

**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  
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**

UNIVERSITATEA „SAPIENTIA” din CLUJ-NAPOCA  
Facultatea de Științe Tehnice și Umaniste din Târgu Mureș  
Programul de studii: Informatică

Viza facultății:

### LUCRARE DE DIPLOMĂ

**Coordonator științific:**

dr. Márton Gyöngyvér

**Îndrumător:** Györfi Ágnes

Candidat: Kovács Bence

Anul absolvirii: 2023

**a) Tema lucrării de licență:**

F1 Ticket Manager - Gestiunea ticketelor

**b) Problemele principale tratate:**

Studiul securității sistemelor de plată online

Studiul protocoalelor de siguranță

Documentarea adecvată a stadiilor de proiectare a aplicațiilor

**c) Desene obligatorii:**

Diagrame de proiectare pentru aplicația software realizată.

**d) Softuri obligatorii:**

Un sistem de plată online

**e) Bibliografia recomandată:**

Npmjs.com - node-rsa. <https://www.npmjs.com/package/node-rsa>.

Márton Gyöngyvér. Kriptográfiai alapismeretek. Scientia, 2008.

**f) Termene obligatorii de consultații:** săptămânal, preponderent online

**g) Locul și durata practicii:** Universitatea „Sapientia” din Cluj-Napoca,  
Facultatea de Științe Tehnice și Umaniste din Târgu Mureș, sala / laboratorul 414

Primit tema la data de: 20.06.2022

Termen de predare: 02.07.2023

Semnătura Director Departament

Semnătura responsabilului  
programului de studiu

Semnătura coordonatorului

Semnătura candidațului

## Declarație

Subsemnatul/a **KOVACS BENCE**, 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 / TÂRGOU MUREȘ**

Data: **30.05.2023**

Absolvent

Semnătura.....**Kovacs**.....

# 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 eseményeken való részvételt, hiszen a vásárlóknak nem kell hosszú sorokban állniuk a jegypénztárnál.

Az F1 Ticket Manager alkalmazás célja egy jegyvásárlási platform biztosítása Forma-1-es versenyekre. Továbbá a vásárlási és beléptetési folyamatok biztonságosabbá tétele, amelyet kétlépcsős jegy hitelesítéssel valósítottam meg.

# 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 confortabil și rapid bilete 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 plăti 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.

Aplicația F1 Ticket Manager are ca scop furnizarea unei platforme de achiziționare a biletelor pentru cursele de Formula 1. În plus, am implementat procese de achiziție și acces mai sigure prin autentificarea în două etape a biletelor.

# 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.

The purpose of the F1 Ticket Manager application is to provide a ticket purchasing platform for Formula 1 races. Additionally, I have implemented secure purchasing and admission processes through two-step ticket authentication.

# 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. Alkalmazások és érdekességek . . . . .	18
2.5. RSA kulccsere protokoll . . . . .	19
2.6. Kutatási kérdések . . . . .	20
2.7. Célkitűzések . . . . .	20
<b>3. Rendszerspecifikáció</b>	<b>22</b>
3.1. Rendszerkövetelmények . . . . .	22
3.1.1. Funkcionális követelmények . . . . .	22
3.1.2. Nem funkcionális követelmények . . . . .	26
3.2. Felhasználói követelmények . . . . .	26
<b>4. A rendszer architektúrája</b>	<b>29</b>
4.1. GitHub frontend architektúra . . . . .	30
4.2. Google Firebase backend architektúra . . . . .	31
4.2.1. Firebase Authentication . . . . .	32
4.2.2. Cloud Firestore . . . . .	32
4.2.3. Firebase Storage . . . . .	33
4.3. Vercel backend architektúra . . . . .	34
<b>5. Tervezés és megvalósítás</b>	<b>36</b>
5.1. Az alkalmazás frontendje . . . . .	36
5.1.1. Kezdőoldal (Homepage) . . . . .	36
5.1.2. Versenynaptár (Schedule) . . . . .	38
5.1.3. Bejelentkezés és regisztráció (Sign in and up) . . . . .	39
5.1.4. Profil (Profile) . . . . .	40
5.1.5. Rendelés (Checkout) . . . . .	42
5.1.6. Beléptetés (Scan) . . . . .	46

<b>Összefoglaló</b>	<b>47</b>
<b>Jelölések</b>	<b>48</b>
<b>Ábrák jegyzéke</b>	<b>49</b>
<b>Irodalomjegyzék</b>	<b>50</b>

# **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, mivel percek alatt el tudják végezni a világ bármely pontjából a nap bármely időpontjában. Az online jegyvásárlás számos előnnyel 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 alkalmazásával, 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 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 megleírható, 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 FIA Formula-1-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, QR kódokat és 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 vész kárba a vásárlás. Ez a rendszer viszont teret ad egy olyan biztonsági kérdésnek, amelyet jelenleg a felhasználó felelősségeire 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 kereskedjenek velük.

Gyakran megtörténik, hogy egy felhasználó több oldalra is ugyanazokat a bejelentkezési adatokat adja meg a regisztráció során. Ez többségében a személyes email fiók felhasználó nevével és jelszavával 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 vannak használva és 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:

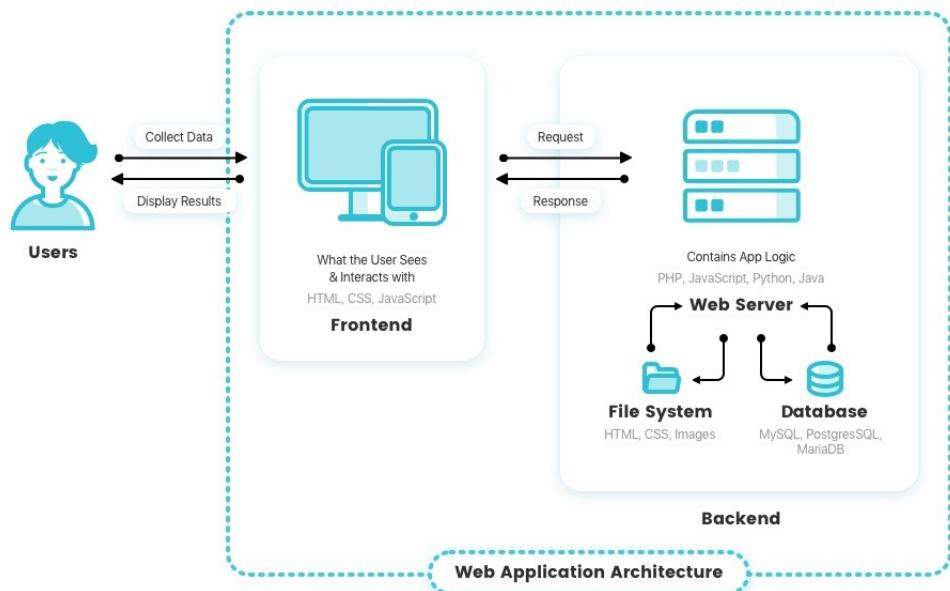
- HTML: Az alapvető struktúrát és tartalmat határozza meg a weboldalakhoz.
- CSS: A megjelenítést és a stílust adja a weboldalaknak, mint például a színek, betűtípusok, elrendezés stb.
- JavaScript: A dinamikus és interaktív funkciókért felelős, például animációk, eseménykezelés, adatmanipuláció. Gyakran 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.
- Keretrendszerek: 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). Nagyobb 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 fél által (Third party) 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, szükségem volt két backend szerverre és egy adatbázis-kezelő rendszerre, amely kiszolgálja a frontendet. Az egyik a Google által fejlesztett Firebase platform, valamint a saját fejlesztésű, Vercel szerveren futó, NodeJS szerver. 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 ki lehet szolgálni az alkalmazás statikus fájljait a világ bármely pontjáról.

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 [AES].

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

Az algoritmus 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 halmozbeli elemet kapunk. Az AES vé-

ges testként használja a bináris együtthatós polinomokat a  $GF(2^n)$  felett, ahol  $n = 8$  fokszámnál kisebb.

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 [Má08].

#### 2.4.2. Az algoritmus elmélete

Az algoritmus 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 nevezünk.

Az altranszformációk három típusúak lehetnek: AddRoundKey, SubBytes és ShiftRows. 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 függvényvel. A ShiftRows lépés során az adatblokk minden sorát egy bizonyos számmal rotáljuk.

Összességeben az AES titkosítási algoritmus magába foglalja a bemeneti adatok blokkokra bontását, majd a bemeneti blokkokon végrehajtja 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.

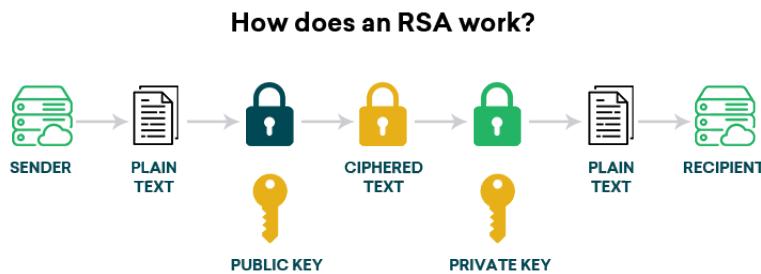
#### 2.4.3. Alkalmazások és érdekességek

- 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.
- Az algoritmus 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 azt együtt használják egyéb kriptográfiai primitívekkel, mint az RSA. 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 késleltetés valószínű, a 128 bites kulcsok jobb választásnak számítanak.

## 2.5. RSA kulcscsere protokoll

Az **RSA** (**Rivest-Shamir-Adleman**) egy kriptográfiai algoritmus, amely nyilvános kulcsokkal dolgozik. Ezt a matematikai alapokon fekvő eljárást számos területen használják, mint a titkosítás, elektronikus aláírások és kulcscsere. Az RSA kulcscsere rendszerben mindenki férnek el egy **publikus** és egy **privát** kulcsa. A nyilvános kulcsot a feladó közzéteszi, míg a privát kulcs titkos marad, amely csak a címzett számára ismert. Az üzenetek titkosításához a feladó használja a címzett nyilvános kulcsát, majd a címzett a saját privát kulcsával tudja megfejteni az üzenetet. Ez biztosítja a biztonságos kommunikációt, mivel csak a címzett képes elolvasni az üzenetet (2.4).



2.4. ábra. RSA protokoll működése

A RSA protokoll alapvetően a prímtényezőkre bontás nehézségén alapul. A kulcsok mérete befolyásolja a rendszer biztonságát, azaz minél hosszabb a kulcs, annál nehezebb a faktorizáció, és így biztonságosabb a rendszer [Zol08].

Fontos megjegyezni, hogy a nagyobb kulcsok használata növeli a számítási időt és az erőforrásigényt, ezért a RSA rendszer lassabb lehet a titkos kulcsú kriptográfiához képest.

Az RSA-t a legtöbb esetben együtt használják olyan privát szimmetrikus kulcsú algoritmusokkal, mint az **AES** (Advanced Encryption Standard), **DES** (Data Encryption Standard), vagy a **Blowfish**. Az RSA titkosítja a szimmetrikus kulcsot, majd ezzel a titkosított kulccsal titkosítja az üzenetet. Ez azért hasznos, mert a szimmetrikus kulcsú algoritmusok hatékonyabbak a hosszú üzenetek titkosításában. Az F1 Ticket Managerben az RSA-t az AES-el együtt használom a jegyek kódjainak titkosítására.

Az alkalmzásomban minden vásárlás esetén generálok egy egyedi RSA kulcspárt, amelyet a Google Firestore-ban tárolok a felhasználók rendeléseinek dokumentumában. A hitelesítés folyamatában meg kell keresni a felhasználó rendelései között az adott egyedi

azonosítóval ellátott rendelést és ezáltal a hozzá tartozó kulcsokat, valamint az **orders** dokumentumon belül az adott versenypályához tartozó rendeléseket, ahol hasonlóan az egyedi azonosító alapján tudja lekérni a rendszer a titkosított kódot.

Fontos megjegyezni, hogy a *gyorsaság* és a *biztonság* nagy mértékben befolyásolják egymást. Ennek értelmében, ha egy rendszer biztonságos, akkor a veszíteni fog a teljesítményéből a bonyolult és adott esetekben sok matematikai számítás során. Ha viszont gyors, akkor az a rendszer kevésbé biztonságos.

Az *F1 Ticket Manager*-ben előtérbe helyeztem a biztonság mértékét a gyorsasággal szemben, mivel érzékeny adatokról van szó. Az alkalmazás teljesítményét ettől függetlenül lehet fejleszteni annak skálázásával, vagyis a szerverek mennyiségek és teherbírásuknak növelésével.

## 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 kezelni a felhasználói jogosultságokat, hogy minden felhasználó csak a saját adataihoz férjen hozzá?
- Hogyan fogom megvalósítani a többlépcsős azonosítást a jegyek validációjánál?
- Hogyan fogom eljuttatni a felhasználóknak a megvásárolt termék(ek) azonosításához szükséges információkat?
- Hogyan tudom a felületet felhasználóbaráttá 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 félreérthető és nincsenek olyan működési aspektusok, amelyek zavarhatják a felhasználói élményt.

Az előbbiek elérésére a kisebb célok egy még pontosabb képet tudnak biztosítani számomra, 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 megoldás után kutatni és kiválasztani az alkalmazás szempontjából megfelelőt.

Továbbá az előzőek mellett megoldást próbáltam kapni 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 elvesztése 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 karbantartását és továbbfejlesztését.

## 3. fejezet

# Rendszerspecifikáció

### 3.1. Rendszerkövetelmények

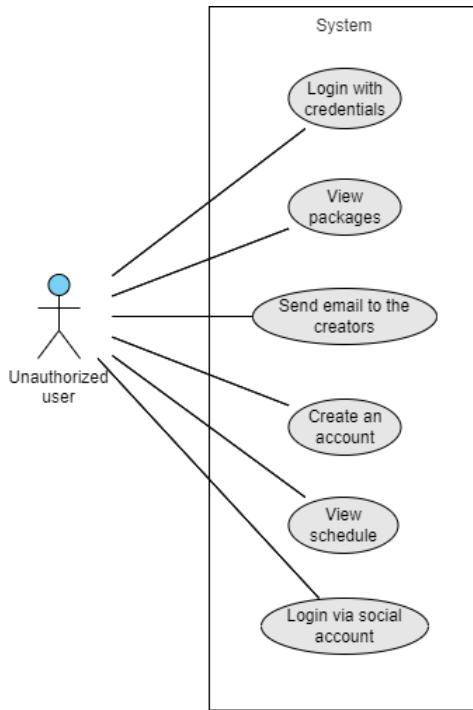
A rendszer specifikációt négy nagy részre oszthatjuk fel, amelyek közé tartoznak a felhasználói felület-, a jegyvásárlás-, a felhasználók- és a biztonság specifikációja. Elsőként a felhasználói felület biztosítja a felhasználó és a rendszer közötti kommunikációt. A jegyvásárlás a rendszer központi funkcionálitása, 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 megfelelően kezelni. A specifikációk prezentálására a Visual Paradigm Online felület segítségével hoztam létre a szükséges diagramokat [[VPO](#)]. 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.

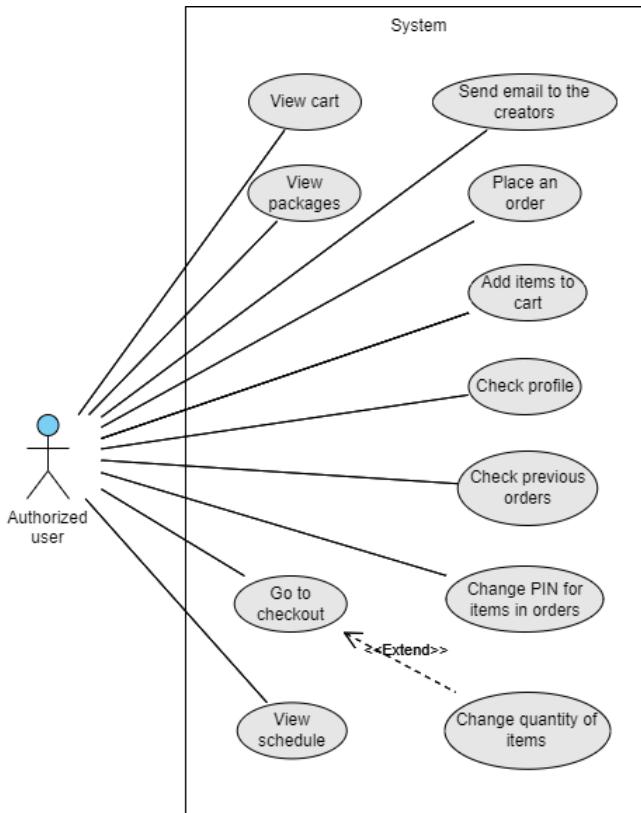
##### a, Felhasználói szerepkörök:

Az oldal böngészésének tekintetében érdemes különböző 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. Ez a funkcionálitás egy továbbfejlesztési lehetőség, mivel érdemes ennek módját egy szélesebb skálán vizsgálni. Emellett fontos, hogy az Anonymous felhasználók be tudjanak jelentkezni az oldalra vagy új felhasználói fiókot tudjanak létrehozni, 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 jegyek kosárba helyezésére, megtekinteni a profiljukat, ahol lehetőségük van felhasználói név és profil kép változtatására, valamint a vásárlási előzményeiket. 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 kosárban levő jegyek mennyiségének módosítására vagy azok törlésére. Itt történik meg a fizetés és a rendelés leadása (3.2).

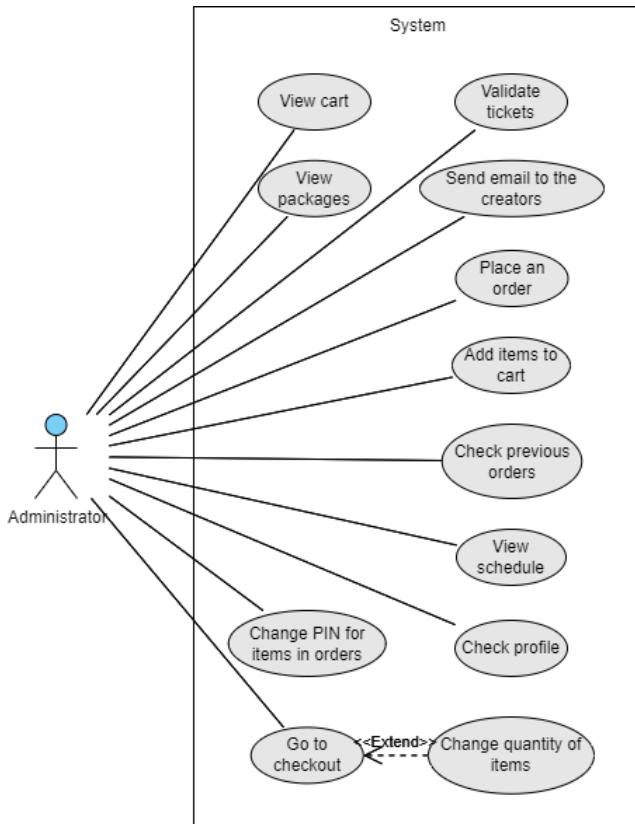


**3.2. ábra.** Use case diagram - Bejelentkezett felhasználó

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 be kell olvasniuk vagy az adatokat manuálisan megadniuk 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, azt kizárolag 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 szociális fiók segítségével. A felhasználóknak lehetőségeük van megadni a regisztrált felhasználónevüket 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 ki kell töltetniük egy regisztrációs űrlapot,

amely tartalmazza az szükséges adatokat, például felhasználónevüket, jelszavukat, email 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 funkcionálisához kapcsolódnak, hanem inkább annak működési jellemzőit, tulajdonságait vagy környezeti tényezőit érintik.

#### **a, Felhasználói szerepkörök:**

A **felhasználói szerepkörök** között egyértelmű határvonal kell legyen, hogy 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 mennyiségű jegykezelésre és skálázhatónak kell lennie, hogy a jegyvásárlás során ne jelentkezzene 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 funkcionálitások 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 funkcionálitások 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 egy felugró ablakban.

**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 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 hibaüzenetek is, amelyek kisebb vagy nagyobb hibák esetén is célravezető módon informálják a felhasználót, hogy ez a hiba miként befolyásolja 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 asztali számítógépekre lett optimalizálva, viszont egy következő továbbfejlesztési lehetőség lenne, hogy reszponzív legyen, ezáltal mobilon és táblagépen is könnyedén kezelhető legyen az oldal. A rendszer kompatibilis a közismert modern böngészők mindegyikével, mint a Microsoft Edge, Google Chrome, Mozilla Firefox, Apple Safari vagy az Opera ([3.4](#)).

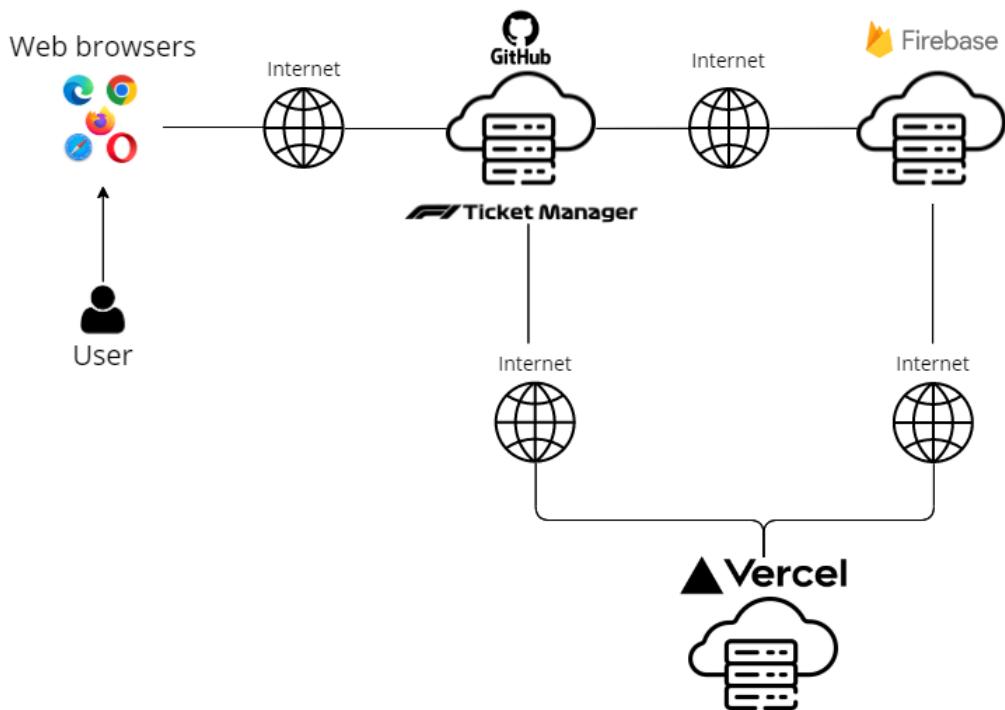


**3.4. ábra.** Web böngészők

## 4. fejezet

# A rendszer architektúrája

Az alkalmazás architektúrája három nagyobb komponensből áll. Mindhárom egy különálló szerver, amelyek egy 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 a GitHub egy optimalizált kódot generál és azt hosztolja [Dep].

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 funkcionálisok 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.

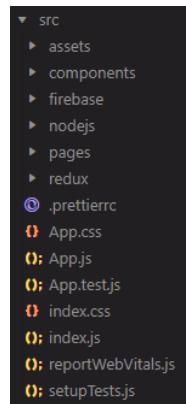
## 4.1. GitHub frontend architektúra

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 kódmezőny 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.

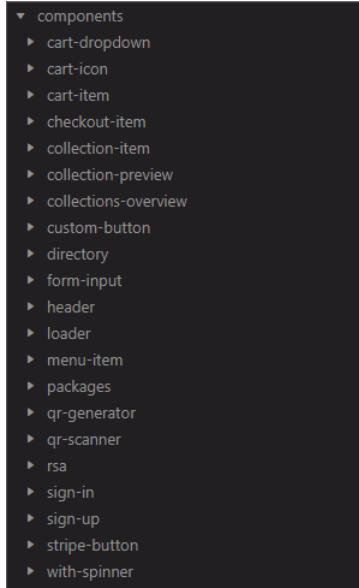


4.3. ábra. Az src mappa struktúrája

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 komponens-ként hozzáadhatjuk a megfelelő mappába anélkül, hogy át kellene írni vagy megváltoztatni a meglévő komponenseket. Ez tisztább és kevésbé összeavaró kódázist eredményez, és minimalizálja a mellékhatásokat vagy a hibák kockázatát.



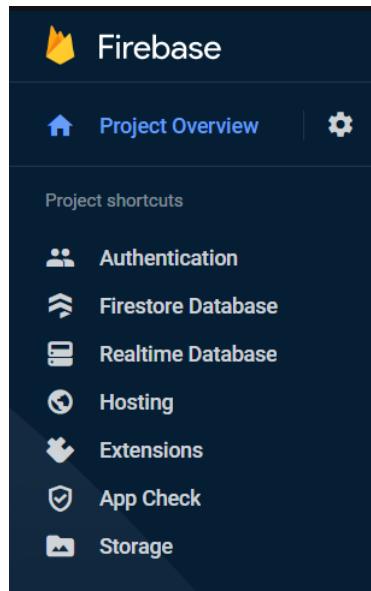
**4.4. ábra.** Komponensek

A **Vercel** szerverrel való kommunikációra az **axios** csomagot használtam. Ennek segítségével tudom kezelnı a hálózaton történő HTTP kéréseket, amelyekhez hozzátartozik 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 üzemeltesenek 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 ke-retrendszerre 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 kérés a Firebase felé és a visszakapott adatok deserializálása. 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 regiszálni 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 leghasználthatóbb, funkcionálitás a **Cloud Firestore**. A Firestore egy dokumentum-orientált adatbázis, amely skálázhatóbb és rugalmasabb lehetőségeket

kínál adatok tárolására és lekérdezésére. Firestore használatakor a fejlesztők struktúrált gyűjteményeket és dokumentumokat hozhatnak létre, és lehetőségük van komplex lekérdezések végrehajtására és az adatok szűrésére. Ilyen 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 kód segítségével ebből a JSON adatcsomagból létrehozok egy többrétegű dokumentumrendszert a Firestoreban.

**4.7. ábra.** Cloud Firestore struktúrája

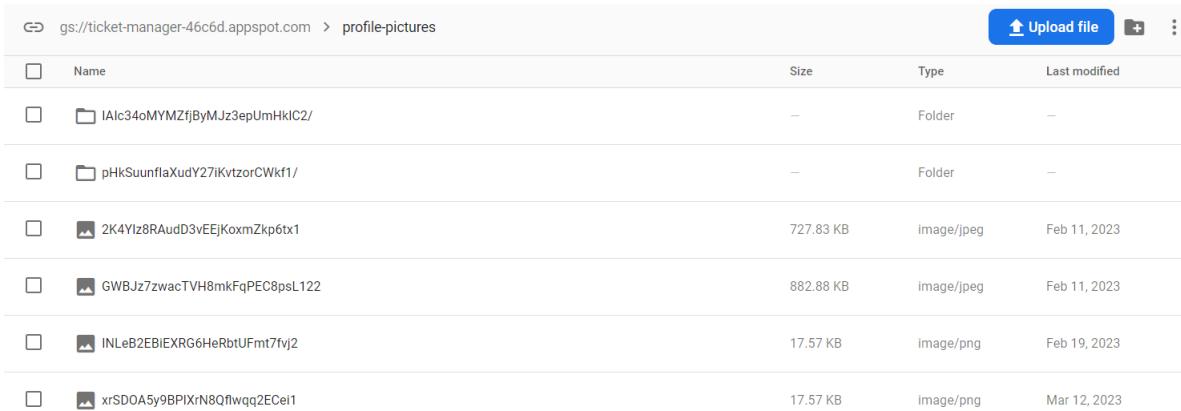
A adatbázis megfelelő biztonságát a **szabályok** (Rules) beállításával lehet megadni. Ezen szabályok tudják biztosítani, hogy az adott dokumentumokhoz milyen feltételek alapján lehet hozzáférni. Ilyen például, hogy egy felhasználó adataihoz mindig csak a *Firebase Authentication* által bejelentkezett 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 three vertical dots. 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 fájlokat. Az én esetemben itt találhatóak 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 rendre a **crypto-js** és **node-rsa** JavaScript könyvtárcsomagokat használom [NRS] [CJS]. 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 vettettem, 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 Firestore-ban tárolom, amelyre olyan szabályok (rules) vannak beállítva, hogy minden 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

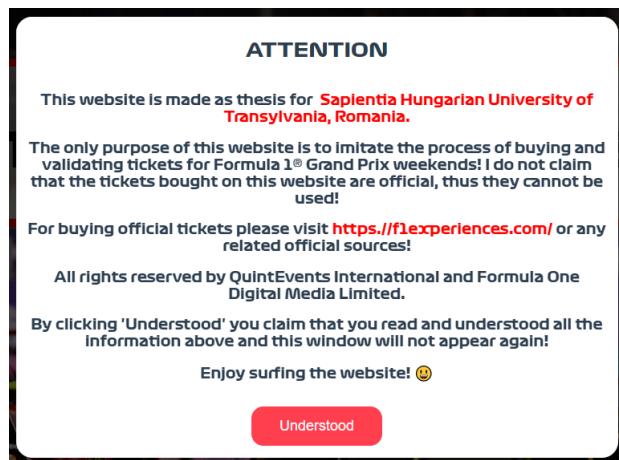
A rendszer architektúrájának tervezése során figyelembe vettetem a 2.6. és 2.7. fejezetekben kitűzött célokat. Az eredeti tervek elkészültével kezdetét vette a megvalósítás. Az implementáció során felmerültek olyan akadályok és újabb ötletek, amelyek kisebb-nagyobb mértékben befolyásolták a terveket. Erre számítottam és éppen ezért úgy próbáltam tervezni, hogy dinamikusan módosíthatóak legyenek 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

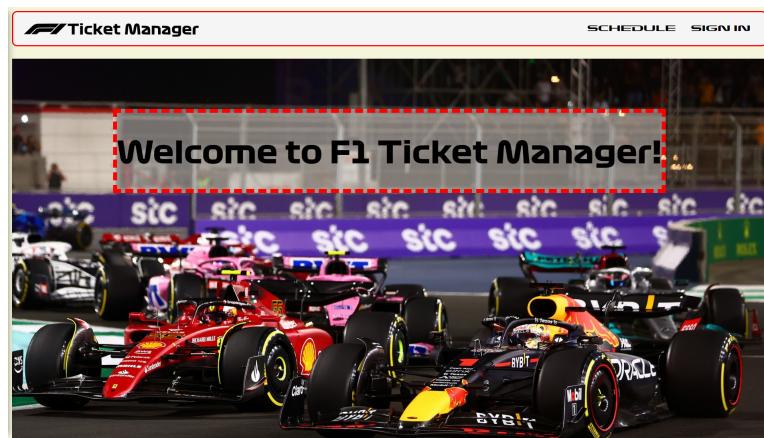
#### 5.1.1. Kezdőoldal (Homepage)

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

Annak érdekében, hogy ez az ablak csak egyszer jelenjen meg **sütiket** (**Cookies**) használok. 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 az ablak.



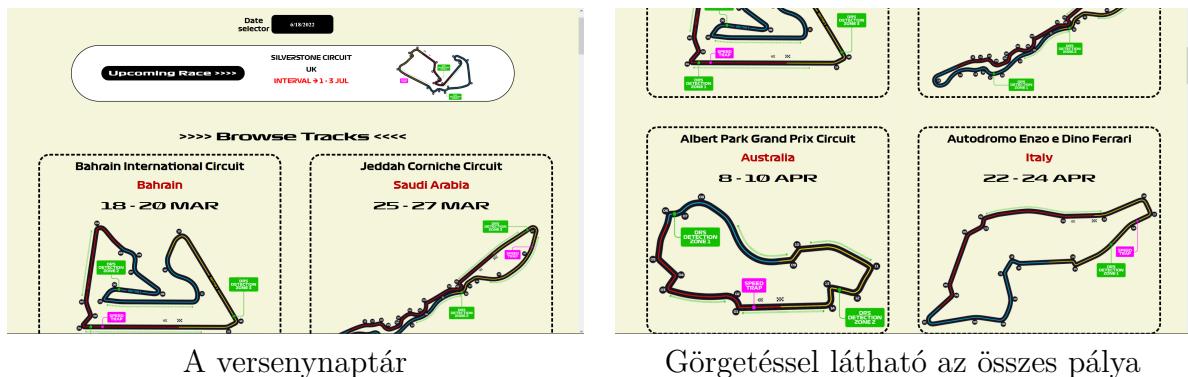
5.1. ábra. Kezdőoldalon felugró ablak



5.2. ábra. Kezdőoldal

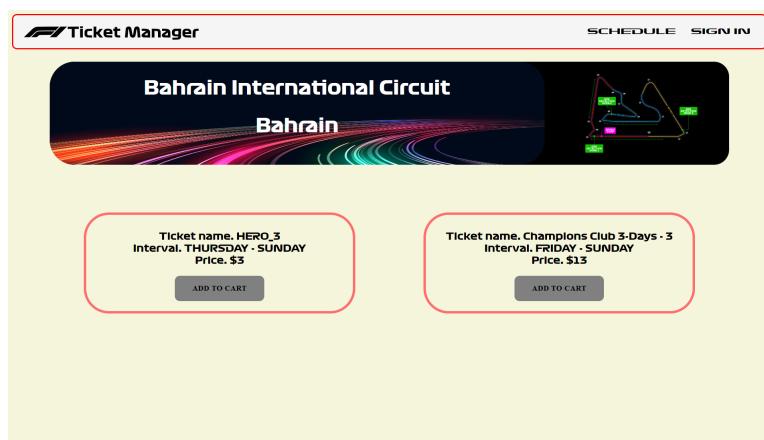
## 5.1.2. Versenynaptár (Schedule)

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). Továbbfejlesztési lehetőség, hogy csak azok a pályák legyenek aktívak, amelyek még hátra vannak.



5.3. ábra. Versenynaptár

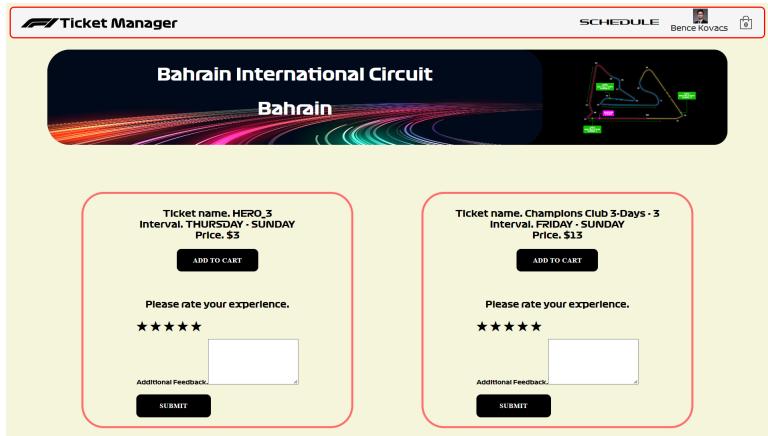
Ha átvisszük az egeret a kártya fölött, akkor megjelenik egy gomb **Browse tickets** szöveggel, amelyre kattintva megtekinthetők a versenyhez tartozó jegy típusok és azok információi (5.4). Ha nincs bejelentkezve a felhasználó, akkor az **Add to cart** gomb szürkén jelenik meg, vagyis nem lehet a kosárba 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. Egy továbbfejlesztési lehetőség, hogy visszajelzést tudjon adni az adott

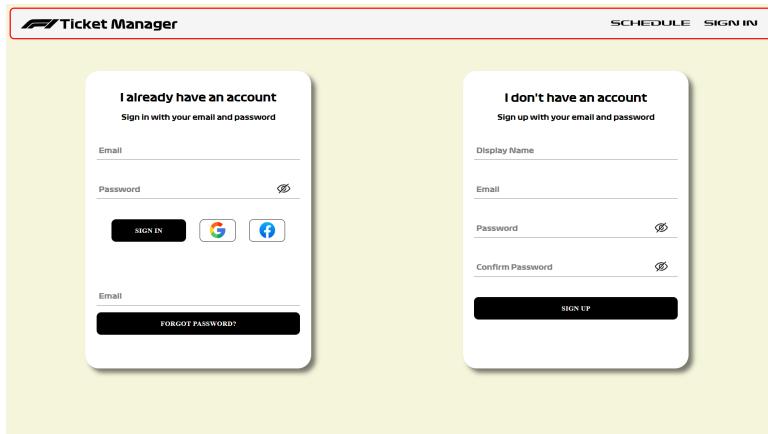
jegyről, élményekről (5.5). A jegyek mennyiségét a **Checkout** oldalon van lehetősége módosítani a felhasználónak. A rendszer ezen része egy továbbfejlesztési terület, hogy adott esetben ellenőrizve legyen, hogy tényleg csak az tudjon visszajelzést adni, akkor vásárolt az adott jegyből.



**5.5. ábra.** Jegy típusok bejelentkezett felhasználóval

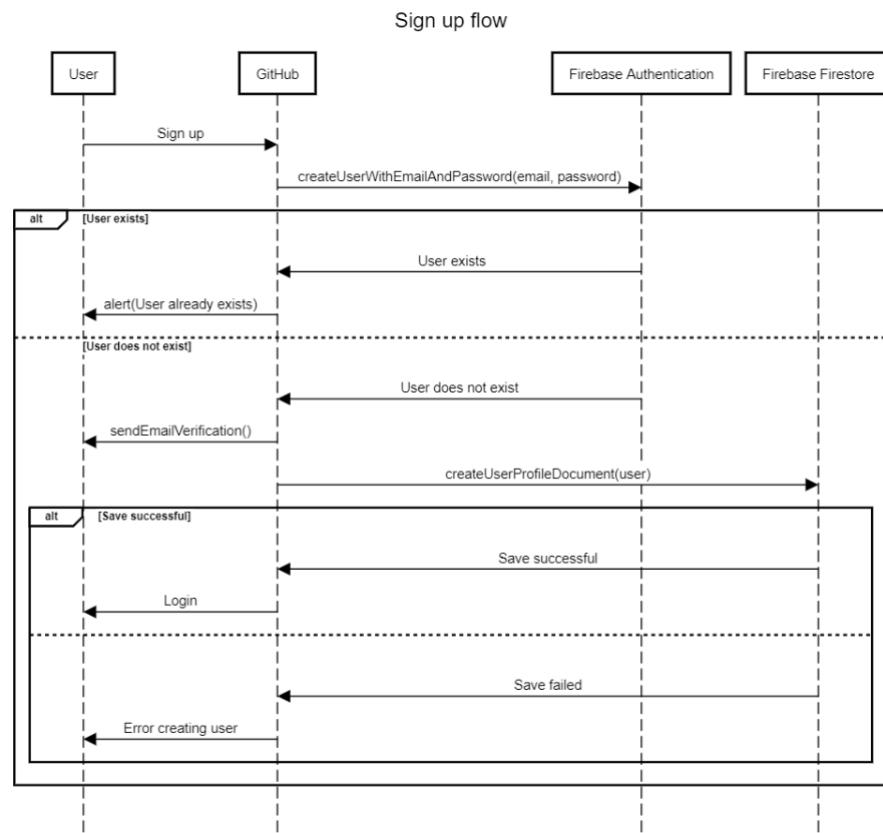
### 5.1.3. Bejelentkezés és regisztráció (Sign in and up)

A bejelentkezési felületen egyben lett feltüntetve a bejelentkezés és regisztráció két külön kártyán. Az email címmel vagy közösségi fiókkal való autentikáció kívül lehetőség van elfelejtett jelszó esetén új megadása. Ennek érdekében a felhasználó megadja az email címét és kapni fog egy levelet, amely tartalmaz egy linket az új jelszó megadásához (5.6).



**5.6. ábra.** Autentikáció

Továbbá lehetőség van a felületre való regisztrációra is a jobb oldali kártyán. Ennek a folyamatnak a bemutatására készítettem egy szekvencia diagramot a **SequenceDiagram.org** online szerkesztő segítségével [Seq].



**5.7. ábra.** Szekvencia diagram - Regisztráció

A regisztráció folyamatát a felhasználó indítja el a kötelező mezők kitöltésével. A *Firebase* szerver első sorban ellenőrzi, hogy az adott email címmel már létezik-e felhasználó és amennyiben igen, a megfelelő üzenetet jeleníti meg az oldal. Ellenkező esetben létrehozza a rendszerben a felhasználót, elmenti a megadott adatokat egyéb meta adatokkal kiegészítve, mint a létrehozás dátuma. A mentés után a rendszer automatikusan be is jelentkezteti a felhasználót (5.7).

#### 5.1.4. Profil (Profile)

A **Profil** oldal elérése csak a bejelentkezett felhasználóval lehetséges (5.8). Ezen az oldalon lehetséges a profilkép és a megjelenített név módosítása az **Edit profile** gombra kattintva.

Az oldal jobb részén láthatóak a **rendelési előzmények**. Ezek görgetéssel böngészhetők és a jobb szélén levő ikonra kattintva tekinthetők meg a vásárlás részletei (5.9).

Amennyiben elvisszük valamely elem fölött az egeret megjelenik egy **More details** gomb, amelyre kattintva megtekinthető a jegy egyedi azonosítója (5.10). 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.

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

No.	Date	Total	Details
1	6/11/2023	\$20	
2	6/10/2023	\$44	
3	6/10/2023	\$44	
4	5/24/2023	\$36	
5	5/18/2023	\$0	

5.8. ábra. Saját profil oldal

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	
<b>Total \$44</b>						

5.9. ábra. Rendelés részletei

**Ticket details**

UID:

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

NEW PIN

CANCEL

5.10. ábra. Jegy részletei

A PIN kód megadásánál kötelezően egy 6 számjegyű számot kell megadni (5.11). 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.



**5.11. ábra.** PIN kód megadása

### 5.1.5. Rendelés (Checkout)

A Rendelés oldalon van lehetősége a felhasználónak véglegesíteni a vásárlást ([5.12](#)). Itt is láthatóak a kosárba helyezett jegyek információi és csak itt módosíthatóak az egyes jegyek mennyisége vagy a kosárba való törlése. A **Pay Now** gombra kattintva a felhasználó meg tudja adni a számlázási adatokat majd a kártyaadatokat a vásárlás véglegesítéséhez. Ezt a **Stripe** szolgáltatás segítségével valósítottam meg. Sikeres vásárlás esetén a felhasználónak küld a rendszer egy emailt, amiben megkapja a QR kódokat. Egy kód tartalmazza a rendelés egyedi azonosítóját kombinálva a felhasználó egyedi azonosítójával. Az emailek küldésére a **Brevo** szolgáltatót vettem igénybe [[Bre](#)].

**5.12. ábra.** Rendelés véglegesítése

A rendelés leadása után az adatok eltárolására használt titkosítás elvégzésére több megoldást is kipróbáltam, amelyek közül az egyik az [5.1](#) kódrészlet, amely az AES algoritmus egyik használati módja JavaScript programozási nyelven a CryptoJS könyvtárcsomag segítségével.

A kód első lépése a *salt* (*só*) generálása, ami egy 128/8 bájt szekvencia, amely segítségevel 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()* függvénytől 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 elegendőek 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()* függvénytől 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);
};
```

### 5.1. 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 bájt van, akkor a blokk utolsó bájtai a 0x03 értéket veszi fel.

Az *iv* az inicializáló vektort tartalmazza, amit korábban generáltunk. A *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 *titkosított 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 (5.2):

```
export const decryptData = text => {
  const json = JSON.parse(text);
  const salt = CryptoJS.enc.Base64.parse(json.salt);
  const iv = CryptoJS.enc.Base64.parse(json.iv);
  const tag = json.tag;
  const ciphertext = CryptoJS.enc.Base64.parse(json.ciphertext);

  const key = CryptoJS.PBKDF2(secretPass, salt, {
    keySize: 256 / 32,
    iterations: 1000,
  });

  const decrypted = CryptoJS.AES.decrypt(
    { ciphertext: ciphertext, salt: salt, iv: iv, tag: tag },
    key,
    {
      iv: iv,
      mode: CryptoJS.mode.CBC,
      padding: CryptoJS.pad.Pkcs7,
      tag: true,
    }
  );
  return JSON.parse(decrypted.toString(CryptoJS.enc.Utf8));
};
```

## 5.2. 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.

A másik opción a **CryptoJS** csomag mellett a **node-rsa** könyvtárcsomag segítségével megvalósított titkosítás és RSA kulcscsere volt **NodeJS** keretrendszerben. A 5.3 kódrészletben az adat titkosítása látható a RSA publikus kulcs segítségével (5.4 kódrészlet), amelyet minden rendelés során a rendszer generál a privát kulcs mellett. A titkosítás során szükség van a *publicKey*-re, amely a generálás során elmentődik az adatbázisba és ezt lekéri a rendszer. A *publicKey* segítségével titkosítom az *AES* kulcsot, amellyel titkosítom az eredeti szöveget. A végeredmény egy **encryptedData** és egy **encrypted-**

**Key** lesz, amelyek hasonlóan bekerülnek az adatbázisba az összesített rendelések közé, az **orders** dokumentumba.

```
const aesKey = CryptoJS.lib.WordArray.random(16).toString();
const encryptedData = CryptoJS.AES
    .encrypt(<unique code with PIN code>, aesKey).toString();

const rsaKey = new NodeRSA();
rsaKey.importKey(publicKey, "public");
const encryptedKey = rsaKey.encrypt(aesKey.toString(), "base64");
```

### 5.3. kódrészlet. RSA kulccscsere és titkosítás.

```
const key = new NodeRSA({ b: 2048 });
const publicKey = key.exportKey("public");
const privateKey = key.exportKey("private");
```

### 5.4. kódrészlet. RSA kulcsok generálása.

A 5.5 kódrészletben lekéri a rendszer a rendeléshez tartozó privát kulcsot, majd annak segítségével visszafejti a titkosított *AES* kulcsot. Ezzel a kulccsal már vissza lehet fejteni a titkosított szöveget, ezáltal megkapni a az eredeti szöveget az egyedi azonosítóval és a PIN kóddal.

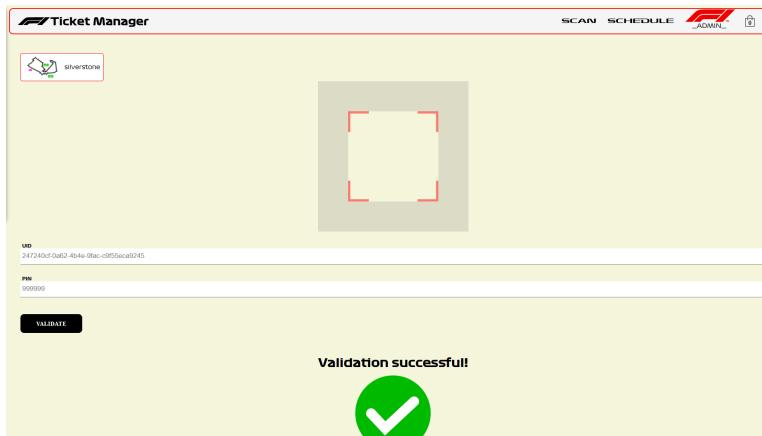
```
const rsaKey = new NodeRSA();
rsaKey.importKey(privateKey, "private");
const decryptedKey = rsaKey.decrypt(data?.encryptedKey, "utf8");

const decryptedData = CryptoJS.AES.decrypt(
    data?.encryptedData,
    decryptedKey
).toString(CryptoJS.enc.Utf8);
```

### 5.5. kódrészlet. RSA visszafejtés.

### 5.1.6. Beléptetés (Scan)

A Beléptetés oldalon a rendszer *Adminisztrátor(a)* tudják a jegyeket hitelesíteni a versenyhelyszíneken az erre a célra kihelyezett terminálon (5.13). A felhasználó a kamera elé helyezi a QR kódot [RQR], majd beírja a PIN kódot, amelyeket a rendszer rögzít és jelzi a hitelesítési folyamat eredményét.



5.13. ábra. Jegyek hitelesítése

# Összefoglaló

Az alkalmazás arra szolgál, hogy egy olyan platformot biztosítson Forma-1-es futamokra. Elsősorban az volt a cél, hogy egy könnyen kezelhető weboldal készüljön figyelembe véve az aktuális elvárásokat és trendeket az ilyen típusú oldalak esetében. Ezek közé tartoznak a bejelentkezés, regisztráció, előzmények megtekintése, valamint a jegyek böngészése. A jegyek elektronikus formában való elküldése a felhasználóknak lehetőséget ad arra, hogy mindenhol elérhető legyen, továbbá kevésbé valószínű az elvesztése.

A hangsúly a jegyek kétrépcsős hitelesítése volt, mivel ez a hivatalos rendszerben nincs jelen. Ez egy olyan plusz biztonságot ad a felhasználóknak, amellyel drasztikusan csökkenhet a jegylopások száma.

GitHub linkek:

[https://github.com/bencekovacs01/F1\\_Ticket\\_Manager](https://github.com/bencekovacs01/F1_Ticket_Manager)  
[https://github.com/bencekovacs01/F1\\_Ticket\\_Manager-NodeJS](https://github.com/bencekovacs01/F1_Ticket_Manager-NodeJS)

# Jelölések

Rövidítés	Angol megnevezés	Magyar megnevezés
F1	Formula One	Formula-1/Forma-1
FIA	International Automobile Federation	Nemzetközi Automobil Szövetség
F1TM	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

# Ábrák jegyzéke

1.1. Online fizetési rendszerek . . . . .	12
2.1. Webes alkalmazás architekúrája . . . . .	15
2.2. React logó . . . . .	16
2.3. Firebase logó . . . . .	17
2.4. RSA protokoll működése . . . . .	19
3.1. Use case diagram - Nem bejelentkezett felhasználó . . . . .	23
3.2. Use case diagram - Bejelentkezett felhasználó . . . . .	24
3.3. Use case diagram - Adminisztrátor . . . . .	25
3.4. Web böngészők . . . . .	28
4.1. Rendszer architekúra . . . . .	29
4.2. Az npm csomagkezelő . . . . .	30
4.3. Az src mappa struktúrája . . . . .	30
4.4. Komponensek . . . . .	31
4.5. Firebase Console . . . . .	32
4.6. Autentikációs módok . . . . .	32
4.7. Cloud Firestore struktúrája . . . . .	33
4.8. Firebase Storage struktúrája . . . . .	34
4.9. Vercel mappa struktúrája . . . . .	34
4.10. Vercel <i>Routes</i> mappa . . . . .	35
5.1. Kezdőoldalon felugró ablak . . . . .	37
5.2. Kezdőoldal . . . . .	37
5.3. Versenynaptár . . . . .	38
5.4. Jegy típusok be nem jelentkezett felhasználóval . . . . .	38
5.5. Jegy típusok bejelentkezett felhasználóval . . . . .	39
5.6. Autentikáció . . . . .	39
5.7. Szekvencia diagram - Regisztráció . . . . .	40
5.8. Saját profil oldal . . . . .	41
5.9. Rendelés részletei . . . . .	41
5.10. Jegy részletei . . . . .	41
5.11. PIN kód megadása . . . . .	42
5.12. Rendelés véglegesítése . . . . .	42
5.13. Jegyek hitelesítése . . . . .	46

# Irodalomjegyzék

- [AES] National institute of standards and technology (nist). <https://nvlpubs.nist.gov/nistpubs/fips/nist.fips.197.pdf>.
- [Bre] Brevo (formerly sendinblue). <https://developers.brevo.com>.
- [CJS] nnpmjs.com - crypto-js. <https://www.npmjs.com/package/crypto-js>.
- [Dep] Github. <https://docs.github.com/en/pages/getting-started-with-github-pages/creating-a-github-pages-site>.
- [Git] Github.com - geshan. <https://github.com/geshan/nodejs-posgresql>.
- [Mod] Dev.to. <https://dev.to/franciscomendes10866/how-to-create-a-modal-in-react-3coc>.
- [Má08] Gyöngyvér Márton. Kriptográfiai alapismeretek. *Scientia*, 2008.
- [NRS] nnpmjs.com - node-rsa. <https://www.npmjs.com/package/node-rsa>.
- [RQR] npmjs.com - react-qr-code. <https://www.npmjs.com/package/react-qr-code>.
- [Seq] Sequencediagram.org. <https://sequencediagram.org>.
- [VPO] Online visual paradigm. <https://online.visual-paradigm.com>.
- [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>.
- [Zol08] Németh L. Zoltán. Hatodik előadás, nyilvános kulcsú kriptográfia i., az rsa. *SZTE, Számítástudomány Alapjai Tansz ny Alapjai Tanszék*, 2008.