**Eötvös Loránd Tudományegyetem**

**Informatikai Kar**

**UML-Conference  
 webes alkalmazás**

**Témavezető:**

Szalai-Gindl János Márk

adjunktus

ELTE Információs Rendszerek Tanszék

**Szerző**:

Szeifert Péter

Programtervező informatikus BSc.

Budapest, 2021

# Tartalomjegyzék

[1. Tartalomjegyzék 2](#_Toc71199098)

[2. Bevezetés 4](#_Toc71199099)

[3. Felhasználói dokumentáció 4](#_Toc71199100)

[3.1 Rendszerkövetelmények 4](#_Toc71199101)

[3.2 A program telepítése 5](#_Toc71199102)

[3.3 A program használata 5](#_Toc71199103)

[3.3.1 Belépés az oldalra 5](#_Toc71199104)

[6](#_Toc71199105)

[3.3.2 A felhasználói felület 6](#_Toc71199106)

[3.3.3 *(1)*A File Browser 7](#_Toc71199107)

[*3.3.4* 8](#_Toc71199108)

[3.3.5 *(2)* A megosztás ablak 9](#_Toc71199109)

[**3.3.6** **Az EDITOR-ról és egy projekt** felépítéséről 10](#_Toc71199110)

[3.3.7 Az Editor felépítése 11](#_Toc71199111)

[3.3.8 Editor funkciói részletesen 13](#_Toc71199112)

[4. Fejlesztői dokumentáció 14](#_Toc71199113)

[4.1 Felhasznált technológiák 14](#_Toc71199114)

[4.1.1 Kliens oldal 14](#_Toc71199115)

[4.1.2 Szerver oldal 15](#_Toc71199116)

[4.1.3 Adatbázis 15](#_Toc71199117)

[4.1.4 Fejlesztői környezet 16](#_Toc71199118)

[4.2 Szerkezeti felépítés 16](#_Toc71199119)

[4.2.1 Client-side 17](#_Toc71199120)

[5. Hivatkozások 22](#_Toc71199121)

# Bevezetés

Sokféle diagram szerkesztő áll rendelkezésünkre manapság, de többrésztvevős alig akad. Nagyobb szoftverek megtervezése esetén fontos, hogy a tervezési folyamat minél dinamikusabb, gyorsabb legyen, ezért a munkám során egy elosztott webes alkalmazást tervezek készíteni, amelyben közösen, akár egyidőben is lehet UML osztálydiagramokat és csomagdiagramokat szerkeszteni.

A program 3 rétegből áll. Egy angular kliensből, Java spring-boot back-end-ből, illetve postgreSQL adatbázisból. Az interaktív szerkesztő felülethez natív, szöveg alapú web socketeket használtam. Az összes többi kommunikáci REST API-n keresztül történik

A dokumentáció tartalmazza a felhasználói dokumentációt, ennek forgatásával egy új felhasználó közelebb kerül és segítséget kap a program helyes és produktív használatához.

Emellett tartalmazza a fejlesztői dokumentációt, amely a fejlesztőktek nyújt segítséget: Leírja a főbb komponensek egymáshoz való viszonyát, ezek főbb struktúrális ismérveit. Ennek a fejezetnek a forgatásával közelebb kerül az olvasó a szoftver működésének megértéséhez illetve reprodukálásához. A tesztelési tervet és a tesztelési eredményeket a fejezet végén olvashatjuk.

A továbbfejlesztéshez ötleteket olvashatunk a dolgozat végén, a forráslista előtt.

# Felhasználói dokumentáció

## Rendszerkövetelmények

Windows esetében a 3 program komponens futtatásához legalább 1,5 GB memóriát ajánlott szabadon hagyni. A java szerver átlagosan 800-1000mb memóriát foglal le, a kliensszerver és az adatbázisszerver memóriafelhasználása elhanyagolható. Processzor teljesítmény tekintetében bármi megfelel, amin az operációs rendszer elfut. A teljes program kevesebb, mint 1GB tárhelyet foglal.

Szükségünk lesz továbbá egy böngészőre. Én a chrome-ot, operát vagy firefox-ot ajánlok a használathoz. Az Internet Explorer és Edge böngészők nem támogatottak.

## A program telepítése

* Telepítsük fel A Java 8-hoz tartozó JRE-t, adjuk hozzá a környezeti változókhoz(fűzzük fel a classpath-ra)
* Telepítsük fel az npm-et, ezt is adjuk hozzá a környezeti változókhoz.
* Telepítsük a PostgreSQL 13-as verzióját. e
* indítsuk el az init.bat fájlt. Ez elindítja a postgresql ada

Nodejs-t ésPostgreSQL szoftvereket. Adjuk hozzá a a postgre /bin mappáját a környezeti változókhoz.

//majd berakok linkeket.

Indítsuk el a start.bat fájlt. <- ez elv így működik, nem kell semmit installálni, mindent betettem a projekt mammába ami kell. java, npm, postgre.

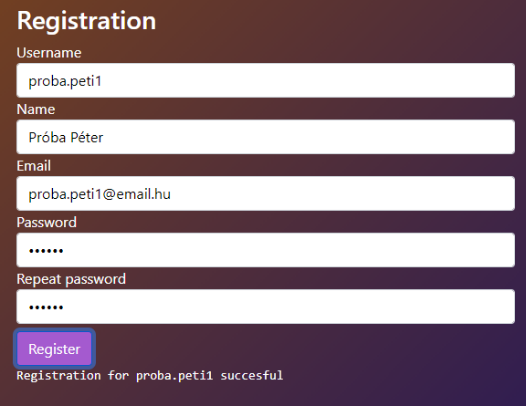
## A program használata

Nyissuk meg a böngészőt, navigáljunk el a <http://localhost:8100> oldalra!

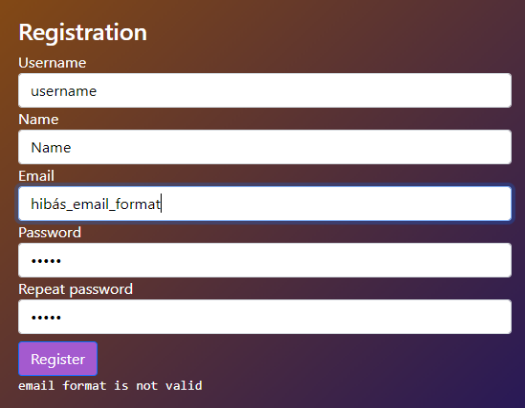
### Belépés az oldalra

|  |  |  |
| --- | --- | --- |
| GIVEN | WHEN | THEN |
| User belép az oldalra | Nincs bejelentkezve senki | Home oldal megjelenik |
| User belép az oldalra | Már be van jelentkezve valaki | A “Home” oldalra kerül a user, ahol a program köszönti őt. |
| User a regisztráció linkre kattint | Kitölti az email, username, Name és password mezőket, majd és a “Register” gombra kattint. | Ha az adatok formátumai megfelelőek, akkor a regisztráció sikeres. Most már be lehet lépni a megadott adatokkal. Nem megfelelő formátumok esetén hibaüzeneteket kapunk. |
| Login oldalon van a user | Beírja a login adatokat, enter | Ha helyes adatokat adtunk meg, a “Home” oldalra kerül a user. ellenkező esetben hibaüzenetet kapunk. |

### 



1. ábra: Sikeres regisztáció



2. ábra Hiba a regisztáció során

### A felhasználói felület

Oldalt,illetve felül fordított L alakban a menu sávban a következő opciók vannak: Home, Editor, Files Belépés után a „Home” felületet látjuk, ahol a program köszönt minket.

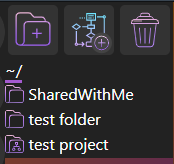
A **“be vagyunk lépve”** feltételt a továbbiakban **“BVL”** jelöléssel jeleztem.

|  |  |  |  |
| --- | --- | --- | --- |
| GIVEN | WHEN | THEN | EXPLANATION |
| BVL | Profile menüpontra kattintunk | **A főképernyőn** megjelennek a Profilunk részletei. | Itt megtekinthetjük, illetve módosíthatjuk az adatainkat; email, jelszó, username, stb… módosítására van lehetőség. |
| BVL | File browser menüpontra kattintunk | **Bal oldali panelen** kinyílik a file browser | link (1) |
| BVL, **nem** az EDITOR felület van megnyitva | Nagy “EDITOR” gombra kattintunk | **A főképernyőn** megjelenik a szerkesztő felület. | Ezen a felületen diagrammjainkat szerkeszthetjük. a fent leírt funkciókat továbbra is használhatjuk a **Bal oldali panelen.** A fájl böngészőben kikereshetjük diagrammjainkat, azokat megnyitva már szerkeszthetjük is. Ez élő socketes kapcsolattal történik, automatikus mentésekkel. Amint valaki más is elkezdi ugyanazt a diagrammot szerkeszteni, auomatikusan bekapcsolódik a session-ba. |

### (1)A File Browser

Itt tudjuk **böngészni** a **projektjeinket**. Vannak velem megosztott, általam megosztott és “privát”(nem megosztott) mappák, projektek.

A panelt kinyitva a gyökérkönyvtárt nyitja meg az oldal. Itt egy fix mappa található *~/sharedWithMe/*.

* A ~/sharedWithMe/ mappában a felénk megosztott tartalmak láthatók. Ezek virtuális linkek, kattintáskor ugyan arra afájlra navigálnak, mint amit a másik felhasználó is lát.

*3. ábra: file browser panel*

A file browser felület 2 részből áll, egy eszköztárból (felül) és nagyobb részt böngészőből (alul). A böngészőben projekteket, mappákat hozhatunk létre. Ezeket törölhetjük, illetve megoszthatjuk másokkal. Megosztáskor a célszemély olvasási jogot kap a mappákhoz illetve szerkesztői jogot a projektekhez. Másik személy projektjét vagy mappáját nem törölhetjük ki.

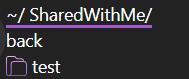
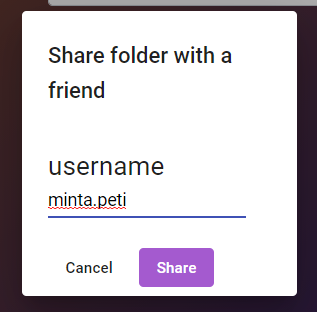
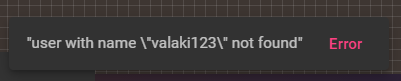
Minden normális fájlkezelőhöz hasonlóan a *File Browser* panel is rendelkezik az alapvető funkcionalitásokkal. Kiválasztás bal kattintással: egy elem, ctrl+ bal klikk-el egyesével lehet jelölgetni illetve jelölést megszűntetni. A kijelölt elemek kék keretet kapnak.

Felül, az eszköztárban 3 gomb van: Mappa létrehozása, Projekt létrehozása, törlés.

|  |  |  |  |
| --- | --- | --- | --- |
| GIVEN | WHEN | THEN | EXPLANATION |
| File Browser panel meg van nyitva | felül, a **mappa**/**projekt** **létrehozása** gombra kattintunk | létrehoz egy mappát/projektet oda, ahol éppen állunk | A lista végén villogó kurzorral várja a program, hogy elnevezzük. Már foglalt vagy üres név esetén a szerver hibaüzenettel válaszol. |
| Kiválasztunk egy, vagy több elemet | Felül a **törlés** gomra kattintunk | Kitörli a kiválasztott elemet a listából | Ilyenkor fa szerűen a beágyazott mappák, projektek is törlődnek. |
| Kiválasztunk egy elemet | Felül az **átnevezés** gomra kattintunk | A választott elem neve villogni kezd, input mező keletkezik. | Átnevezés esetén, ha előtte meg volt osztva az elem másokkal, akkor náluk is átnevezésre kerül. |
| Kiválasztunk egy mappát | A **megosztás** gomra kattintunk | Megnyílik egy párbeszéd ablak, ahol a barátunk nevét meg kell adni. | [(2) Megosztás ablak](#_(4)_A_megosztás) |

### 

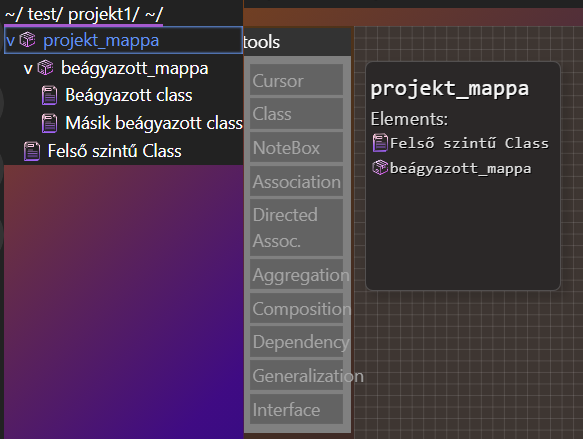
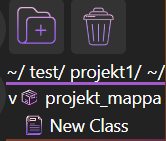
### (2) A megosztás ablak

Megosztott elemeket nem lehet tovább osztani. Ezt csak az eredeti tulajdonos teheti meg. Csak mappákat oszthatunk meg. projekteket önmagában nem. Ha a szerver megtalálta a nevet, akkor a mappa bekerül az ő vele megoszott mappák közé. Ha nem, a szerver hibaüzenetet küld kis popup ablakban.

4. ábra: nem létező felhasználóval akarunk megosztani valamit

5. ábra: megosztás ablak

6. ábra: a mappa megosztás sikeres (minta.peti user)

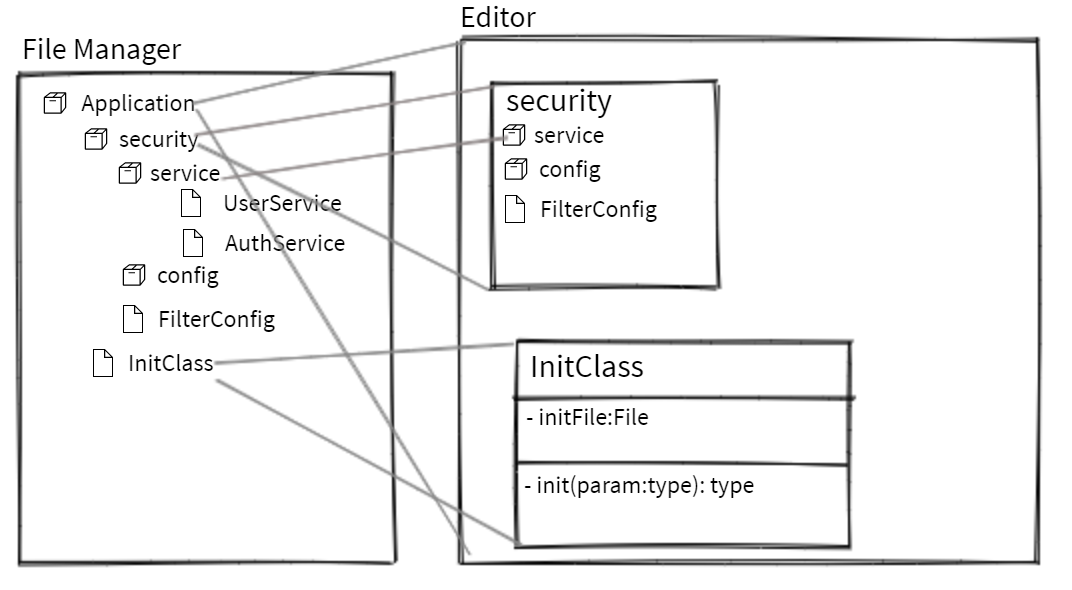
Ha a mappa, amiben állunk egy **projektmappa** (a projekt gyökér mappája is ilyen), akkor az eszköztár részen már csak 2 gomb látható. A **projektmappa létrehozása** gomb, illetve a **törlés**. Ezzel a két gombbal csak a projektmappák manipulálhatók. Továbbá, ha a manager ablakban létrehozunk egy projektmappát, akkor **az automatikusan megjelenik a session**-ban. Ha egy osztály definíciót létrehozunk, (Toolbox->Class) akkor az a szülő mappa alá virtuálisan bekerül a manager-ben, hogy jobban átlátható legyen, mi hova tartozik. Továbbá a mappa ikonja mellett balra található kis nyilacskával kinyithatók, illetve becsukhatók a mappában található elemek. Osztálydefinítiók szerkesztésekor a nagyszülő diagramban automatikusan frissülnek a csomag definíciók (akár élő kapcsolat esetén is)

7. ábra: Project management

8. ábra: A Manager és az Edtor szoros kapcsolata

### Az EDITOR-ról és egy projekt felépítéséről

Egy projekt konkrétan diagrammok struktúrált halmaza. Egy projekt lényegében egy mappa, amelyben almappák(diagrammok) vannak. Egy diagram lényegében osztálydefiníciók, megjegyzések, kapcsolatok  **és beágyatott diagrammok** egy csomagja, halmaza. Egy projekt struktúrája pl. a következő képpen nézhet ki:

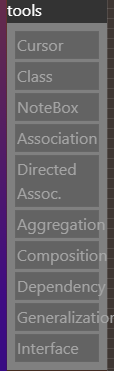


9. ábra: egy példa projekt felépítése (vázlat)

### Az Editor felépítése

Az Editor felépítése áll egy diagram szerkesztő felületből, illetve 2 mozgatható ablakból;

* Toolbox ablak



10. ábra Eszköztár ablak

Itt Osztályleíró dobozok(Class Obect), csomagok (Package) megjegyzések(note) illetve kapcsolati vonalak beszúrását választhatjuk ki.

Vonal típusok:

* + - Asszociáció 
    - Irányított asszociáció 
    - Aggregáció 
    - Kompozíció 
    - Függőség 
    - Öröklődés 
    - Interface 
* Log ablak

Itt az esetleges hibákról kapunk értesítéseket. Pélául nem szerkeszthető elemet próbálunk módosítani, vagy, ha a magas ping miatt hibás kéréseket kapunk, akkor a visszaállításokról itt kapunk tájékoztatást. Az egyes sorok fölé húzva az egeret az halvány kiemelést kap, illetve a tárgyban szereplő objektum is az Szerkesztő felületen.

11. ábra: Log ablak, működés közben

### Editor funkciói részletesen

|  |  |  |  |
| --- | --- | --- | --- |
| GIVEN | WHEN | THEN | EXPLANATION |
| Megosztott diagrammot szerkesztünk, egyedül | Valaki más is elkezdi szerkeszteni ugyanazt a diagrammot | Ő is **bekapcsolódik** a szerkesztésbe. | Ilyenkor az elemek **szerkesztését** **zárolásokkal** látja el a program, ezt a szerver a **memóriában** jegyzi. |
| Megosztott diagrammot szerkesztünk, többen | Osztályt leíró vagy komponens **dobozra** **kattintunk**, mert mozgatni, szerkeszteni vagy törölni szeretnénk. | **Ha nincs zárolva**, akkor **új zárat kap**(a benne lévő összes szövegmező is), csak szerkesztőnek engedélyezi a **módosítást** (pl.: kapcsolati vonalak módisítása), **mozgatást**, **törlést**.  Ha zárolva van, akkor valahogyan értesül a felhasználó erről. | A zárolás alatt álló elemek nem módosíthatók, törölhetők vagy mozgathatók. Ezek az elemek a többiek számára egy lakatot kapnak, illetve egy **keretet**, piros színnel. |
| **Szövegmezőre** **kattintunk** egy osztályban vagy komponens dobozban. | **Ha nincs zárolva**, akkor **új zárat kap.** Csak a szerkesztőnek engedélyezve a szerkesztést. | Ha szövegmezőt szerkesztünk, akkor szülőként maga a tartalmazó doboz **különleges lock-ot** kap: egy lakattal ellátott keretet, hogy ne lehessen kitörölni, illetve mozgatni, de a doboz **többi szövegmezője** attól még **szerkeszthető** marad a többiek számára. |

# Fejlesztői dokumentáció

## Felhasznált technológiák

### Kliens oldal

A Kliens oldalt, amit a végfelhasználó lát teljes mértékben AngularJS keretrendszerben készítettem. Azért az angulart választottam, mert nagyon megtetszett a moduláris felépítése, a rengeteg újrafelhasználhatóság, nézetek egymásba ágyazása könnyedén és átláthatóan. A nézetmodel (.ts fájl) elkülönül a nézettől (.html) illetve a stíluslaptól (.css vagy scss). Az angularról annyit elmondanék, hogy a többi mai front-end keretrendszerrel ellentétben az Angular alapértelmezett módon typescript-et használ, amelynek sok előnye van. Főleg, ha valami nagyobb, enterprise méretű applikációt szeretnénk készíteni. A typescipt tulajdonságaiból kiemelnék néhányat.

1. Lehet vele statikusan típusos javascript kódot írni.
2. A típusozott már „typescript” kódot a saját fordítója mezei javascript kóddá generálja.
3. Intellisense figyelmeztet a típusok betartására és helyes használatára. Ezzel rengeteg időt lehet megspórolni. Egyrészt a folyamatos autocomplete funkcióval gyorsabb a kódírás, másrészt típusibákra, elgépelésekre már fordítási időben figyelmeztet a fordító.

### Szerver oldal

A szerver oldalt, ahol az üzleti logika tartózkodik, Java nyelvben írtam meg, Spring Boot keretrendszert használva. A Spring Boot egy nyílt forráskódú Java alapú keretrendszer, amelyet leginkább a microservce-k építésére használnak az iparban. A Spring Boot a népszerű Spring keretrendszert használja a motorháztető alatt, a legszembetűnőbb különbség az, hogy míg Spring XML alapú Bean (objektum) definiálást használ, addig SpringBoot-ban ezt @annotációk megadásával sokkal egyszerűbben és átláthatóbban megtehetjük. Továbbá Rengeteg konfigurációt out of the box megírtak nekünk, hogy csak a Controller, Service és Repository rétegeket kelljen megírnunk. Természetesen minden Spring-es funkció elérhető és igény szerint konfigurálható.

Springen belül rengeteg dependenciát „könyvtár csomagot” használtam ezek közül a legfontosabbak:

* spring-boot-starter-data-jpa Az adatbázis kommunikációhoz
* spring-boot-starter-web Az alap RestController funkcionalitáshoz
* spring-boot-starter-test A teszteléshez
* spring-websocket Az editor socketes kommunikációjához
* jackson-databind A Json-Java POJO könnyed konverziójához
* spring-boot-starter-security A biztonsági alrendszerhez.

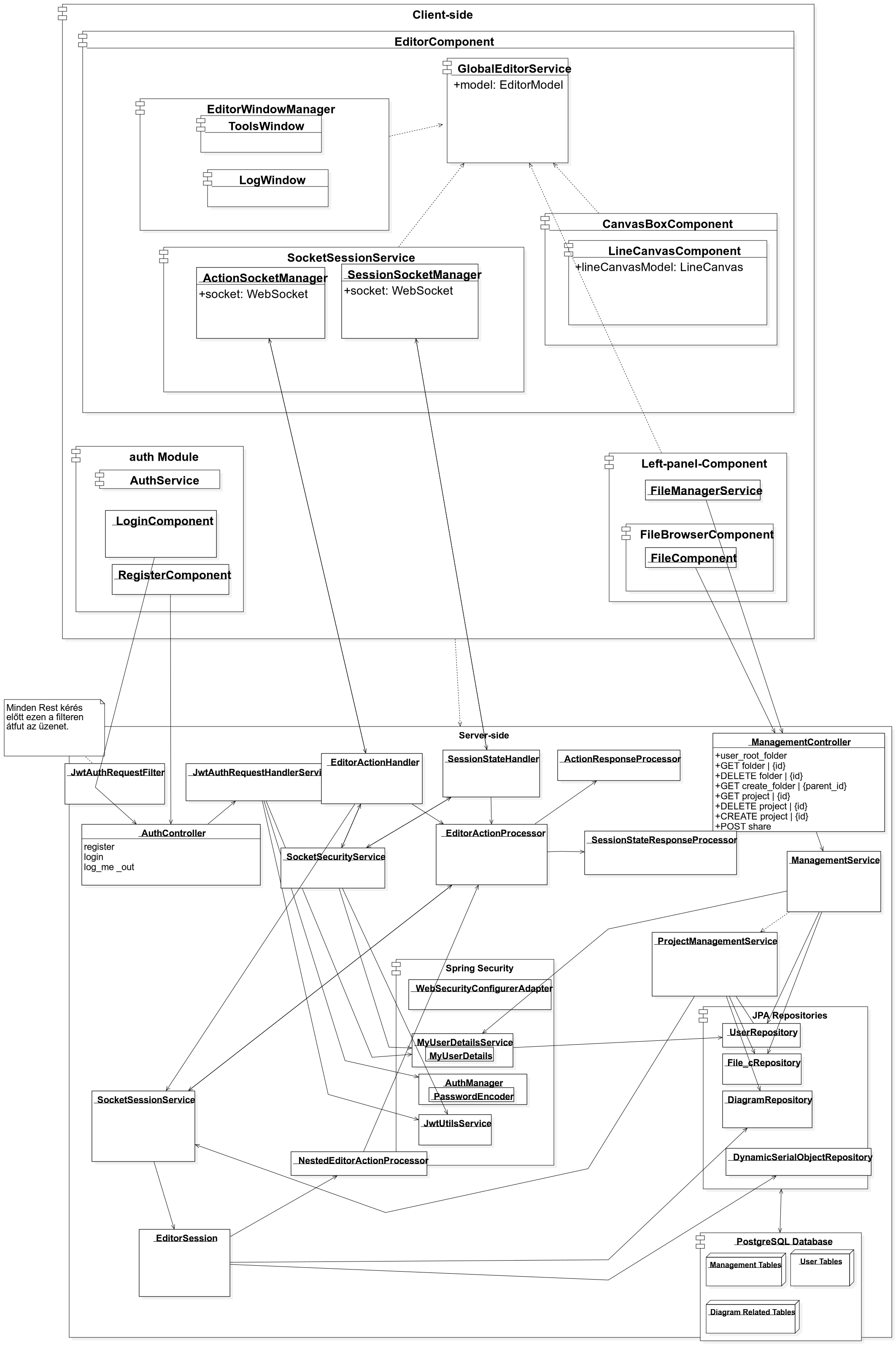
### Adatbázis

A perzisztencia réteg feladatait a PostgreSQL adatbáziskezelőre bíztam. Itt fontos megjegyeznem, hogy esetemben a spring JPA és hibernate könyvtárcsomagjai kommunikálnak az adatbázissal SQL nyelven. A táblákat és kapcsolatokat Közvetlenül a java forráskódban, @annotációk segítségével lehet létrehozni pl.: @Entity(tábla), @OneToMany, stb... Röviden, az adattáblák és kapcsolatok SQL-ben való megírásának terhét levették a vállamról.

### Fejlesztői környezet

A Kliens oldal fejlesztéséhez Visual studio code-t használtam (<https://code.visualstudio.com>) néhány angular illetve html bővítmény telepítésével. A spring-boot szerver fejlesztéséhez Apache NetBeans 12.0-t(<https://netbeans.apache.org/download/nb120/>), a PostgreSQL adatbázis menedzseléséhez pedig PgAdmin4-et használtam.

## Szerkezeti felépítés



NestedActionProcessor hiba van ->SocketSessionService-hez kéne közni, SharedialogComponent, EditorSocketControllerService rosszul van írva.

### Client-side

A kliens oldal megtervezésekor fontosnak tartottam, hogy egyértelmű, magától érthetődő ikonokat, gombokat válasszak. Olyanokat amelyek nem törik meg a felhasználói szokásokat. Az ikonok nagyrészét a <https://www.flaticon.com> oldalról töltöttem le. Az angular komponenseket úgy terveztem meg, hogy közülük minnél több újra felhasználható legyen. Ez az editor komponensben nagyobb értelmet nyer majd. Egy átlagos Angular komponens 4 fájlból tevődik össze: x.component.html, x.component.css, x.component.ts, x.component.ts

* x.component.html
  + Az x komponens nézet kódja, ez nem csak egy mezei html fájl. Rengeteg angular splecifikus attribute-ot írhatunk az egyes <tag>-ekbe pl.:<div ng-If=”this.value==’1’ ”> ezek közül néhány példa:
    - ng-If=”” a ”” jelek közé ts kódot írhatunk. Ezt futási időbenkiértékeli a javascript engine, ha igaz akkor az adott DOM elem megjelenik.
    - ng-For=”let item of this.list” egy listán végig iterálunk, az adott DOM elemet minden lista elemre létrehozza és egymás után fűzi.
* x.component.css
  + Itt definiálhatjuk azokat stílus szabályainkat, amelyek csak az adott komponens html kódjára lesznek érvényesek. Nem kötelező css formátumot használni, az angular projekt létrehozásánál scss illetve sass spreadsheet nyelveket is választhatunk én scss-t választottam. A globális szabályokat az src/app/styles.css fájlba írhatjuk bele.
* x.component.ts
  + Ez a komponens nézetmodellje. Itt definiálva van egy xComponent osztály amely @Component annotációval meghivatkozza a hozzá tartozó html és css fájlokat. Itt fogalmazhatjuk meg a kliens oldali logikát, a felhasználói eseteket, interakciókat, gomblenyomásokat, egérmozgatásokat, kattintásokat és még sok minden mást.
* x.component.spec.ts
  + Itt a komponenshez tartozó teszteseteket fogalmazhatjuk meg. Az angular default esetben Jasmine teszt keretrendszert ad a kezünk alá, Karma tesztfuttató szerverrel.

A teljes angular dokumentációt itt olvashatjuk: <https://angular.io/docs>

#### Auth module

Ez a modul a felhasználó authentikációjával foglalkozik. Két komponensből, illetve egy service-ből áll:

* LoginComponent
* RegisterComponent
* AuthService

##### Login Component

Ez a komponens semmi másért nem felel, csak azért, hogy a html kódban megírt form segítségével a felhasználó bejelentkezhessen. A .ts kódban a sendLogin( a:AuthRequest) fügvényhívással bejelentkezési kérelmet küldünk a /login endpoint-ra

##### RegisterComponent

Ez a komponens szintén semmi másért nem felel, csak azért, hogy a html kódban megírt form segítségével a felhasználó beregisztrálhassén. A .ts kódban a sendRegister( a:RegistrationRequest) fügvényhívással regisztrációs kérelmet küldünk a /register endpoint-ra

##### AuthService

Ez egy service. A servicek olyan ts osztályok, amelyek más osztályokba injektálhatók, dependency-ként. Ezt a @Injectable kulcsszó teszi lehetővé. Esetünkben az AuthService-nek egy lényeges Függvénye van: logOut(). Értelem szerűen ezt meghívva kijelentkezünk. Fontos megjegyezni, hogy itt nem csak arról van szó, hogy kitöröljük a bejelentkezéskor kapott authetication tokent, hanem szerver oldalon a tokenünk feketelistára kerül. úgyhogy ha valaki időközben valaki megszerzi a tokenünket, kijelentkezéskor az szerver oldalon érvénytelenné válik. Feketelistára kerül, amelyet az adatbázisban tárolunk, egy trigger felel azért, hogy a token lejárati dátumát meghaladva a rekord törlődjön. A trigger kódja megtalálható a back-end/src/main/resources/sql\_triggers.sql fájlban.

#### Bevezetés, Ősosztályok, Ősinterface-k

A következőkben megismerjük a fő kliensmodul működését, az editor modult, ezért fontosnak tartom az alap építőkövek előzetes megismerését:

##### ŐsInterface-k

* SessionInteractiveItem
* SessionInteractiveContainer
* LogInteractive\_I

A **SessionInteractiveItem** interface tartalmazza az összes olyan deklarációt, amely egy interaktív Item típusú komponens működéséhez kell.

A **SessionInterActiveContainer** kiterjeszti a SessionInteractive Interface-t néhány extra függvénydeklarációval. Ezek közül a legfontosabbak:

* createItem(model: DynamicSerialObject, extra?: any),
* restoreItem(item\_id: string, model: DynamicSerialObject)
* deleteItem(item\_id: string).

A LogInteractive\_I interface egyetlen metódust tartalmaz: highlightMe(on: boolean, color: string) Ennek csupán annyi a jelentősége, hogy a megadott boolean kapcsolóval és színnel az implementációban a nézetet úgy frissítse, hogy a színnek megfelelő árnyékkal tűntesse ki magát.

##### Ősosztályok

* InteractiveItemBase
* InteractiveContainerBase

Az **InteractiveItemBase** egy olyan abstract Bázis osztály, amely implementálja, deklarálja az összes olyan függvényt, methódust, amely egy interaktív elemhez kell. Itt külön kiemelném az editBegin() és EditEnd() függvényeket, melyekben általában rendre SELECT és UPDATE akciók küldése történik. Továbbá konstruktorban magába injektálja a EditorSocketControllerService-t és a CommonService-t előbbi a socket alapú kommunikációért és annak állapotának tárolásáért felel, utóbbi a logolásért. Az előbbihez hasonlókat lehet elmondani az **InteractiveContainerBase**-ről is, ugyanis kiterjeszti az **InteractiveItemBase** osztályt, de emellett implementálja az InteractiveContainer interface-t.

#### A legfontosabb service-k a fő modulban

##### GlobalEditorService

Ez talán a legfontosabb része a kliens oldalnak itt történik a diagramok lekérése és tárolása. Ha egy projektben navigálunk, projektmappák között váltogatunk minden esetben az initFromServer(dg\_id) függvény fog lefutni. Itt event kezelés is történik, bárki feliratkozhat bármilyen eseményre. A diagram lekérésekor a ’diagram\_fetch’aliasra hallgató eventet triggereli a service. Erre az eventre két komponens is fel van iratkozva:

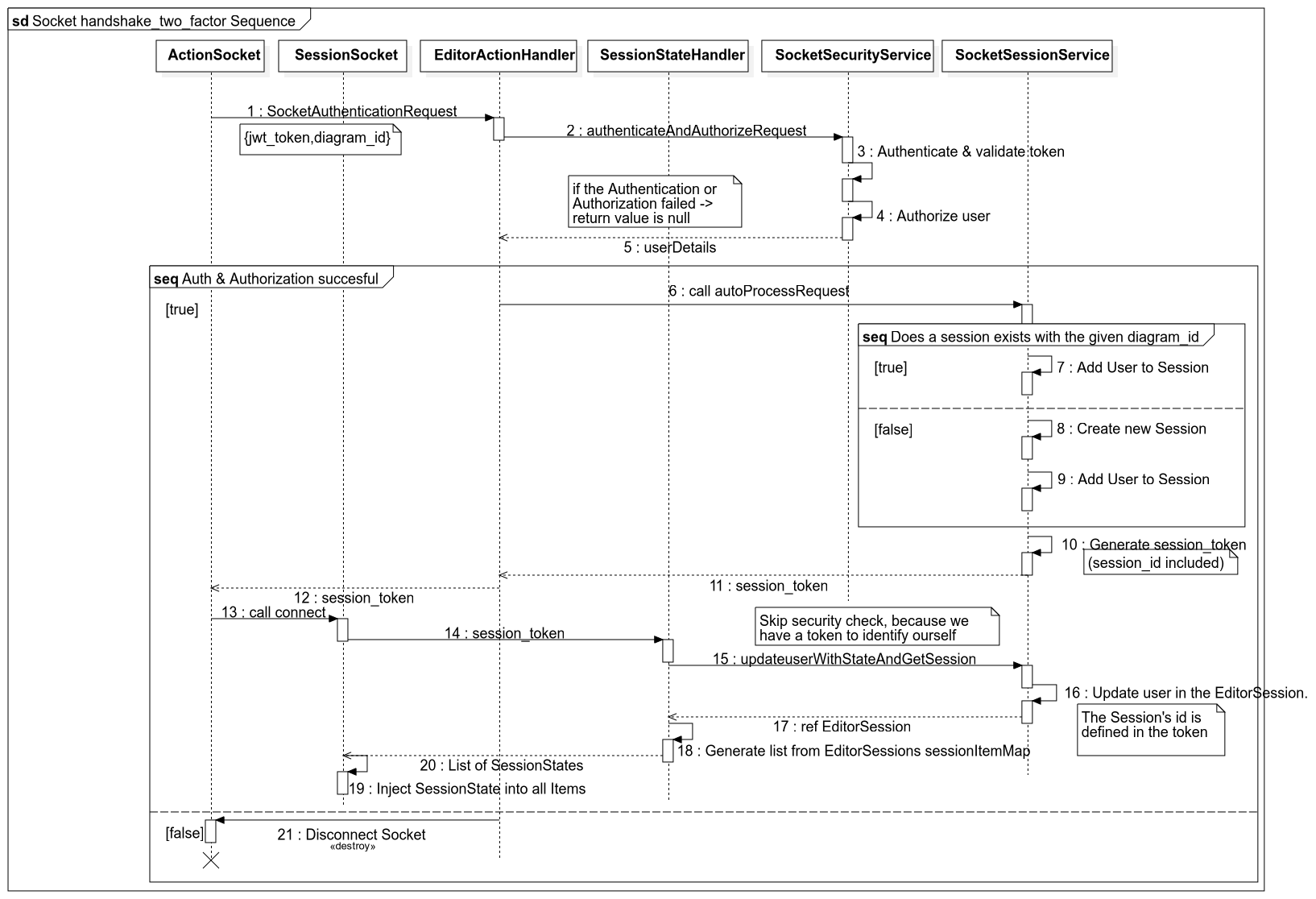
* LineCanvasComponent: az event hatására üríti az előző session vonalait, helyükre a frisseket inicializálja be.
* EditorSocketControllerService: az event hatására Létrehoz 2 socketet és csatlakozik a szerver websocket endpointjaihoz (action és state)

Egy event listener hozzáadása nagyon egyszerű. csak meg kell hívni az addListenerToEvent(target,fn,alias) függvényt, ahol az fn-nek egy lambda függvénynek kell lennie 1 paraméterrel. Amikor a triggerEvent(trigger\_alias) függvényt bárhonnan meghívjuk, akkor az összes lambda függvény lefut target paraméterrel, ahol a tárolt alias megegyezik a trigger\_alias-al. A konstruktorban egy lambda függvénnyel a ’canvas\_size\_update’ eseményre, amely tüzeléskor frissíti a canvas és a lineCanvas dimenzióit, hogy az összes objektum megjeleníthető legyen.

##### EditorSocketControllerService

Ez a szervíz az editor összes elemével kapcsolatban van, rajta keresztül fut a socket alapú aktív kommunikáció, itt kell felregisztrálniuk illetve leregisztrálniuk az egyes Itemeknek, konténereknek, amelyekkel interaktálhatunk a szerkesztőfelületen. ezeket a register, unregister, unregisterContainer és registerContainer függvények felparaméterezett hívásával tehetjük meg. A service a nyílvántartott elemeket típusuknak megfelelően a következő listákba teszi:

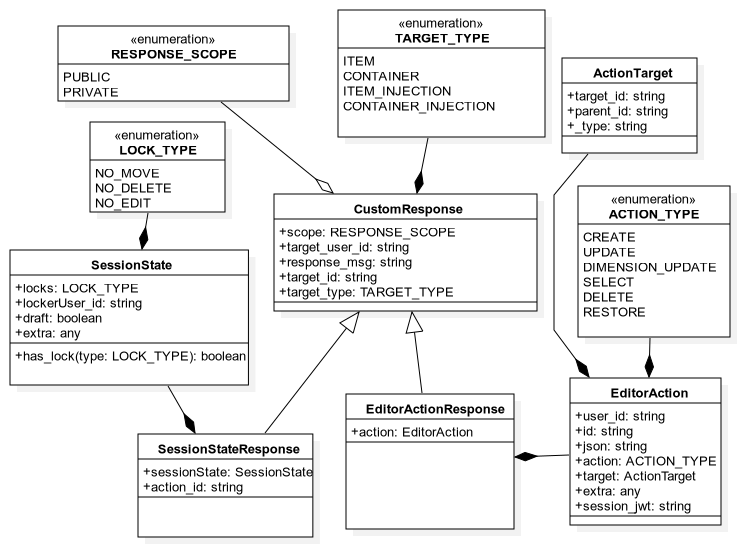
* itemViewModelMap:Pair<String,SessionInteractiveItem>[]
* containerViewModelMap:Pair<String,essionInteractiveContainer>[]

A constructor szintén kiemelt fontossűgú. A ’diagram\_fetch’ esemény hatására a connect() függvény fut le, amely csatlakoztatja az ActionSocket-et. Ha az ActionSocket csatlakoztatása sikeres és van hozzáférési jogunk a kért diagramhoz, akkor kapunk egy session\_token-t válaszként. Ennek a tokennek a felhasználásával második lépésként csatlakozhatunk a SessionSocket-el a szerver oldali session-hoz a token hashelve tartalmazza a felhasználónevünket illetve a szerver oldali session azonosítóját. Erről mellékelek egy szekvencia diagrammot, alább:

12. ábra: Kapcsolódás a socketekkel, azonosítás

A 'pre\_setup' esemény hatására újra inicializálódik az összes socket státusza, törlődnek a regzsztrált elemek, majd lecsatlakozik mind a 2 socket, amennyiben fenn áll aktív kapcsolat.

###### ActionSocket

Az interaktív szerkesztőfelület központi hídja ez az osztály. Implementálja a **SocketWrapper\_I interface-t**, amelyben minden fontos függvény deklarálva van, amely egy socketes interakcióhoz kellhet. A connect függvénnyel csatlakoztathatjuk a socketünket a paraméterben kapott címhez. A socket default függvényeit, mint például az onmessage, onopoen vagy onclose, felüldefiniáljuk. Ha a socketes kézfogás, illetve http2.0-s protokollfejlesztés megtörtént, akkor az onopen hatására elküldjük az egyedi authentikációs tokenünket, amelyet a bejelentkezéskor kaptunk és a cookie-k közé mentettünk. A folyamat többi részét fentebb már ismertettem. 

13. ábra: Response objektumok felépítése

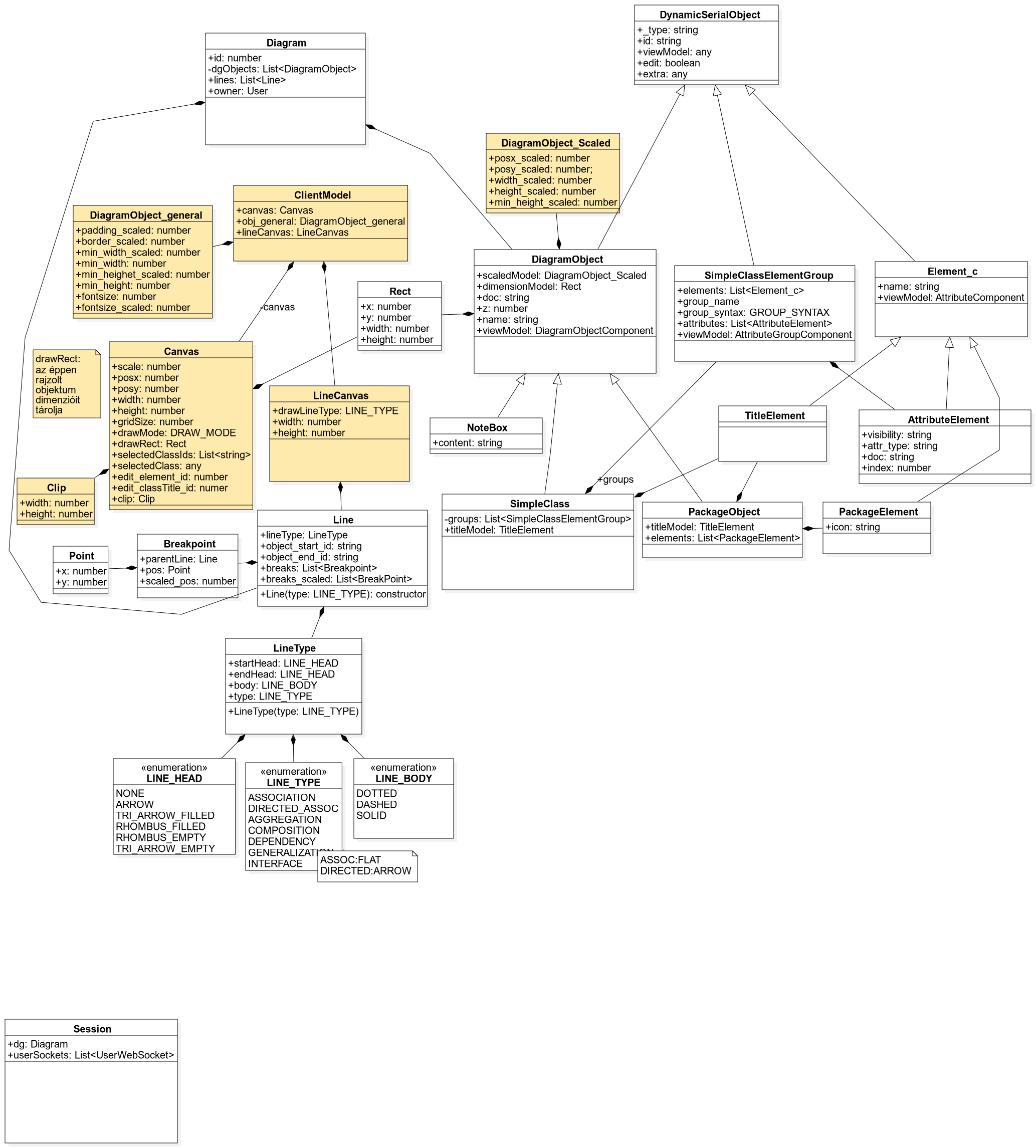
Bármikor, amikor a szerver küld egy akciót, akkor az **onmessage(e:any)** függvény fut le. Az üzenetet a függvény egyből egy EitorActionResponse objektummá alakítja. A **switch**-ben négy féle Akció típust különböztetünk meg:

* ACTIONTYPE.UPDATE
  + Valami módosítás történt. Az EditorSocketControllerService-től elkérjük a módosítandó objektum nézetmondelljét (id alapján), majd nemes egyszerűséggel az updateModel-t felparaméterezve meghívjuk.
* ACTION\_TYPE.RESTORE
  + Az előbbihez hasonlóan járunk el, annyi különbséggel, hogy itt a restoreModel-t hívjuk meg felparaméterezve. Ha az objektum törölve lett, amikor nem volt rá lakatunk, akkor a szülőobjektumot (container) megkérjük, hogy hozza létre újra a response.json-ból kiparsolt komponenst.
* ACTION\_TYPE.CREATE
  + A létrehozandó objektumot a response.json adattagban találjuk meg, ezt javascript objektummá kell alakítani. Ha a létrehozó user kapja meg az üzenetet, akkor a régi, ideiglenes id-t ki kell cserélni a szerver oldali id-ra a már létrehozozott item-ben. A régi id tárolására a response.action.extra objektum áll rendelkezésünkre, amelybe a szerver bele tette ’old\_id’ kulccsal a régi azonosítót. Ez alapján kikeresi a nézetmodellt majd frissít rajta. Ha nem mi vagyunk a tulajdonosok, akkor szimplán a szülő objektumot kérjük ki a response.action.target.parent\_id alapján az EditorSocketControllerService-től és rajta meghívjuk a createItem() függvényt felparaméterezve.
* ACTION\_TYPE.DELETE
  + Az objektumot id alapján elkérjük EditorSocketControllerService-től és a deleteSelfFromParent() függvényt meghívjuk rajta, aminek hatására a szülő container kitörli magából a törlendő gyermek objektumot.

###### SessionSocket

A 12-es ábra bemutatja a socketes kapcsolódás folyamatát az azonosítással együtt. Ennek a folyamatnak a második része a SessionSocket csatlakoztatása a szerverhez (/state websocket végpont). Ha ez megtörtént, akkor az ActionSockethez hasonlóan itt is A connect függvény inicializálja a socketet illetve felülírja a socket legfontosabb handler függvényeit, amiket a javascript virtuális gép hívogat majd számunkra. azonban itt a this.socket.onmessage=this.oninitmessage; paranccsal először egy ideiglenes handler függvénnyel írjuk felül az onmessage handler-t. az oninitmessage csak az első üzenetet kezeli, amelyik ugyanis egy listát tartalmaz az összes objektum állapotáról (state). Az átvétel után elkéri az interaktív elemek listáját az EditorSocketControllerService-től, majd id alapján beinjektálja az összes elem állapotát.Ennek a műveletnek a hatásáratűnnek el a null kifejezések a szerkesztőből. Az inicializálás után a this.parent.socket.onmessage = this.parent.onmessage; paranccsal átállítja a socket **onmessage** handler-jét az egyedi, általános handler-re. Az **onmessage** handler a státusz változásokat kezeli. Státuszváltozásnak számít az, amikor valaki létrehoz egy Objektumot (ezt nevezzük draft állapotnak), vagy amikor valaki kiválaszt valamit (zárol).

A lenti ábrán az alkalmazás adatmodelljei láthatók. A csak kliens oldalon létező modelleket sárga színnel emeltem ki. A back-end rétegen az adatmodellek kis mértékben eltérnek. A Minimális eltérés a perzisztancia réteg igényei miatt keletkeztek, ez a legtöbb esetben azt jelenti, hogy a gyermek objektum mindig hordoz egy referenciát a szülő objektumra. Ez azért szükséges, hogy a JPA könyvtárcsomag el tudja készíteni az adattáblákat az @annotációkkal kiegészített Java osztályokból (Entitásokból).



14. ábra: Az alkalmazás adatmodelljei (Objektum orientált megözelítésben)

#### CanvasBoxComponent

Ez a komponens implementálja a SessionInteractiveContainer interface-t, tehát ő is egy Interaktív konténer, lényegében a Diagram adatmodell nézetmodelljének az egyik fele. A szerkesztői funkcionalitás nagy része ebben a komponensben van leimplementálva. Néhány fonotsabb függvény:

* setup(): Feliratkozik egy lambdával a ’canvas\_size\_update’ eseményre, amely bármilyen DiagramObject pozíciófrissítése esetén tüzel. a lambda frissíti a Clip dimenzióit, amely a szerkesztőfelület keretét adja.
* zoom(e): A görgetési eseményeket kezeli le, módosítja a clientModel nagy részét.
* onMouseMove(e), onMouseUp(e), onMouseDown(e): a szerkesztői logika és az ezekhez tartozó interakciók logikáit tartalmazzák.
* cursorModel(e), drawClassMode(e),drawNoteMode(e): ezek a függvények a toolbox ablakban felsorolt műveletek implementációit tartalmazzák. A drawClassMode(e) és drawNoteMode(e) a sorok között elküldik a megrajzolt DiagramObjetumot socketen keresztül, a szervernek.
* corrigateTargetClassPosition(), corrigateTargetClassDimension(): az éppen megfogott doboz pozícióját illetve méretét a tervezőtrácshoz igazítja.
* repositionCanvas(): amikor a vásznat mozgatjuk (egér lenyomása + mozgatás), akkor módosítja a canvas pozícióját.
* corrigateCanvasPosition(): A szerkesztő vásznat vissza igazítja a keretbe (Clip), ha esetleg kihúznánk belőle.

##### LineCanvasComponent

A szerkesztői funkciók maradék része ebben a komponensben kaptak helyet. Implementálja a SessionInteractioveContainer interface-t, tehát ez is ez konténer. Miket tárolunk? Vonalakat. Ez a komponens a Vonalak megjelenítéséről gondoskodik. A megjelenítéshez html5 canvas-t használtam 2 dimenziós kontextusban, és a beépített beginPath(), moveTo(x,y), stroke(), stb... funkciókra támaszkodva rajzolom a vonalakat. A komponens az előbb ismertetett CanvasBoxComponent részét képezi, ezért a legtöbb felhasználói eseménykezelést (pl.: kattintás, egérmozgatás, billentyű leütés, stb..) a szülő komponens delegálja és továbbítja. Néhány fontosabb függvény:

###### init()

Az angular speciális @ViewChild(’valami\_id’) annotációját használva hivatkozhatunk bármely DOM elemre a html sablonban, ha azt #valami\_id jelzéssel elláttuk. Jelen esetben a #canvas-ra hivatkozunk, amelytől elkérjük a 2d-s Context-et. Itt különösen vigyáznom kellett arra, hogy a context-et csak akkor próbáljam el elkérni, amikor a cavas már teljes mértékben inicializálódott. Erre is van az angularnak megoldása, mégpedig az AfterViewinit interface, amely csak az ngAfterViewInit() függvényt deklarálja. Ez a függvény akkor fog lefutni, miután a teljes html sablon legenerálódott. Ez nekünk pont kapóra jön, úgyhogy itt hívjuk meg az init()-et. Az init ezen felül még a ResourceLoaderService segítségével betölti a Vonalvégződések svg képeit.

###### drawBegin(e,type), drawMove(e), drawEnd(e):

Ők új vonalak konkrét megrajzolásáért felelősek. A drawEnd(e) teszi fel az i-re a pontot, ugyanis itt jön létre a Vonal Saját nézetmodellje (this. createLineWithControllerLocally (this.lineInstance)) és kerül elküldésre (this.sendLineCreated(this.lineInstance)).

###### MathHelper

A matematikai számolásokért, collision detektálásért, vektor metszéspont és egyebek számolásáért a MathHelper segédosztály felel.

###### LineController

Ezt az osztályt úgy kell elképzelni, mint egy angular komponenst, csak nincs hozzá html template, sem css fájl. Implementálja az InteractiveItemBase interface-t ami felruházza őt minden olyan tulajdonsággal, hogy egy Session-ban interaktívan lehessen használni. A vonalak a megjelenítésükért maguk felelnek. Ennek a kötelességnek drawLine() függvény tesz eleget.

drawLine()

Ez a függvény, amennyiben nincsenek töréspontok (BreakPoint) a vonal 2 végpontja közé, amely 2 darab DiagramObject, húz a canvas-on egy egyenes vonalat. Ha vannak töréspontok, akkor végig iterál a skálázott töréspontokon, (breaks\_scaled) (mert ugyan is van zoom funkciónk), és azok összekötésével alakul ki a spline. ###

#### LeftPanelComponent

Ez a komponens tartalmazza a bal oldalipanelben elhelyezkedő file browsert. a ts fájl tartalmazza a kattintás illetve gomb események kezelését, a szerver rest endpontjait hívogató függvények egy részét, ezek a következők:

* createProjectToActual(name: string)
* createFolderToActual(name: string)
* createProjectFolderToActual(name: string)
* deleteFile(id, \_type)
* getFile(id, \_type)
* getRootFolder()  
  Egy tipikus rest kérés a következő képpen néz ki:

this.http

      .get<FileResponse>(environment.api\_url\_http +... {

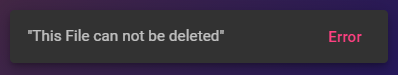
        headers: { 'Authorization': 'Bearer ' + getCookie("jwt\_token") }

      })

      .pipe(catchError(this.handleError<FileResponse>(this, '...', ...)))

      .subscribe((r) => { ... }

A fenti példában látszik, hogy http GET kérést indítunk a default api url-re, az Authorizációhoz a bejelentkezéskor kapott token-t mellékelni kell a kéréshez, különben 403-as Forbidden http választ kapunk. A további authorizáció a token segítségével szerver oldalon történik, amely hashelve tartalmazza többek között a felhasználónevet is. A komponens esetében az összes http kérés során fellépő hibát a handleError<T>(view: LeftPanelComponentComponent, operation = 'operation', result?: T) függvény kezeli le, illetve egy snackBar segítségével közvetíti a hibaüzeneteket a felhasználó felé. Erre egy példa:



15. ábra: Példa a snackBar használatára

#### FileManagerService

Ez a service néhány http rest kérést intéző függvényt tartalmaz, amelyet közösen használ a FileComponent és a LeftPanelComponent.

#### FileComponent

Ez a komponens a fájlok megjelenítésére szolgál. A kapott fájl típusa és az ICON enum értéke segítségével szerint dönti el a megjelenítés formáját.

##### A fájlok megjelenítése

A típusok a következők lehetnek:

* FolderDto
* folder
* project
* ProjectFolderDto
* projectFileDto
* ProjectFile
* ProjectFolder

Az osztálytípust a json szövegbe futási időben a \_type mezőbe írja a jackson csomag. Ezt az összes fájltípus ős interfészében definiáltam @JsonTypeInfo és @JsonSubTypes annotációk segítségével. Ha az objektumokat A Jackson saját ObjectMapperjével konvertáljuk json formátumba, akkor az annotációk figyelembevételével dolgozik. Egy folder például így néz ki:

{

...

icon: "FOLDER",

id: 3,

name: "SharedWithMe",

owner: {id: 1, userName: "user", email: "example@email.hu",   
 name:"Peter"},

\_type: "folder"

...

}

A megjelenítés mellett néhány event is helyet kapott a komponensben. Mivel előfordulhat, hogy a FileComponent-ek beágyazott módon helyezkednek el egymásban, ezért kiemelkedően fontos, hogy tudjuk melyik gyermek komponens indította a hívást és mit szeretne.

##### ShareDialogComponent

Ez a kis komponens a FileComponent mellett kapott helyet, mivel másol nem használtam fel. A nézetmodelljét szintén a file.component.ts fájlban lehet megtalálni, a nézet külön html-template-t kapott: ./dialog-share.html.

# Hivatkozások

Angular dokumentáció <https://angular.io/docs>