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

SAPIENTIA
ERDÉLYI MAGYAR
TUDOMÁNYEGYETEM

Cronotus

# DIPLOMADOLGOZAT

Témavezető:
Dr. Jánosi-Rancz Katalin Tünde,
Egyetemi adjunktus

Végzős hallgató:
Ábrám Szilveszter Levente

**2024**

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

UNIVERSITATEA
SAPIENTIA

Cronotus

# LUCRARE DE DIPLOMĂ

Coordonator științific:
Dr. Jánosi-Rancz Katalin Tünde,
Lector universitar

Absolvent:
Ábrám Szilveszter Levente

**2024**

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

SAPIENTIA
HUNGARIAN UNIVERSITY
OF TRANSYLVANIA

# Cronotus

## BACHELOR THESIS

Scientific advisor:
Dr. Jánosi-Rancz Katalin Tünde,
Lecturer

Student:
Ábrám Szilveszter Levente

**2024**

# Declaraţie

Subsemnata/ul ....., absolvent(ă) al/a specializării ……., promoţia…… cunoscând prevederile Legii Educaţiei Naţionale 1/2011 şi a Codului de etică şi deontologie profesională a Universităţii Sapientia cu privire la furt intelectual declar pe propria răspundere că prezenta lucrare de licenţă/proiect de diplomă/disertaţie se bazează pe activitatea personală, cercetarea/proiectarea este efectuată de mine, informaţiile şi datele preluate din literatura de specialitate sunt citate în mod corespunzător.

Localitatea,

Data: Absolvent

Semnătura……

# Kivonat

A dolgozat célja egy internetes felület biztosítása sport események szervezésére. A mindennapi életünk egy rendkívül fontos, mondhatni elengedhetetlen része a mozgás. Ez a tevékenység sok mindenben megmutatkozhat; lehet ez sétálás futás, vagy akármás sportok űzése is.

Ha sportolni szeretnénk, akkor rendkívül sok lehetőségünk van különböző sportok közül válogatni, viszont van egy tényező, amely valamennyi sportban közös. Mindenképp szükség van egy társaságra a sportoláshoz. Amennyire nyilvánvaló ez a kijelentés, sokszor annyira okozhat kihívást is. Megtörténik, hogy minden indíték, motiváció megvan arra bennünk, hogy mozogni menjünk, de társaság hiányában ezt nem tehetjük meg.

A Cronotus egy olyan internetes felületet biztosít, ahol az emberek könnyedén kiküszöbölhetik ezt a problémát. Lehetőséget biztosít sport események szervezésére, illetve ezek böngészésére, hogy biztosak lehessünk abban, hogy többé senki nem marad otthon azért, mert nem volt akivel sportolni menjen.

A projekt továbbá szem előtt tartja azt is, hogy egyes események könnyedén követhetőek legyenek, lehetőséget biztosít a szervezőknek arra, hogy naprakész információkkal lássák el sportolóikat, továbbá egy visszajelzéseken alapuló rendszert is kínál, ami segíthet a közösségnek abban, hogy a számukra legmegfelelőbb eseményeket találják meg.

# Rezumat

Scopul acestei teze este de a oferi o platformă de internet pentru organizarea evenimentelor sportive. Exercițiul fizic este o parte extrem de importantă, am putea spune indispensabilă, a vieții noastre de zi cu zi. Această activitate necesară și recomandată poate lua mai multe forme; poate fi mersul pe jos, alergarea sau chiar practicarea unui anumit sport.

Dacă dorim să facem sport, există multe sporturi diferite din care putem alege, dar există un factor comun tuturor sporturilor. Ai nevoie de o companie pentru a-l putea practica. Pe cât de evidentă este această afirmație, pe atât de provocatoare poate fi de multe ori. Se întâmplă ca noi să avem toată motivația și dorința de a ieși să facem mișcare, dar fără companie nu o putem face.

Cronotus oferă o platformă online unde oamenii pot elimina cu ușurință această problemă. Aceasta vă permite să organizați și să navigați prin evenimente sportive, astfel încât să vă asigurați că nimeni nu rămâne acasă pentru că nu are cu cine să iasă.

De asemenea, proiectul își propune să facă anumite evenimente ușor de urmărit, oferindu-le organizatorilor posibilitatea de a le oferi sportivilor lor informații actualizate și un sistem de feedback care poate fi folosit de comunitate pentru a găsi cele mai bune evenimente.

# Abstract

The aim of this thesis is to provide an internet platform for the organisation of sports events. Exercise is an extremely important, one might say indispensable, part of our daily lives. This necessary and recommended activity can take many forms; it can be walking, running or even doing some kind of sport.

If we want to do sports, we have many different options to choose from, but there is one factor that all sports have in common. You need a company in order to play it. As obvious as this statement is, it can often be just as challenging. It may happen to the best of us, that we have every necessary motivation to go out and get some movement in, but without a team to accompany us we are unable to do so.

Cronotus provides an online platform where people can easily eliminate this problem. It allows people to organise and browse sporting events, so that we can make sure that no one is left at home because they don't have someone to go out with.

The project also aims to make sports events easy to follow, giving organisers the opportunity to provide their athletes with up-to-date information, furthermore it offers a viable feedback system that can be used by the community to find the best events.

# Tartalomjegyzék

**1. Bevezető** **10**

**2. Felhasznált technológiák** **11**

2.1. A szerver . . . . . 11

2.1.1. A rétegelt architektúra előnyei . . . . . 12

2.1.2. Az adatbázis . . . . . 14

2.1.3. A Cronotus API . . . . . 14

2.1.4. Hitelesítés és Engedélyezés . . . . . 15

2.1.5. Globális hibakezelés . . . . . 16

2.2. A felhasználói felület . . . . . 17

2.2.1. Felhasznált technológia . . . . . 17

2.2.2. Kommunikáció a szerverrel . . . . . 17

2.2.3. Oldalak közötti navigálás . . . . . 18

2.2.4. Munkamenetkezelés JWT tokenekkel . . . . . 19

**Köszönetnyilvánítás** **22**

# 1. fejezet

# Bevezető

Annak ellenére, hogy az információ közlése és elérése napjainkban rendkívül gyors és olcsó, mégis előfordul, hogy alapvető tevékenységeket igen csak maradi módon kezelnek az emberek. Többek között idetartozik egyes sportesemények koordinálása is. A résztvevők gyakran hajlamosak egy nem erre a célra készített platformra hagyatkozni, ami természetesen nem mindig működik úgy, ahogy ideális lenne. Ennek eredményeként megtörténik, hogy lemaradnak az emberek olyan információkról, ami egyébként releváns lett volna számukra.

A Cronotus projekt ennek a problémának a megoldását egy ingyenes és könnyen kezelhető platfrom biztosításával közelíti meg, ami arra törekszik, hogy kiküszöbölje az esetelges információk elvesztését, vagy akár figyelmen kívül hagyását.

A dolgozat elsősorban bemutatja a projekt általános működését, párhuzamba helyezi más ehhez hasonló internetes platformokkal, szemünk elé tárja az esetleges előnyöket és hátrányokat más szolgáltatásokkal szemben.

A továbbiakban a dolgozat ismerteti velünk mélyebben a Cronotus szerkezetét, ahol felépítés szerinti rétegekre bontja, majd ezeket részletekbe bocsátkozva tárgyalja. Bemutatja azt, hogy hogyan működik a web kliens-t kiszolágló szerver, milyen biztonsági intézkedéseket biztosít, illetve hogyan kezeli a felmerülő hibákat. Betekintést biztosít abba, hogy hogyan érkezik el a kívánt információ az adatbázistól egészen a felhasználó felületig.

A következő fejezetek segítségével szeretném bemutatni a felhasznált eszközöket és módszereket, melyek elősegítik a Crontous helyes működését. Végezetül egy következtetést vonok le, illetőleg továbbfejlesztési lehetőségeket tárok az olvasó elé, melyek fényt derítenek a projekt jövőbeli élettartamára.

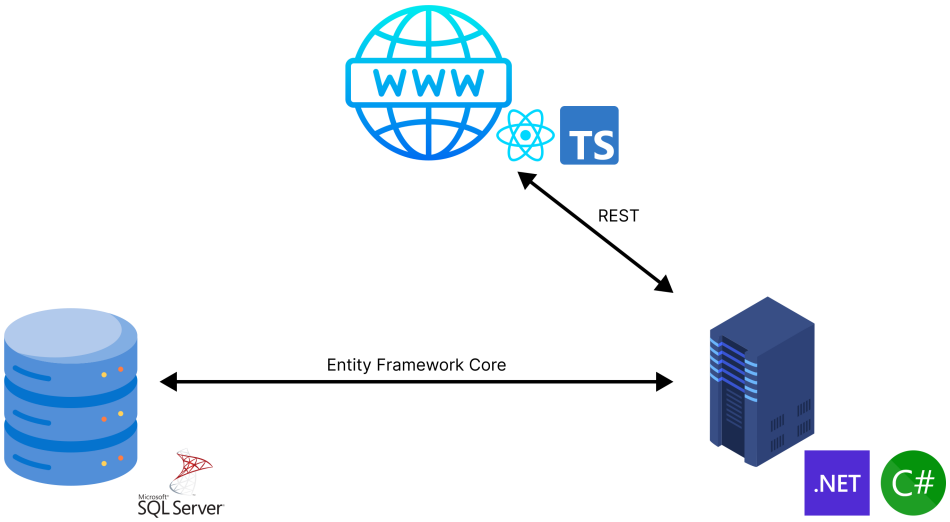
A paltform készítése 2023 októberében kezdődött a Codespring szoftverfejlesztő cég koordinálásával. Megemlítendő, hogy a felhasznált technológiák visszatükrözik azt a tudást, amit a Marosvásárhelyi Codespring Mentorprogram keretein belül sajátíthattam el, ami 2022 őszi időszakában vette kezdetét. A projektet a koordináló cég segítségével egyedül fejlesztettem.

Az alkalmazás neve, azaz a Cronotus, egy összetett lingvisztikai ihletből merített szó. A név első fele a görög mitológiai Krónosztól származik, akit az idő isteneként ismerünk, míg a név második fele a „mozgás” szó latin fordításából, a 'motus' szóból ered.

# 2. fejezet

# Felhasznált technológiák

A Cronotus alkalmazás két jól elkülöníthető részre bontható fel. Amint a 2.1-es ábrán is látható, a felhasználói felület weben keresztül érhető el, mely a React technológia segítségével lett megalkotva TypeScript nyelven, míg az alkalmazás szerver oldali része egy .NET keretrendszeren alapuló program, mely C# nyelven lett megírva és az Entity Framework Core által lefektetett szabályrendszert használva kommunikál egy Microsoft SQL Server adatbázissal, ahol minden adat eltárolásra kerül. Ahhoz, hogy ezeket az adatokat a felhasználó elérhesse, a kliens és szerver oldal között egy REST API segítségével történik meg a kommunikáció.

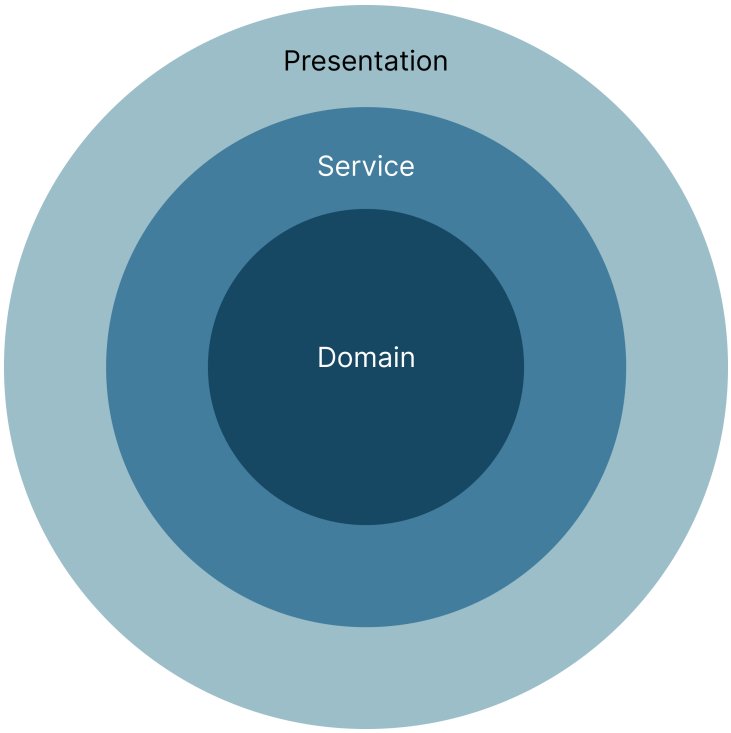



**2.1. ábra.** Rendszer architektúra komponensekre bontva

## 2.1. A szerver

A szerver feladata az, hogy kiszolgálja a webalkalmazásról beérkező kéréseket. Ez a feladat alapszinten viszonylag könnyen elvégezhető lenne, viszont nem elég csak csupán elvégezni, hanem a legmegfelelőbb módon kell ezt véghez vinni. Fontos szem előtt tartani a biztonságot, kérésenkénti időt, illetve megbízhatóságot. Az első fontos döntés a szerverrel

kapcsolatban az, hogy milyen architekturát valósít meg. A Cronotus esetében egy rétegelt architekturáról beszélhetünk, mely három fő elemre bontja a szerver működését.




**2.2. ábra.** A rétegelt szerver architektúrája

A három réteg a 2.2-es ábrán feltüntetett módon helyezkedik el. A legalsó szint a Domain, melynek mindössze annyi a feladata, hogy az adatbázissal kommunikáljon. Ezen a szinten történik meg az adatok kinyerése, vagy akár az alapszintű műveletek véghezvitele, például létrehozás, vagy törlés.

A kör középpontjától kifele haladva a következő a Service réteg. Ez az a szint, amely a biznisz logikát végrehajta a kapott adatokon. Abban az esetben, ha a kliens oldal egy kérést küld valameny adat lekérésére, úgy a kéréshez szükséges eredmény ezen a szinten formálódik olyan alakba, amely megfelelő a fogyasztónak. A szint felelősége úgy is felfogható, mint egy tolmács, mely a kliens és Domain között működik. Főbb feladatai a komplex struktúrájú adatok egyszerű alakra és utasításokra bontása, illetve egyszerű adatok előkészítése és prezentálható formába szervezése a kliens számára.

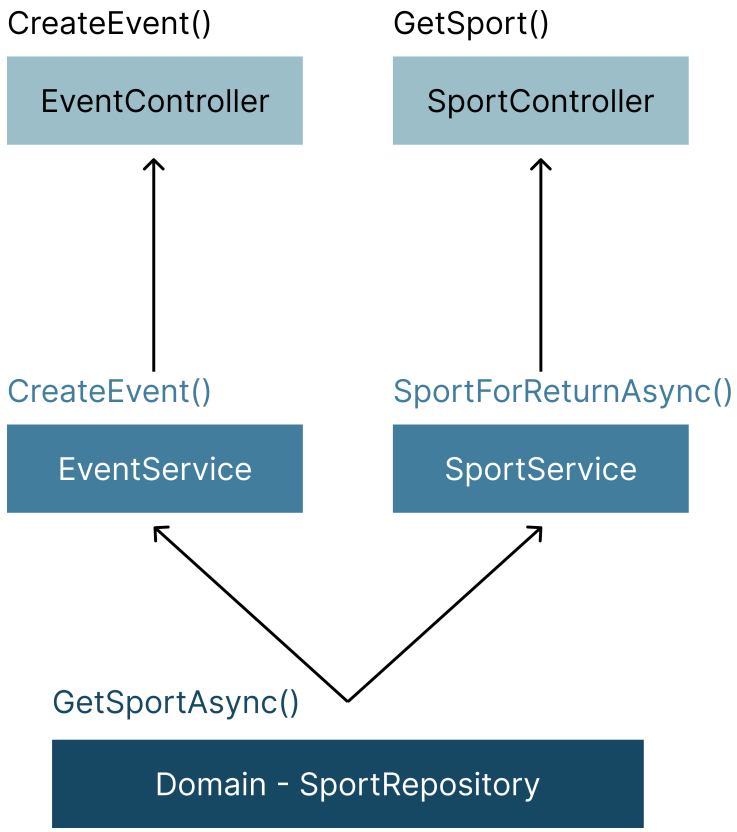
A legfelső, a Presentation réteg feladata a fogyasztó irányába történő kérésekkel kapcsolatos visszajelzések megfelelő kezelése, illetve a Service rétegtől kapott adatok átirányítása a helyes HTTP kóddal.

### 2.1.1. A rétegelt architektúra előnyei

Felmerülhet a kérdés, hogy miért érdemes az előbbiekben [2.2] bemutatott formába szervezni a szerver felépítését. Ahhoz, hogy ezt bemutassuk, fontos először megvizsgálni az architektúrán belüli egyes rétegek közti függőségek folyását. Ha ezt a felépítést követjük, akkor a függőségek a legbelső szintektől a felsőbb szintek irányába folynak, azaz a

Domain réteg nem függ semmitől. Ez betudható szabályként is, tehát egy réteg minél alacsonyabban van a felépítésben, annál kevesebb függősége van.

Továbbá minden réteg csakis kizárólag az alatta levő rétegre hivatkozhat. Ez azért hasznos, mert így nem kall aggódnunk afelől, hogy hogyan fog egy felsőbb szintű művelet implementálni egy alacsonyabb osztályút, elég kontraktokat felállítanunk, melyeket teljesíthetnek a felső rétegek. Ez fontos tulajdonsága az architektúrának, hiszen nagy szabadságot nyújt egyes szintek elkészítésében, hiszen elég csak az oda tartozó logikára figyelnünk.



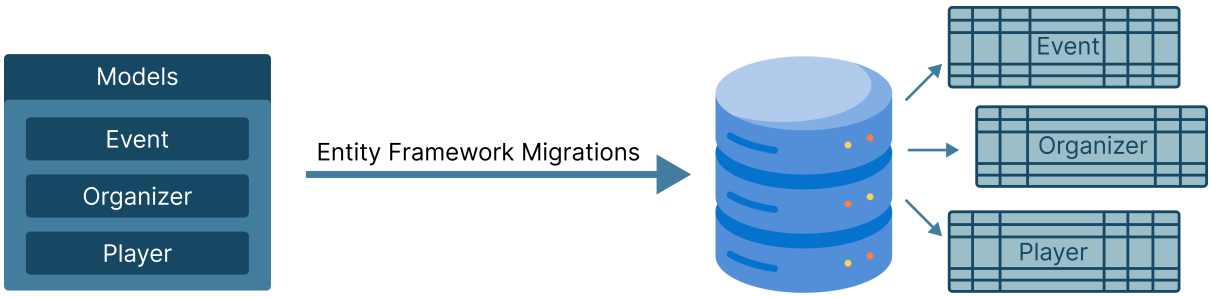

**2.3. ábra.** A SportRepository egy Domain szintű műveletének eredményének kétféleképpen való felhasználása felsőbb szinteken

A 2.3-as ábrán látható egy példa, mely a rétegelt architektúra egy előnyét használja fel. A SportRepository-ban rendelkezésünkre áll egy függvény, mely egy Sport Entity-t térít vissza. A Repository-nak ebben az esetben nem kell tudnia, hogy hogyan fogják a felsőbbrendű rétegek felhasználni az innen kapott adatot. Az EventService arra használja fel az adatot, hogy ellenőrzés alá vesse a létrehozandó eseményhez kiválasztott sportot, vajon létezik-e? A SportService viszont arra használja fel ugyanazt az eredményt, hogy egy prezentálható formában vissza térítse a Presentation szintnek a sportot, ami egy Controller formájában elérhetővé teszi a kliens számára, hogy lekérhessen egy sportot ID alapján.

### 2.1.2. Az adatbázis

A rendszer adatai egy Microsoft SQL Server relációs adatbázisban vannak eltárolva. Mivel a rendszer személyes információkat tárol el a felhasználóiról, szükséges volt egy olyan adatbázist választani, mely megfelelő képpen tudja kezelni az egyes felhasználók és felhasználókhoz kötődő entitások közötti kapcsolatokat. Továbbá az is fontos, hogy biztonságosan legyenek az adatok tárolva. Mindezeket figyelembe véve egy relációs adatbázis volt a legjobb választás. A konkrét döntés azért esett a Microsoft SQL Server termékre, mivel a szerver fő keretrendszere alapból Microsoft termék, így a Microsoft-készített adatbázis egy jó integrálhatósági lehetőségnek bizonyul. Továbbá a technológiák megegyezése elősegít a későbbi felhőbe való kitelpítésben is.

Az adatbázis strukturális létrehozása a *code first approach* elv segítségével hajtódik végre, ahogyan a 2.4-es ábrán is látható. Az Entity Framework Core lehetővé teszi azt, hogy adatmodellek és adatmodellek közti kapcsolatok alapján migrációk segítségével változtassunk az adatbázis sémán, illetve az abba tartozó adatokon.




**2.4. ábra.** A Code First Approach áttekintése

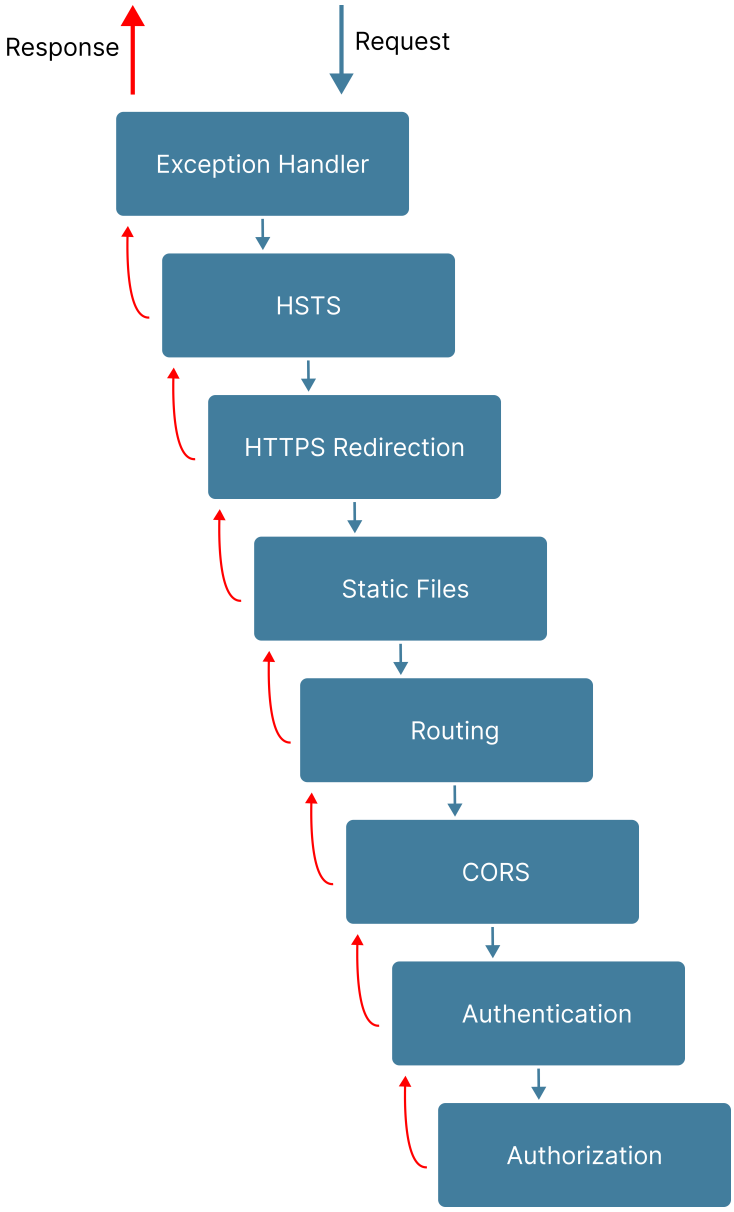
Technikailag megvizsgálva, a Code First Approach [2.4-es ábra] az Entity Framework Core egyik hozománya, mely az ORM (Object Relational Mapper) segítségével képes a *DbContext* osztályon keresztül kapcsolatba lépni az adatbázissal, továbbá a *Repository-Base* absztrakt osztályon végrehatjtandó CRUD műveletek segítségével tudja az itt tárolt entitásokat kezelni, törölni vagy újakat létrehozni.

### 2.1.3. A Cronotus API

A kliens oldali kéréseket a szerver a Presentation [2.1-es ábra] rétege fogadja és szolgálja ki. A *pipeline*-ja HTTPS (HyperText Transfer Protocol Secure) protokolt követve üzemel. A kliens HTTP kéréseket küldve tud adatokat küldeni, lekérni, illetve szerveri erőforrásokat igénybe venni adatokkal kapcsolatos műveletekhez. Az adatokat DTO-k (Data Transfer Object) formájában cserélgeti a kliens a szerver oldallal, mely JSON (JavaScript Object Notation) formájában valósul meg.

A szerver minden beérkező kérést átfuttat egy ellenőrzésrendszeren [2.5-ös ábra], mely az Application Pipeline formájában van felépítve. Itt biztonsági és adatintegritási szempontból fontos vizsgálat alá veti a kéréseket. A szerver a pipeline-ban érvényesíti azt, hogy HTTP helyett biztosan HTTPS protokllon keresztül történjen meg a kommunikáció, hiszen így egy harmadik fél nem tudja leolvasni a küldött kérések tartalmát, úgy sem ha hozzáférése van ezekhez, mivel kódolva vannak. A pipeline továbbá más hasznos beállításokat is érvényesít, például azt, hogy csak megfelelő URL címekről legyen engedélyezett

a hozzáférés a szerverhez, illetve jogosultságot vizsgál *Authentication* és *Authorization* segítségével.




**2.5. ábra.** A pipeline, melyen keresztül kell mennie minden szerver fele érkező kérésnek

### 2.1.4. Hitelesítés és Engedélyezés

Az rendszer JWT Tokenek formájában kezeli a felhasználói munkamenetet és az ezekbe kódolt információk segítségével végzi el a felhasználók hitelesítését és bírálja el az *endpoint*-okhoz való hozzáférésüket.

Egy ilyen token fontos információkat tartalmaz, például felhasználói jogok, jogokhoz kapcsolódó id-k, illetve a token lejárati dátuma.

A kliens oldalról kapott kérések mindegyikéhez csatolva van egy ilyen JWT token, amit a rendszer egy kérés beérkeztekor dekódol, majd az innen kinyert információk segítségével megnézi, hogy érvényes-e a token, majd ellenőrzi, hogy a felhasználónak van-e joga végrehajtani a kívánt műveletet. Egy token élettartama 10 perc, ezután egy *Refresh Token* segítségével a kliens újat kell kérjen. Ez a fajta viszonylag rövid szsavatossági idő egy fontos biztonsági intézkedés, mivel ha egy harmadik fél meg is tud szerezni egy ilyen érvényes tokent, akkor is csak maximum 10 perce van arra, hogy hozzáférése legyen a szerverhez.

### 2.1.5. Globális hibakezelés

Annak érdekében, hogy a kód átlátható és könnyen karbantartható legyen a hiba és kivételkezelést a projekt egy globális hibakezelő middlware segítéségvel oldja meg, mely részét képezi az application pipeline-nak, mely be lett mutatve a 2.5-ös ábán.

Ha ezt a kezelési módszert használjuk, akkor nincs szükség *try - catch* kódblokkok használatára, hiszen ha valamilyen kivétel merül fel, akkor azt elég csak tovább dobni, majd a middleware automatikusan lekezeli. A kezelés során a middleware beállítja a visszatérítendő objektumba a fellépő hiba HTTP státuszkódját, majd a kiindulási pontból dobott hiba üzenetét, mindezt annak érdekében, hogy kliens oldalon könnyen olvasható legyen, hogy egyes hiba miből származhat. Ezt a folyamatot a **??**-es kódrészlet szemlélteti.

```
1 private static Task HandleExceptionAsync(HttpContext httpContext, int statusCode,
      string errorMessage)
2 {
3     httpContext.Response.StatusCode = statusCode;
4     httpContext.Response.ContentType = "application/json";
5     return httpContext.Response.WriteAsync(new ErrorDetails
6     {
7         message = errorMessage
8     }.ToString());
9 }
```

**2.1. kódrészlet.** A visszatérítendő válasz felépítése

## 2.2. A felhasználói felület

A felhasználói felület egy webalkalmazás formájában nyilvánul meg. A felhasználók itt végezhetnek eseményekkel kapcsolatos műveleteket, például létrehozás, csatlakozás, vagy kommentelés. Ez az alfejezet a felhasználói felület felépítését és működését tárgyalja.

### 2.2.1. Felhasznált technológia

A webalkalmazás TypeScript nyelven íródott, React könyvtár felhasználásával. A TypesScript a JavaScript nyelvet bővíti ki egy szintaxis csomaggal, amely lehetővé teszi a változók és objektumok típuskezelését, ezzel egy jobban átlátható és biztonságosabb fejlesztési procedúrát nyújtva.

A React könyvtár segítségével a webalkalmazás egy Single Page Application (SPA)-ként működik. A SPA az egész weboldalt egyetlen HTML dokumentumban építi fel, majd változás során ennek a tartalmát frissíti és cseréli. Ez előnyös megoldás, mivel így nem kell minden oldalcsere, illetve tartalomfrissítés után újratölteni az alkalmazást, így gyorsabb lesz, javítja a felhasználói élményt.

Az alkalmazáshoz felhasznált további könyvtárcsomagok kezelése, a webalkalmazás futtatása, illetve építése a Vite segítségével van megoldva. A Vite egy gyors és egyszerű build eszköz, mely alkalmas React webalkalmazások kezelésére. Lehetővé tesz Live Reloading-ot mely hasznos a fejlesztés során, illetve úgy van felépítve, hogy minimalizálja az alkalmazás építéséhez szükséges időt, így mégjobban gyorsítva és javítva a fejlesztési időt, élményt.

Az alkalmazás felhasználói felületének egy szerves része továbbá a React könyvtárra épített Material UI komponens csomag, mely segítségével könnyen és gyorsan lehet előre megépített megjelenítő komponenseket felhasználni.

### 2.2.2. Kommunikáció a szerverrel

A webalkalmazás HTTPS protokollon keresztül kommunikál a szerverrel. A HTTPS protokoll a HTTP protokoll biztonságos formája. Itt a HTTP-vel ellentétben az adatok nem egyszerű szöveges formátumban vannak küldve, hanem kódolt blokkokként, melyek biztosítják azt, hogy út közben egy rosszindulatú harmadik félnek ne legyen hozzáférése bármilyen kommunikációhoz.

Az alkalmazás GET metódusokra az SWR - Data Fetching könyvtárcsomagot használja. Az SWR rövidítés az angol „stale-while-revaildate” szókapcsolatoban ered, mely annyit tesz, hogy „elavult-közben-újraérvényesít”. Az SWR adatlekérési stratégia alapján egy kérés után először a potenciális elavult adatot megkapjuk egy cache (stale) tároló rendszerből, majd elküldjük a kérést a legfrissebb adatért (revalidate), majd végül előállunk ezzel.

A könyvtárcsomag lehetőséget biztosít arra is, hogy az adatlekéréssel kapcsolatos státuszokat kezeljük, például kérés alatt töltés ideje, illetve hibákat. Ezt egyszerűen megtehetjük a függvényhívás esetén, amint a 2.1-es kódrészleten is látszik.

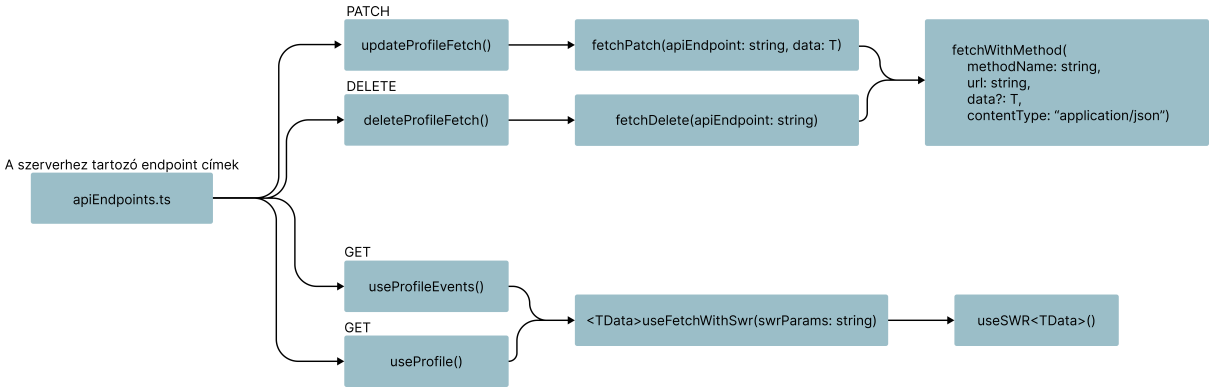
```
const { data: profileInfo, isLoading, error, mutate } = useProfile(userId!);

{isLoading ? (
            <ProfileTitleIsLoading />
          ) : (
            <ProfileHead
              profileInformation={profileInfo!}
              className="profile-head"
            />
          )}
```

**2.2. kódrészlet.** SWR használata profil adatok lekérésére

A 2.1-es ábrán látható, hogy abban az esetben, ha az adatok még lekérés alatt vannak, a komponens a „ProfileTitleIsLoading” komponenset tölti be, majd amint a a betöltés befejeződik, megjeleníti a betöltött adatokat.

Amennyiben a szerverhez intézett HTTP kérés nem GET metódus, úgy az alkalmazás nem SWR-t használ, hanem egy egyszrű, dinamikusan felépített fetchert. Ez a fetcher függvény újrahasznosítható, így elég minden egyedi kérés esetében csak megadni a megfelelő adatokat, endpoint címet, és meghívni az ehhez tartozó fetchert. Ez a folyamat látható a **??**-os ábrán is.




**2.6. ábra.** A profil oldallal kapcsolatos néhány kérés működése

### 2.2.3. Oldalak közötti navigálás

A webalkalmazáson való oldalak közötti navigációért a React Router könyvtárcsomag felelős. Ez lehetővé teszi azt, hogy könnyen és zökkenőmentesen tudjunk oldalak között navigálni, illetve ennek során információt cserélni egyes oldalak, komponensek között.

A React Router lehetővé teszi azt, hogy deklaráljunk minden létező oldalt egy „Router” komponensben, ahonnan aztán a Single Page Application dinamikusan cserélni tudja a pillanatnyilag megjelenített oldalt. Ez a deklarálási módszer látható a 2.2-as kódrészletben.

```
const Router = (props: { onLogout: () => void }) => {
  return (
    <Routes>
      <Route path="/" element={<Layout />}>
        <Route index element={<Home />} />
        <Route path="browser" element={<Browser />} />
        <Route path="create-event" element={<CreateEvent />} />
        <Route path="event/:id" element={<SportsEvent />} />
        <Route path="profile" element={<Profile onLogout={props.onLogout} />} />
        <Route path="*" element={<NoMatch />} />
      </Route>
    </Routes>
  );
};
```

**2.3. kódrészlet.** Route-ok deklarálása

Abban az esetben, ha kliens oldalon böngészőn belül nem a megfelelő cím kerül megadásra, úgy a „NoMatch” route kerül betöltésre. Ez hasznos, mivel ha a felhasználó egy nem létező oldalt akar betölteni, akkor nem egy érthetetlen hibaüzenet kerül elé, hanem egy általunk felépített érthető visszajelzés.

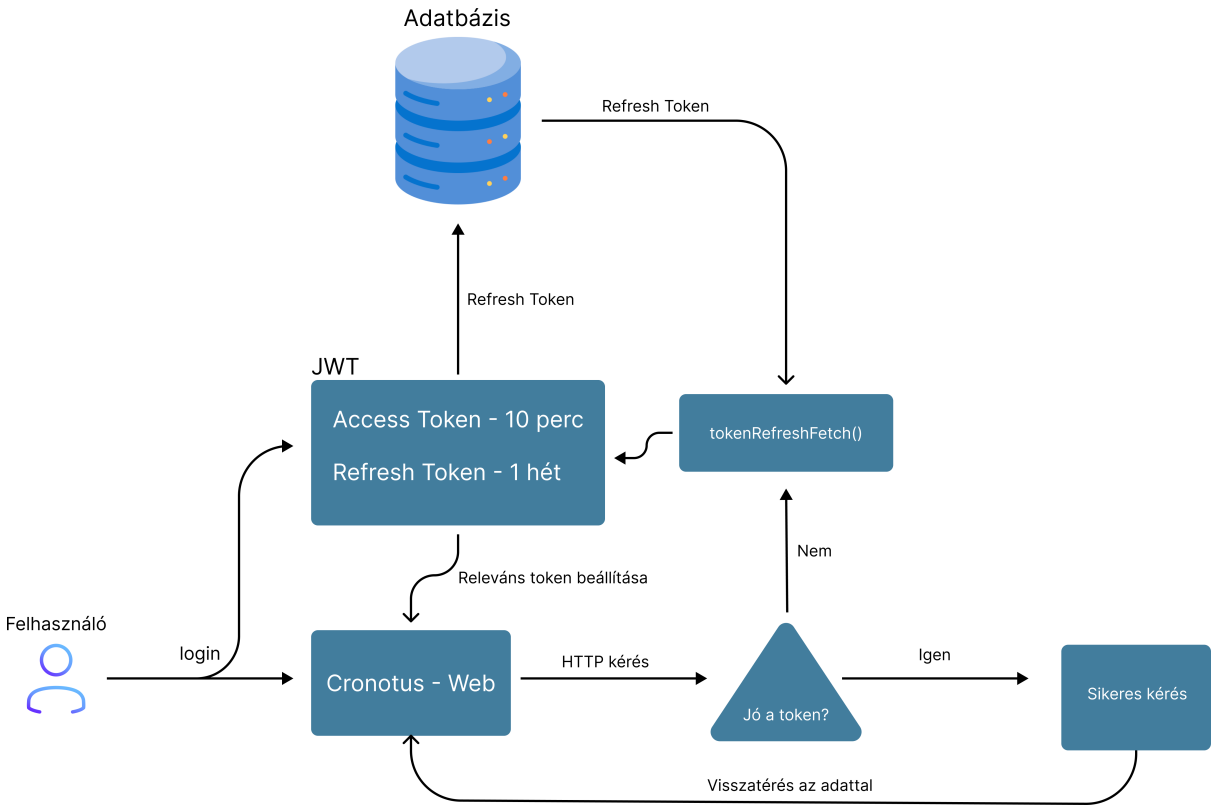
### 2.2.4. Munkamenetkezelés JWT tokenekkel

A webalkalmazás biztonságos működését JWT (JSON Web Token)-ek segítik elő. A JWT egy szerkezetileg három részből álló metódusa annak, hogy két fél biztonságosan információt válthasson egymás között.

A webalkalmazás szintjén ahhoz, hogy a Authentication és Authorization-el kapcsolatos feladatokat elvégezzük ilyen tokeneket használunk. Minden szerverhez intézett kéréshez a HTTP kérés Header részébe egy ilyen token van csatolva, amely segítségével elérhetőek a jogrendszer szerinti védett endpointok. A tokenekbe bele van kódolva a felhasználókhoz kapcsolódó egyedi azonosítók, jogok, illetve a token lejárási ideje.

A lejárási idő a webalkalmazás szintjén fontos, mivel ennek függvényében lehet a munkamenetet követni. Abban az esetben ha egy token lejár, a felhasználó alap esetben arra van kérve, hogy jelentkezzen be újra. Ez azért hasznos, mert ha egy harmadik rosszindulatú fél meg is szerez egy ilyen tokent, csak limitált ideje van kárt tenni a rendszerben, esetünkben 10 perc.

Ahhoz, hogy egy jó felhasználói élményt nyújtson a webalkalmazás, két féle tokent használunk. Van access token, illetve refresh token. Az access tokenek magukba foglalnak minden olyan információt és működési elvet, melyet a fentiekben tárgyaltunk. Azért, hogy egy access token lejártakor ne kelljen újra bejelentkeznie a felhasználóknak, refresh tokeneket használ az alkalmazás, melyek adatbázisban tárolva vannak minden felhasználó számára. Ezek segítségével új access tokent lehet generálni, így nem szükségeltetve az újboli bejelentkezést 10 percenként. Ezt a folyamatot szemlélteti a **??**-es ábra is.




**2.7. ábra.** Standard JWT kezelési folyamat a webalkalmazáson

# Köszönetnyilvánítás

Ez nem kötelező, akár törölhető is. Ha a szerző szükségét érzi, itt lehet köszönetet nyilvánítani azoknak, akik hozzájárultak munkájukkal ahhoz, 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.