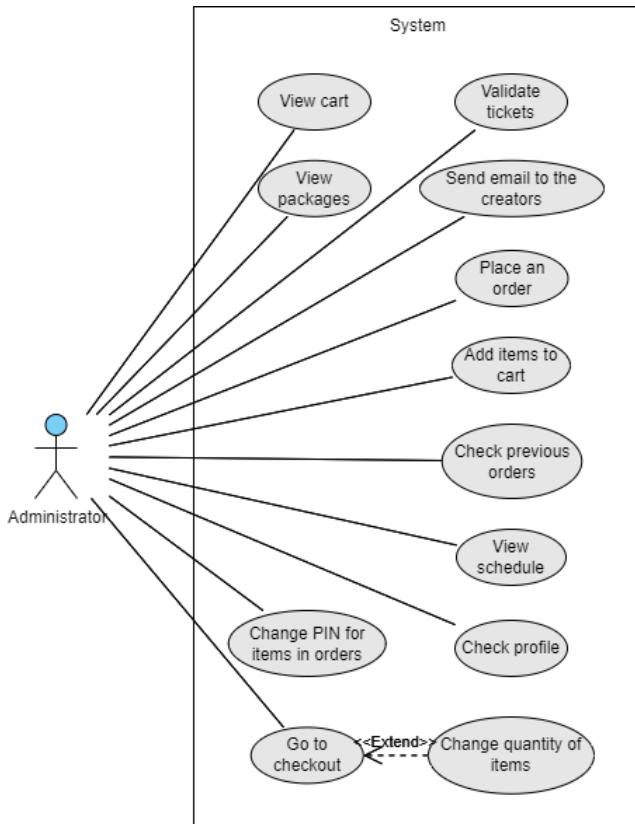


**3.2. ábra.** Use case diagram - Bejelentkezett felhasználó

Az adminisztrátorok a harmadik kategóriát alkotják a rendszerben, és különleges jogosultságokkal rendelkeznek. Az **Admin** felhasználóknak joguk van az összes korábban említett funkcionálishez, mint például a jegyek kosárba helyezése, visszaüzenetek írása és értékelése, valamint a profiljuk kezelése és vásárlási előzményeik megtekintése. Azonban az Admin felhasználóknak további feladataik is van, nevezetesen a jegyek hitelesítése.

Az Admin felhasználóknak lehetőségük van a jegyek hitelesítésére, amelyhez a QR kódot kell olvasniuk és meg kell adniuk a PIN kódot. Ez a folyamat biztosítja, hogy csak érvényes jegyek kerüljenek elfogadásra és használatra a rendszerben. Ez a szerepkör manuálisan adható hozzá a rendszerhez, és szükség esetén módosítható.

Fontos megjegyezni, hogy bár az adminisztrátoroknak lehetőségük van jegyek vásárlására, ezt kizárálag a rendszer tesztelésére és karbantartására szolgáló célokra használható. Az adminisztrátorok elsődleges felelőssége a rendszer megfelelő működésének biztosítása és a jegyek hitelesítése (3.3).



**3.3. ábra.** Use case diagram - Adminisztrátor

#### b, Jegyvásárlás:

A **jegyvásárlás** során a felhasználóknak számos funkcionálitást kell biztosítani. Először is, lehetőséget kell adni nekik, hogy kiválaszthassák a jegyeket és azokat kosárba helyezhessék. Emellett fontos, hogy a felhasználók módosíthassák a jegyek mennyiségét vagy akár törölhessék azokat, ha szükséges. A böngészés során pedig lehetővé kell tenni számukra, hogy részletes információkat kapjanak a jegyekről, mint például a helyszín, az időpont vagy az ár. Ez segíti őket a megfelelő döntéshozatalban és a kívánt jegyek megtalálásában.

#### c, Bejelentkezés és regisztráció:

A **bejelentkezés** és **regisztráció** lehetővé teszik a felhasználók számára, hogy saját profiljuk legyen a rendszerben, amellyel vásárlásokat tudnak végrehajtani és nyomon tudják követni azokat.

A **bejelentkezés** folyamata magában foglalja a felhasználónév és jelszó megadását vagy egy harmadik féltől származó fiók segítségével. A felhasználóknak lehetőségeük van megadni a regisztrált felhasználónévüket és a hozzájuk tartozó jelszavukat a bejelentkezéshez. A felhasználói felületnek tartalmaznia kell egy bejelentkezés gombot, amelyre kattintva a felhasználó bejelentkezik a rendszerbe. Ezután a rendszer ellenőrzi az adatok helyességét és hitelesíti a felhasználót, hogy hozzáférjen a felhasználói funkciókhöz.

A **regisztráció** lehetőséget ad a felhasználóknak arra, hogy új fiókot hozzanak létre a rendszerben. Ehhez a felhasználóknak kitölteniük kell egy regisztrációs űrlapot, amely tartalmazza az szükséges adatokat, például felhasználónevüket, jelszavukat, e-mail címüket. A felületnek tartalmaznia kell egy regisztrációs gombot, amelyre kattintva a rendszer regisztrálja a felhasználót és létrehozza az új fiókját.

### **3.1.2. Nem funkcionális követelmények**

A nem funkcionális követelmények olyan aspektusokra fókusznak, amelyek nem közvetlenül a rendszer funkcionálitásához kapcsolódnak, hanem inkább annak működési jellemzőit, tulajdonságait vagy környezeti tényezőit érintik.

#### **a, Felhasználói szerepkörök:**

A **felhasználói szerepkörök** között egyértelmű határvonal kell legyen, hogy melyeknek mihez van jogosultságuk. A felhasználóknak könnyen és zökkenőmentesen kell tudniuk használni az oldalt függetlenül a szerepkörüktől. Fontos továbbá, hogy az egyes szerepkörökhöz tartozó jogosultságok és hozzáférési szintek megbízhatóak és biztonságosak legyenek, hogy a felhasználók csak azokhoz az információkhoz és funkciókhoz férjenek hozzá, amelyek az adott szerepkörhöz kötött.

#### **b, Jegyvásárlás:**

A **jegyvásárlási** folyamatnak gyorsnak és hatékonynak kell lennie. A rendszernek képesnek kell lennie a nagyobb mennyiségi jegykezelésre és skálázhatónak kell lennie, hogy a jegyvásárlás során ne jelentkezzenek teljesítménybeli problémák. Emellett a jegyvásárlás folyamatának biztonságosnak is kell lennie, és megfelelő védelmi intézkedéseket kell tartalmaznia az adatbiztonság érdekében. Ilyen intézkedések például a titkosított adatátvitel vagy a vásárlói adatok megfelelő védelme.

#### **c, Bejelentkezés és regisztráció:**

A **bejelentkezés és regisztráció** folyamata kapcsán a nem funkcionális követelmények között szerepel, hogy a folyamatnak biztonságosnak kell lennie. A bejelentkezési és regisztrációs folyamatnak megfelelő hitelesítési és azonosítási mechanizmusokat kell tartalmaznia. Emellett a folyamatnak gyorsnak és felhasználóbarátnak kell lennie, hogy a felhasználók könnyedén és zökkenőmentesen tudjanak bejelentkezni vagy regisztrálni. A felhasználóknak egyszerű és intuitív felhasználói felületet kell biztosítani a bejelentkezéshez és regisztrációhoz, hogy könnyen megadhassák szükséges információikat és végrehajthassák az adott folyamatot.

## **3.2. Felhasználói követelmények**

#### **a, Funkcionalitások:**

A főbb funkcionalitások tervezésénél, amelyekről szó volt a funkcionális rendszer követelmények alfejezet (3.1.1) keretében, figyelembe vettettem, hogy azok a lehető legjobban betöltség a feladatukat, így a felhasználók zökkenőmentesen tudják használni a rendszert. Figyelembe vettettem az implementáció során a funkcionalitások teljesítményének, megbízhatóságának és használhatóságának a fontosságát.

**b, Felhasználói felület:**

A rendszer felhasználói felületének fejlesztése első sorban arra irányult, hogy intuitív és könnyen érthető legyen, ezáltal a felhasználók ne tapasztaljanak nehézségeket vagy zavarokat az alkalmazás használata során. Egy felhasználó első benyomása határozza meg a legnagyobb mértékben a rendszerről alkotott véleményét, ezért kellő figyelmet fordítottam arra, hogy az oldal letisztult legyen, elkerülve a félrevezető utasításokat és funkcionálitásokat. A felhasználó az oldal első látogatásakor értesül az oldal céljáról és további információkról a hivatalos oldalak elérésére.

**c, Teljesítmény és megbízhatóság:**

Az alkalmazás teljesítménye több komponensből áll össze, amelyek közé tartoznak a reakcióidő, válaszkészség, stabilitás és rendelkezésre állás. Fontosnak tartottam, hogy az alkalmazás gyorsan és megbízhatóan működjön, hogy a felhasználók ne érezzenek frusztráló lassulásokat vagy hibákat. Persze a hibák teljes mértékű elkerülése majdnem hogy lehetetlen a szoftver rendszerek esetében, mivel az emberi hiba lehetősége minden jelen van. Arra törekedtem, hogy minimalizálva legyen a hibák előfordulásának lehetősége és megelőző intézkedéseket vezettem be. Ide tartoznak azok a megfelelő hibaüzenet is, amelyek kisebb vagy nagyobb hibák esetén is célravezető módon informálják a felhasználót, hogy ez a hiba miként folyásolja be a rendszer további működését vagy adott esetben mit tud tenni a felhasználó a hiba elkerülése ellen.

**d, Profilkezelés:**

A felhasználók számára fontos szempont, hogy hozzáférjenek a saját adataikhoz egy oldalon és módosítani tudják őket. Erre a célra hoztam létre egy aloldalt, ahol kényelmesen lehet profil képet, felhasználónévét módosítani és a rendelési előzeteseket visszanézni. Továbbá lehetőségük van, hogy az egyes rendelésekhez új PIN kódot rendeljenek.

## e, Kompatibilitás:

Az alkalmazás elsősorban a desktop számítógépekre lett optimalizálva, viszont egy következő továbbfejlesztési lehetőség lenne, hogy reszponzív legyen, ezáltal mobilon és táblagépen kezelhetőbb. A rendszer kompatibilis a közismert modern böngészők mindegyikével, mint a Microsoft Edge, Google Chrome, Mozilla Firefox, Apple Safari vagy az Opera ([3.4](#)).

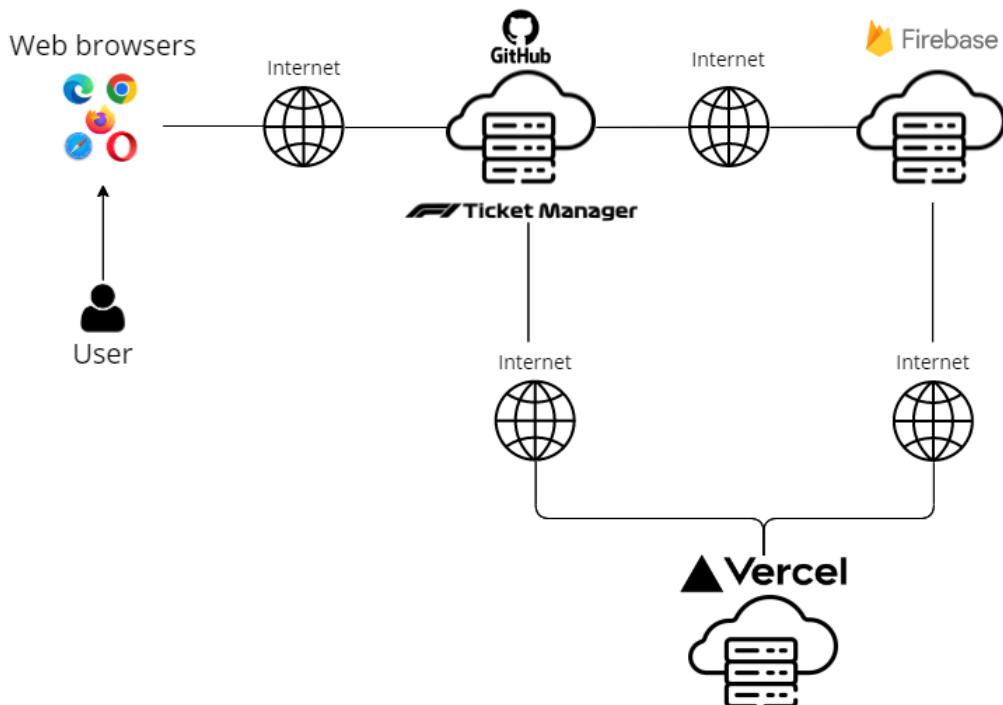


**3.4. ábra.** Web böngészők

## 4. fejezet

# A rendszer architektúrája

Az alkalmazás architektúrája három nagyobb komponensből áll. Mindhárom egy különálló szerver, amelyek egy teljesen összefüggő gráfként képesek kommunikálni egymással (4.1).



4.1. ábra. Rendszer architekúra

Az rendszer fő komponense a weboldalt üzemeltető szerver, a **GitHub Pages**. Ez felel az oldal rendereléséért és valós időben elérhetőségéről. Erre azért van lehetőség, mivel a forráskód megtalálható a GitHub-on, amelyet összekötöttem a GitHub Pages-el. A deployment fázis során egy optimalizált deploy verziót készít a kód ból és azt hosztolja. Ez a szerver

A második komponens a **Google Firebase** CDN alapú felhőalapú szolgáltató. Segítségével meg lehet valósítani felhasználók autentikációját, adatok tárolását egy NoSQL adatbázis-kezelő rendszerben és még sok más. Azért ezeket emeltem ki, mivel ezek a funk-

cionalitások játszanak szerepet a rendszerben. A weboldal folyamatosan kommunikál a Google Firebase-el, hogy valós időben és hatékonyan tudja kiszolgálni a felhasználókat. Az **F1 Ticket Manager** egyik központi eleme a jegyek kezelése, amely során titkosítási algoritmusok futnak a felhasználók adatainak biztonsága érdekében. Mivel ezeket az algoritmusokat nem lehetséges, hogy a Google Firebase ingyenes verziójával igénybe vegyem, ezért szükségem volt egy másik backend szerverre is, ahol el tudom végezni ezeket a műveleteket.

A harmadik és egyben utolsó komponens a **Vercel** felhőalapú szolgáltató, amelyen fut egy NodeJs szerver, amelyet én fejlesztettem. Ennek a szervernek a API-jai felelnek a titkosítási folyamatokért. Mivel az adatok csak a Google Firestore-ban vannak tárolva, ezért elengedhetetlen, hogy kommunikáljon vele.

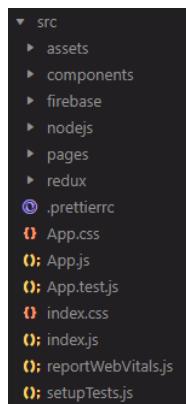
## 4.1. GitHub frontend architektúra

A rendszer frontendjét szolgáló komponens, amelyet a GitHub Pages hosztol, egy **create-react-app** alkalmazás. Ez egy olyan **npm** (4.2) JavaScript futási környezet csomagkezelője [Wika] által forgalmazott projekt sablon, amely tartalmazza az alapvető konfigurációkat és fájlokat, amelyek szükségesek egy **React** alkalmazás készítéséhez.



4.2. ábra. Az npm csomagkezelő

Az F1TM frontend kódmezőnyének gyökérkönyvtára az *src* mappa (4.3), amely tartalmazza az alkalmazás komponenseit, oldalait, a többi szerver végpontjaira kapcsolódó függvényeket és a React **Redux** kezelésére használt kódokat.

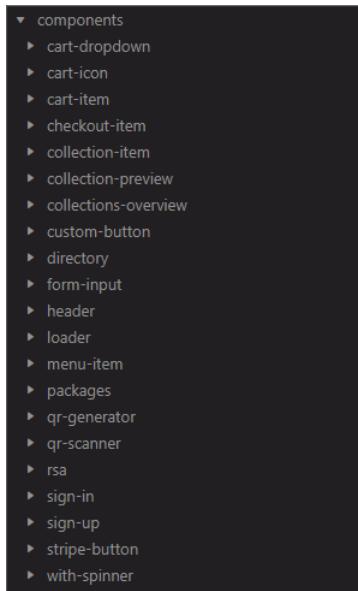


4.3. ábra. Az src mappa struktúrája

A mappa struktúrába rendezett komponensek (4.4) áttekinthetőbbé teszik a rendszert. Az adott funkcióhoz tartozó összes fájl vagy modul egy helyen vannak, így könnyen megtalálhatóak és karbantarthatóak. Ez különösen hasznos, ha több fejlesztő dolgozik ugyanazon a rendszeren, vagy ha idővel vizsgálni vagy módosítani kell a komponenseket.

Az alaposan megszervezett struktúra lehetővé teszi a komponensek könnyű újrafelhasználását. Ez jelentősen csökkentheti a fejlesztési időt és erőforrásokat, mivel nem kell minden újra implementálni vagy átírni.

Továbbá egy új funkció bevezetésénél nagyon megkönnyíti a mapparendszer, hogy könnyen megtaláljam a megfelelő helyét az új fájloknak. Az új funkciót egy új komponensként hozzáadhatjuk a megfelelő mappába anélkül, hogy át kellene írni vagy megváltoztatni a meglévő komponenseket. Ez tisztább és kevésbé összeavaró kódmezőt eredményez, és minimalizálja a mellékhatásokat vagy a hibák kockázatát.



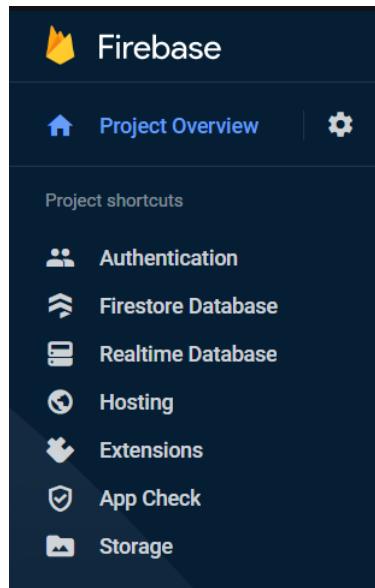
**4.4. ábra.** Komponensek

A **Vercel** szerverrel való kommunikációra az **axios** csomagot használtam. Ennel segítségével tudom kezeln a hálózaton történő HTTP kéréseket, amelyekhez hozzátarozik az url, body és egyéb paraméterek megadása.

## 4.2. Google Firebase backend architektúra

A **Google Firebase** olyan platform, amely lehetővé teszi a fejlesztők számára, hogy könnyedén építsenek és üzemeltessenek felhőalapú alkalmazásokat. A Firebase architektúrája több szolgáltatásból áll, amelyek együttműködnek, hogy biztosítsák a fejlesztők számára a skálázhatóságot, megbízhatóságot és az alkalmazások széleskörű funkcióit (4.5).

A Google Firebase funkcionalitásainak elérésére léteznek direkt JavaScript alapú kezretrendszerre könyvtárcsomagok. ReactJs-re a *firebase* és annak almoduljai a */storage*, */compat/app*, */compat/auth* vagy a */compat/firestore*, míg NodeJS-ben a *firebase-admin*. Ezekben a modulokban implementálva minden olyan kérés a Firebase felé és a visszakapott adatok deserializálása, amelyek lehetségesek a Goolge szerverek fele kód alapon. Például a **storage** modullal lehetséges a Cloud Storage kezelése és az **auth**-al a felhasználók bejelentkeztetése és az adataik kezelése.



**4.5. ábra.** Firebase Console

### 4.2.1. Firebase Authentication

A rendszerben az egyik központi szerepet játszó szolgáltatás a **Firebase Authentication**. Ennek segítségével a felhasználók regisztrálhatnak, bejelentkezhetnek és hitelesíthetik magukat az alkalmazásban. Ez a szolgáltatás támogatja az email-alapú, és közösségi média alapú hitelesítési módokat is, és lehetővé teszi a felhasználói adatok kezelését. Az F1 Ticket Managerben lehetőség van regisztrálni email cím, Google- vagy Facebook fiók segítségével (4.6). Amennyiben saját email fiókkal történik a regisztráció, szükség van annak vissza igazolására, amely a megadott fiókra érkező levélben megadott linkkel lehetséges.

Sign-in providers		Add new provider
Provider	Status	
Email/Password	Enabled	
Google	Enabled	
Facebook	Enabled	

**4.6. ábra.** Autentikációs módok

A 4.2 fejezetben is említett `/compat/auth` modul által lehetőségem van a rendszer bármely pontján elérni az aktuálisan bejelentkezett felhasználó adatait a `firebase.currentUser`-en keresztül.

### 4.2.2. Cloud Firestore

A második, általam leghasználatabb, funkcionálitás a **Cloud Firestore**. A Firestore egy dokumentum-orientált adatbázis, amely skálázhatóbb és rugalmasabb lehetőségeket

kínál adatok tárolására és lekérdezésére. Firestore használatakor a fejlesztők strukturált gyűjteményeket és dokumentumokat hozhatnak létre, és lehetőségük van komplex lekérdezések végrehajtására és az adatok szűrésére. Ilyen a dokumentumokban tárolom a felhasználói adatokat, amelyek nagyrésze meg is jelenik a felületen, a pályák és jegyek adatait, valamint a rendeléseket (4.7). A pályaadatokat egy JSON fájlban gyűjtöttem össze és abba lehetséges a módosításuk is, majd egy általam írt algoritmust segítségével ebből a JSON adatcsomagból létrehozok egy többrétegű dokumentumrendszer a Firestoreban.

The screenshot shows the Cloud Firestore interface. On the left, there's a sidebar with '+ Start collection' and a list of collections: 'Circuits', 'orders', and 'users'. Below 'Circuits' is a list of document IDs: 7mjct1x0psadhJ0BDiXN, 9pKITwkgw4PrLtcZ8BQe, AHPR9ujZKy8Ge4hWlAoT, AgROgDDY6yBfnRgK0ubE, GH5FI7J8ilsXBpbTXUAD, LLhjrJkJieBPVvkRg5wQF, NRx0ajJ9ihXFP0RcaUId, OpyodY1aZ9WWZuLKzI6, PCyhWqMdQbEhPR3MjbYj, QZAI9HbnIFYVks97P9Pk, XGSMexCKwog2Gr8d3M0h, ibsq2HI0U5dfV4BcM3I5, and mRbP3zYdfnxXPWYRGuJd. The main area shows a document named '4NsHMTPsDt16KzhtbmHq' with fields: 'CircuitName' (Marina Bay Street Circuit), 'Country' (Singapore), 'Date' (with 'end' and 'start' properties), 'Locality' (Marina Bay), 'circuitId' (marina\_bay), and a 'packages' array containing one element: 'name' (HERO\_10), 'period' (THURSDAY - SUNDAY), and 'price' (10).

**4.7. ábra.** Cloud Firestore struktúrája

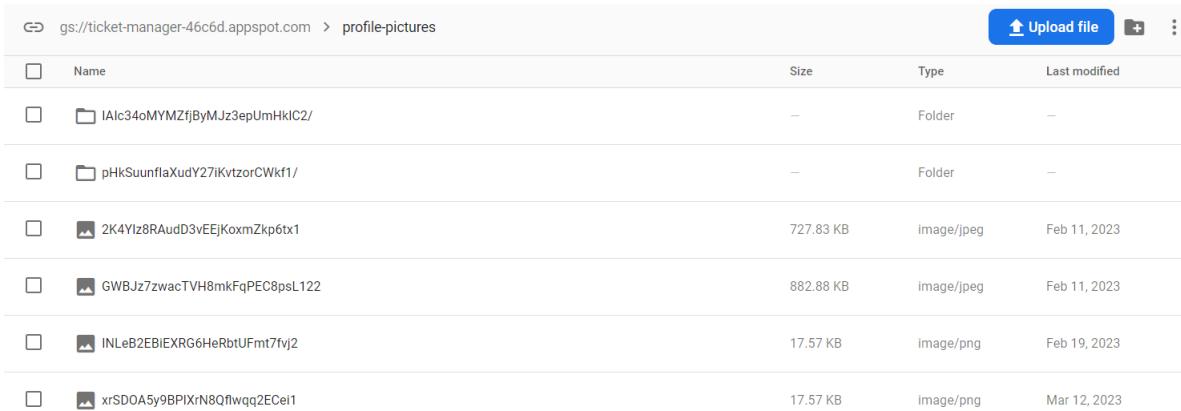
A adatbázis megfelelő biztonságát a **szabályok** (Rules) beállításával lehet megadni. Ezen szabályok tudják biztonsítani, hogy az adott dokumentumokhoz milyen feltételek alapján lehet hozzáférni. Ilyen például, hogy egy felhasználó adataihoz mindig csak a *Firebase Authentication* által bejelentkezett felhasználó férjen hozzá az egyedi azonosító alapján (auth.uid) (4.1).

```
match /databases/{database}/documents {
  match /users/{userId} {
    // Allow users to only read or write their own documents
    allow read, write: if request.auth.uid == userId;
  }
}
```

**4.1. kódrészlet.** Firestore szabályok.

### 4.2.3. Firebase Storage

A **Firebase Storage** szolgáltatás lehetővé teszi a felhasználói fájlok, például képek, videók vagy hangfájlok tárolását és kezelését. A Firebase Storage nagyobb méretű fájlok tárolására és letöltésére specializálódott, amelyeket a Google Cloudban tárolja. Az én alkalmazásomban a feltöltött profilképek kerülnek a Storage-ban tárolásra (4.8).



The screenshot shows the Firebase Storage interface. At the top, there's a header with the URL 'gs://ticket-manager-46c6d.appspot.com' and a path 'profile-pictures'. On the right side of the header are buttons for 'Upload file', '+', and '⋮'. Below the header is a table with columns: 'Name', 'Size', 'Type', and 'Last modified'. The table lists several items:

Name	Size	Type	Last modified
IAlc34oMYMZfjByMJz3epUmHkIC2/	—	Folder	—
pHkSuunflaXudY27iKvtzorCWkf1/	—	Folder	—
2K4YIz8RAudD3vEEjKoxmZkp6tx1	727.83 KB	image/jpeg	Feb 11, 2023
GWBJz7zwacTVH8mkFqPEC8psL122	882.88 KB	image/jpeg	Feb 11, 2023
INLeB2EBIEXRG6HeRbtUFmt7fvj2	17.57 KB	image/png	Feb 19, 2023
xrSDOA5y9BPIXrN8Qflwqq2ECel1	17.57 KB	image/png	Mar 12, 2023

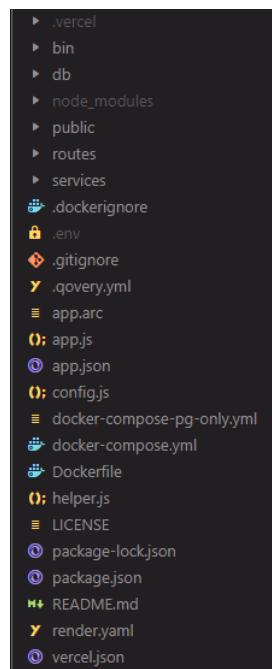
**4.8. ábra.** Firebase Storage struktúrája

### 4.3. Vercel backend architektúra

A **Vercel** által kiszolgált egyik backend szerver, amely hasonlóan a frontendhez (4.1) JavaScript nyelven van írva **NodeJS** keretrendszerben (4.9). A szerver kiinduló pontja az *app.js*, amely tartalmazza az alapértelmezett konfigurációkat a HTTP hívások kezelésére és a végpontokat tartalmazó fájlok helyét. A szerver textbfExpress alapú.

A Vercel-en több sablon alapján lehet szervert létrehozni. Én egy egy third-party repository segítségével hoztam létre, amely megtalálható a GitHubon a **geshan** nevű felhasználó jóvoltából [Git].

Az Express.js, vagy egyszerűen csak Express, egy backend webalkalmazás-keretrendszer RESTful API-k létrehozásához a Node.js segítségével, amelyet ingyenes és nyílt forráskódú szoftverként adnak ki az MIT-licenc alatt [Wikb].



**4.9. ábra.** Vercel mappa struktúrája

A **routes** mappa tartalmazza a alkalmazás működéséhez szükséges JSON, CSS és egyéb JS fájlok. Az én esetben itt található az *index.js* és a *quotes.js* fájlok ([4.10](#)). Az *index.js* tartalmazza a szerver végpontjait és az üzleti logikát. A *quotes.js* pedig a Google Firebase szerverrel való kapcsolat kialakításához szükséges konfigurációkat, amelyek lefutnak a szerver indulásakor.



**4.10. ábra.** Vercel *Routes* mappa

A titkosítási és kulcscsere algoritmusok használatára a rendre a **crypto-js** és **node-rsa** JavaScript könyvtárcsomagokat használom. A node-rsa egy optimális és kényelmes használatot nyújt az RSA publikus kulcscsere protokoll beépítéséhez a rendszerbe. Ez egy emelt szintű biztonságot biztosít a felhasználók számára. Azt is figyelembe vettetem, hogy ez a kulcscsere csak akkor biztonságos, ha a kulcsokat is biztonságos módon tároljuk. Erre azt a megoldást alkalmaztam, hogy a publikus és privát kulcsokat is a Google Firestoreban tárolom, amelyre olyan szabályok (rules) vannak beállítva, hogy minden csatlakozó csak az adott autentikált felhasználó férjen hozzá, ahogyan arról már említést tettem a [4.2](#) fejezetben a [4.1](#) kód részlet segítségével.

## 5. fejezet

# Tervezés és megvalósítás

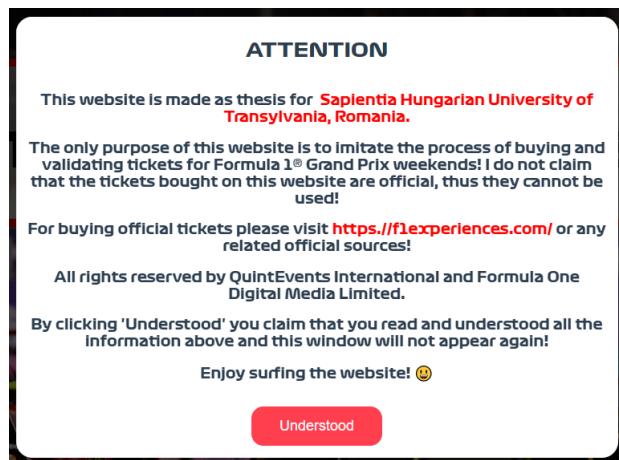
A rendszer architektúrájának tervezése során figyelembe vettetem a 2.6. és 2.7. fejezetekben kitűzött célokat. Az eredeti tervek elkészültével kezdetét vette a megvalósítás. Az implementáció során felmerültek olyan akadályok és újabb ötletek, amelyek kisebb-nagyobb mértékben befolyásolták a terveket. Erre számítottam és éppen ezért úgy próbáltam tervezni, hogy dinamikusan módosíthatóak legyenek és bővítés vagy módosítás esetén. A tervezési és megvalósítási folyamat hónapokat vedd igénybe annak érdekében, hogy minden jelentkező problémát és az újabb ötleteket legyen idő átgondolni.

### 5.1. Az alkalmazás frontendje

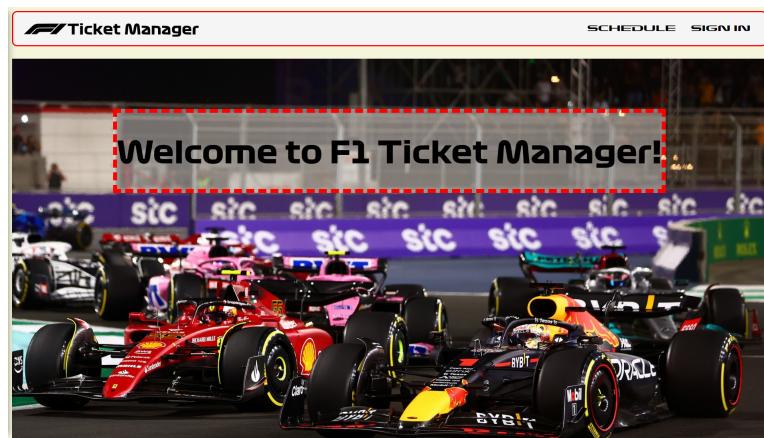
#### 5.1.1. Kezdőoldal (Homepage)

Tekintettel arra, hogy az alkalmazás bizonyos funkcionálitásai használhatóak bejelentkezés nélkül is, a kezdőoldal egy üdvözlő oldal egy navigációs menüvel az oldal tetején. Amennyiben egy eszközről első alkalommal lépik az oldalra egy felhasználó, akkor egy információs ablakkal találkozik, amely tartalmazza az alkalmazás célját, a hivatalos forrásokat a jegyvásárlásra és ezeknek a jogtulajdonosait (5.1).

Annak érdekében, hogy ez az ablak csak egyszer jelenleg meg **sütiket** (**Cookies**) használjon. Ezek a sütiket a böngésző tárolja a felhasználó eszközén és a weboldal betöltésekor a böngésző betölti az elmentett sütiket, ha vannak. Ezek kulcs-elem párok, amely értelmében egy tetszőleges nevű kulcs alatt tudok elmenteni majdnem bármilyen adatot. Ha a felhasználó rágattnak a **Understood** gombra, akkor egy **modalAccepted** kulcs alatti sütiben eltárolom a **true** értéket és amikor betöltődik az alkalmazás ezt az értéket ellenőrzöm. Mint említettem ezek a sütik lokálisan vannak tárolva a felhasználónál és ha a felhasználó kitörli azokat, amelyek ehhez az oldalhoz tartoznak, akkor egyértelműen újra elő fog jönni egy az ablak.



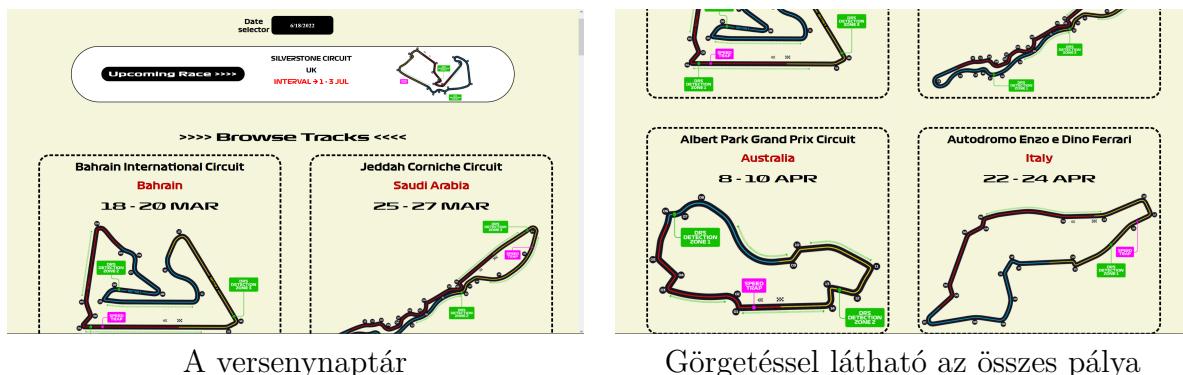
5.1. ábra. Kezdőoldalon felugró ablak



5.2. ábra. Kezdőoldal

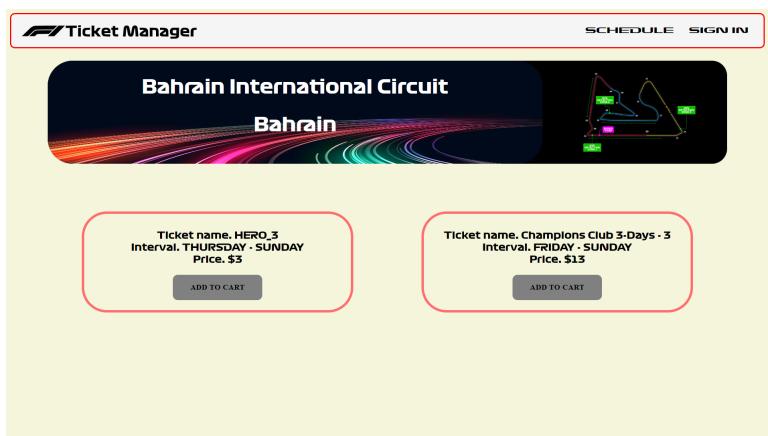
## 5.1.2. Versenynaptár (Schedule)

A **Versenynaptár** oldalon lehet böngészni a 2022-es év versenyeinek listáját a megrendezési sorrendben. Az első kártya, ami megjelenik az oldalon az a következő verseny, amely automatikusan kerül ki az aktuális dátum alapján. Teszt jelleggel van egy dátum kiválasztó gomb is, hogy ki lehessen próbálni, hogy egy adott dátumhoz melyik verseny lesz a legközelebb. Erre a kártyára kattintva meg lehet tekinteni a részleteket. minden kártyán látszanak a legfontosabb információk az adott versenyről, mint a pálya neve, a rendező ország neve, az intervallum, amikor az esemény zajlani fog és a pályarajz (5.3).



5.3. ábra. Versenynaptár

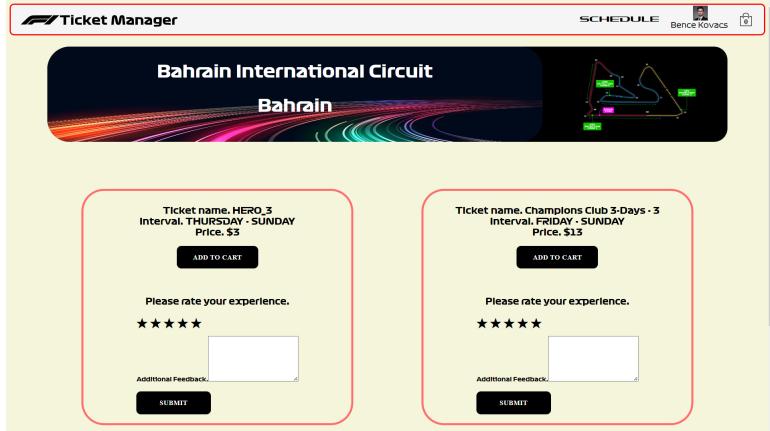
Ha átvisszük az egeret a kártya fölött, akkor megjelenik egy gomb **Browse tickets** szöveggel, amelyre kattintva megtekinthetők a versenyhez tartozó jegy típusok és azok információi (5.4). Ha nincs bejelentkezve a felhasználó, akkor az **Add to cart** gomb szürkén jelenik meg, vagyis nem lehet a kosárba tenni, viszont ha rákattint a felhasználó, akkor a rendszer jelezni fogja egy **alert** (figyelmeztetés) ablakban, hogy autentikáció szükséges, ha ezt a funkciót szeretné használni.



5.4. ábra. Jegy típusok be nem jelentkezett felhasználóval

Abban az esetben, ha a felhasználó be van jelentkezve, lehetőség van a jegyet a kosárba helyezni vagy visszajelzést adni az adott jegyről, élményekről (5.5). A jegyek mennyiségeit a **Checkout** oldalon van lehetősége módosítani a felhasználónak. A rend-

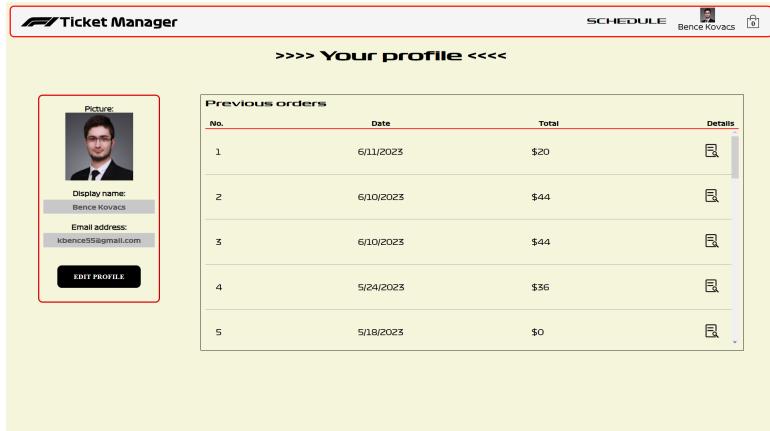
szer ezen része egy továbbfejlesztési terület, hogy adott esetben ellenőrizve legyen, hogy tényleg csak az tudjon visszajelzést adni, akkor vásárolt az adott jegyből.



**5.5. ábra.** Jegy típusok bejelenkezett felhasználóval

### 5.1.3. Profil (Profile)

A **Profil** oldal elérése csak a bejelentkezett felhasználóval lehetséges (5.6). Ezen az oldalon lehetséges a profilkép és a megjelenített név módosítása az **Edit profile** gombra kattintva.



**5.6. ábra.** Saját profil oldal

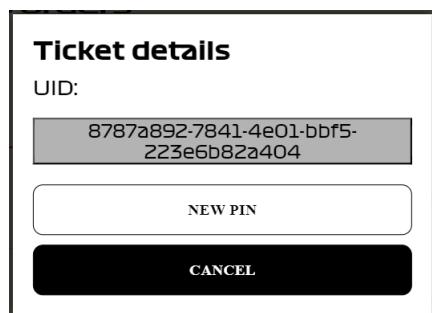
Az oldal jobb részén láthatóak a **rendelési előzmények**. Ezek görgetéssel böngészhetők és a jobb szélén levő ikonra kattintva tekinthetők meg a vásárlás részletei (5.7).

Amennyiben elvisszük valamely elem fölött az egeret megjelenik egy **More details** gomb, amelyre kattintva megtekinthető a jegy egyedi azonosítója (5.8). Ez a funkcionális azért fontos, mivel megtörténhet, hogy a felhasználó elveszíti vagy nem kapja meg a QR kódot emailben, ami ezt az azonosítót tartalmazza. Ha ez bekövetkezne, akkor innen is lehetőség van kimásolni és a megszokott módon azonosítani a PIN kóddal.

Previous orders					
Circuit	Ticket	Quantity	Price	Price Total	Interval
	HERO_17	1	\$17	\$17	THURSDAY - SUNDAY
	Champions Club 3-Days -17	1	\$27	\$27	FRIDAY - SUNDAY
Total \$44					

5.7. ábra. Rendelés részletei

Ezen a felugró ablakon továbbá lehetősége van a felhasználónak egy új PIN kódot megadni, ha a meglévő elfelejtette vagy úgy érzni, hogy valaki megszerezte. Ezután egy újabb felugró ablakban lehetséges a PIN kód megváltoztatása.



**Ticket details**

UID:  
8787a892-7841-4e01-bbf5-223e6b82a404

NEW PIN

CANCEL

5.8. ábra. Jegy részletei

A PIN kód megadásánál kötelezően egy 6 számjegyű számot kell megadni (5.9). Ezt manuálisan is be lehet írni vagy a *pálca* ikonra kattintva generálódik egy véletlenszerű szám. A **Submit** gombra kattintva frissítheti a felhasználó a jegyhez tartozó kódot, ami teljes mértékben felülírja az előzőt, ezért nagyon figyelmes kell lenni a megadásakor, de erre a pirossal írt szöveg is figyelmezteti a vásárlót.



**Please set a PIN code**

Please keep in mind that this PIN code is highly confidential, thus it should NOT be shared with anyone!  
This PIN code CANNOT BE SEEN in the future but you can create a new one at any time in your order details on the profile page!

PIN 

SUBMIT

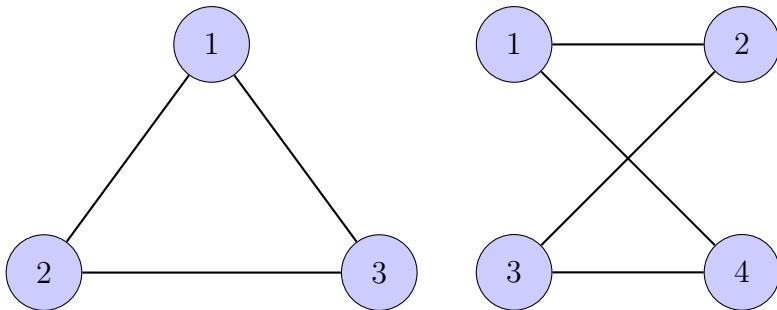
CANCEL

5.9. ábra. PIN kód változtatás ablak

////////— PROGRESS —////////

**Matlab ode45 előnyei:**

- nagyon egyszerű a használata, nem igényel komoly programozási ismereteket
- könnyű beépíteni és összekötni más Matlab programokkal
- a kapott eredményeket mátrix vagy vektor típusokban téríti vissza, ami könnyűvé teszi az eredmények további kezelését
- az eredményeket grafikus felületen azonnal meg tudjuk jeleníteni
- jobb eredményeket produkált, mint az Odeint
- jól dokumentált, sok példa van a használatára



**5.10. ábra.** Egyszerű gráf TIKZ segítségével

#### Matlab ode45 hátrányai:

- komoly hátránya az Odeinttel szemben, hogy **fizetni kell a használatáért**
- nagyon sok memóriát használ, komolyan igénybeveszi a számítógép erőforrásait

#### Odeint előnyei:

- ingyenes és nyílt forráskódú, használható személyi és kereskedelmi célokra egyaránt
- nagyon rugalmas, absztrak, így könnyedén változtatható a bemeneti adatok típusa vagy struktúrája
- C++ nyelven íródott, támogatva a modern programozási technológiákat (Generikus programozás, Template Metaprogramming)

#### Odeint hátrányai:

- használata nehezebb, mint a Matlab ode45 programé, szükséges a C++ programozási nyelv ismerete
- a kapott eredményekkel nem olyan könnyű bánni, mint a Matlab esetében
- absztraktsága miatt nehéz a felmerülő problémákat megoldani
- dokumentáltsága jóval szegényesebb, mint a Matlabé

## 5.2. Saját szoftverek esetén

Saját szoftverek esetében sikerült négy különböző technológia segítségével megvalósítani a Dorman-Prince differenciálegyenlet megoldó algoritmust (lásd ??-. alfejezetet). A felhasznált technológiák: Matlab, Java, C++ és Android voltak.

Ezen technológiák közül az Androidos szoftverrel kapcsolatban már előzetes félelmek voltak, mivel azt feltételeztük, hogy bármennyire is fejlettek napjainkban a mobileszközök, mégis hardveresen nem lesznek elegek ahhoz, hogy versenybe tudjanak szállni a számítógépekkel. Néhány teszt után feltételezések beigazolódtak, láthatjuk a ??-. alfejezet teszteseteiben is, hogy az Android szoftver mennyire gyengén teljesített (mind a Motorola Moto E2, mind a Samsung Galaxy Core Prime esetében). Összehasonlítva a többi algoritmussal az átlagidőket nézve 150 – 160 - szor lassabb a Javanál és 300 – 350 - szer a C++ - nál! Tehát azt a következtetést vonhatjuk le, hogy nem érdemes mobileszközökön differenciálegyenleteket oldani, mivel túlságosan nagy a hardver igénye és egyelőre ezen a téren nem képesek tartani a lépést a számítógépekkel.

A további három technológia közül (Matlab, Java, C++) meglepő módon itt is a Matlab teljesített a legjobban, igaz, hogy ebben az esetben már nem használtuk az ode45 programot, hanem megírtam én a saját függvényemet. Ez a teljesítményen is meglátszott, mert az általam írt függvény nem tudott jobban teljesíteni a tesztek alatt, mint az ode45. Mindezek ellenére 35 – 40 - szer gyorsabb volt a Javanál és megközelítőleg 15 – 20 - szor gyorsabb a C++ - nál.

A Java és C++ szoftvereket összehasonlítva elmondhatom, hogy az esetek többségében a C++ körülbelül 2 - szer volt gyorsabb a Javanál, ami megfelel az előzetes elvárásoknak.

Amit nagyon fontosnak tartok kihangsúlyozni, hogy az általam megvalósított C++ szoftver a tesztek során nagyon jól teljesített, felvette a versenyt az Odeint könyvtárral és az elért átlagok is csak nagyon kicsivel maradnak el az Odeint által produkált eredményektől (??-. alfejezet).

Továbbá megvalósítottam az Euler és Runge-Kutta módszerek párhuzamosított változatait is CUDA technológia segítségével. Ebben az esetben a tesztek azt mutatták, hogy az Euler módszer esetében többe kerül a sok CPU és GPU memória közötti másolás művelete, mint amennyit nyerünk a számítások elvégzése során. Tehát ebben az esetben ez a fajta párhuzamosítási megközelítés nem éri meg. Ezzel ellentétben a Runge-Kutta módszer esetében a megközelítés eredményesnek bizonyult abban az esetben, ha az egyenletek száma nagy és a lépésköz kicsi. A ?? alfejezetben láthattuk, hogy abban az esetben ha az egyenletek száma  $n = 10$  és a lépésköz  $h = 0.001$ , a GPU-n megközelítőleg 5 és fél perccel hamarabb lefutott az algoritmus, mint a CPU-n. Ezzel a párhuzamosítási módszerrel nem tudtuk kihasználni a videókártya által nyújtott maximális számítási kapacitást, de így is jelentős különbséget sikerült elérni a futási időket nézve, lásd [K08].

## 5.3. Összességében

A fentieket összegezve elmondhatom, hogy megéri előre megírt szoftvereket vagy könyvtárakat használni differenciálegyenletek megoldására. Nagyon megkönnyíthetik az eltünetet egyszerűségükkel és nagy teljesítményükkel. Viszont fontos elmondani, hogy használatuk problémákkal is járhat, például fizetni kell értük vagy nem lehet belenyúlni az

algoritmusokba kedvünk szerint, esetleges fellépő hibák esetén nagyon nehéz lenyomozni a hiba forrását (vagy szinten lehetetlen).

Saját algoritmusok terén bátran elmondhatom, hogy megéri a C++ technológiát választani és ezen a vonalon továbbhaladni egy esetleges saját könyvtár megírása, megvalósítása felé. Láthattuk, hogy az általam megírt C++ szoftver is felvette a versenyt az Odeint könyvtárral, ami szintén C++ technológiát alkalmaz, láasd [[Ant07](#)].

Egy másik vonal, amit érdemes sokkal jobban felderíteni az a CUDA technológiával és grafikus kártyával történő differenciálegyenlet megoldása. Láthattuk, hogy egy kis párhuzamosítás is jelentős időbeli különbséget jelenthet bizonyos algoritmusok esetében. Annak tudatában is, hogy a differenciálegyenletek megoldása nem a legjobban adatpárhuzamosítható feladatok közé sorolható azt mondjam, hogy megéri ezzel a technológiával foglalkozni. Ennek kapcsán a legnagyobb motiváció számomra a jövőre nézve a parciális differenciálegyenletek párhuzamosításának megvalósítása és tanulmányozása, mivel ezeknél az egyenleteknél jobban ki lehet használni a videókártya ráccsal szerkezetét.

# 6. fejezet

## Szoftver

---

### 1. algoritmus: Euclidean Algorithm

---

**Input:** Integers  $a \geq 0$  and  $b \geq 0$

**Output:** GCD of  $a$  and  $b$

```
1 while  $b \neq 0$  do
2    $r \leftarrow a \bmod b;$ 
3    $a \leftarrow b;$ 
4    $b \leftarrow r;$ 
5 end
```

---

---

### 2. algoritmus: How to write algorithms

---

**Data:** this text

**Result:** how to write algorithm with L<sup>A</sup>T<sub>E</sub>X2e

```
1 initialization;
2 while not at end of this document do
3   read current;
4   if understand then
5     go to next section;
6     current section becomes this one;
7   else
8     go back to the beginning of current section;
9   end
10 end
```

---

---

### 3. algoritmus: IntervalRestriction

---

**Data:**  $G = (X, U)$  such that  $G^{tc}$  is an order.

**Result:**  $G' = (X, V)$  with  $V \subseteq U$  such that  $G'^{tc}$  is an interval order.

```
1 begin
2   |   V ← U
3   |   S ← ∅
4   |   for  $x \in X$  do
5   |   |   NbSuccInS( $x$ ) ← 0
6   |   |   NbPredInMin( $x$ ) ← 0
7   |   |   NbPredNotInMin( $x$ ) ←  $|ImPred(x)|$ 
8   | end
9   | for  $x \in X$  do
10  | | if  $NbPredInMin(x) = 0$  and  $NbPredNotInMin(x) = 0$  then
11  | | | AppendToMin( $x$ )
12  | | end
13  | end
14 while  $S \neq \emptyset$  do
15   | remove  $x$  from the list of  $T$  of maximal index
16   | while  $|S \cap ImSucc(x)| \neq |S|$  do
17   | | for  $y \in S - ImSucc(x)$  do
18   | | | { remove from  $V$  all the arcs  $zy$  : }
19   | | | for  $z \in ImPred(y) \cap Min$  do
20   | | | | remove the arc  $zy$  from  $V$ 
21   | | | | NbSuccInS( $z$ ) ← NbSuccInS( $z$ ) - 1
22   | | | | move  $z$  in  $T$  to the list preceding its present list
23   | | | | { i.e. If  $z \in T[k]$ , move  $z$  from  $T[k]$  to  $T[k - 1]$  }
24   | | | end
25   | | | NbPredInMin( $y$ ) ← 0
26   | | | NbPredNotInMin( $y$ ) ← 0
27   | | | S ←  $S - \{y\}$ 
28   | | | AppendToMin( $y$ )
29   | | end
30   | | RemoveFromMin( $x$ )
31   | end
32   | end
33   | end
34 end
35 end
```

---

## 6.1. A szoftver bemutatása

Az általam írt szoftver egy **Java nyelven**, **NetBeans IDE 8.0.1** fejlesztői környezetben írt asztali alkalmazás, amelynek fő funkcionálitása a kezdetiérték-probléma típusú differenciálegyenletek numerikus megoldása és ezen megoldások grafikus felületen való ábrázolása. A fő funkcionálitás mellett a szoftver tartalmaz még két kisebb funkcionálitást is, ezek közül az egyik a kétdimenziós függvényábrázolási lehetőség, a másik pedig a háromdimenziós függvények megjelenítésének lehetősége.

Az szoftver a differenciálegyenletek megoldásához a 3. fejezetben leírt numerikus eljárásokat alkalmazza.

A grafikus felhasználói felület megalkotásához a **Swing** (Java) komponens készletet használtam. A Swing használatával célom az volt, hogy egy felhasználóbarát és könnyen kezelhető felületet hozzak létre, amelyen a felhasználó könnyedén eligazodhat. Továbbá e komponenskészlet használata mellett szól az is, hogy a későbbiekben bemutatásra kerülő könyvtárak, melyek az ábrázolás megvalósítására használtam szintén Swing komponensekkel vannak megvalósítva.

Felhasználói felület, valamint a három funkcionálitás bemutatása képekben:

---

### 4. algoritmus: Switch használata

---

```
1 switch order do
2   case bloody mary do
3     Add tomato juice;
4     Add vodka;
5     break;
6   case hot whiskey do
7     Add whiskey;
8     Add hot water;
9     Add lemon and cloves;
10    Add sugar or honey to taste;
11    break;
12  otherwise do
13    | Serve wine;
14  end
15 end
```

---

## 6.2. A szoftver megírásához használt könyvtárak

A szoftver elkészítésénél szükségem volt néhány előre megírt osztálykönyvtárra, amelyek megkönnyítették a munkámat. Ezekről tudni kell, hogy nyílt forráskódúak, tehát bárki számára elérhetőek az interneten, továbbá azt is, hogy ezek is Java nyelvben íródottak, hasonlóan, mint az általam írt alkalmazás. A továbbiakban szeretném bemutatni ezeket a könyvtárakat és azt, hogy mire- és hogyan használtam fel őket.

- JMathPlot (<https://sites.google.com/site/mulabsltd/products/jmathplot>):
  - Java könyvtár, amelyet interaktív megjelenítésre, ábrázolásra fejlesztettek

- gyors és könnyű utat biztosít tudományos adatok megjelenítésére Swing komponensek segítségével (nem használ OpenGL-t)
- az általa biztosított saját komponenseket úgy lehet használni, mint bármely más Swing komponenst
- a számomra legfontosabb tulajdonsága az, hogy két- és háromdimenziós ábrázolási lehetőséget biztosít, ezt használtam fel az alkalmazásomban
- JMathArray (<https://sites.google.com/site/mulabsltd/products/jmatharray>):
  - olyan Java könyvtár, amely alapvető matematikai, lineáris algebrai műveleteket biztosít számunkra
  - a könyvtár által biztosított statikus metódusok tömbökre alkalmazhatóak
  - a szoftverben arra használtam, hogy egy megadott intervallum két végpontja között egy bizonyos lépésközzel haladva egy tömböt tudjak feltölteni (inkrementálás)
- JEP (<http://www.cse.msu.edu/SENS/Software/jep-2.23/doc/website/>):
  - szintén egy Java könyvtár, amelyet különböző elemzésekre és kiértékelésekre fejlesztettek
  - segítségével egy szövegként (sztring-ként) megadott kifejezést könnyedén kiértekelhetünk, elvégezhetünk
  - a szövegként megadott kifejezésből a háttérben egy kifejezésfát épít fel, majd a későbbiekben ennek a fának a segítségével dolgozik
  - emellett sok általános matematikai függvény és konstans is bele van építve, amiket szintén könnyedén elérhetünk
  - az általam fejlesztett szoftverben a függvények sztringként adhatók meg egy beviteli mezőn keresztül, a JEP könyvtárat ezen függvények „parszolására” használtam fel

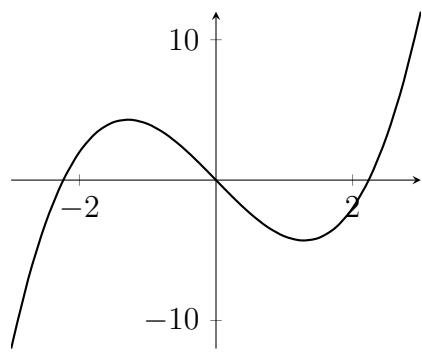
## **6.3. Diagramok**

### **6.3.1. Use Case diagram**

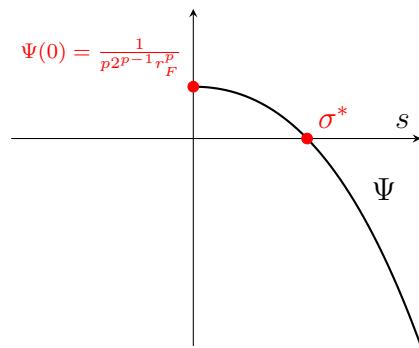
### **6.3.2. Osztálydiagram**

A szoftver szerkezetileg két nagyobb részből (csomagból) áll, az egyik a felhasználói felület megalkotásához szükséges osztályokat tartalmazza, a másik pedig a differenciál-egyenletek megoldására szolgáló osztályokat és a parszer osztályt, mely egy sztringként megadott függvény kiértékelésére szolgál.

Az implementációnál a felhasználói felület elemeit tartalmazó csomagot „View”-nak, a numerikus módszereket és a parszert tartalmazó csomagot „Model”-nek neveztem, emellett a 6.7-es ábrán megjelenik egy harmadik csomag is, amely tartalmazza a „MainClass”-t és egyben a main() metódust is. Az alábbi két diagramon láthatjuk a felsorolt csomagokat és a bennük lévő osztályokat, illetve a köztük lévő kapcsolatokat.

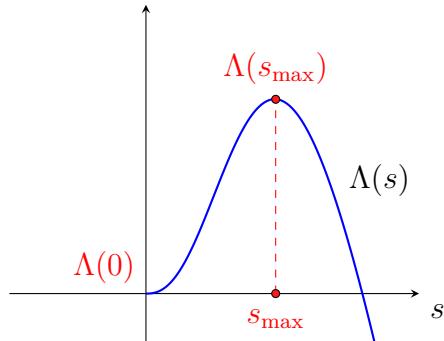


**6.1. ábra.** Az  $x^3 - 5x$  függvény grafikus képe PGFPLOT-al

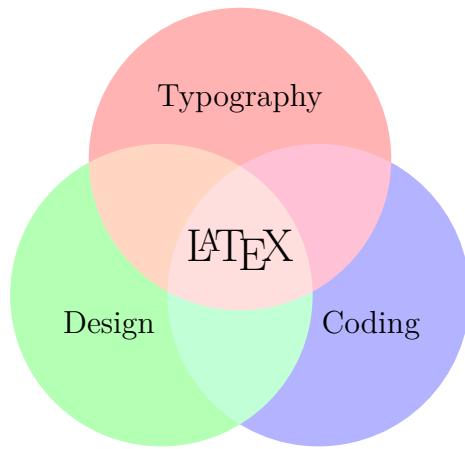


**6.2. ábra.** A  $\Psi$  grafikus képe

### 6.3.3. Szekvencia diagram



6.3. ábra. A  $\Lambda(s)$  grafikus képe



6.4. ábra. Venn diagram TIKZ segítségével

**6.3.1. Definíció.** Legyen  $(X, d)$  és  $(Y, \rho)$  két metrikus tér, legyen  $T : X \rightarrow Y$  egy leképezés. Azt mondjuk, hogy a  $T$  leképezés Lipschitz tulajdonságú, ha létezik egy olyan  $L > 0$  szám amelyre

$$\rho(Tx, Ty) \leq Ld(x, y) \quad \forall x, y \in X.$$

Az  $L$  számot Lipschitz állandónak nevezzük.

Ha  $T : X \rightarrow Y$  leképezés Lipschitz tulajdonságú, és az  $L < 1$  akkor a  $T$  operátort **kontrakciónak** nevezzük. Azt mondjuk, hogy  $x^* \in X$  fixpontja a  $T$  operátornak ha

$$Tx^* = x^*.$$

**6.3.1. Tétel.** *Banach féle fixponttételet Legyen  $(X, d)$  teljes metrikus tér és  $T : X \rightarrow X$  leképezés egy kontrakció az  $L < 1$  állandóval. Ekkor igazak a következő állítások:*

1.  *$T$ -nek egy és csakis egy  $x^*$  fixpontja.*
2. *Bárholg választunk meg egy  $x_0 \in X$  elemet, a  $x_{k+1} = Tx_k$  sorozat konvergens és  $Tx_k \rightarrow x^*$ , ahol  $k$  természetes szám.*
3. *Igaz, hogy*

$$d(x_k, x^*) \leq \frac{L^k}{1-L} d(x_0, Tx_0).$$

# Összefoglaló

Dolgozatomban differenciálegyenletek megoldásával foglalkoztam, amelyet különböző programozási technológiák segítségével valósítottam meg. Először ismertettem a differenciálegyenletek numerikus megoldásának elméleti alapjait, majd megvizsgáltunk és levezettünk három numerikus módszert az Euler-, a Runge-Kutta és a Dormand-Prince módszereket. Ezek közül a mai technológiákban leginkább használatos Dormand-Prince algoritmus esetében megnéztük, hogy milyen szoftverekben találhatjuk meg, mint alapértelmezett differenciálegyenlet megoldó. A továbbiakban ismertettem két modellt, a leukémia betegség alap modelljét és a hullámmozgás modelljét, ezzel is kihangsúlyozva a téma fontosságát, hogy mennyire fontos az időtényező bizonyos problémák egyenleteinek medoldásánál. Ezek után részletesen is megnéztük, hogy milyen szoftvereket alkalmaztam és alkottam a differenciálegyenletek és rendszerek megoldására. A szoftvereket két kategóriába osztottottuk fel, az első a már létező szoftverek kategóriája, a másik pedig az általam megvalósított szoftverek csoportja volt. Az első kategóriában ismertettem két technológiát, a Matlab által nyúltott ode45 beépített megoldót és a Boost könyvtárcsomagban található Odeint nevű könyvtárat. Az általam írt szofverek csoportjában négy megvalósítást mutattam be ezek a Matlab, Java, C++ és Android technológiák segítségével készültek. Emellett megnéztük, hogy mennyire hatékonyan lehet párhuzamosítani a differenciálegyenletek megoldását CUDA technológia segítségével és a grafikus kártyát (GPU-t) felhasználva. Végül kiértékeltük a tesztelés során kapott eredményeket és összehasonlítottuk a különböző programokat, kiemelve azok erősségeit és gyengéit. Majd levontuk a következetéseket, hogy melyik technológia irányában érdemes tovább haladni és melyik az, amelyikkel nem éri meg foglalkozni.

Jövőbeli terveimet illetően szeretném jobban elmerülni a GPU-n történő differenciálegyenletek megoldásának módszereiben, valamint ezek alkalmazását kipróbálni és tanulmányozni a parciális differenciálegyenletek területén (PDE). Továbbá érdemesnek tartom a C++ szoftver továbbfejlesztését és egy differenciálegyenlet megoldó könyvtár megalkotását, amely ingyenesen használható és nyílt forráskódú lenne.

# Köszönetnyilvánítás

Ez nem kötelező, akár törölhető is. Ha a szerző szükségét érzi, itt lehet köszönetet nyilvánítani azoknak, akik hozzájárultak munkájukkal azzal, hogy a hallgató a szakdolgozatban vagy diplomamunkában leírt feladatokat sikeresen elvégezze. A konzulensnek való köszönetnyilvánítás sem kötelező, a konzulensnek hivatalosan is dolga, hogy a hallgatót konzultálja.

# Jelölések

Rövidítés	Angol megnevezés	Magyar megnevezés
F1	Formula One	Formula-1/Forma-1
FIA	International Automobile Federation	Nemzetközi Automobil Szövetség
F1TM	F1 Ticket Manager	F1 Ticket Manager
EDI	Electronic Data Interchange	Elektronikus Adatcsere
PDF	Portable Document Format	Hordozható Dokumentum Formátum
QR Code	Quick Response Code	Quick Response-kód (=gyors válasz)
E2EE	End-To-End Encryption	Végpontok Közötti Titkosítás
PIN	Postal Index Number	Postai Indexszám
DOM	Document Object Model	Dokumentum Objektum Modell
OSS	Open-Source Software	Nyílt Forráskódú Szoftver
NoSQL	Not only Structured Query Language	Nem csak Strukturált Lekérdezőnyelv
API	Application Programming Interface	Alkalmazásprogramozási Felület
CDN	Content Delivery Network	Tartalomelosztó Hálózat
IDE	Integrated Development Environment	Integrált Fejlesztői Környezet
DBMS	Database Management Systems	Adatbázis-kezelő Rendszer (ABKR)
UI	User Interface	Felhasználói Felület
UX	User Experience	Felhasználói Élmény
HTML	HyperText Markup Language	Hiperszöveges Jelölőnyelv
CSS	Cascading Style Sheets	Lépcsőzetes Stíluslapok
HTTP	HyperText Transfer Protocol	Hiperszöveges Szállítási Protokoll
BCM	Block Cipher Mode	Blokktitkosítási Mód
AES	Advanced Encryption Standard	-
RSA	Rivest–Shamir–Adleman	-
REST	Representational State Transfer	-
MIT	Massachusetts Institute of Technology	Massachusettsi Műszaki Egyetem

**6.1. táblázat.** Jelölések

# Ábrák jegyzéke

1.1. Online fizetési rendszerek . . . . .	12
2.1. Webes alkalmazás architekúrája . . . . .	15
2.2. React logó . . . . .	16
2.3. Firebase logó . . . . .	17
3.1. Use case diagram - Nem bejelentkezett felhasználó . . . . .	24
3.2. Use case diagram - Bejelentkezett felhasználó . . . . .	25
3.3. Use case diagram - Adminisztrátor . . . . .	26
3.4. Web böngészők . . . . .	29
4.1. Rendszer architekúra . . . . .	30
4.2. Az npm csomagkezelő . . . . .	31
4.3. Az src mappa struktúrája . . . . .	31
4.4. Komponensek . . . . .	32
4.5. Firebase Console . . . . .	33
4.6. Autentikációs módok . . . . .	33
4.7. Cloud Firestore struktúrája . . . . .	34
4.8. Firebase Storage struktúrája . . . . .	35
4.9. Vercel mappa struktúrája . . . . .	35
4.10. Vercel <i>Routes</i> mappa . . . . .	36
5.1. Kezdőoldalon felugró ablak . . . . .	38
5.2. Kezdőoldal . . . . .	38
5.3. Versenynaptár . . . . .	39
5.4. Jegy típusok be nem jelentkezett felhasználóval . . . . .	39
5.5. Jegy típusok bejelenkezett felhasználóval . . . . .	40
5.6. Saját profil oldal . . . . .	40
5.7. Rendelés részletei . . . . .	41
5.8. Jegy részletei . . . . .	41
5.9. PIN kód változtatás ablak . . . . .	41
5.10. Egyszerű gráf TIKZ segítségével . . . . .	43
6.1. Az $x^3 - 5x$ függvény grafikus képe PGFPLOT-al . . . . .	51
6.2. A $\Psi$ grafikus képe . . . . .	51
6.3. A $\Lambda(s)$ grafikus képe . . . . .	52
6.4. Venn diagram TIKZ segítségével . . . . .	52
F.1.1A TeXstudio L <sup>A</sup> T <sub>E</sub> X-szerkesztő . . . . .	60

# Táblázatok jegyzéke

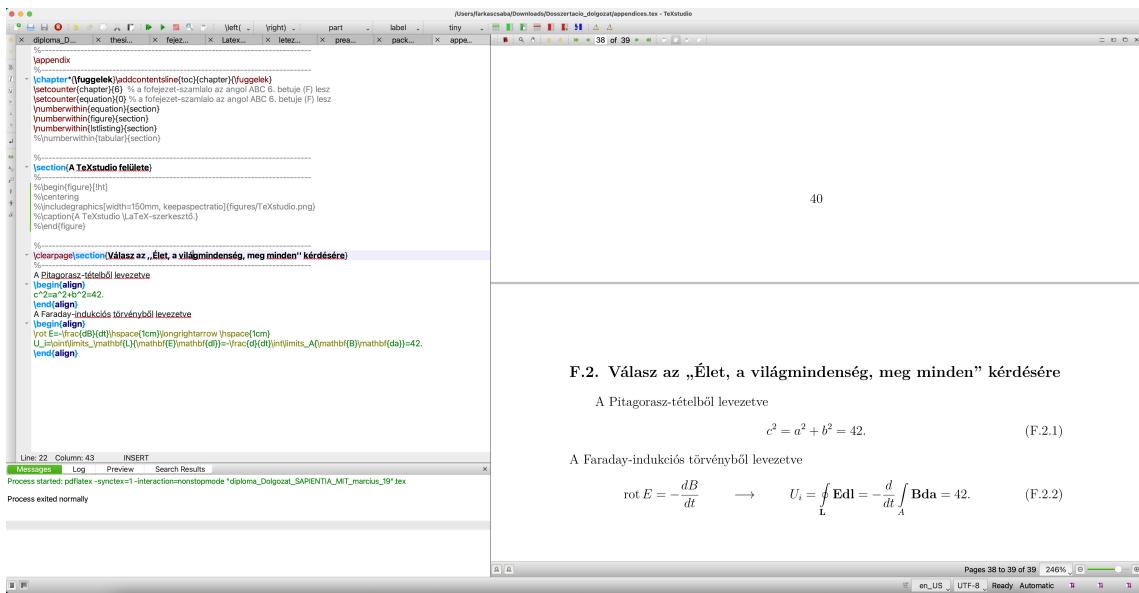
6.1. Jelölések . . . . .	56
--------------------------	----

# Irodalomjegyzék

- [Ant07] Margit Antal. Toward a simple phoneme based speech recognition system. *Stud. Univ. Babeş-Bolyai Inform.*, 52(2):33–48, 2007.
- [Git] Github.com - geshan. <https://github.com/geshan/nodejs-posgresql>.
- [K08] Zoltán Kátai. Dynamic programming as optimal path problem in weighted digraphs. *Acta Math. Acad. Paedagog. Nyhazi. (N.S.)*, 24(2):201–208, 2008.
- [Wika] Wikipedia - the free encyclopedia. [https://en.wikipedia.org/wiki/Npm\\_\(software\)](https://en.wikipedia.org/wiki/Npm_(software)).
- [Wikb] Wikipedia - the free encyclopedia. <https://en.wikipedia.org/wiki/Express.js>.

# Függelék

## F.1. A TeXstudio felülete



F.1.1. ábra. A TeXstudio L<sup>A</sup>T<sub>E</sub>X-szerkesztő.

### F.2. Válasz az „Élet, a világmindenség, meg minden” kérdésére

A Pitagorasz-tételből levezetve

$$c^2 = a^2 + b^2 = 42. \quad (\text{F.2.1})$$

A Faraday-indukciós törvényből levezetve

$$\text{rot } E = -\frac{dB}{dt} \quad \rightarrow \quad U_i = \oint_{\text{L}} \mathbf{E} \cdot d\mathbf{l} = -\frac{d}{dt} \int_A \mathbf{B} \cdot d\mathbf{a} = 42. \quad (\text{F.2.2})$$

## F.2. Válasz az „Élet, a világmindenség, meg minden” kérdésére

A Pitagorasz-tételből levezetve

$$c^2 = a^2 + b^2 = 42. \quad (\text{F.2.1})$$

A Faraday-indukciós törvényből levezetve

$$\operatorname{rot} E = -\frac{dB}{dt} \quad \longrightarrow \quad U_i = \oint_{\mathbf{L}} \mathbf{E} d\mathbf{l} = -\frac{d}{dt} \int_A \mathbf{B} d\mathbf{a} = 42. \quad (\text{F.2.2})$$