



EÖTVÖS LORÁND TUDOMÁNYEGYETEM

INFORMATIKAI KAR

PROGRAMOZÁSELMÉLET ÉS SZOFTVERTECHNOLÓGIAI  
TANSZÉK

## Shalendar

*Témavezető:*

Pintér Balázs

egyetemi adjunktus, PhD

*Szerző:*

Kertész János

programtervező informatikus BSc

*Budapest, 2025*

## SZAKDOLGOZAT TÉMABEJELENTŐ

### Hallgató adatai:

Név: Kertész János

Neptun kód: AM2VZ8

### Képzési adatok:

Szak: programtervező informatikus, alapképzés (BA/BSc/BProf)

Tagozat : Nappali

Belső témavezetővel rendelkezem

### Témavezető neve: Pintér Balázs

munkahelyének neve, tanszéke: ELTE IK, Programozásmélet és Szoftvertechnológia Tanszék

munkahelyének címe: 1117, Budapest, Pázmány Péter sétány 1/C.

beosztás és iskolai végzettsége: egyetemi adjunktus, PhD

A szakdolgozat címe: Shalendar

### A szakdolgozat témája:

(A témavezetővel konzultálva adja meg 1/2 - 1 oldal terjedelemben szakdolgozat témájának leírását )

A dolgozat témája egy mindennapi életben használható time management szoftver megvalósítása. A projekt felépítése három fő felületre és egy bejelentkezési oldalra oszlik, az utóbbi lehetővé teszi több felhasználó számára a profil kezelését és az egymás közötti naptár megosztást.

#### Főoldal:

Az oldalon egy naptár található, mellette pedig minimum egy oszlop, amely a feladatkezelő szoftverekből ismert lista formátumot követi. A felhasználó igényei szerint több, saját tematikáinak megfelelő oszlopot is felvehet. Az oszlopban kártyák helyezhetők el, amelyeket a felhasználó szabadon hozzáadhat vagy törölhet. A kártyákon kötelezően megadható cím, valamint opcionálisan kezdeti dátumok, határidők és prioritások. Ezek a kártyák a naptár megfelelő napjaira húzhatók, így segítve a feladatok ütemezését.

#### Napi nézet:

A naptár adott napjára kattintva megjelenik a nap részletezése. Itt két lista található: Az egyik lista egy időjelző sávval rendelkező feladatlista, amelyben a naphoz tartozó, időponthoz kötött feladatok jelennek meg. A másik lista olyan teendőket tartalmaz, amelyek nem kötöttek időponthoz. Mindkét listában a feladatok „elvégeztnek” jelölhetők.

#### Több naptár kezelése:

A naptár mellett található egy plusz gomb, amely lehetővé teszi több naptár létrehozását és kezelését. Ezekhez a naptárakhoz további felhasználók is hozzáadhatók, valamint lehetőség nyílik a naptárakból egyes kártyák vagy teljes naptárak importálására a saját naptárba.

Budapest, 2024. 10. 05.

# Tartalomjegyzék

<b>1. Bevezetés</b>	<b>4</b>
1.1. Problémafelvetés és motiváció . . . . .	4
1.2. Célkitűzés . . . . .	5
1.3. A dolgozat felépítése . . . . .	5
1.4. Fogalmi és formai irányelvek . . . . .	5
1.4.1. Fogalomhasználat . . . . .	5
1.4.2. Stíluskonvenciók . . . . .	7
<b>2. Felhasználói dokumentáció</b>	<b>9</b>
2.1. A szoftver ismertetése . . . . .	9
2.2. Célközönség . . . . .	10
2.3. Általános felhasználói tájékoztató . . . . .	10
2.4. A rendszer funkcióinak bemutatása oldalanként . . . . .	11
2.4.1. Regisztráció és Bejelentkezés . . . . .	11
2.4.2. Navigációs sáv funkciói . . . . .	12
2.4.3. Főoldal funkciói . . . . .	13
2.4.4. Napi nézet funkciói . . . . .	15
2.4.5. Ticketek funkciói . . . . .	18
2.4.6. Naptárak funkciói . . . . .	19
2.4.7. Profil funkciói . . . . .	19
2.5. Futás közbeni rendszerüzenetek . . . . .	20
2.5.1. Hibaüzenetek és jelentésük . . . . .	20
2.5.2. Figyelmeztetések . . . . .	20
<b>3. Fejlesztői dokumentáció - Tervezés és megvalósítás</b>	<b>21</b>
3.1. Rendszer architektúrája . . . . .	21
3.1.1. Alrendszerek és rétegek szerepei és felelősségei . . . . .	21
3.1.2. Alkalmazott technológiák és eszközök . . . . .	22

3.1.3.	Fejlesztési módszertan . . . . .	24
3.1.4.	Csomag diagram ( <i>UML</i> ) . . . . .	25
3.2.	A rendszer működése . . . . .	26
3.2.1.	Kommunikációs diagram ( <i>UML</i> ) . . . . .	26
3.2.2.	Tevékenységi diagram ( <i>UML</i> ) . . . . .	27
3.3.	Adatbázis modell . . . . .	29
3.3.1.	Az adatmodell áttekintése . . . . .	29
3.3.2.	Tábla-szintű leírás . . . . .	29
3.4.	Modul- és osztályszerkezet . . . . .	32
3.4.1.	<i>Backend</i> modulok és rétegek . . . . .	32
3.4.2.	Osztálydiagram ( <i>UML</i> ) . . . . .	33
3.4.3.	Főbb osztályok leírása és implementációja . . . . .	34
3.5.	A felhasználói felület . . . . .	38
3.5.1.	Csomagdiagram ( <i>UML</i> ) . . . . .	39
3.5.2.	Képernyők navigációs logikája . . . . .	39
3.5.3.	Felhasználói események kezelése . . . . .	41
3.6.	Telepítési folyamat leírása . . . . .	42
3.6.1.	Fejlesztési telepítési útmutató . . . . .	42
<b>4.</b>	<b>Fejlesztői dokumentáció - Tesztelés</b>	<b>43</b>
4.1.	Tesztelési stratégia . . . . .	43
4.1.1.	Tesztfájlok fizikai elhelyezkedése ( <i>UML</i> ) . . . . .	44
4.1.2.	Teszteléshez használt eszközök . . . . .	44
4.2.	Egységtesztek . . . . .	45
4.3.	Integrációs tesztek . . . . .	46
4.4.	Rendszertesztek - automatizált . . . . .	46
4.5.	Rendszertesztek - Manuális . . . . .	46
4.6.	Tesztelési eredmények . . . . .	54
4.6.1.	<i>Backend coverage</i> . . . . .	54
4.6.2.	<i>Frontend e2e</i> teszt megtekintése . . . . .	57
<b>5.</b>	<b>Összegzés</b>	<b>58</b>
5.1.	A projekt céljának összefoglalása . . . . .	58
5.2.	További fejlesztési lehetőségek . . . . .	58
5.3.	Személyes tapasztalat . . . . .	59

5.4. Zárógondolat . . . . .	59
-----------------------------	----

# 1. fejezet

## Bevezetés

A szakdolgozat alapja egy *time management* alkalmazás, amely a professzionális *kanban board*hoz hasonló módszertant ötvözi a hagyományos naptárvezetési szokásokkal. A szoftveren túl belátást nyerünk a motivációtól kezdve az apró tervezési részleteken át a komplex és robusztus rendszer harmonikus működésének mivoltjába is. A felhasználói dokumentációban színes és letisztult ábrák segítik a megértést a mindennapi felhasználó számára, míg a fejlesztői dokumentációban a legapróbb részleteket is feltáró diagramok, valamint azokhoz kapcsolódó leírások támogatják a szakértő szemeket.

### 1.1. Problémafelvetés és motiváció

Számos szoftver létezik, amely segít a felhasználó időbeosztásában, azonban gyakran ezek nagyon komplex, úgynevezett *enterprise* eszközök. Az átlagember számára nehezen átlátható rendszerek. Sok *kanban* típusú táblához naptárnézet is elérhető, azonban az ottani tervezés, *Ticket* mozgatás meglehetősen körülményes. A *Shalendar* célja megtalálni az egyensúlyt a letisztult, könnyen érthető szoftver dizájn és a professzionális *time management* szoftverek által kínált lehetőségek között egy átlag felhasználó számára. A *Ticketek* közvetlen naptár napjára való kiosztása és az ott található *ScheduledList* valamint *ToDoList* párhuzamos megléte is viszonylag ritka, de praktikus megoldás. A szoftver elkészítésének motivációja ezen "hézagok" kitöltése valamint a kevésbé elterjedt funkciók kombinációjának kipróbálása.

## 1.2. Célkitűzés

A szoftver célja, hogy a felhasználók egyszerűen létrehozassanak saját naptárakat, kezelhessék feladataikat egy személyre szabható környezetben, valamint együttműködhessenek más felhasználókkal. A közös programok könnyedén importálhatóak saját naptárba egy megosztottból. Az engedélykezelés lehetővé teszi, hogy biztonságosan meg lehessen osztani naptárakat nagyobb csoportokkal.

## 1.3. A dolgozat felépítése

A diplomamunka célja nem csupán egy működő alkalmazás létrehozása, hanem annak bemutatása is mind a felhasználó, mind a fejlesztő részére. Továbbá, hogy hogyan lehet egy gyakorlati problémára átfogó, jól dokumentált, moduláris szoftveres megoldást készíteni a legmodernebb webes technológiák segítségével.

A szakdolgozat első része felhasználói szempontból mutatja be a rendszer működését. Ezt követi a fejlesztői dokumentáció, amely részletesen tárgyalja a rendszer architektúráját, az alkalmazott technológiákat és a fejlesztés lépéseit. A dolgozat végén a rendszer tesztelésének eredményei olvashatóak.

## 1.4. Fogalmi és formai irányelvek

A dolgozatban az alábbi formai és jelentésbeli konvenciók alkalmazása következetesen történik. Amennyiben egy fogalom nincs megmagyarázva annak közvetlen használata után, vagy ismétlődő, megtalálható az alábbi fogalomhasználati leírásban.

### 1.4.1. Fogalomhasználat

A dolgozatban használt fogalmak két csoportra oszthatók: az első részben olyan elnevezések szerepelnek, amelyek a *Shalendar* rendszeréhez szorosan kötődnek, és egyedi módon kerülnek definiálásra. A második részben pedig olyan általánosan elfogadott informatikai vagy fejlesztői terminológiák szerepelnek, amelyek konvencionális jelentéssel bírnak.

## Szoftverspecifikus elnevezések

Ezen elnevezések minden említéskor nagy kezdőbetűvel jelennek meg.

- **„Ticket”** - A felhasználók által létrehozott egyedi teendőket jelenti, amelyek tartalmazhatnak címet, leírást, prioritást, naptári napra való kiosztás után kezdő és befejező időpontot.
- **„Dashboard”** - A főoldali nézet elnevezése, ahol a felhasználó a naptárt és a hozzá tartozó listákat látja, és interakcióba léphet velük.
- **„CalendarView”** - A napra kiosztott *Ticketek* megjelenítésére, valamint a naptárral kapcsolatos interakciókra szolgáló komponens. A *Dashboard* elsődleges eleme.
- **„CalendarLists”** - Ez a *Dashboard* második eleme. A *CalendarListek* kezelésére szolgál.
- **„CalendarList”** Egy tematikusan elkülönített lista, amely több *Ticketet* tartalmazhat. Felhasználónként több *CalendarList* is létrehozható.
- **„DayView”** - Egy adott nap részletes nézete, ahol az időponthoz kötött és kötés nélküli teendők külön oszlopban jelennek meg.
- **„ScheduledList” & „ToDoList”** - A *DayView* részeként megjelenő két lista, ahol a *ScheduledList* az időponthoz kötött, a *ToDoList* pedig a szabadidejű feladatokat tartalmazza.
- **„Calendars”** - Azon oldal melyet egy adott naptár *Manage* gombjáról lehet elérni, a naptárak kezelésére szolgál.
- **„Owner”, „Write”, „Read”** - A naptárakhoz kötött engedélyek szintjei, sorban tulajdonosi (mindent csinálhat), írói (a *CalendarListekkel* valamint naptárengedélyekkel kapcsolatos műveleteken kívül tevékenykedhet), olvasói (csak a naptár megtekintésére, és annak/elemeinek másolására korlátozott engedélyszint).

## Általánosan elfogadott terminológia

- **„Backend”** - A rendszer háttérreége, amely a szerveroldali logikát, adatkezelést és API-kiszolgálást valósítja meg.



- **„Frontend”** - A felhasználó által közvetlenül látott és használt felület, amely lehetővé teszi az interakciót a rendszerrel.
- **„Szerver”** - Egy fizikai vagy virtuális gép (illetve szolgáltatás), amelyen a backend fut. Fogadja a *kliensek* kéréseit, és válaszokat küld vissza.
- **„Kliens”** - Az a program vagy eszköz, amely kapcsolatba lép a szerverrel. Jellemzően ebben fut a *frontend*.
- **„Modal”** - Egy felugró ablak, amely új entitások (pl. *Ticket*, naptár) létrehozását vagy meglévők szerkesztését teszi lehetővé.
- **„Navbar”** - Az oldal tetején található navigációs sáv, amely a főbb nézetek közötti váltást teszi lehetővé.
- **„Drag-and-drop”** - Egy felhasználói interakciós minta, amely lehetővé teszi objektumok (pl. *Ticketek*) mozgatását a felhasználói felületen.
- **„UML”** - *Unified Modeling Language*, azaz egységes modellezési nyelv, amely vizuálisan írja le a szoftver struktúráját vagy működését.
- **„Happy-path”** - Az az eset, amikor egy funkció a várt módon, hibák nélkül teljesül.
- **„Edge-case”** - Szélsőséges, ritkán előforduló, vagy különleges bemeneti/környezeti feltételek.
- **„e2e”** - Egy tesztelési stratégia. Az *end-to-end* teszt a teljes rendszer működését vizsgálja a felhasználói interakcióktól kezdve egészen a *backend* válaszokig, mintha egy valódi felhasználó használná az alkalmazást.

#### 1.4.2. Stíluskonvenciók

- Fekete aláhúzott szöveg - A dokumentáción belüli hivatkozásokat jelöli, amennyiben rájuk kattintunk a hivatkozott részhez ugrik.
- *Dólt betűs szöveg* - Általános fogalmakat, technológiákat jelöl (pl. *Navbar*, *drag-and-drop*, *DayView*).
- **Kódformátum** - Kódrészletek, fájlnevek, változók, vagy osztálynevek esetén használatos (pl. `JwtHelper`, `calendarId`, `App.vue`).

- **Ábrák magyarázata** - A leírások mindig az ábrák előtt találhatóak, mivel ezek sokszor a megértéshez szükséges információkat tartalmazznak.
- **"(x.y)" típusú jelölések** - A szöveghez tartozó ábrát jelölik, ezzel könnyebben értelmezhető melyik szöveg melyik ábrához tartozik.

## 2. fejezet

# Felhasználói dokumentáció

### 2.1. A szoftver ismertetése

A *Shalendar* egy webes alkalmazásként működő időmenedzsment eszköz, amely célja, hogy a felhasználók számára egyesítse a professzionális teendőlista-kezelés előnyeit a hagyományos naptár alapú időbeosztással. A rendszer lehetőséget biztosít egyéni és közös naptárak létrehozására, valamint ezekhez kapcsolódó feladatlisták és események kezelésére.

Az alkalmazás három fő felületből épül fel: a nyitó oldal (*Dashboard*), a napi nézet (*DayView*), valamint a több naptár kezelő nézet (*Calendars*). Emellett rendelkezik egy bejelentkezési és regisztrációs oldallal valamint egy a felhasználói profil kezelésére szolgálóval.

A *Dashboard* felületen a felhasználók létrehozhatnak tematikus feladatlistákat (*CalendarLists*) a kiválasztott naptárhoz, és az ezekhez tartozó feladatokat (*Ticketeket*) *drag-and-drop* technikával akár közvetlenül a naptár egy adott napjára is ütemezhetik. A napi nézetben az időponthoz kötött és kötés nélküli teendők elkülönítve jelennek meg, így biztosítva a részletes, mégis áttekinthető napirendet.

A *DayView* felület a kiválasztott nap teendőinek részletes kezelésére szolgál. Két elkülönített oszlopot tartalmaz: az első az időponthoz rendelt teendőket jeleníti meg (*ScheduledList*), míg a második azokat a feladatokat, amelyekhez nem tartozik konkrét időintervallum (*ToDoList*).

A *Calendars* nézet a felhasználó által létrehozott és megosztott naptárak áttekintésére és kezelésére szolgál. Ebben a felületen lehetőség nyílik új naptárak létrehozására, meglévők törlésére, valamint más felhasználók meghívására és jogosultságainak

beállítására. A megosztott naptárakon végzett módosítások azonnal szinkronizálódnak minden érintett felhasználónál.

## 2.2. Célközönség

A program elsődlegesen a rendszerezettségre törekvő magánszemélyek számára készült. Azokat célozza meg, akik szeretik előre, vizuálisan átlátni napjaikat, heteiket vagy akár teljes hónapjukat is, és igénylik a feladataik egy helyen történő kezelését. A célcsoport jellemzően a 18-60 év közötti korosztály, amely aktívan használ digitális eszközöket, és értékeli az egyszerű, de hatékony időgazdálkodási eszközöket.

A rendszer különösen hasznos lehet:

- diákoknak, akik párhuzamosan kezelik tanulmányi és magánéleti feladataikat
- fiatal felnőtteknek, akik munka, tanulás és személyes projektek között szeretnének strukturált áttekintést
- szabadúszóknak vagy kreatív szakembereknek, akik projektalapú munkát végeznek, és fontos számukra a naptár és a feladatlista kombinációja
- olyan felhasználóknak, akik csapatban is dolgoznak, de nem szeretnének bonyolult vállalati szoftvereket használni

A *Shalendarnak* nem célja a nagyvállalati projektmenedzsment kiváltása, ehelyett egy könnyen kezelhető, személyes időmenedzsment platformként szolgál, amely kiscsoportos együttműködésre is lehetőséget biztosít.

## 2.3. Általános felhasználói tájékoztató

Az oldal tartalmaz *tooltipeket*. Amennyiben a felhasználó egy elem fölé helyezi a kurzorát és kis ideig nem mozgatja, megjelenik egy üzenet amely segít az ott elérhető interakciók értelmezésében.

Az oldalon található *Ticketekkel* kapcsolatban *drag-and-drop* mechanizmus él. Azaz a felhasználó az egér lenyomásával meg tudja fogni az adott *Ticketet* és mozgatni azt. (pl.: naptárba kiosztani, vagy az adott listán belül újrapozicionálni.)

A kötelezően kitöltendő input mezők neve mellett piros csillag jelenik meg. Amennyiben ezek mégsem kerülnek kitöltésre hibaüzenetek jelzik azok hiányát.

## 2.4. A rendszer funkcióinak bemutatása oldalanként

(2.1., 2.2., 2.3., 2.4., 2.5., 2.6., 2.7.) A képeken nyilak jelzik a felhasználói interakciót és annak következményeit. Amennyiben *type* szerepel a nyílon, abban az esetben csak az input mezők kitöltése történik. Egyébként pedig a kattintás következményét tárja elénk. A szaggatott nyíl a *drag-and-drop* funkcionalitást jelzi.

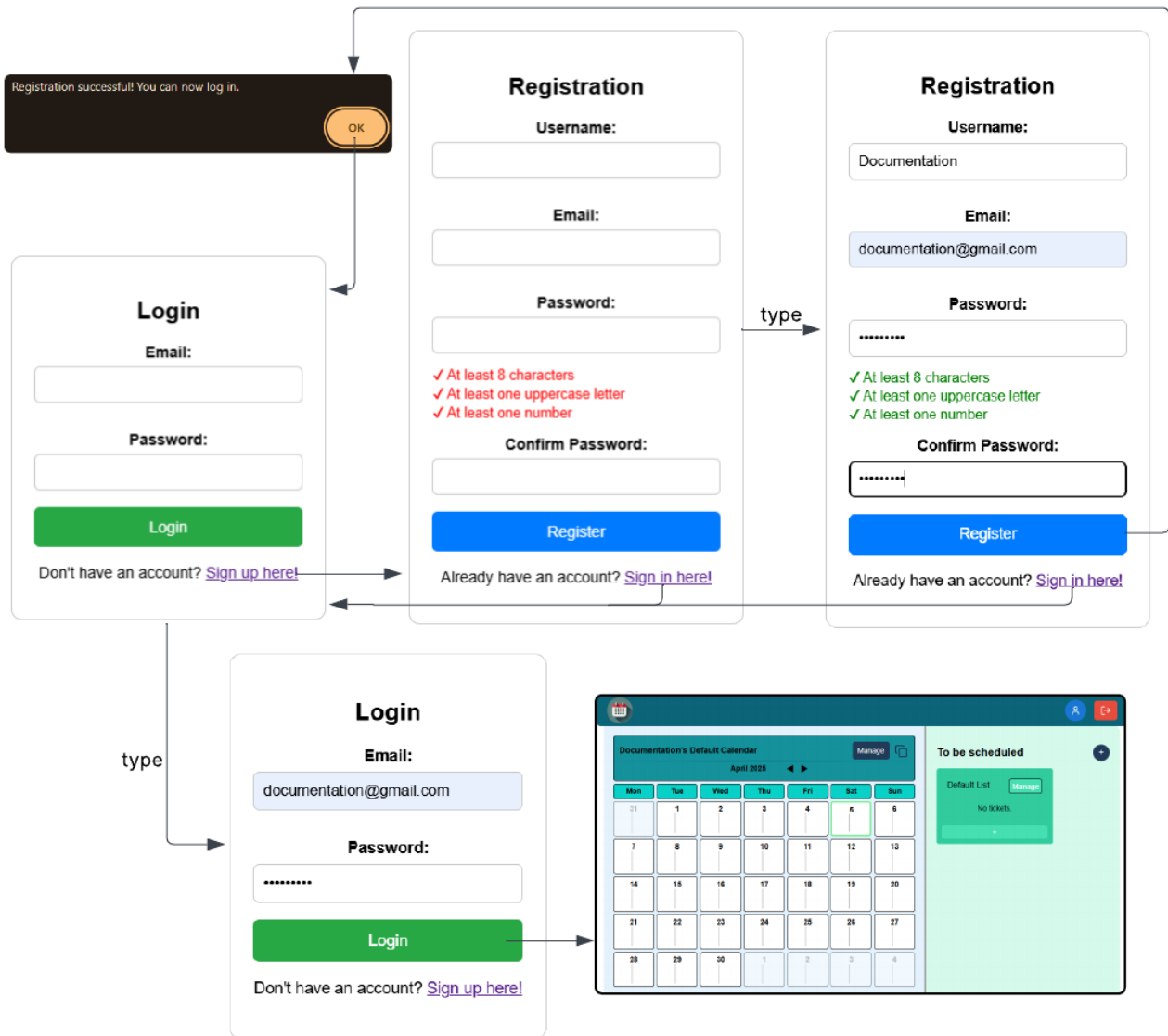
A fő oldalak fekete egybefüggő kerettel vannak jelölve, míg a felugró ablakok (*Modalok*) szaggatott sötétkéssel.

Az adott oldal használatának részletes megértésében minden ábra előtt rövid magyarázó szöveg segít.

### 2.4.1. Regisztráció és Bejelentkezés

(2.1) Az oldal megnyitásakor a bejelentkezési oldal nyílik meg, ahova email címmel és a regisztrációnál megadott jelszóval lehet belépni.

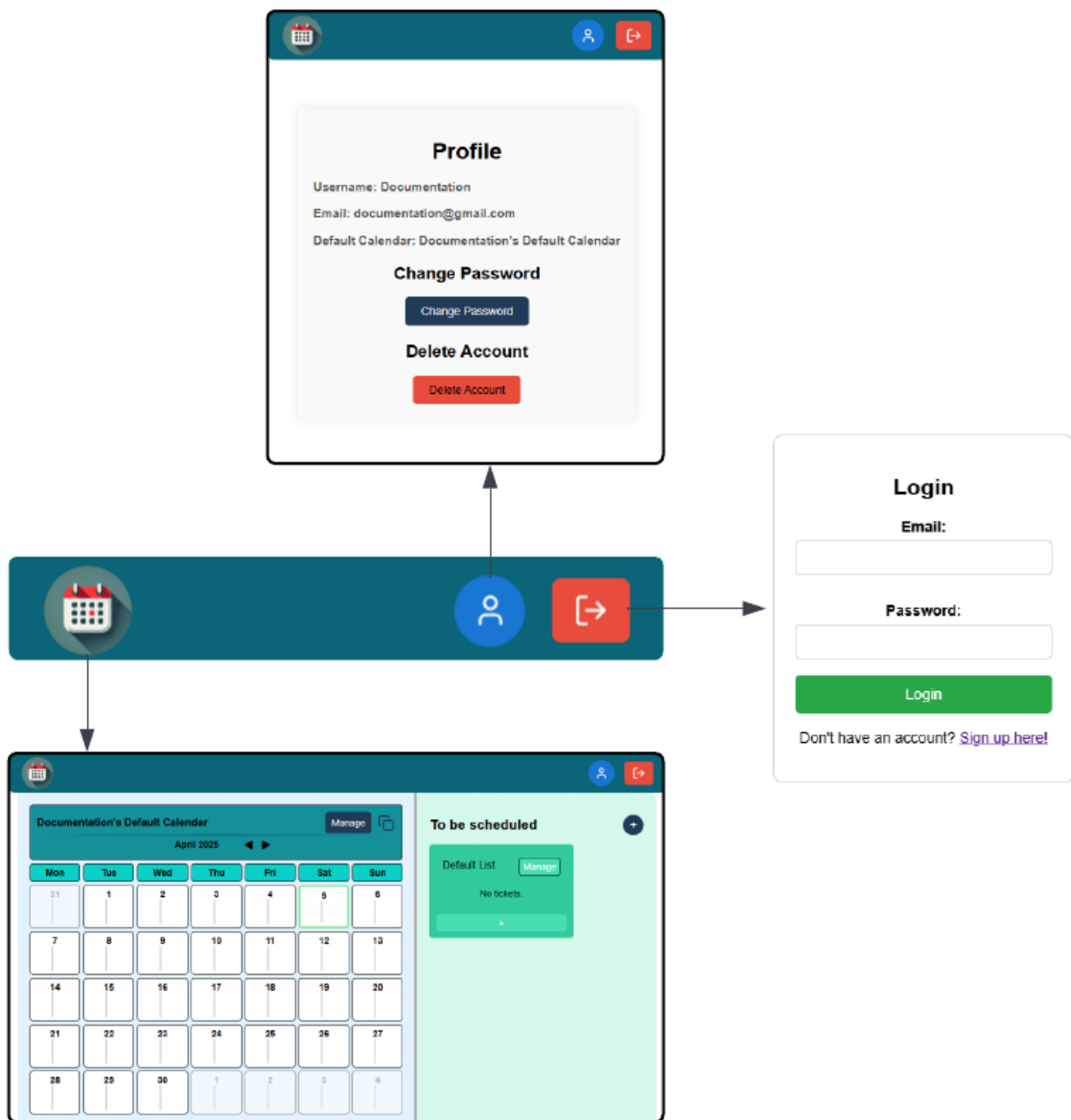
Regisztráció során egy még nem regisztrált felhasználót lehet regisztrálni a képernyőn látható jelszó kritériumok mellett. A frissen regisztrált felhasználó egy alap naptárral és egy hozzá tartozó listával kezd. Melyet a bejelentkezés után fog látni.



2.1. ábra. Registration & Login

## 2.4.2. Navigációs sáv funkciói

(2.2) A bal oldali naptár ikonnal lehet bármikor visszatérni a *Dashboard*-ra. A jobb oldali gombok a profil valamint a kijelentkezés funkciók.



2.2. ábra. *Navbar*

### 2.4.3. Főoldal funkciói

<sup>(2.3)</sup> A naptár másolása gombra kattintva (📅) megnyílik az ábrán látható *Modal*, melynek a legördülő listáján a felhasználó *Owner* és *Write* engedéllyel rendelkező naptárjai jelennek meg. A *Copy* gomb megnyomásával a naptár összes *Tickete* másolásra kerül a duplikátumok szűrésével.

A naptár bármelyik napjára való kattintással az adott naphoz tartozó DayView jön elő.

A szaggatott nyilak azt jelölik, hogy a *Ticketek* a naptár napjaira szabadon kioszthatóak, vagy az adott listán belül átrendezhetőek.





2.3. ábra. Dashboard

#### 2.4.4. Napi nézet funkciói

(2.4) Amennyiben új *Ticketet* szeretne felvenni a felhasználó a konzisztens műkö-

dés érdekében szükséges azt valamelyik tematikus listájához kötni.

A *Ticket* helye attól függ, hogy meg lett-e adva időintervallum. Amennyiben igen, automatikusan a *ScheduledListbe* kerül, ellenkező esetben a *TodoListbe*.



2.4. ábra. DayView

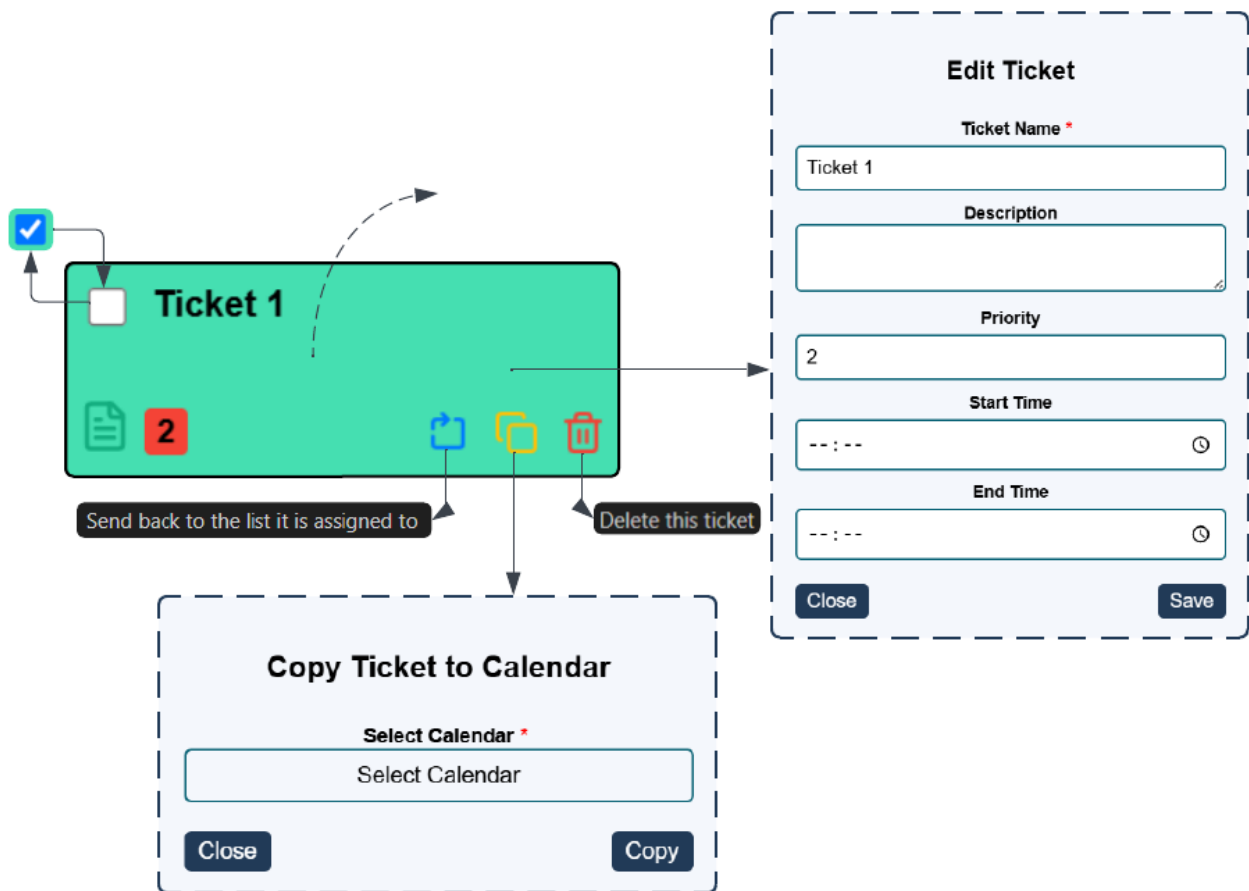
### 2.4.5. Ticketek funkciói

<sup>(2.5)</sup> A *Ticketek* funkciói kibővülnek a *DayView* nézetben. A *Dashboardon* annyi eltérés van, hogy nem látható a visszaküldés gomb (🔄), valamint a *Ticket* szerkesztésénél nem állítható kezdő és záró időpont.

A *Ticket* másolását jelző gomb (📄) megnyomását követően megnyílik az ábrán látható *Modal*, melynek a legördülő listáján a felhasználó *Owner* és *Write* engedéllyel rendelkező naptárai jelennek meg, a *Copy* gomb megnyomásával az adott *Ticket* másolásra kerül amennyiben egy pontosan olyan még nem létezik a naptárban amibe másoltuk.

*DayViewban* való *Ticket* szerkesztés esetén az időpont hozzáadásával áthelyezhető a másik listába a *Ticket*, valamint az időpont kitörlésével az ellenkező irányba.

A jegy elvégzettnek jelölhető, a *CalendarViewban* ezek a *Ticketek* halványabban és áthúzva jelennek meg.

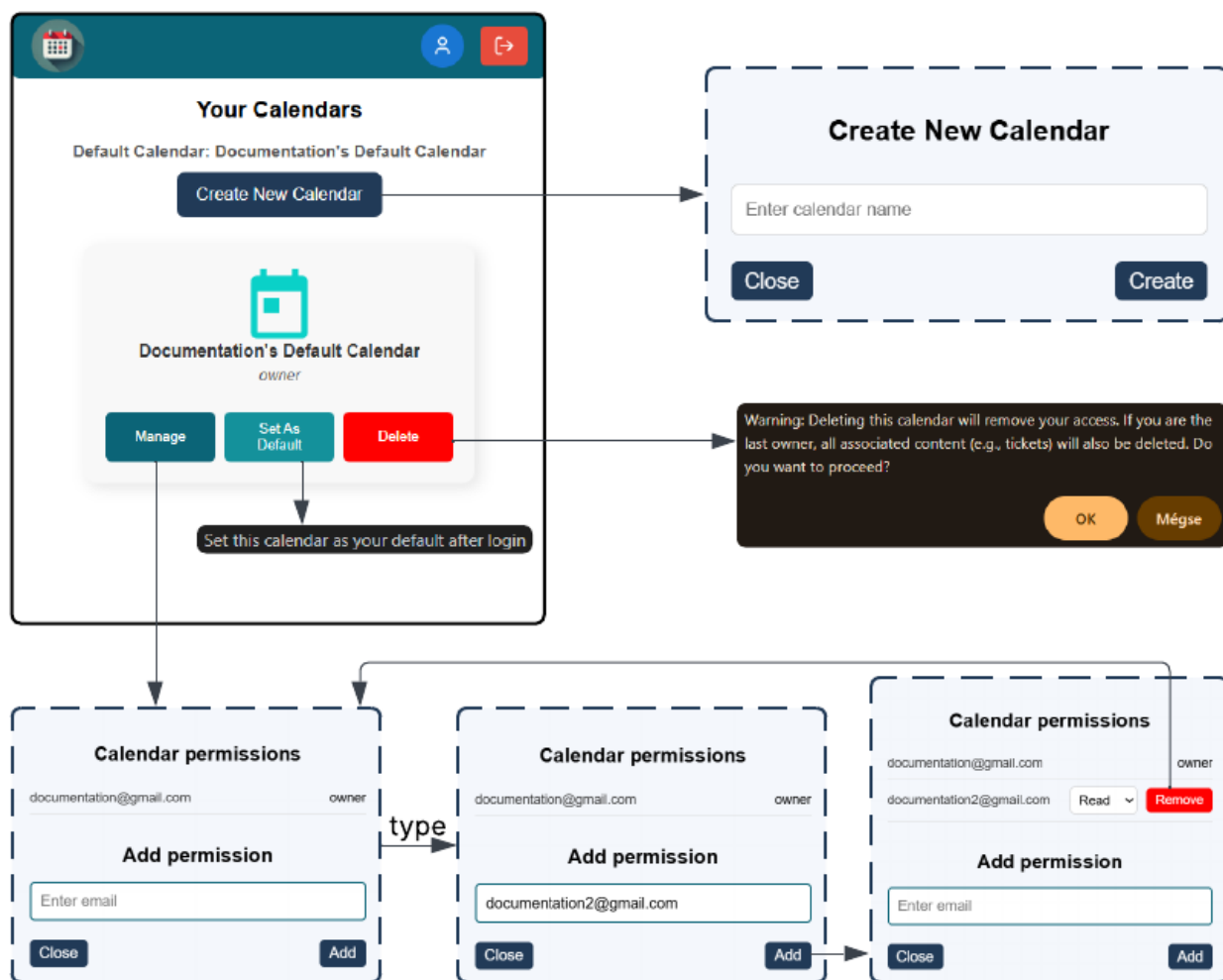


2.5. ábra. *Ticketek*

### 2.4.6. Naptárak funkciói

<sup>(2.6)</sup> Amennyiben a felhasználó új profilt rendel a naptárjához *Read* engedéllyel kerül bele, melyet később tud szabadon módosítani.

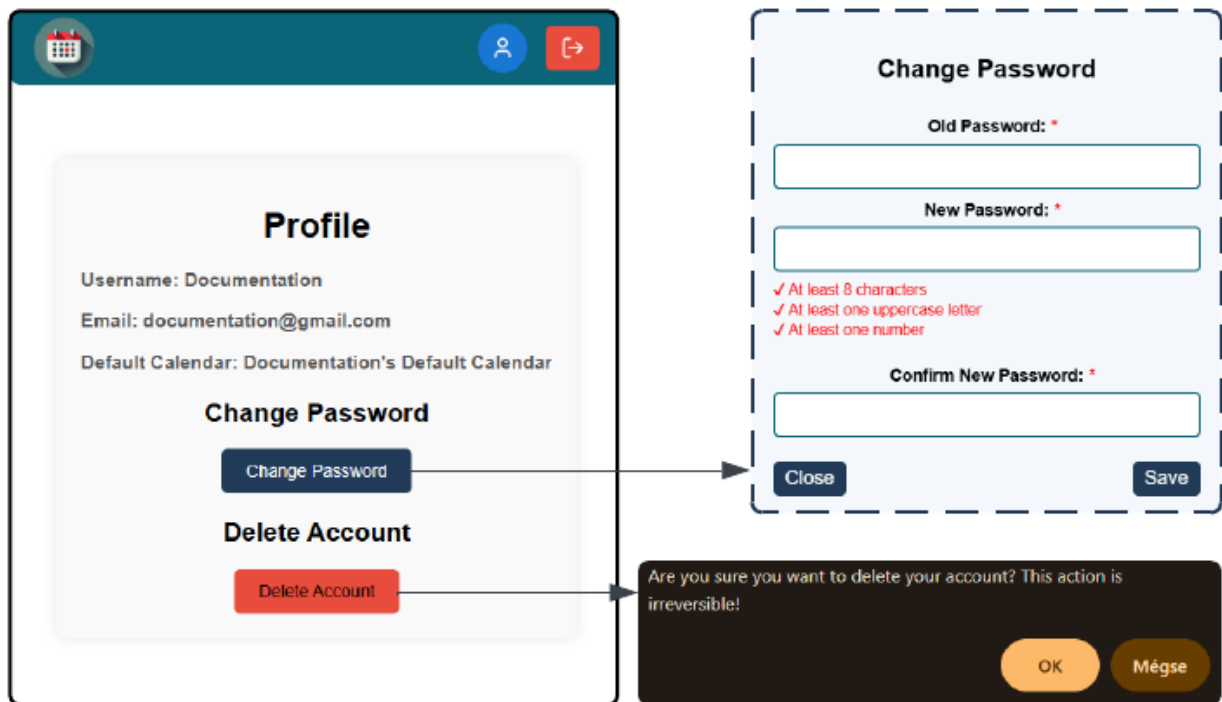
A naptár ténylegesen csak akkor törlődik ha a felhasználó az utolsó *Owner* jogosultsággal rendelkező felhasználó, ellenkező esetben csak a saját hozzáférését semlegesíti meg.



2.6. ábra. *Calendars*

### 2.4.7. Profil funkciói

<sup>(2.7)</sup> A profil törlése végleges vagyis a hozzá tartozó naptárak elvesznek, amennyiben a törölt profil volt az utolsó *Owner* jogosultsággal rendelkező felhasználó. Így az alacsonyabb jogosultságokkal rendelkezők sem fogják látni többé a naptárat.



2.7. ábra. Profile

## 2.5. Futás közbeni rendszerüzenetek

### 2.5.1. Hibaüzenetek és jelentésük

A hibaüzenet valamilyen kritikus hibára hívja fel a figyelmet. Leggyakrabban akkor ha valamilyen adat helytelenül kerül kitöltésre, vagy amennyiben egy olyan naptárban szeretne interakciót végrehajtani a felhasználó amelyhez nincs megfelelően magas engedélye.

A hibaüzenetek megjelenésüket követően 5 másodperccel eltűnnek.

### 2.5.2. Figyelmeztetések

Az oldal fontos, vissza nem csinálható műveletek előtt megerősítést kér arról, hogy a felhasználó biztosan el szeretné-e végezni az adott műveletet. Ilyen például *CalendarListek*, naptárak vagy felhasználói profilok törlése.

## 3. fejezet

# Fejlesztői dokumentáció - Tervezés és megvalósítás

### 3.1. Rendszer architektúrája

Az alkalmazás a *kliens-szerver* modellt követve 3 fő komponensből áll: a *Microsoft SQL Server* adatbázis, az *ASP.NET Web API* alapú *backend* és a *Vue.js*-alapú *frontend*. Az első két komponens a szerveret, míg a harmadik a *klienst* képviseli.

A 3 komponens tisztán szétválasztható, ezzel biztosítva a moduláris fejlesztést. A *frontend* és a *backend API* hívások segítségével kommunikál. A *backend* és az adatbázis pedig közvetlen kapcsolatban állnak az *Entity Framework*nek köszönhetően.

#### 3.1.1. Alrendszerek és rétegek szerepei és felelősségei

##### ***Frontend***

Feladata a felhasználói interakciók kezelése valamint az üzleti logika megjelenítése. Utóbbi elérésére *axios* HTTP kéréseken keresztül történik. A kérés *headerjébe* automatikusan integrálja az autentikációhoz szükséges adatokat (*JWT token*) valamint a naptár azonosítóját. Ezzel segítve, hogy a felhasználó csak számára elérhető adatokhoz férhessen hozzá.

## **Backend API**

Felelősségei közé tartozik, hogy a *JWT token* megfelelően generálva legyen a bejelentkezés során, tartalmazza a felhasználó azonosítóját, email címét, egyedi azonosítóját (pl. *GUID*) valamint a felhasználó naptárakhoz való engedélyeit a token megszokott jellemzői mellett. (*Issuer, Audience, Expiration...*)

Feladata továbbá, hogy kezelje a *kliens* által küldött HTTP kéréseket. Ahol szükséges ellenőriznie, hogy a felhasználó rendelkezik-e érvényes tokennel, valamint a naptárakkal kapcsolatos tevékenységes esetén figyelje az írási (*Write*), olvasási (*Read*) vagy tulajdonosi (*Owner*) engedélyek meglétét. A token ellenőrzését a beépített `[Authorize]` attribútummal végzi. Az engedély ellenőrzés pedig az adatbázisban szereplő engedélyek és a fejlécben kapott naptár azonosító összehasonlításával történik.

Amikor megtörtént az adatok validálása és az üzleti logika végrehajtása a *backend* feladata, hogy értesítse az összes *klienst* a változásról amely az adott naptár valamelyik nézetén tartózkodik .

## **Adatbázis**

Az adatbázis szerepe, hogy hosszútávon, jól strukturáltan tárolja az adatokat a felhasználókról valamint azok naptáiról, engedélyeiről, *Ticketeiről*.

Indexek, kulcsok és idegen kulcsok valamint megszorítások segítségével biztosítsa a következetességet és teljesítményt. Ezek tényleges kapcsolatát a 3.3. Adatbázis modell című pont alatt tárgyaljuk.

### **3.1.2. Alkalmazott technológiák és eszközök**

Alábbiakban csak a főprogramban használt technológiák vannak felsorolva. A teszteléshez használtak a 4.1.2. Teszteléshez használt eszközök pontban olvashatóak.

#### **Programozási nyelvek, keretrendszerek**

- **.NET 8.0 SDK** - A *backend* teljes projektje .NET 8.0 épül.
- **C#** - *backend* logika és *API* megvalósítására.
- **ASP.NET Core** - *REST API* keretrendszer.



- **Entity Framework Core** - *ORM* a relációs adatbázis kezelésére.
- **JavaScript** - *frontend* logika.
- **Vue.js** - *JavaScript* keretrendszer.
- **HTML, CSS** - struktúra és stílus.
- **Vite** - *frontend build* és *hot-reload*.

### Fejlesztői eszközök

- **Visual Studio 2022** - *backend* fejlesztés, hibakeresés, teszt.
- **Visual Studio Code** - *frontend* fejlesztés, *Vue* komponensek.
- **Node.js** - *frontend* futtatási környezet.
- **Postman** - *APIk* kipróbálásához és manuális teszteléshez.
- **Git + GitHub** - verziókövetés.
- **SSMS** - SQL szerver kezelése, tesztelés, lekérdezések írása.

### Külső csomagok (*NuGet* és *npm*)

A projekt több külső könyvtárat használ, melyeket *NuGet* illetve *npm* segítségével kezel.

#### *Backend (NuGet csomagok):*

- **Microsoft.AspNetCore.Authentication.JwtBearer** (v8.0.0) - *JWT tokenek* feldolgozásához és hitelesítéshez.
- **Microsoft.AspNetCore.SignalR** (v1.2.0) - *kliens* oldali értesítések a *backendből SignalR hubok* felhasználásával
- **Microsoft.EntityFrameworkCore** (v9.0.2) - Az adatbázissal való kommunikáció során objektum-relációs leképázést használunk (*Entity Framework*), az az adatbázis entitások C# osztályokként jelennek meg, így *LINQ*-szintű lekérdezés is lehetséges.
- **Microsoft.EntityFrameworkCore.SqlServer** (v9.0.2) - SQL Server-specifikus *EF Core provider*.

- **Microsoft.EntityFrameworkCore.Tools** (v8.0.11) - *EF Core* migrációs és *scaffold* eszközök. Ezek csak fejlesztési célra vannak használva.
- **Microsoft.VisualStudio.Web.CodeGeneration.Design** (v8.0.7) - *scaffold* eszközök a *Web API* fejlesztéshez.
- **Swashbuckle.AspNetCore** (v6.4.0) - *Swagger* generálása és dokumentáció *REST API*-hoz.
- **System.IdentityModel.Tokens.Jwt** (v8.5.0) - *JWT* tokenek létrehozása és kezelése.

**Frontend (npm csomagok):**

- **vue** (v3.5.13) - A *Vue.js* 3 keretrendszer magja.
- **vue-router** (v4.5.0) - Oldalak közötti navigáció *Vue*-ban.
- **axios** (v1.7.9) - HTTP kliens API hívásokhoz.
- **@microsoft/signalr** (v8.0.7) - SignalR a kliens valós idejű frissítéséhez. A *backend*ől érkező értesítések alapján
- **jwt-decode** (v4.0.0) - *JWT* tokenek tartalmának *frontend* oldali dekódolása.
- **lucide-vue-next** (v0.479.0) - Ikonkészlet *Vue* 3-hoz.
- **vuedraggable** (v4.1.0) - *Drag-and-drop* funkcionalitás *Vue* komponensekhez.

**Fejlesztői függőségek:**

- **vite** (v6.0.11) - *Build* és *hot-reload* rendszer *Vue*-hoz.
- **@vitejs/plugin-vue** (v5.2.1) - *Vue* támogatás *Vite*-hez.
- **vite-plugin-vue-devtools** (v7.7.1) - *Vue* fejlesztői eszközök bővítménye.

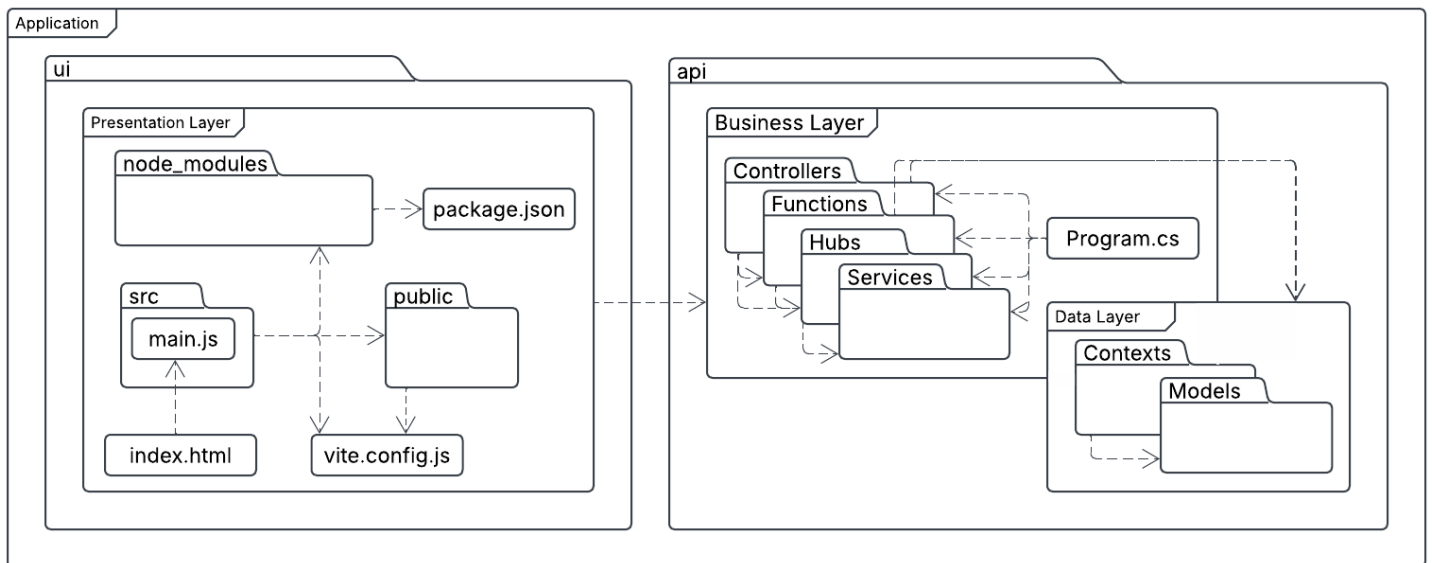
### 3.1.3. Fejlesztési módszertan

A fejlesztés elsődleges célja egy működőképes *MVP* (minimális prototípus) létrehozása volt, amely kizárólag az alapvető funkcionalitásra koncentrált. Ennek elkészültét követően fokozatosan kerültek bevezetésre a további funkciók, tematikusan

a nézetek szerint csoportosítva. Minden új funkció implementálása manuális teszteléssel, szükség esetén hibakereséssel és *refaktorálással* zárult, mielőtt a következő fejlesztési egység elkezdődött volna. A felhasználói felület (*UI*) szépítése és a felhasználói élmény (*UX*) kialakítása csak a funkcionalitás teljes implementációját követően kezdődött meg. Végül a rendszerhez egység- (*unit*), integrációs- (*integration*) és *e2e* rendszertesztetek is készültek.

### 3.1.4. Csomag diagram (*UML*)

(3.1.) A csomag diagramokban amennyiben a mappa ikon látható (📁), egy valós fizikai mappára utal, amennyiben a másik jelölés (📁) akkor pedig egy nem fizikai azaz logikai egységről beszélünk. Minden egyéb jelölés valós fizikai fájlokat takar. A nyíl a importáló csomagtól közvetlenül az importált elemre mutat. Amennyiben nem egy elemre hanem egy logikai rétegre/mappára mutat az adott komponens mindegyike importálhatja azt.



3.1. ábra. Applikáció szintű csomag diagram

A program belépési pontja az `index.html`, amely a `src/main.js` mappáját importálja, a részletesebb megértés érdekében a [3.5.1. Csomagdiagram \(UML\)](#) pont alatt a 3.20. ábra mutatja az `src` mappa csomag diagramját. A *backend* diagram megfelelően lefedi az összes szükséges információt, így annak egyik eleme sem kerül később csomag szinten kifejtésre.

## 3.2. A rendszer működése

(3.2., 3.3, 3.4, 3.5) A működés 2 féle diagrammal kerül bemutatásra, az első típus<sup>(3.2., 3.3)</sup> hangsúlyt fektet a kommunikáció módjának ismertetésére míg a második típus<sup>(3.4., 3.5)</sup> a kommunikáció sorrendjét, valamint hibakezeléseket hivatott szemléltetni.

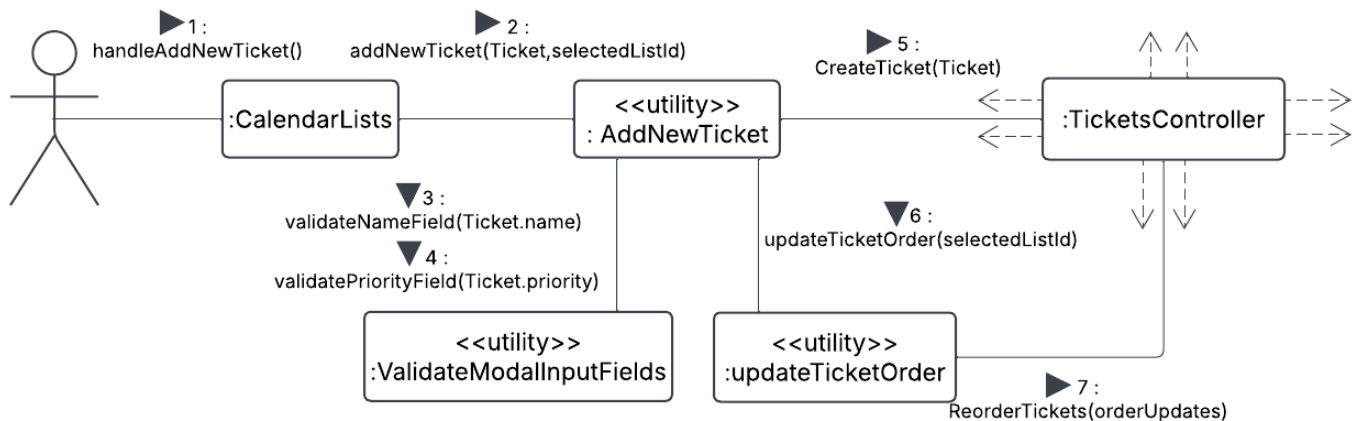
A *frontend Vue 3* keretrendszeren alapuló implementációt reprezentál, amely a komponenseket *JavaScript* objektumokként valósítja meg. Mivel ez általában névtelen objektumokat generál ezért a komponensre annak a fájlnek a nevével hivatkozik, melyben található. A folyamatban részt vesznek fontos, objektumhoz nem kötött segédfüggvények is, melyeket «utility» sztereotípiával és a fent említett névadási konvenciókkal jelenít meg a lent látható diagramokban.

A diagramok azt ábrázolják, hogy hogyan kezeli a program azt amikor a felhasználó egy új *Ticketet* kíván felvenni valamelyik *CalendarListbe*.

### 3.2.1. Kommunikációs diagram (UML)

(3.2.) Az alábbi diagramon egy sikeres folyamat működése látható. A diagramnak hála könnyen leolvashatóak a kommunikációban részt vevő objektumok kapcsolatai valamint a kommunikáció sorrendje is megjelenik (bár a hangsúly itt nem ezen van).

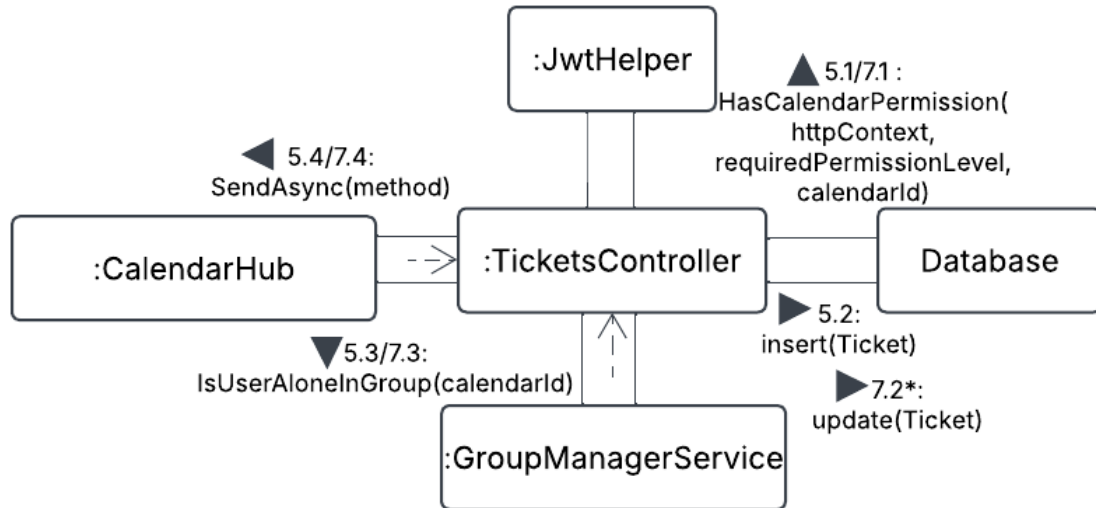
A függvényhívások paraméterezése során csak a sikeres hívás, valamint az adat továbbítás szempontjából releváns információk jelennek meg, ezzel könnyítve a diagram átláthatóságát.



3.2. ábra. Kommunikációs diagram, *kliens*

(3.3.) A kontrollerek nagy része a lent látható logikát követve épül fel. A *JwtHelper* felel a naptárral kapcsolatos engedélyek ellenőrzéséért, a *GroundManagerService*

valamint a `CalendarHub` pedig a *kliens* értesítéséről ha változás történik egy adott naptár azonosító szerinti csoportban. A 7.2-es lépésben a "\*" szimbólum az iterációs lehetőséget jelzi, azaz esetünkben a hívás a frissíteni kívánt *Ticketek* száma szerint fut le.



3.3. ábra. Kommunikációs diagram, *szerver*

### 3.2.2. Tevékenységdiagram (UML)

<sup>(3.4.)</sup>Az alábbi diagramban *ref(...)* kulcsszó jelöli azon objektumokat melyek változása esetén frissül a felhasználói interfész is. Az olvashatóság érdekében nincs feltüntetve, de ilyen objektum a *CalendarLists* is, mely szintén frissül amikor hozzáadjuk a felhasználó által meghatározott *Ticketet*.

Könnyebb követhetőség érdekében a `TicketsController` hívások kék és zöld színekkel lettek jelölve, hogy leolvasható legyen a későbbi diagramon<sup>(3.5.)</sup> is, hogy pontosan melyik tevékenység melyikhez tartozik.

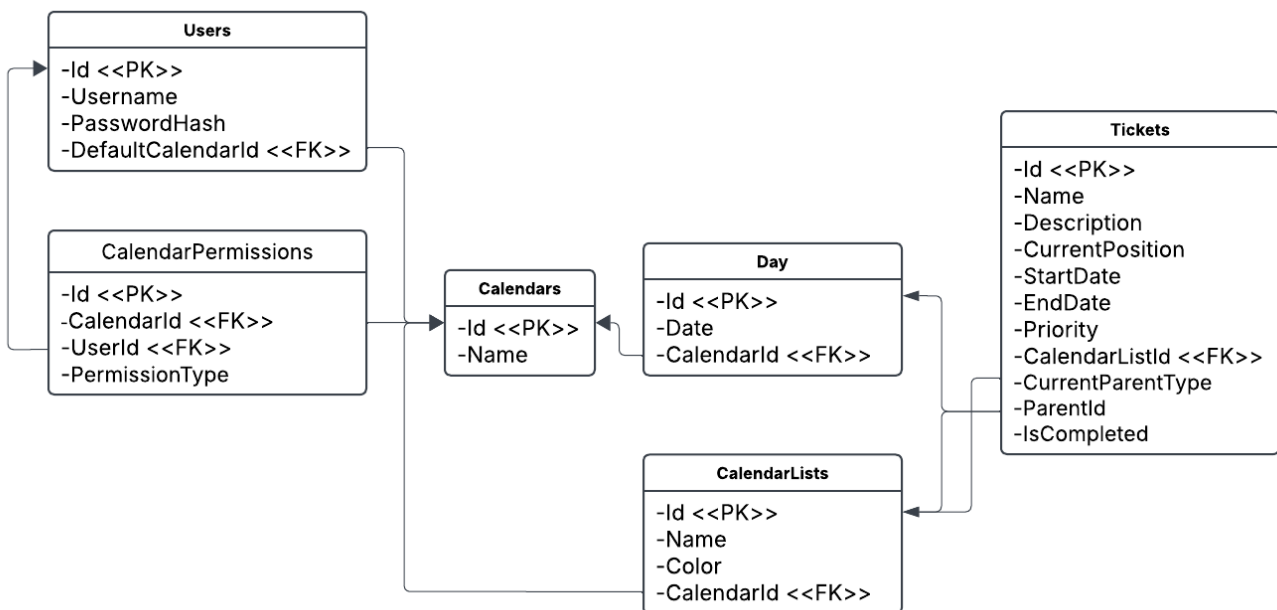


### 3.3. Adatbázis modell

Az adatbázis 5 táblából áll, nincs a csoporttól elkülönülő tábla, azaz az összes kapcsolatban áll legalább 1 másikkal. Úgy lett tervezve, hogy dinamikusan és hatékonyan kezelhető legyen a felhasználókhoz tartozó naptárak, listák és jegyek száma, valamint azok tulajdonságai. Az adatbázis tervezése során fontos szempont volt a redundancia kerülésére, valamint a komplexitás és a letisztultság között meghúzódó egyensúly megtalálása.

(3.6.) Az alábbi diagramon a nyilak a táblák közötti kulcskapcsolatokat jelölik. A «PK» a *Primary Key*-t (elsődleges kulcsot), míg a «FK» a *Foreign Key*-t (idegen kulcsot) szimbolizálja.

#### 3.3.1. Az adatmodell áttekintése




3.6. ábra. Adatbázis entitások és kapcsolatok

#### 3.3.2. Tábla-szintű leírás

(3.7., 3.8., 3.9., 3.10., 3.11., 3.12.) Az alábbiakban minden adatbázis entitást külön vizsgálunk. A képeken láthatóak a tábla argumentumnevei, típusai, NULL értéket engedélyező beállításai (*nullable*), valamint az elsődleges és másodlagos kulcsok is. A táblák a hivatkozási hierarchia aljától kezdve kerülnek bemutatásra,



hogy az idegen kulcsok értelmezésekor a hivatkozott tábla szerkezete már ismert legyen. Az elsődleges kulcsot egy sárga kulcs, míg a másodlagosat egy kék pont jelöli.

(3.7.) A naptárakat tartalmazó tábla az egyetlen, amely nem rendelkezik idegen kulccsal. Illetve jellemzően az idegenkulcsok ennek a táblának az azonosítójára hivatkoznak.

	Column Name	Data Type	Allow Nulls
	Id	int	<input type="checkbox"/>
	Name	varchar(255)	<input type="checkbox"/>



3.7. ábra. *Calendars*

(3.8.) A napokat reprezentáló tábla napjai mindig egy naptárhoz vannak kötve, elsősorban azonosító alapján kezeljük a velük kapcsolatos tevékenységeket. Azonban előfordul, hogy a `date` és `calendarId`-val azonosítjuk egyértelműen. A naptáraknak csak azon napjai léteznek adatbázisban eltárolt objektumként melyekhez legalább 1 jegy tartozik.

	Column Name	Data Type	Allow Nulls
	Id	int	<input type="checkbox"/>
	Date	date	<input type="checkbox"/>
	CalendarId	int	<input checked="" type="checkbox"/>

3.8. ábra. *Days*

(3.9.) A naptárlisták esetében kiválasztható a kívánt szín a felhasználó részéről, a választott megjelenést *HEX* kódok formájában tároljuk.

	Column Name	Data Type	Allow Nulls
	Id	int	<input type="checkbox"/>
	Name	varchar(255)	<input type="checkbox"/>
	Color	varchar(50)	<input checked="" type="checkbox"/>
	CalendarId	int	<input checked="" type="checkbox"/>



3.9. ábra. *CalendarLists*

(3.10.) A *Ticketek* esetében megfigyelhető egyedül, hogy másik azonosítóra hivatkozik idegenkulcsuk. Ez azt a naptárlistát jelöli amelyhez létrehoztuk a *Ticketet*.






Innen kapja például a színét vagy amikor egy napra kiosztott *Ticketet* visszateszünk a naptár nézetre ennek a mezőnek köszönhetően fogja megtalálni a helyét.

A `parentId` tartalmazhatja egy nap vagy egy naptár lista azonosítóját is, azt hogy pontosan melyiket azt a `currentParentType` argumentum alapján dől el, melynek értékei lehetnek "CalendarList", "ScheduledList" és "TodoList". Utóbbi kettő esetén tartalmazza a nap azonosítóját, a megkülönböztetés azért van, hogy egyértelmű legyen, hogy melyik listában szeretnénk megjeleníteni a napi nézeten belül.

	Column Name	Data Type	Allow Nulls
	Id	int	<input type="checkbox"/>
	Name	varchar(255)	<input type="checkbox"/>
	Description	text	<input checked="" type="checkbox"/>
	CurrentPosition	int	<input checked="" type="checkbox"/>
	StartTime	time(7)	<input checked="" type="checkbox"/>
	EndTime	time(7)	<input checked="" type="checkbox"/>
	Priority	int	<input checked="" type="checkbox"/>
	CalendarListId	int	<input checked="" type="checkbox"/>
	CurrentParentType	varchar(50)	<input checked="" type="checkbox"/>
	ParentId	int	<input checked="" type="checkbox"/>
	IsCompleted	bit	<input type="checkbox"/>


3.10. ábra. *Tickets*

(3.11.) Az felhasználók esetén alkalmazva van egy egyediségi (*unique*) megszorítás az email mezőre, tekintve hogy ez alapján történik a felhasználónk azonosítása / megkülönböztetése. A jelszót a *backend hash*-elve, titkosított formában tárolja.

	Column Name	Data Type	Allow Nulls
	Id	int	<input type="checkbox"/>
	Username	varchar(255)	<input type="checkbox"/>
	PasswordHash	varchar(255)	<input type="checkbox"/>
	Email	nvarchar(255)	<input type="checkbox"/>
	DefaultCalendarId	int	<input checked="" type="checkbox"/>

3.11. ábra. *Users*

(3.12.) A naptárakhoz való jogosultságok kezeléséért ez a tábla felelős, a *backenden* található `HasCalendarPermission` segédfüggvény ezen tábla értékei alapján dolgozik.

	Column Name	Data Type	Allow Nulls
	Id	int	<input type="checkbox"/>
•	CalendarId	int	<input checked="" type="checkbox"/>
•	UserId	int	<input checked="" type="checkbox"/>
	PermissionType	varchar(50)	<input checked="" type="checkbox"/>

3.12. ábra. *CalendarPermissions*

## SQL és C# típusok megfeleltetése Entity Framework-ben

Amikor az *EF* felvesz egy adatbázis entitást, az alábbi típuskonverziós konvenciók szerint teszi azt.

SQL típus	C# típus (EF-ben)
bit	bool
int	int
varchar(n)	string
nvarchar(n)	string
text	string
date	DateTime
time(7)	TimeSpan

## 3.4. Modul- és osztályszerkezet

### 3.4.1. *Backend* modulok és rétegek

A *backend* a `Program.cs` fájlban építi fel az *API* fő komponenseit és azok között a megfelelő kommunikációt. Az `ApplicationBuilder` példány létrehozásával kezdődik, amely az alkalmazás konfigurálásának alapját képezi. Majd sorra kerülnek a *CORS* szabályok, *SignalR* szolgáltatás, Adatbázis-kapcsolat, *JWT* alapú hitelesítés konfigurálása valamint további egyedi szolgáltatások regisztrálása *Dependency Injection*nel (innenről csak *DI*). Amint ezek mind helyesen beállításra kerültek az alkalmazás futtatása következik. A `WebApplication` amikor objektumpéldányra van szüksége megkéri a *DI* konténert hogy a Builderben meghatározott szabályok szerint adjon neki egy objektumpéldányt.

### 3.4.2. Osztálydiagram (*UML*)

Amennyiben egy objektum rendelkezik egy adattaggal amely valamelyik másik objektumot igényli az *DI*-al fog bekerülni az adott objektumba példányosítás során. Ilyen az *UML*-ben 3 helyen szerepel «injected» sztereotípiával jelölve, ezen kapcsolat egyszerűsítve van az olvashatóság kedvéért, a 3.4.3. Főbb osztályok leírása és implementációja pontban kifejtésre kerül pontosan melyik objektum hova injektálódik.

#### Nem szokványos sztereotípiák magyarázata:

(3.13.)

- «registers scoped», «registers singleton»

A `WebApplicationBuilder` végzi ezeknek az osztályoknak a regisztrálását a *DI* konténerbe. A «registers scoped» jelzi, hogy az osztály példánya minden egyes HTTP-kéréshez külön jön létre. Míg a «registers singleton» azt, hogy az alkalmazás teljes futása alatt csupán egyetlen példány létezik.

- «build»

A `WebApplicationBuilder` a `Build()` metóduson keresztül hozza létre a `WebApplication` példányt.

- «routes»

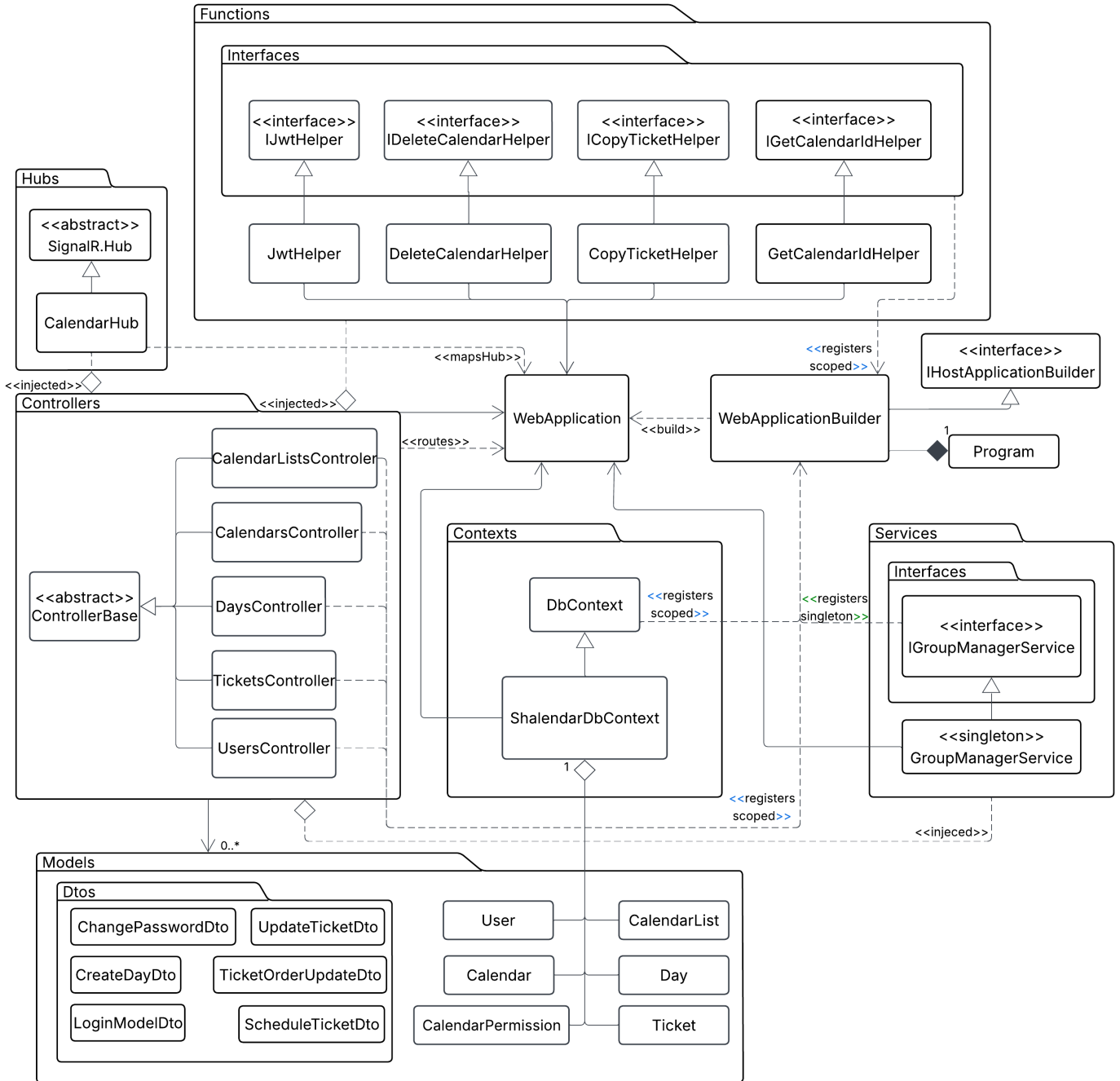
A `WebApplication` automatikusan hozzárendeli a HTTP útvonalakat a `Controller` típusú osztályokhoz a `MapControllers()` segítségével.

- «mapsHub»

Jelzi, hogy egy *SignalR* Hub végpont (`/calendarHub`) regisztrálásra kerül a `WebApplication` `pipeline`-jában.

- «injected»

Azt mondja hogy ezek az objektumok bele injektálódtak a mutatott objektumba a `WebApplication` *DI* konténere által.

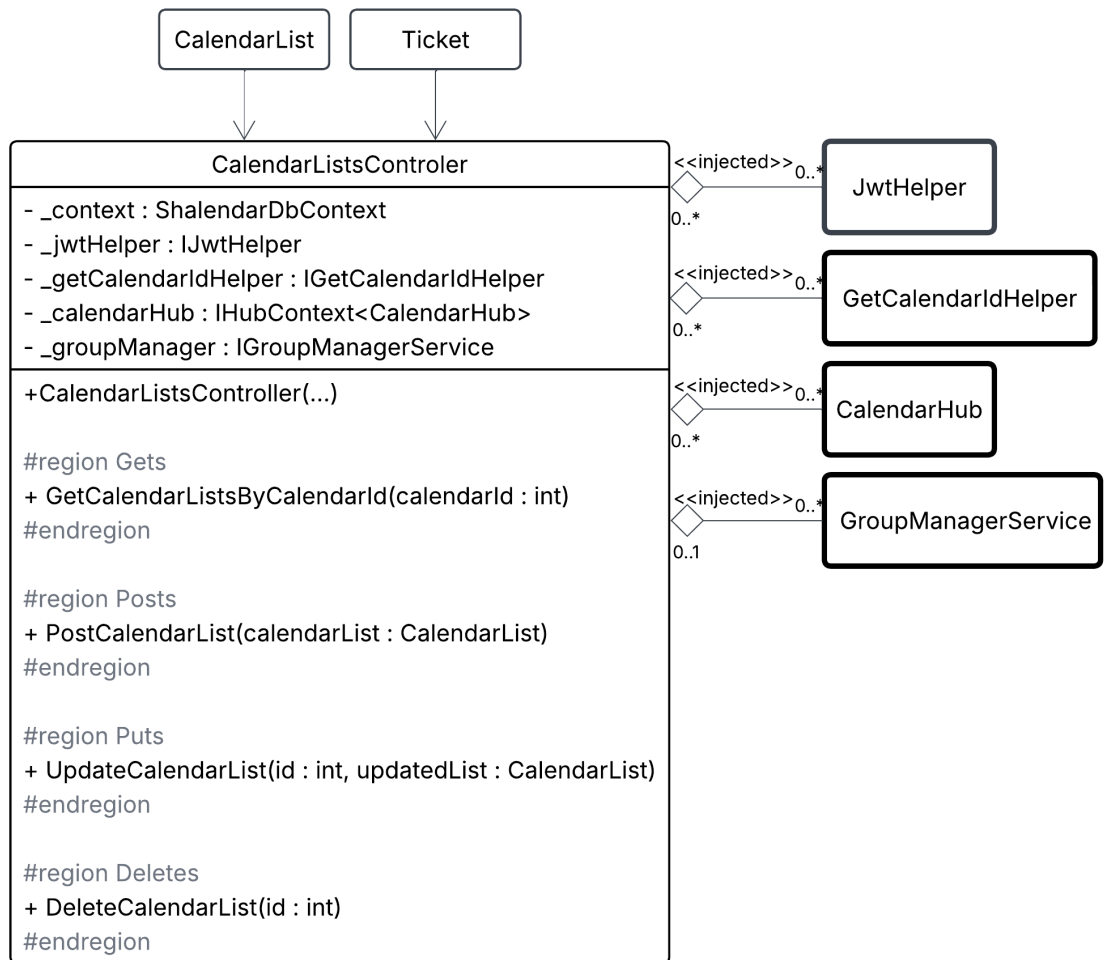


3.13. ábra. API UML osztálydiagram

### 3.4.3. Főbb osztályok leírása és implementációja

(3.14., 3.15., 3.16., 3.17., 3.18., 3.19.) Alábbiakban látható a kontrollerek és fontosabb osztályok részletes felépítése valamint a modellekkel, *DTO*-kal és segédfüggvényekkel való kapcsolataik. A *DTO*-k szaggatott vonalas objektumként vannak jelölve a helper objektumok pedig vastag vonallal a könnyebb átláthatóság érdekében. A kont-

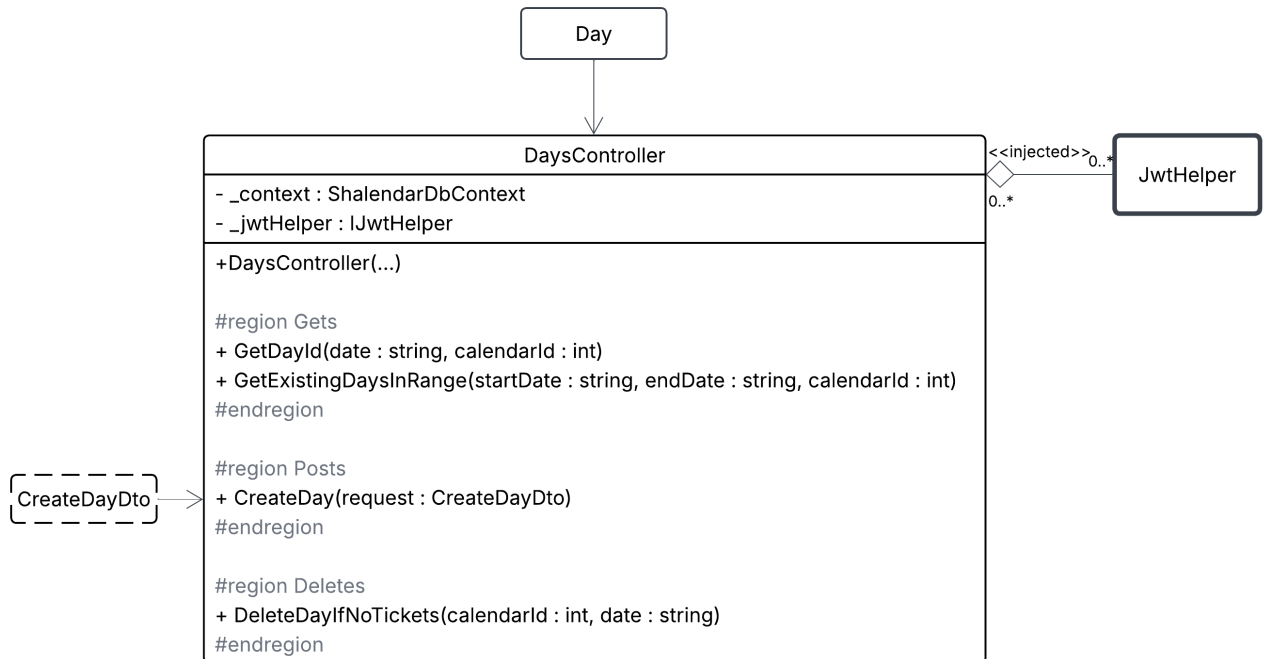
rollerekben láthatóak **regionok**, ezek a logikai elkülönítés vizuális szemléltetésére vannak, a forráskódban szintén szerepelnek.



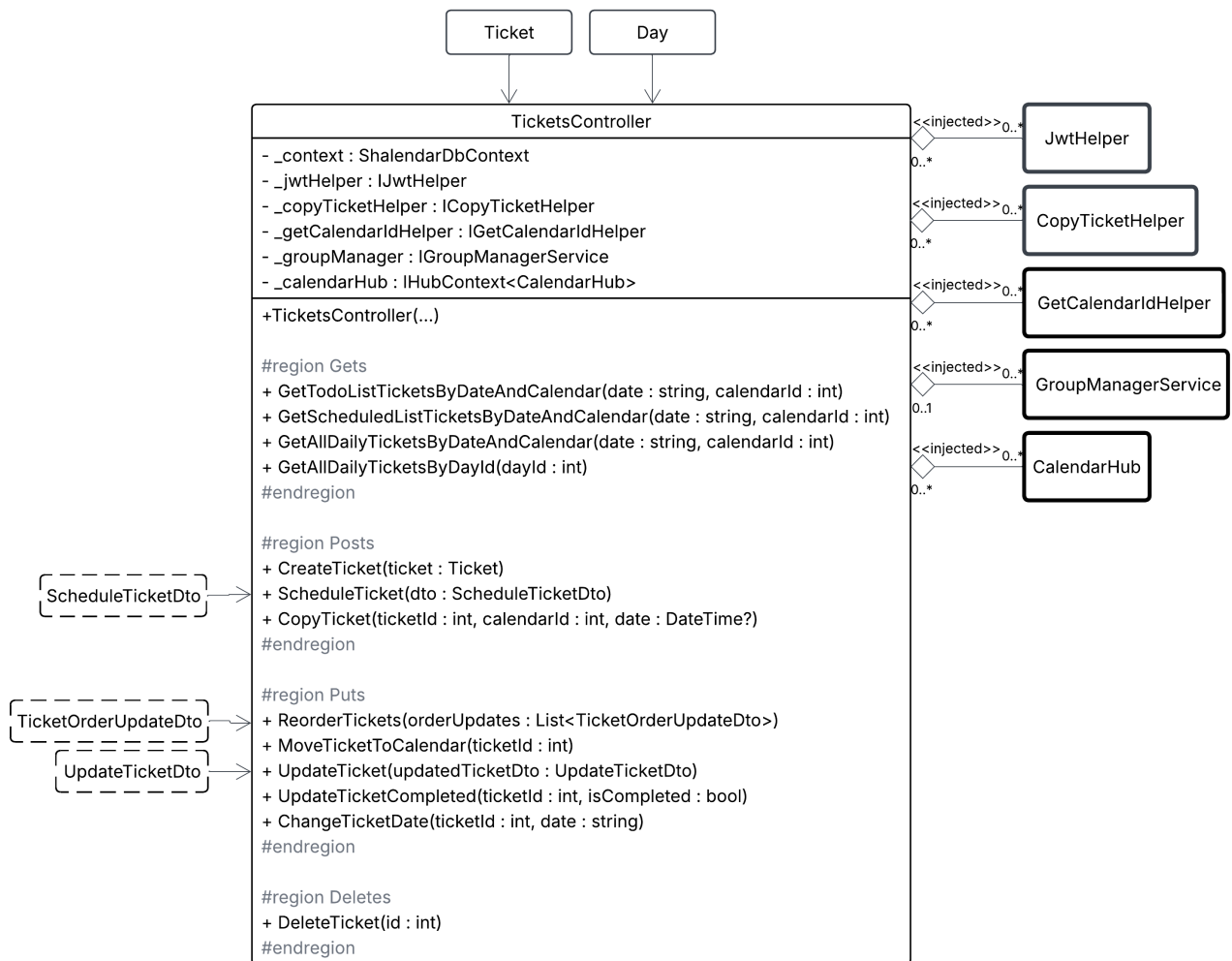
3.14. ábra. CalendarListsController



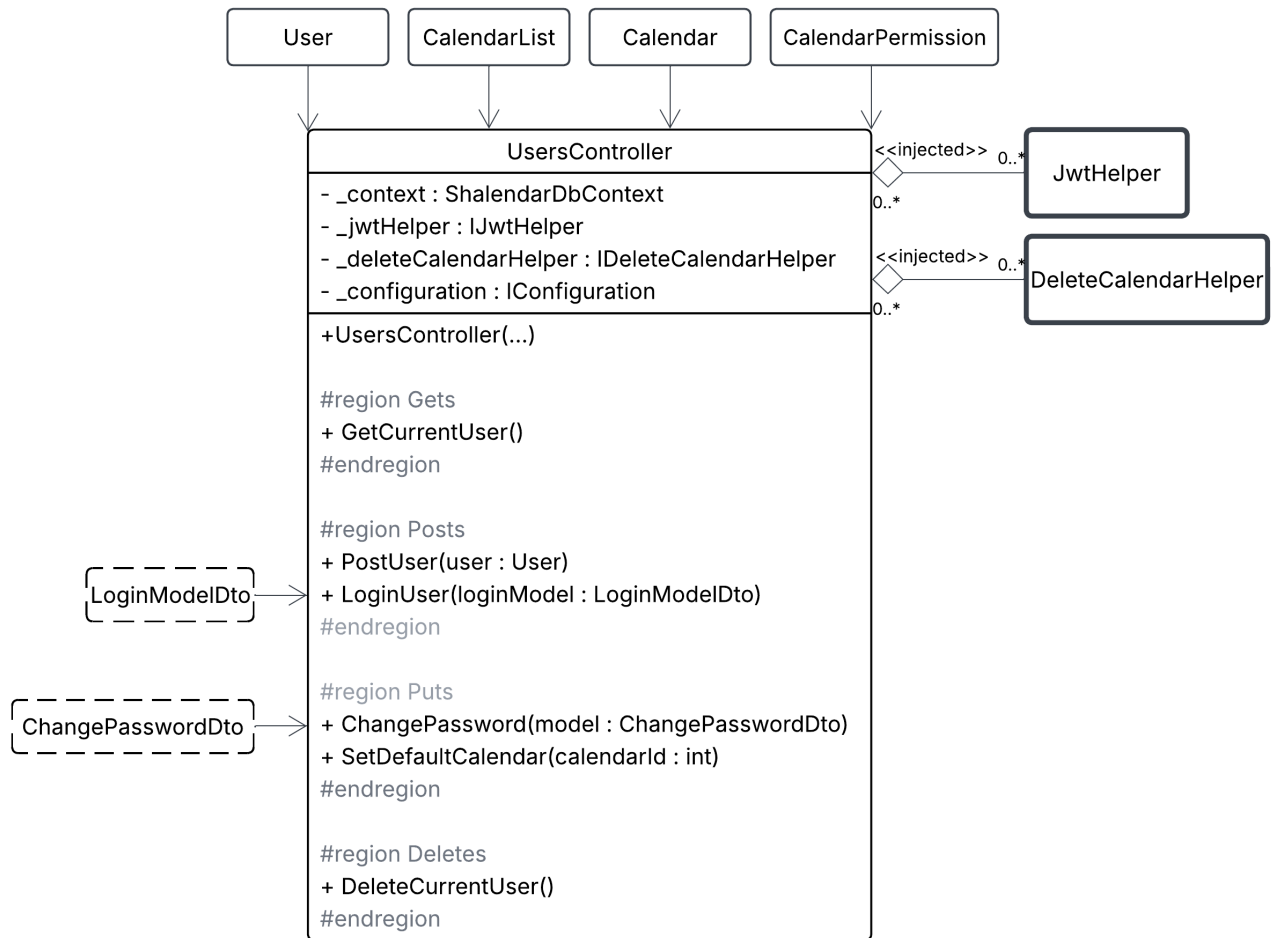
3.15. ábra. CalendarsController



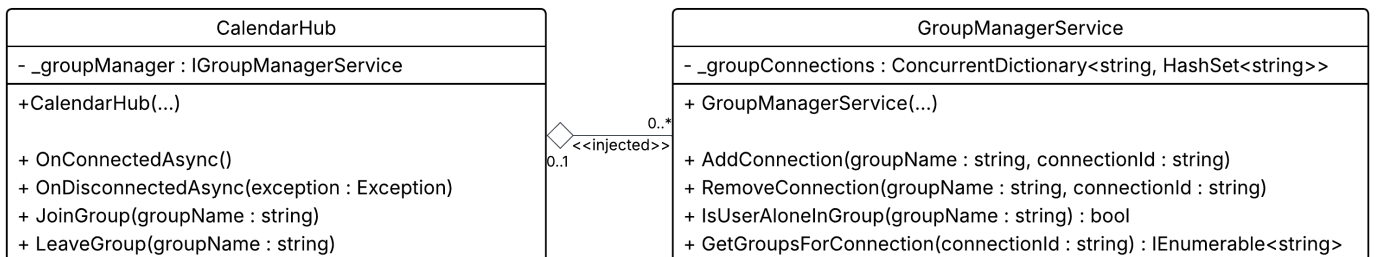
3.16. ábra. DaysController



3.17. ábra. TicketsController



3.18. ábra. UsersController



3.19. ábra. CalendarHub & GroupManagerService

## 3.5. A felhasználói felület

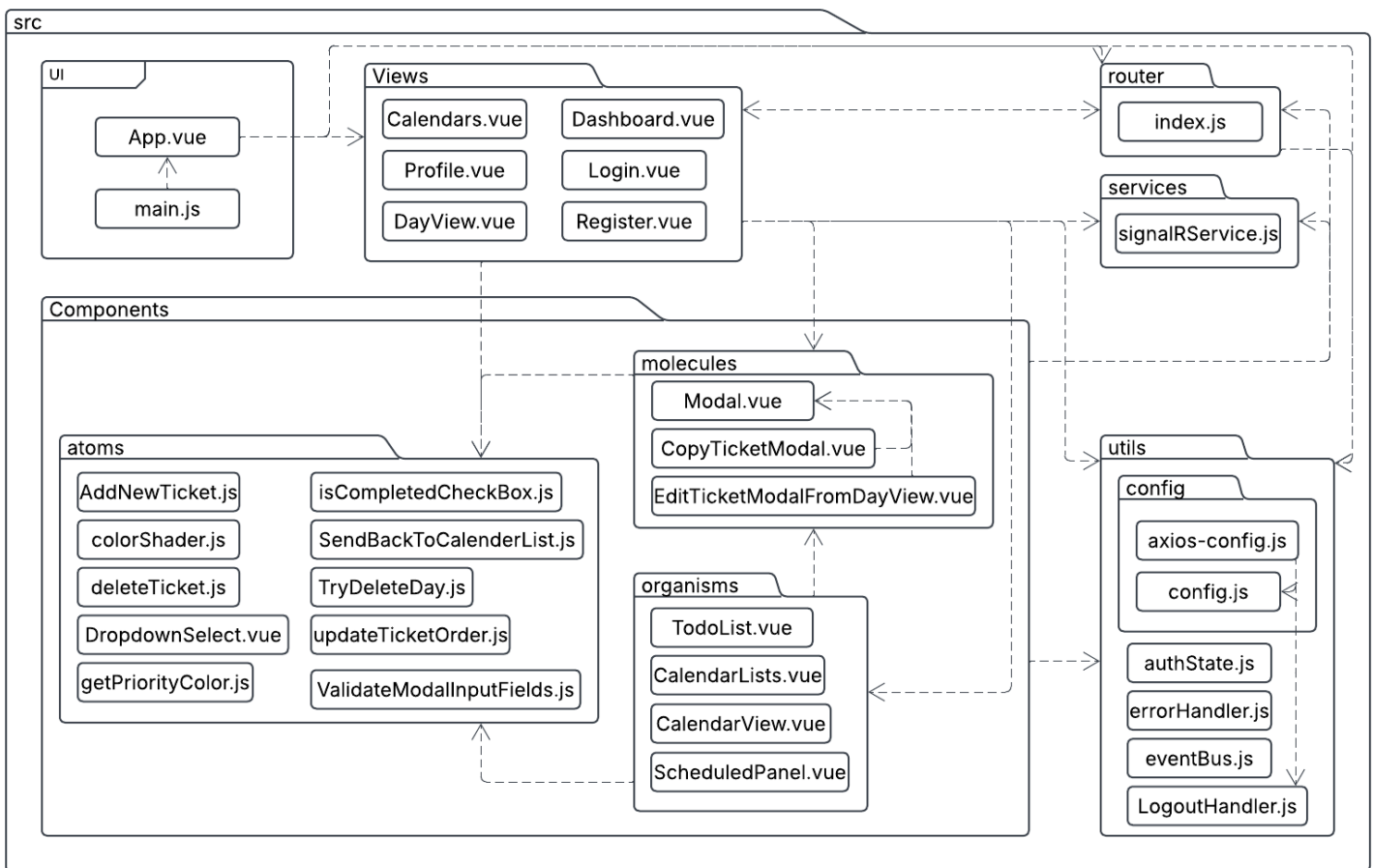
A felhasználó az `App.vue`-ban meghatározott *navbart* látja amely értelemszerűen csak a bejelentkezést követően jelenik meg. Az `App.vue` tölti be a *navbar* alá a további komponenseket. 3 fő nézet van, ezek a nézetek között tud navigálni a felhasználó. Ezek közül speciális a *DayView* valamint a *Dashboard*, mi-



vel ők 2 további fő komponensből állnak. A `Views/DayView.vue` első eleme az `organism/ScheduledPanel.vue` míg második része a `organism/ToDoList.vue`. A `Views/Dashboard.vue` pedig szintén ez a logika alapján vizsgálva a `organism/CalendarView.vue` és `organism/CalendarLists.vue` elemekből állnak.

### 3.5.1. Csomagdiagram (UML)

A csomagdiagram értelmezésében a 3.1.4. Csomag diagram (UML) pontban leírtak segítenek.

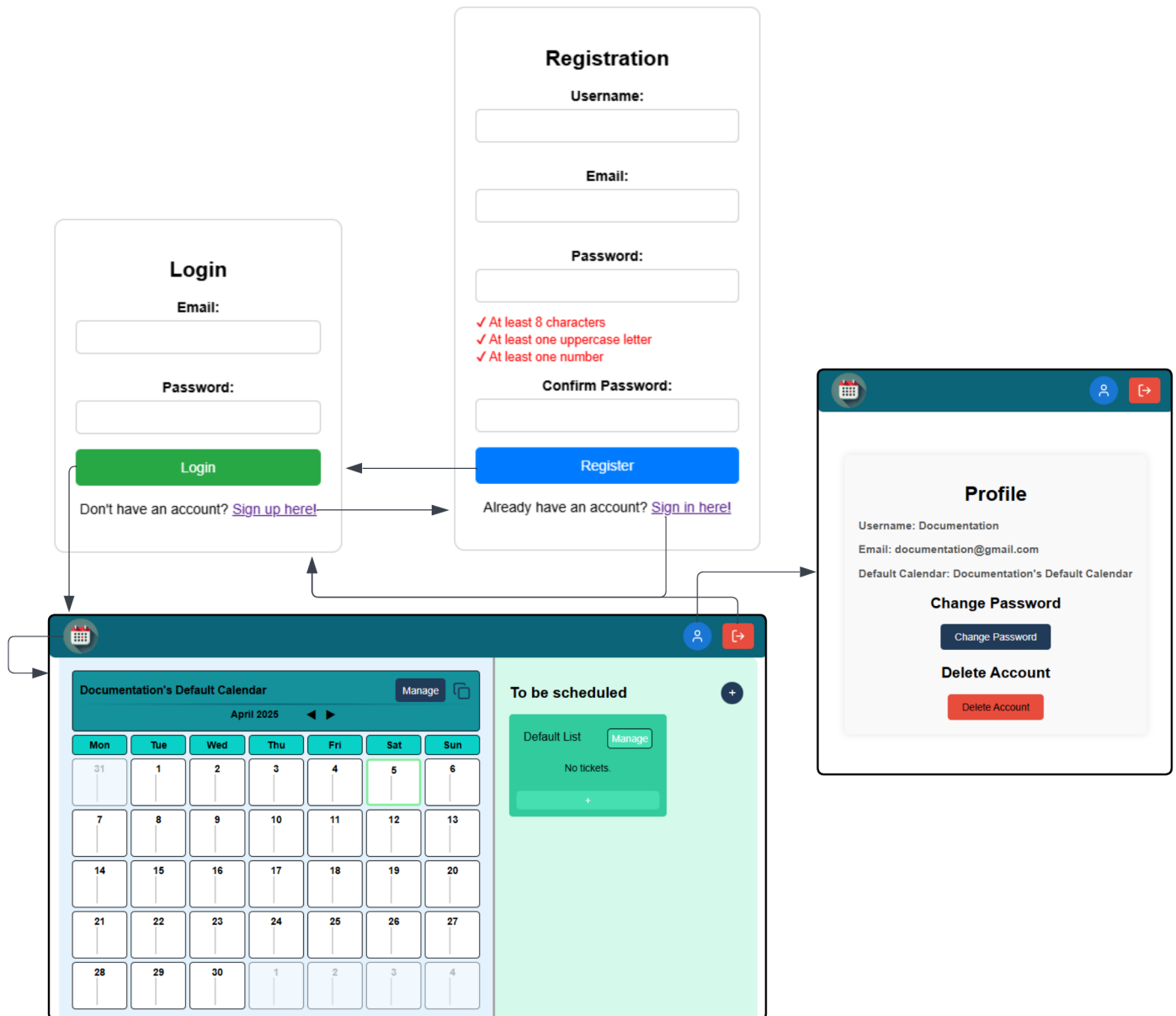


3.20. ábra. Src mappa csomag diagramja

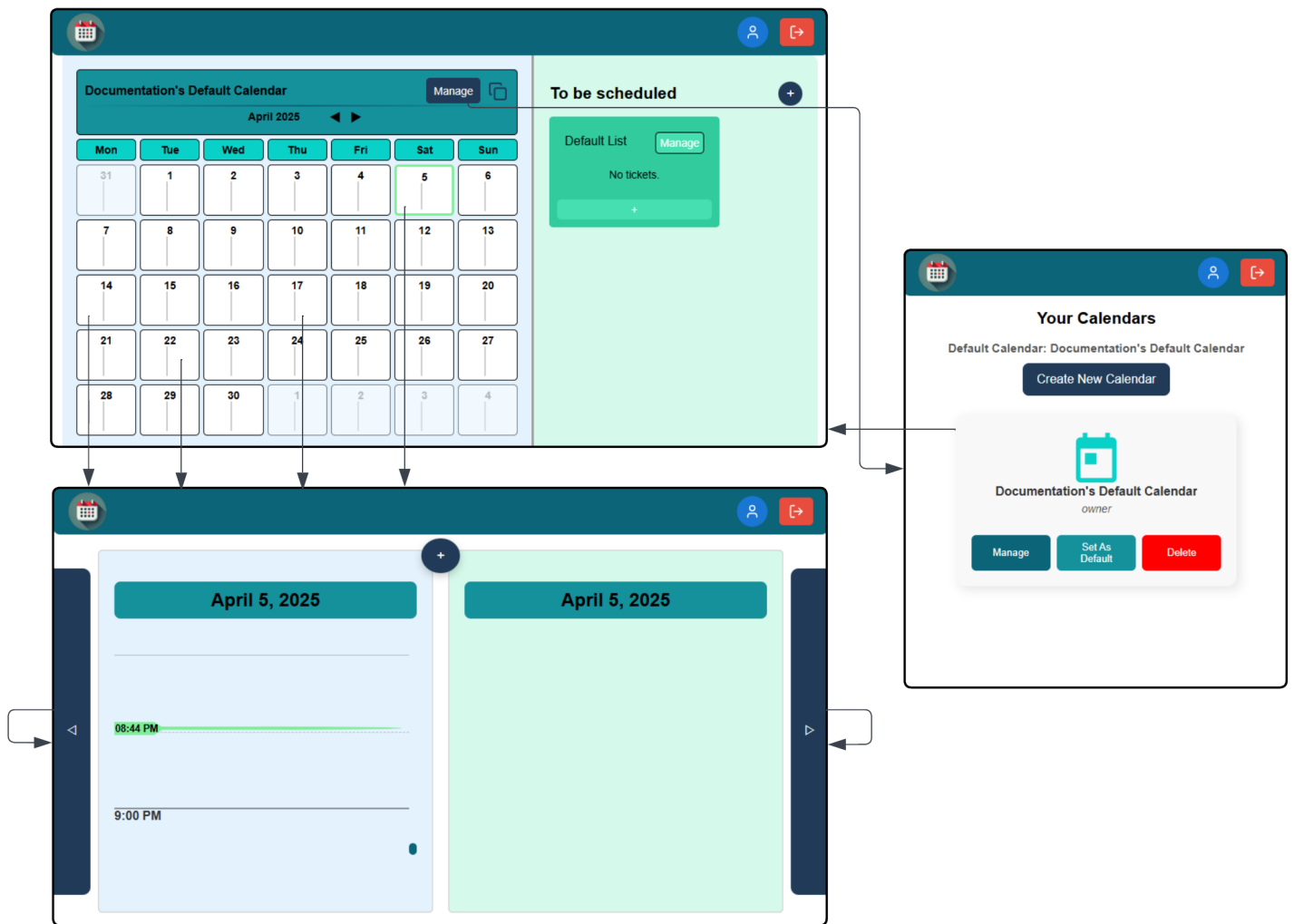
### 3.5.2. Képernyők navigációs logikája

(3.21., 3.22.) A fent említett 6 képernyő közötti navigáció bemutatására az alábbi navigációs térkép szolgál. Az első ábrán<sup>(3.21.)</sup> látható a bejelentkezési/regisztrációs funkciók, valamint a *navbar*. A másodikon<sup>(3.22.)</sup> a további oldalak közötti navigációs logika.

A felhasználó navigációt igénylő tevékenységeiért a `router/index.js` felelős, melyben be vannak állítva a `route`-k. A routerben a token alapján automatikusan átirányításra kerülnek az autentikációt igénylő kérések. (Természetesen az *API* is ellenőrizné ezt, azonban már itt is lekezeljük ezt.)



3.21. ábra. *Registration & Login & Navbar* navigációs logikája



3.22. ábra. További képernyők navigációs logikája

### 3.5.3. Felhasználói események kezelése

#### Főbb interakciók leírása

- **Gombok nyomása:** Amennyiben a felhasználó megnyom egy gombot a felületen leggyakrabban az alábbi 2 dolog történik: vagy betölti neki a kívánt oldalt, vagy megnyílik egy *Modal*.
- **Drag-and-drop:** A ticketekkel kapcsolatban különböző "húzási" funkciók vannak. Például *Ticketek* újrendezése egy *CalendarListen* belül, vagy a nap-tár napjára való kiosztás, esetleg a *DayView* oldalsó nyílaira való ejtés során az egy nappal való előre/hátra ütemezés.
- **Tooltips:** A legtöbb interakció, melyet a felhasználó elvégezhet *tooltippekkel* van ellátva, azaz ha a kurzort egy helyben az elem felett tartja megjelenik egy

üzenet, amely leírja milyen műveletek és hogyan végezhetőek az adott elemmel.

- **Scroll események:** Hosszú naplista, jegylista esetén, vagy ha valamilyen *UI* elem nem fér el a képernyőben megjelenik egy görgetősáv.

## Hibakezelés

A rendszer nem tud a felhasználó számára kezeletlen hibát megjeleníteni. Minden esetben pirosan megjelenik egy specifikus hibaüzenet, majd 5 másodperc után eltűnik. Ezen hibakezelési logikát a `utils/errorHandler.js` végzi.

## 3.6. Telepítési folyamat leírása

### 3.6.1. Fejlesztési telepítési útmutató

Az alábbi pontok feltételezik, hogy a fejlesztő rendelkezik a 3.1.2. Alkalmazott technológiák és eszközök pont alatt felsorolt fejlesztői eszközök és függőségek mindegyikével.

1. Klónozd a projekt Git repóját:

```
git clone https://github.com/KerteszJanos/Shalendar.git
```

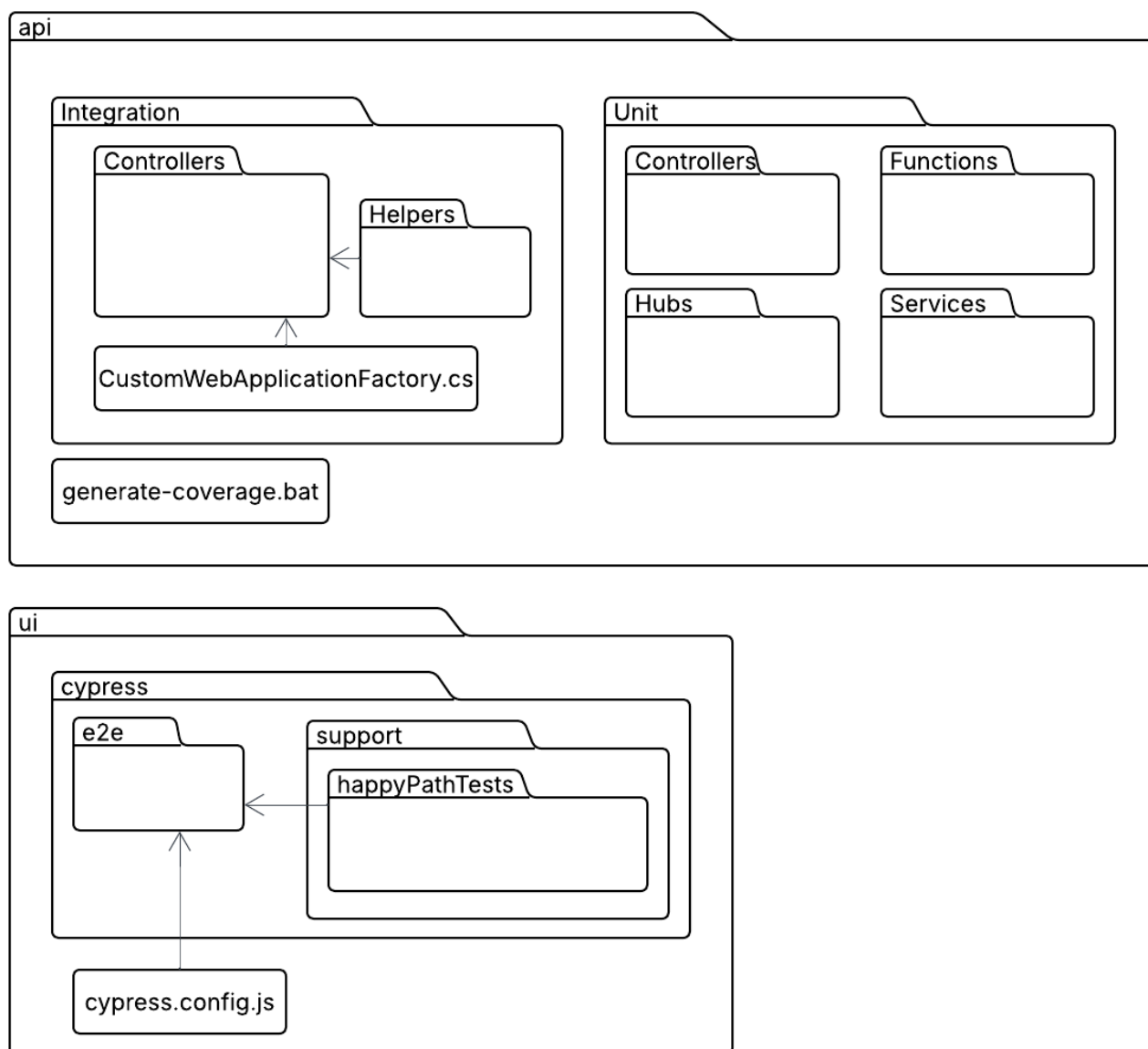
2. A *backend* mappában futtasd: `dotnet restore`, majd `dotnet build`, végül `dotnet run`, vagy tedd meg ezt manuálisan Visual Studióban.
3. Az adatbázis eléréséhez állítsd be a `appsettings.json`-ban az `ConnectionStrings.DefaultConnection` értékét.
4. A *frontend* mappában futtasd: `npm install`, majd `npm run dev`.
5. A *frontend* alapértelmezett címe: `http://localhost:5173`, a *backendé* pedig: `https://localhost:7192`

## 4. fejezet

# Fejlesztői dokumentáció - Tesztelés

### 4.1. Tesztelési stratégia

A projekt megvalósít egység- (*Unit*) valamint integrációs (*Integration*) tesztelést a *backend* oldalon, valamint *e2e* rendszertesztelést a *frontend* oldalon. Az első tesztelés fehérdobozos (*White-box*), míg az utóbbi kettő feketedobozos (*Black-box*) megközelítést alkalmaz.

4.1.1. Tesztfájlok fizikai elhelyezkedése (*UML*)

4.1. ábra. Tesztfájlok és elhelyezkedésük

## 4.1.2. Teszteléshez használt eszközök

A tesztelést elősegítő keretrendszerek, csomagok listája az alábbi, ahol nem láthatóak azok az eszközök, amelyek a fő programban is megjelennek, de itt is fel vannak használva. A fő programban használtokról a [3.1.2. Alkalmazott technológiák és eszközök](#) pontban lehet olvasni.

### Teszteléshez használt keretrendszer

- **xUnit** - Az egység- és integrációs teszteléshez használt tesztelési keretrendszer.

### NuGet csomagok

- **coverlet.collector** (v6.0.4) - kódfedettség (*code coverage*) mérésére szolgáló eszköz.
- **FluentAssertions** (v8.2.0) - olvashatóbb és kifejezőbb **assert**-ek írását teszi lehetővé.
- **Microsoft.AspNetCore.Mvc.Testing** (v8.0.5) - lehetővé teszi az *ASP.NET Core* alkalmazások integrációs tesztelését, a **WebApplicationFactory** osztályon keresztül.
- **Microsoft.EntityFrameworkCore.InMemory** (v9.0.3) - memóriaalapú adatbázis *EF Core*-hoz, tesztelési célokra, integrációs tesztekhez
- **Microsoft.NET.Test.Sdk** (v17.13.0) - az *xUnit* tesztelési keretrendszer működéséhez szükséges *alapsdk*, amely a tesztek futtatását támogatja.
- **Moq** (v4.20.72) - *mock* objektumok létrehozására szolgáló keretrendszer, egy-egységtesztekhez.
- **xunit** (v2.5.3) - xUnit.net tesztelési keretrendszer, a .NET ökoszisztémában elterjedt egységteszt eszköz.
- **xunit.runner.visualstudio** (v2.5.3) - Visual Studio integráció az xUnit tesztek futtatásához és eredményeinek megjelenítéséhez.

### Frontend teszteléshez használt eszközök

- **Cypress** - *End-to-end (e2e)* tesztelési keretrendszer, amely lehetővé teszi a felhasználói interakciók automatizálását a böngészőben.

## 4.2. Egységtesztek

A vezérlők metódusai különböző bemeneti feltételek és *edge-case*-ek mentén vannak tesztelve. A tesztek során *InMemory* adatbázis szimulálja a valódi működ-

dését az *Entity Framework Core* támogatásával, valamint a külső függőségek (pl. *IJwtHelper*, *ICopyTicketHelper*) Moq segítségével kerültek *mockolásra*.

### 4.3. Integrációs tesztek

A rendszer integrációs tesztjei során a teljes HTTP-kérések és -válaszok tesztelése a valós környezethez hasonló módon történik. A *CustomWebApplicationFactory* osztályon keresztül egy test-specifikus *ASP.NET Core* alkalmazás példány jön létre, amely *InMemory* adatbázist és külön konfigurációt használ. Ez lehetővé teszi az autentikáció, a jogosultságkezelés, valamint az adatbázis-kommunikáció teljes körű vizsgálatát izolált környezetben.

A vezérlők metódusai különböző bemeneti feltételek és *edge-case*-ek mentén vannak tesztelve. A *happy-path* tesztek esetében vizsgálva van az adatbázisra kifejtett hatásuk is.

### 4.4. Rendszertesztek - automatizált

Az automatizált rendszertesztek a *UI* mappájában találhatóak. *Cypress* segítségével szimulálják 2 felhasználó segítségével az összes lehetséges interakciót. A tesztek futásának követelménye az aktív adatbázis megléte, valamint a futó *backend* illetve *frontend* rendszerek. A tesztelésben egy a tesztelő által figyelhető profillal is megoszsa a naptárat, ezáltal a *SignalR* működése is ellenőrizhető. A teszteléshez létrehozott felhasználói profil belépési adatai az alábbiak:

- Email: `CypressTestTester@example.com`
- Jelszó: `Password123`

### 4.5. Rendszertesztek - Manuális

A tesztelés *Given-When-Then* struktúrát követve van leírva, melyet a tesztelő manuálisan kell végrehajtson. A táblázatok a *happy-pathet* tartalmazzák, közvetlenül alattuk pedig a lehetséges *edge-casek*, valamint azon engedély szintje szerepel amivel már nem elvégezhető a művelet. A tesztelést ajánlott 2 felhasználóval egy időben végezni, így a releváns funkciók *SignalR* értesítéseinek kezelése is jól látható lesz.



## Bejelentkezési oldal funkciói

### 4.1. táblázat. Regisztráció

<b>Given</b>	A felhasználó megnyitotta az alkalmazást.
<b>When</b>	A regisztrációs lehetőséget választja.
<b>Then</b>	A rendszer belépteti a főoldalra, ahol láthatja a naptárát és a <i>CalendarList</i> jeit.

(4.1.) **Ellenőrizendő hibakezelések:** A felhasználónév üres. Az email cím már regisztrálva van. A jelszavak nem egyeznek. A jelszó 8 karakternél rövidebb. A jelszó nem tartalmazza: nagybetű, szám.

**Engedélykezelés:** -

### 4.2. táblázat. Bejelentkezés

<b>Given</b>	A felhasználó megnyitotta az alkalmazást.
<b>When</b>	A bejelentkezés lehetőséget választja, majd bejelentkezik.
<b>Then</b>	A rendszer elvégzi a regisztrációt, majd a bejelentkezési oldalra irányítja a felhasználót.

(4.2.) **Ellenőrizendő hibakezelések:** Az email cím nincs regisztrálva. A jelszó nem megfelelő.

**Engedélykezelés:** -

## Főoldal funkciói

### 4.3. táblázat. Új *CalendarList* létrehozása

<b>Given</b>	A felhasználó a főoldalon van.
<b>When</b>	A <i>CalendarList</i> hozzáadása opciót választja (+ gomb a <i>CalendarListek</i> felett) majd megfelelően paraméterezi a megnyíló <i>Modalt</i> .
<b>Then</b>	A <i>CalendarList</i> megjelenik a főoldalon.

(4.3.) **Ellenőrizendő hibakezelések:** A *CalendarList* neve üres vagy csak szóköz.

**Engedélykezelés:** *Write* hozzáférési joggal nem elvégezhető a művelet.

4.4. táblázat. *CalendarList* módosítása

<b>Given</b>	A felhasználó a főoldalon van.
<b>When</b>	Az adott <i>CalendarList</i> en a <i>Manage</i> opciót választja, majd a felnyíló <i>Modal</i> t megfelelően paraméterezi.
<b>Then</b>	A <i>CalendarList</i> valamint a hozzá tartozó <i>Ticketek</i> megváltoznak (a <i>Ticketek</i> csak akkor ha változott a <i>CalendarList</i> színe).

(4.4.) **Ellenőrizendő hibakezelések:** A *CalendarList* neve üres vagy csak szóköz.

**Engedélykezelés:** *Write* hozzáférési joggal nem elvégezhető a művelet.

4.5. táblázat. *CalendarList* törlése

<b>Given</b>	A felhasználó a főoldalon van.
<b>When</b>	Az adott <i>CalendarList</i> en a <i>Manage</i> opciót választja, majd a felnyíló <i>Modal</i> on a törlés gombra nyom.
<b>Then</b>	A <i>CalendarList</i> valamint a hozzá tartozó <i>Ticketek</i> törlődnek.

(4.5.) **Ellenőrizendő hibakezelések:**

**Engedélykezelés:** *Write* hozzáférési joggal nem elvégezhető a művelet.

4.6. táblázat. *Ticket* létrehozása az adott *CalendarList*hez

<b>Given</b>	A felhasználó a főoldalon van.
<b>When</b>	A <i>CalendarList</i> en új <i>Ticket</i> létrehozására kattint, majd megfelelően paraméterezi azt.
<b>Then</b>	A <i>Ticket</i> megjelenik a kívánt oszlopban.

(4.6.) **Ellenőrizendő hibakezelések:** A *Ticket* neve üres vagy csak szóköz.

**Engedélykezelés:** *Read* hozzáférési joggal nem elvégezhető a művelet.

4.7. táblázat. *Ticket* törlése az adott *CalendarList*ről

<b>Given</b>	A felhasználó a főoldalon van.
<b>When</b>	A <i>Ticket</i> et kitörli.
<b>Then</b>	A <i>Ticket</i> törlődik a <i>CalendarList</i> ről

(4.7.) **Ellenőrizendő hibakezelések:** -

**Engedélykezelés:** *Read* hozzáférési joggal nem elvégezhető a művelet.

4.8. táblázat. *Ticket* másolása saját naptárba

<b>Given</b>	A felhasználó a főoldalon van.
<b>When</b>	A naptár másolása gombra kattint majd kiválasztja a felugró <i>Modalon</i> melyik naptárba szeretne másolni.
<b>Then</b>	A rendszer lemásolja az adott <i>Ticketet</i> feltéve hogy egy pontosan ilyen még nem létezik az adott naptárban.

(4.8.) **Ellenőrizendő hibakezelések:** Nincs kiválasztott naptár.

**Engedélykezelés:** -

4.9. táblázat. *CalendarList Ticketjeinek* újrendezése

<b>Given</b>	A felhasználó a főoldalon van.
<b>When</b>	Az adott <i>CalendarListen</i> megfog egy <i>Ticketet</i> majd a listán belül mozgatva új pozícióba teszi.
<b>Then</b>	A <i>CalendarListben</i> szereplő <i>Ticketek</i> sorrendje frissül.

(4.9.) **Ellenőrizendő hibakezelések:** A *Ticket* rossz helyre való ejtése nem okoz hibát. (pl.: egy másik listába)

**Engedélykezelés:** *Read* hozzáférési joggal nem elvégezhető a művelet.

4.10. táblázat. *Ticket* áthelyezése

<b>Given</b>	A felhasználó a főoldalon van és van egy oszlop egy meglévő <i>Tickettel</i> .
<b>When</b>	A felhasználó a <i>Ticketet</i> a naptár valamelyik mezőjére húzza. (Balra időponthoz köti ami egy <i>Modal</i> megfelelő paraméterezésével történik.)
<b>Then</b>	A <i>Ticket</i> átkerül a naptár adott napjára.

(4.10.) **Ellenőrizendő hibakezelések:** Időponthoz osztás közben nincs kitöltve mind a 2 input mező. Időponthoz osztás közben az **EndTime**-hoz nem legalább 15 percnél későbbi időpont van írva mint a **StartTime**-hoz.

**Engedélykezelés:** *Read* hozzáférési joggal nem elvégezhető a művelet.

4.11. táblázat. Naptár napjának megnyitása

<b>Given</b>	A felhasználó a főoldalon van.
<b>When</b>	A naptár napjára kattint.
<b>Then</b>	A naphoz tartozó napi nézet megnyílik.

(4.11.) **Ellenőrizendő hibakezelések:** -

**Engedélykezelés: -**

4.12. táblázat. További naptárak kezelése

<b>Given</b>	A felhasználó a főoldalon van.
<b>When</b>	A naptáron lévő <i>Manage</i> gombra kattint.
<b>Then</b>	A további naptárak oldal megnyílik.

(4.12.) **Ellenőrizendő hibakezelések: -**

**Engedélykezelés: -**

4.13. táblázat. Naptár másolása saját naptárba

<b>Given</b>	A felhasználó a főoldalon van.
<b>When</b>	A naptár másolása gombra kattint majd kiválasztja a felugró <i>Modalon</i> melyik naptárba szeretne másolni.
<b>Then</b>	A rendszer lemásolja a naptárhoz tartozó összes <i>Ticketet</i> a duplikátumok szűrésére odafigyelve.

(4.13.) **Ellenőrizendő hibakezelések:** Nincs kiválasztott naptár.

**Engedélykezelés: -**

## Napi nézet funkciói

4.14. táblázat. *Ticket* létrehozása

<b>Given</b>	A felhasználó a napi nézet oldalon van.
<b>When</b>	Az új <i>Ticket</i> létrehozása gombra (+) kattint, majd megfelelően paraméterezi a felugró <i>Modalt</i> .
<b>Then</b>	A <i>Ticket</i> megjelenik a <i>ToDoListben</i> ha nem lett időponthoz kötve, amennyiben igen, a <i>ScheduledListben</i> lesz látható.

(4.14.) **Ellenőrizendő hibakezelések:** A *Ticket* neve üres vagy csak szóköz. Nincs kiválasztott *CalendarList*. Csak az egyik *time* mező van kitöltve. *EndTime*-hoz nem legalább 15 percnél későbbi időpont van írva mint a *StartTime*-hoz. Ha nincs a naptárhoz tartozó *CalendarList* megjelenik egy hibaüzenet ami arra kér hogy hozz létre egyet.

**Engedélykezelés:** *Read* hozzáférési joggal nem elvégezhető a művelet.

4.15. táblázat. *Ticket* törlése

<b>Given</b>	A felhasználó a napi nézet oldalon van.
<b>When</b>	A <i>Ticketet</i> kitörli.
<b>Then</b>	A <i>Ticket</i> törlődik.

(4.15.) **Ellenőrizendő hibakezelések:**

**Engedélykezelés:** *Read* hozzáférési joggal nem elvégezhető a művelet.

4.16. táblázat. *ToDoList Ticketjeinek* újrendezése

<b>Given</b>	A felhasználó a napi nézet oldalon van.
<b>When</b>	A <i>ToDoListben</i> megfog egy <i>Ticketet</i> majd a listán belül mozgatva új pozícióba teszi.
<b>Then</b>	A <i>ToDoListben</i> szereplő <i>Ticketek</i> sorrendje frissül.

(4.16.) **Ellenőrizendő hibakezelések:** A *Ticket* rossz helyre való ejtése nem okoz hibát.

**Engedélykezelés:** *Read* hozzáférési joggal nem elvégezhető a művelet.

4.17. táblázat. *Ticket* megjelölése elvégzettként

<b>Given</b>	A felhasználó a napi nézet oldalán van.
<b>When</b>	A <i>Ticketet</i> megjelöli elvégzettként.
<b>Then</b>	A <i>Ticket</i> elvégzettként lesz megjelenítve.

(4.17.) **Ellenőrizendő hibakezelések:** -

**Engedélykezelés:** *Read* hozzáférési joggal nem elvégezhető a művelet.

4.18. táblázat. *Ticket* időpontra osztása

<b>Given</b>	A felhasználó a napi nézet oldalán van és található legalább 1 <i>Ticket</i> a <i>ToDoListben</i> .
<b>When</b>	A felhasználó a <i>Ticketet</i> szerkesztésre nyitja, majd időponthoz köti.
<b>Then</b>	A <i>Ticket</i> a megfelelő időpontban megjelenik a <i>ScheduledListben</i> .

(4.18.) **Ellenőrizendő hibakezelések:** Időponthoz osztás közben nincs kitöltve mind a 2 input mező. Időponthoz osztás közben az **EndTime**-hoz nem legalább 15 percnél későbbi időpont van írva mint a **StartTime**-hoz.

**Engedélykezelés:** *Read* hozzáférési joggal nem elvégezhető a művelet.

4.19. táblázat. *Ticket* visszaküldése a főoldalra

<b>Given</b>	A felhasználó a napi nézet oldalán van és található legalább 1 <i>Ticket</i> ami a főoldal valamelyik <i>CalendarListjében</i> volt.
<b>When</b>	A felhasználó a <i>Ticketet</i> visszaküldi a főoldalra.
<b>Then</b>	A <i>Ticket</i> visszakerül az eredeti <i>CalendarListjébe</i> .

(4.19.) **Ellenőrizendő hibakezelések:** -

**Engedélykezelés:** *Read* hozzáférési joggal nem elvégezhető a művelet.

4.20. táblázat. *Ticket* másolása saját naptárba

<b>Given</b>	A felhasználó a napi nézet oldalán van és található ott legalább 1 <i>Ticket</i> .
<b>When</b>	A naptár másolása gombra kattint majd kiválasztja a felugró <i>Modalon</i> melyik naptárba szeretne másolni.
<b>Then</b>	A rendszer lemásolja az adott <i>Ticketet</i> feltéve hogy egy pontosan ilyen még nem létezik az adott naptárban.

(4.20.) **Ellenőrizendő hibakezelések:** Nincs kiválasztott naptár.

**Engedélykezelés:** -

### Több naptár kezelése funkciói

4.21. táblázat. Új naptár létrehozása

<b>Given</b>	A felhasználó a több naptár fülön van.
<b>When</b>	Az új naptár létrehozása gombra kattint.
<b>Then</b>	A rendszer létrehoz egy új naptárat, amelyhez további felhasználókat is hozzáadhat.

(4.21.) **Ellenőrizendő hibakezelések:** A naptár neve üres vagy csak szóköz.

**Engedélykezelés:** -

4.22. táblázat. Naptár engedélyeinek kezelése

<b>Given</b>	A felhasználó a több naptár fülön van.
<b>When</b>	A naptár <i>Manage</i> gombjára kattint.
<b>Then</b>	Megnyílik egy <i>Modal</i> amely lehetőséget biztosít a naptárhoz kapcsolódó engedélyek kezelésére, bővítésére.

(4.22.) **Ellenőrizendő hibakezelések:** -

**Engedélykezelés:** *Write* hozzáférési joggal nem elvégezhető a művelet.

4.23. táblázat. Naptár törlése vagy követésének megszüntetése

<b>Given</b>	A felhasználó a több naptár fülön van.
<b>When</b>	A naptár törlése gombra kattint.
<b>Then</b>	Amennyiben a felhasználó az utolsó <i>Owner</i> jogosultsággal rendelkező profil, a naptár és minden hozzá tartozó adat törlődik, ellenkező esetben csak az adott felhasználó hozzáférése.

(4.23.) **Ellenőrizendő hibakezelések:** -

**Engedélykezelés:** *Write* hozzáférési joggal nem elvégezhető a művelet.

4.24. táblázat. Naptár beállítása alapértelmezettnek

<b>Given</b>	A felhasználó a több naptár fülön van.
<b>When</b>	A naptár beállítása alapértelmezettnek gombra kattint.
<b>Then</b>	A naptár alapértelmezettnek lesz beállítva. (Azaz belépéskor ez töltődik be.)

(4.24.) **Ellenőrizendő hibakezelések:** -

**Engedélykezelés:** -

## Profil funkciói

4.25. táblázat. Jelszó változtatása

<b>Given</b>	A felhasználó a profil fülön van.
<b>When</b>	A jelszó változtatása gombra kattint majd megfelelően paraméterezi a <i>Modalt</i> .
<b>Then</b>	A jelszava megváltozik.

(4.25.) **Ellenőrizendő hibakezelések:** jelszavak nem egyeznek. A jelszó 8 karakternél rövidebb. A jelszó nem tartalmazza: nagybetű, szám. A régi jelszó nem megfelelő.

**Engedélykezelés:** -

4.26. táblázat. Profil törlése

<b>Given</b>	A felhasználó a profil fülön van.
<b>When</b>	A profil törlése gombra kattint.
<b>Then</b>	A felhasználó és minden hozzá kapcsolódó adat törlődik. A hozzá tartozó naptárak a naptár törlési szabályai szerint kerülnek kezelésre.

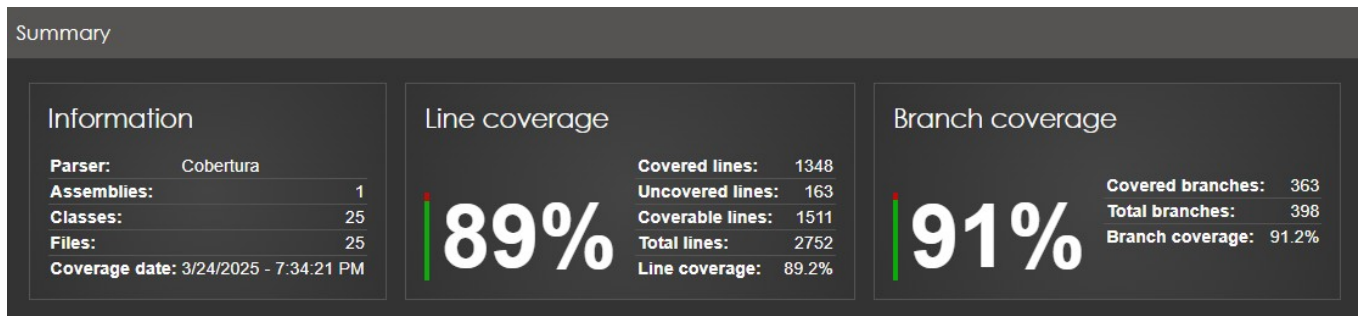
(4.26.) Ellenőrizendő hibakezelések: -

Engedélykezelés: -

## 4.6. Tesztelési eredmények

### 4.6.1. *Backend coverage*

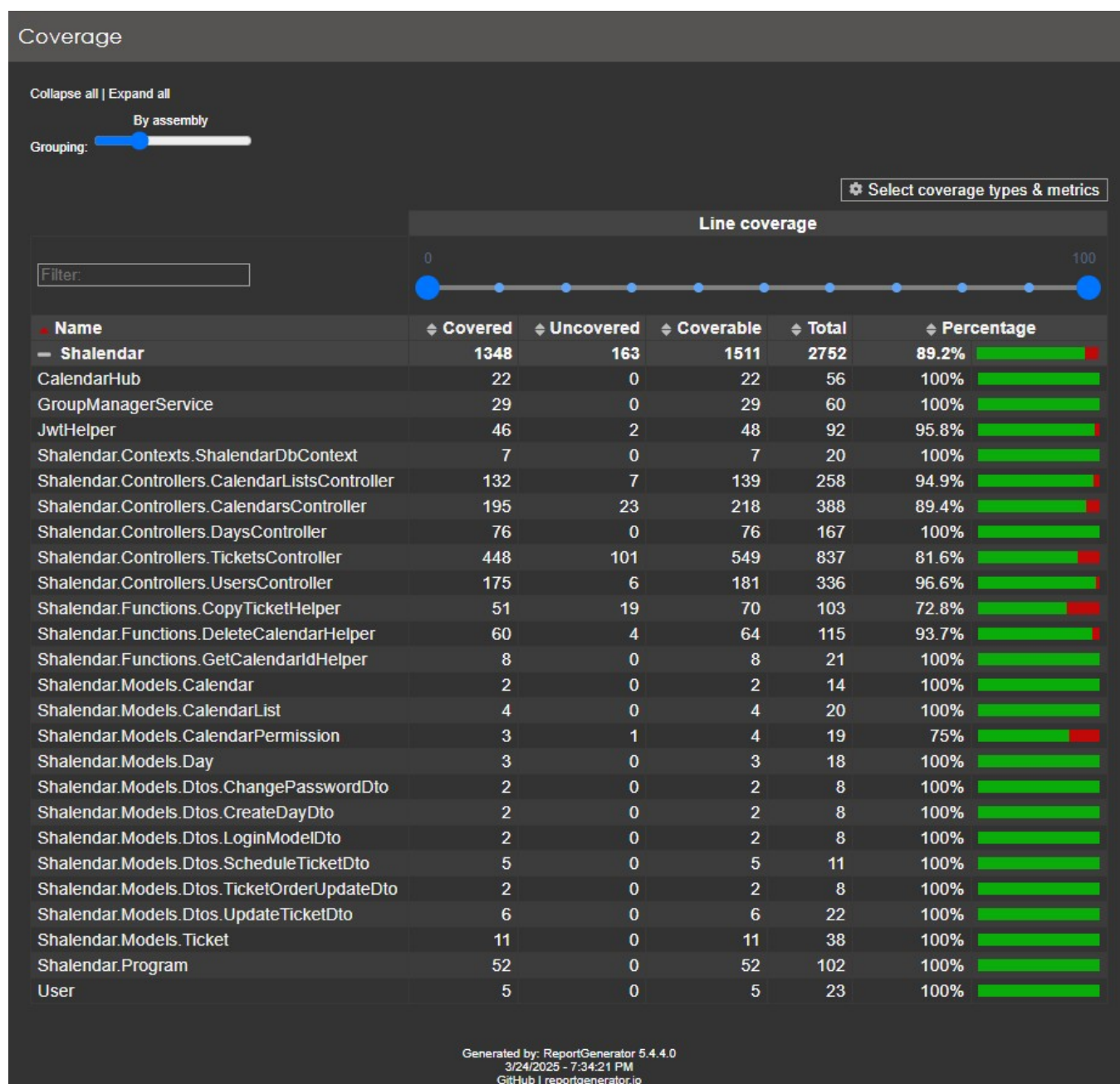
(4.2.) A *backend* tesztfájlokról *coverage report* készíthető a `generate-coverage.bat` futtatásával. Külön report készül az egység- és integrációs tesztekéről, valamint a 2 egyesítéséről. Az alábbi képen a teljes projekt lefedettsége látszik mind a 2 típusú teszt által. A reportok az alábbi úton érhetőek el: `.../api/Shalendar.Tests/coveragereport`



4.2. ábra. Lefedettségi mutató összefoglaló

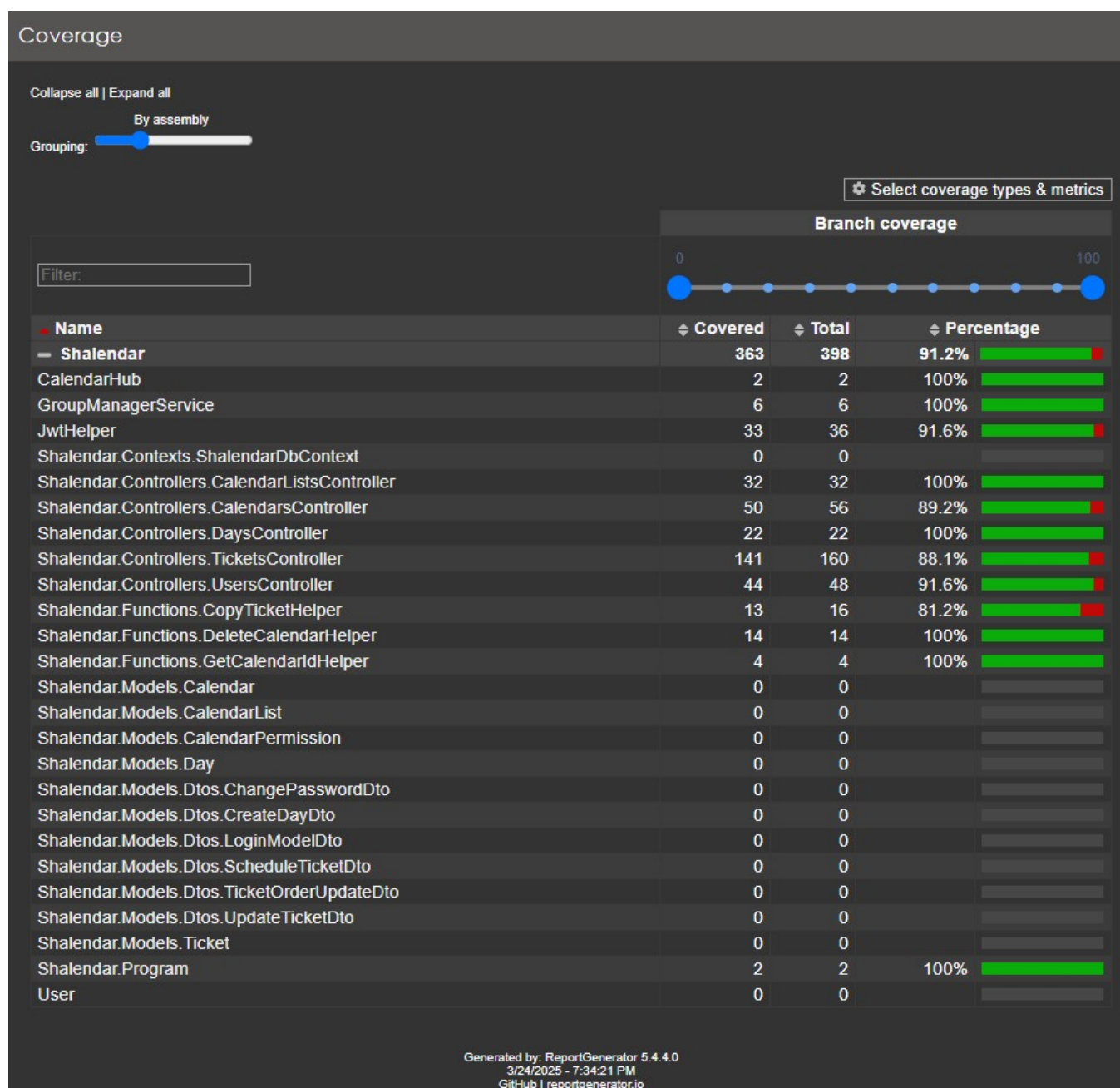
(4.2.) Az alábbi képen a részletes, fájlokra lebontott sor lefedettség olvasható.





4.3. ábra. Részletes lefedettségi kimutatás sorokra

(4.3.) Az alábbi képen a részletes, fájlokra lebontott ág lefedettség olvasható.



4.4. ábra. Részletes lefedettségi kimutatás ágakra

(4.4.) A kimutatásokban található fájlok megnyithatóak, ahol vizuálisan látható a sorok lefedettsége.

```
1 68      DateTime selectedDate = parsedDate.Date;
69
1 70      bool dayExists = await _context.Days
1 71      .AnyAsync(d => d.CalendarId == calendarId && EF.Functions.DateDiffDay(d.Date, selectedDate) == 0);
72
1 73      if (!dayExists)
1 74      {
1 75          return Ok(new List<object>());
76      }
77
0 78      var tickets = await _context.Tickets
0 79      .Where(t => t.CurrentParentType == "TodoList"
0 80      && t.StartTime == null
```

4.5. ábra. Lefedettségi mutató egy adott fájlban

#### 4.6.2. *Frontend e2e* teszt megtekintése

[Shalendar e2e teszt videó megtekintése \(Vimeo\)](#)



## 5. fejezet

# Összegzés

### 5.1. A projekt céljának összefoglalása

A Shalendar alkalmazás sikeresen megvalósult, ezzel egy kiváló eszközt adva azon emberek kezébe akik szeretik egy letisztult felületen, könnyedén és gyorsan, professzionális módon megtervezni hétköznapijaikat. Mindezek mellett létrejött egy jól strukturált, részletes dokumentáció is, mely stabil alapul szolgálhat minden jövőbeli felhasználó és fejlesztő számára. A dolgozat olvasását és megértését könnyíti a benne található 41 ábra, és 27 táblázat, mely a nagy átfogó képtől a legapróbb részletekig tárgyalja a körülbelül 15.000 soros szoftver működését.

### 5.2. További fejlesztési lehetőségek

A tapasztalatok alapján a rendszer jól bővíthető és skálázható. Továbbfejlesztési lehetőségei megszámlálhatatlanok.

Az alap funkcionalitás bővítése céljából bevezethető lenne egy új *Event* típus a *Ticket* mellé, amellyel az ismétlődő tevékenységeket lehetne felvenni napi, heti, havi vagy akár évi rendszerességgel, ezzel könnyen számontartva azokat. A

felhasználói élmény tovább fokozása érdekében a naptárakhoz készíthető több színtéma, ezzel támogatva azt, hogy a felhasználó még jobban magáénak érezhesse a programot.

Végül, de nem utolsó sorban a meglévő funkciók finomhangolása is fontos. Például a megjelenő hibaüzenetek optimalizálása a kellemesebb hatás érdekében, vagy regisztráció során egy ellenőrző email küldése.

A szoftver nem véletlenül lett *Web API backenddel* írva, mivel így könnyedén újrafelhasználható egy mobilos alkalmazás alapjaként, a kód bármilyen jellegű módosítása nélkül. Ezáltal felhasználóbarátabbá téve a mobiltelefonos használatot, így elérve az emberek nagyobb csoportját.

### 5.3. Személyes tapasztalat

A fejlesztési folyamat során számos új technológiát sajátítottam el, ezek közé tartozik a *Vue.js* keretrendszer használata, melyre a teljes *frontend* épül, vagy a különböző *NuGet* csomagok által biztosított *SignalR*, *xUnit*, *EntityFramework* technológiák. Megértettem, megtanultam átlátni a komplex rendszerek működését, valamint a különböző rétegek harmóniáját.

### 5.4. Zárógondolat

Összességében a *Shalendar* projekt nemcsak egy gyakorlati problémára kínál megoldást, hanem egyben lehetőséget adott számomra a modern webfejlesztési eszközök mélyebb megismerésére és alkalmazására. Ezek előkészítése, megvalósítása, tesztelése valamint dokumentálása a gyakorlati tudás megszerzésén túl, lehetővé tette, hogy Programtervező informatikusként hivatkozhaszak magamra.