Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Automatizálási és Alkalmazott Informatikai Tanszék

ADATRÖGZÍTŐ ÉS -LEKÉRDEZŐ ALKALMAZÁS FEJLESZTÉSE SPRING ÉS ANGULAR PLATFORMON

Önálló labor dokumentáció

Gyuricska Milán

KONZULENS

IMRE GÁBOR

BUDAPEST, 2020

Tartalomjegyzék

1 Bevezetés

A feladatom, egy adatrögzítő és -lekérdező alkalmazás fejlesztése volt Spring és Angular platformon. A fő célom a projekttel, az ezen két keretrendszerhez kapcsolódó, jelenleg népszerű technológiák megismerése. Emellett az is fontos számomra, hogy amit csinálok, az ténylegesen felhasználható és praktikus legyen a területtel foglalkozók számára. Az Önálló laboratórium című tárgy keretében készíteném el ennek a szoftvernek a legalapvetőbb részeit, ezt később kiegészítve új funkciókkal, hasznos lehetne a programom.

1.1 A területről

Az alkalmazás Internet of Things (IoT) eszközöktől érkező adatokat rögzít. Ezekre az eszközökre node, datasource vagy adatforrás néven hivatkozok. Jellemző ezekre a node-okra, hogy korlátozott képeségekkel rendelkeznek, processzor teljesítmény, hálózati sávszélesség, valamint rendelkezésre állás tekintetében. Ezen okok miatt érdemes minimalizálni a logikát, amit ezektől az eszközöktől elvárunk, beleértve ebbe a kommunikációs és titkosítással kapcsolatos protokollokat is.

Az adatforrásokra szenzorok kapcsolódnak. A szenzorok általános tulajdonságokkal rendelkeznek, amik alapján értelmezhető a mérésük eredménye, ilyen tulajdonságok például a mértékegység, értékkészlet, érzékenység, offset stb. Ezen tulajdonságok névvel felcímkézett csoportját a szenzor paraméterének nevezzük. Egyszerűsítve, az adatforrás paraméterének. Például, ha ez a szenzor egy lázmérő, akkor a paraméter neve testhőmérséklet, mértékegysége °C, értékkészlete (-5°C – +50°C), érzékenysége 0.1°C.

Az adatforrások méréseket végeznek, ezeket időnként összecsomagolják és elküldik egy adat gyűjtő szolgáltatás felé, mint amilyen a dolgozat tárgyát képező szoftver is. Az elküldött adatcsomagok gyakran átalakításokon esnek át különböző köztes állomásokon.

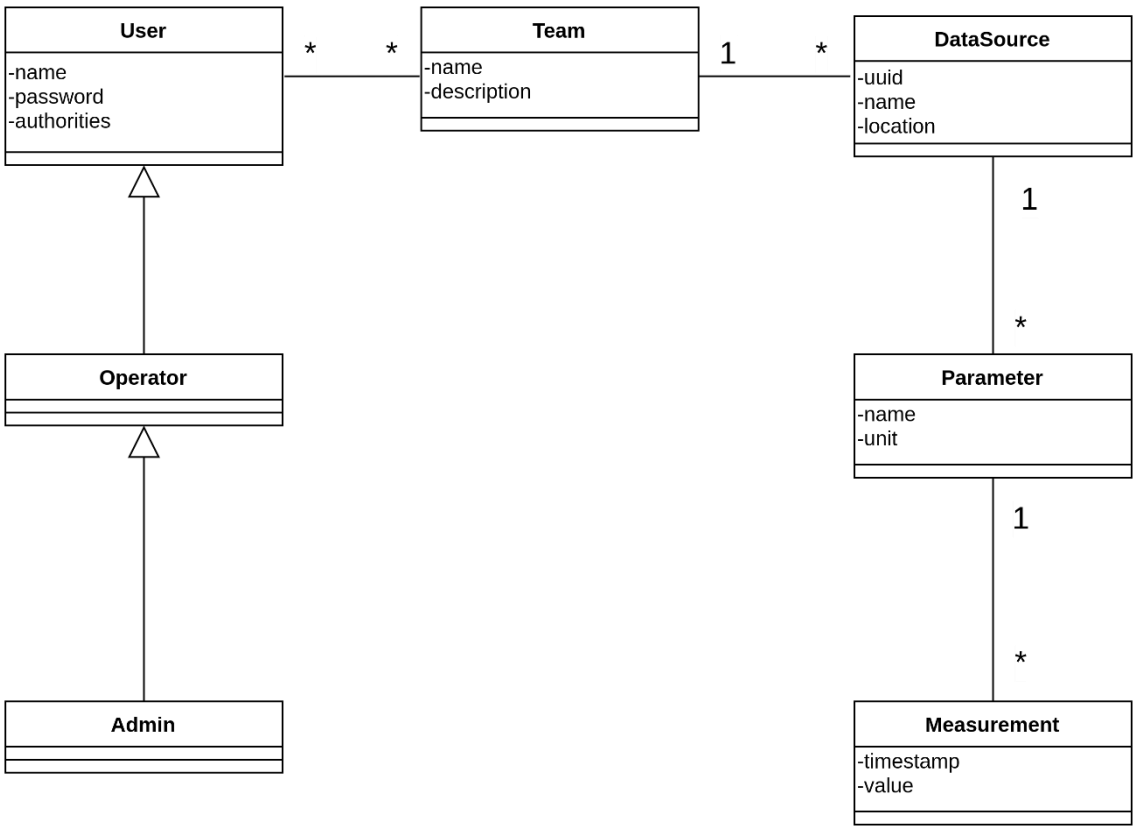
Ahhoz, hogy hasznosítható legyen a megérkezett adat, arra van szükség, hogy a mért értékek egyértelműen hozzárendelhetőek legyenek egy paramérhez, valamint ismert

legyen a mérés ideje is. Az érték, paraméter, időbélyeg hármast mérésnek, measurement-nek nevezzük. Ezt szeretnénk tárolni az adatbázisban.

2 Specifikáció

2.1 Funkcionális specifikáció

A data-collector alkalmazásom hálózaton keresztül érkező Internet of Things (IoT) adatokat rögzít adatbázisba, felhasználók által definiált szigorú struktúra szerint. Az alábbi UML osztálydiagramon látható a tárolt adatok szerkezete. Az itt megjelenő entitások megegyeznek az 1.1 Területről fejezetben leírt objektumokkal.



ábra 1: Adatbázisban tárolt entitások

A rendszer HTTP REST API interfészén keresztül fogad adatokat. A beérkező adatcsomagoknak meghatározott, szabványos json formátumban kell lenniük. Erre a formátumra példák:

```
{
  "uuid": "b525b1b7-1a16-4d01-96aa-6ee6a2814167",
  "payload": {
    "moisture_A_5cm": 13.55,
    "temperature_A_5cm": 14.39,
    "dielectric_A_5cm": 8.4,
```

```
    "pump_state": 1.0
  }
}

{
  "uuid": "0d5d96d4-d630-4788-b9df-048d04317c40",
  "payload": {
    "absolute_air_pressure": 944.5,
    "air_temperature": 3.6,
    "position_latitude": 47.69,
    "position_longitude": 18.91,
    "supply_voltage": 12.4
  }
}
```

A uuid mező küldése kötelező. A uuid az adatforrás egyedi azonosítója, egyben az adatküldéshez szükséges titkos api kulcs is. Az adatok küldéséhez nincs szükség felhasználói fiókra vagy bejelentkezésre, adatküldésnél kizárólag a uuid mező alapján történik az autentikáció és autorizáció. A payload objektum mező nevei az adatforráshoz tartozó paraméterek name tulajdonságával kell hogy megegyezzen. A payload objektum mezőinek értéke json formátumú szám kell hogy legyen. Lehet kevesebb paraméter nevet küldeni, mint amennyivel az adatforrás rendelkezik, ilyenkor csak a küldött paraméterek mentődnek el. Több paraméter küldése is engedélyezett, azokat a payload mezőket ignorálja a program, amihez nem tartozik az adatbázisban paraméter név.

Sikeres payload mező név és paraméternév hozzárendelés esetén új mérés kerül felvételre az adatbázisban a szerver saját órája szerinti időbélyeggel. Sikeres és sikertelen adatcsomag feldolgozás esetén is 200-as http kódot küld vissza az alkalmazás, hogy ne lehessen uuid-t kimerítő támadással felderíteni.

A rendszer többfelhasználós, minden felhasználónak csak a saját adataihoz van hozzáférése, ezeket jogosultsági szintjének megfelelően szerkesztheti. A felhasználók rendelkeznek egyedi névvel és jelszóval. A felhasználók teamekbe szerveződnek, ezek a jogosultságkezelést és az információ megosztást hivatottak szolgálni. Három féle felhasználó van, ezek jogosultságaik szempontjából különböznek.

A User felhasználótípusnak csak olvasási jogosultságai vannak. Bejelentkezhet az alkalmazásba, lekérdezheti saját adatait, kilistázhatja a teamjeit, azonban a teamben lévő többi felhasználót nem láthatja, olvashatja a team adatforrásait, az adatforrások paramétereit és a paraméterekhez tartozó méréseket.

Az Operator felhasználó rendelkezik a User jogosultságaival, ezen felül meghatározott jogosultságai vannak a saját teamjeiben. Lekérdezheti a saját teamjeiben lévő felhasználók listáját. Létrehozhat, módosíthat, törölhet a saját teamjével kapcsolatban álló adatforrást, valamint ezekkel az adatforrásokkal kapcsolatban álló paramétert.

Az Admin felhasználó rendelkezik az Operátor felhasználó tulajdonságaival, ezen felül minden más művelet végrehajtására is jogosult.

2.2 Nem funkcionális követelmények

Az alkalmazás adatszerkezete megtekinthető és szerkeszthető böngészős alkalmazásban. Ez az alkalmazás kis képernyőn is elfogadható felhasználói élményt nyújt. A kliens alkalmazás az Angular alkalmazások mintáit követi.

A backend alkalmazás a Spring alkalmazások mintáit követi és nem tartalmaz anti-pattern-eket. Az adatréteg elérése a Spring Data JPA interfészen keresztül történik és a lehetőségekhez képest viszonylag kevés, illetve hatékony SQL lekérdezést generál. A REST API a best practice-ek szerint működik.

A jelszavak nem tárolódnak a szerveren plain-text formátumban, helyette valamilyen salted hash algoritmussal vannak kódolva. Minden http kommunikáció szerver oldali TLS felett közlekedik. Az alkalmazás védekezik a cross site scripting (XSS) és a cross site request forgery (CSRF) támadásokkal szemben.

3 Hasonló szoftver megoldások

3.1 Ydoc-insights

A Ydoc-insights egy Windowsra írt adatgyűjtő és -vizualizációs alkalmazás. FTP, TCP, soros port, illetve szöveges file -ok formájában fogad adatokat. Az adatokat időbélyeg, paraméter azonosító, mértékegység, érték négyesek formájában várja, saját .ydoc file formátumában. A feldolgozott adatokat relációs adatbázisba menti.

Az adatforrásokat térképen megjeleníti, az adatokból grafikonokat és jelentéseket készít, SMS és Email riasztások leadására is képes. A windowsos vastagkliens mellett böngészős klienseket is támogat távoli eléréshez, ehhez webszervert futtat.

Régi szoftver, már nem nyújtanak hozzá terméktámogatást, sokszor nagyon lassú és instabil, nem nyílt forráskódú.

3.2 Mosquitto + Telegraf/Node-red + InfluxDB + Grafana

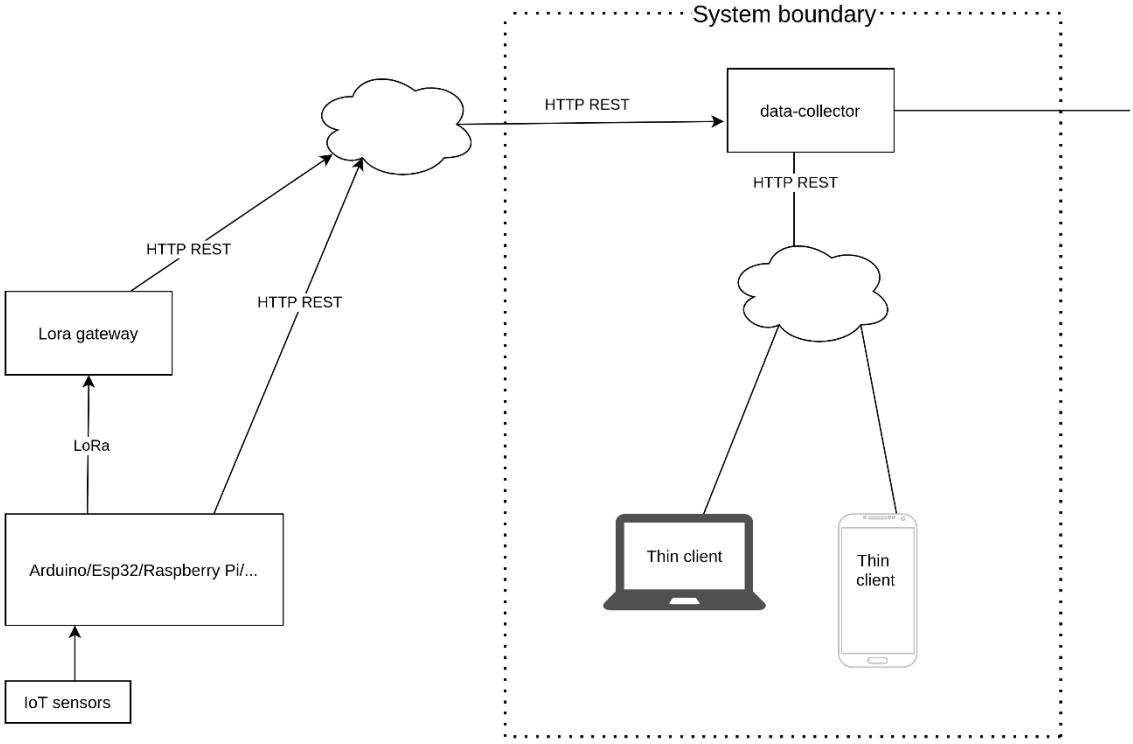
Manapság a legnépszerűbb megoldás. Nyílt forráskódú szoftverek, gyakran dockeres környezetben használják őket. A Mosquitto egy MQTT broker, fogadja és rövid ideig tárolja az IoT eszközöktől érkező üzeneteket. A Telegraf vagy Node-red kiolvassa az MQTT broker üzeneteit, ezeket feldolgozza és elmenti adatbázisba. Az adatbázis gyakran InfluxDB, ami egy idősoros adatbázis. Az idősoros adatbázisok optimalizálva vannak időbélyeggel ellátott értékek tárolására és lekérdezésére. Általában tömörítve tárolják a régebbi értékeket, a tömörítés gyakran intervallumokra vett átlagolást jelent, ezzel a leggyakoribb lekérdezésekhez jelentősen csökkenthető a háttértárról felolvasott adatpontok száma. A Grafana adatvizualizációs szoftver, segítségével változatos adatforrásokból tudunk grafikonokat és dashboardokat rajzolni.

3.3 Sensori.cloud

Alfa verzióban lévő adatgyűjtő és -vizualizációs alkalmazás. Cloud alapú, szoftver mint szolgáltatás, node -ok száma alapján számláz. Http, sms és ftp adatbevitelt támogat, változatos file formátumokban. Csoportok hozhatók létre benne felhasználóknak, a tárolt adatokhoz jogosultságkezelést ad. Előre definiálható eseményekről riasztásokat küld. Grafikonok és dashboardok készíthetők segítségével. A többi alkalmazáshoz képest bonyolultabb a kezelőfelülete.

4 Tervezés, implementálás

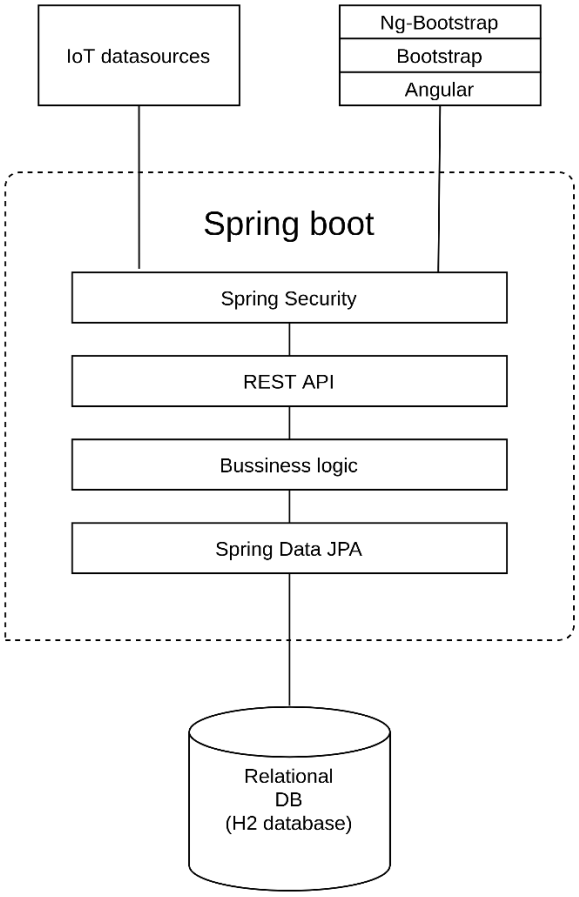
A tervezést a rendszer határainak meghatározásával kezdtem, ahogy ábra 2 is mutatja, kliens-szerver architektúrájú alkalmazást terveztem, az adatküldő IoT kliensekkel nem foglalkoztam.



ábra 2: A rendszer felépítése

4.1 Architektúra

Meghatároztam a felhasznált keretrendszereket. A szerver alkalmazáshoz a Spring boot keretrendszert választottam népszerűsége és kis meredekségű tanulási görbéje miatt. A biztonsági követelményekhez a Spring Security könyvtárat adtam hozzá a projekthez. Az objektum relációs leképezéshez (ORM) a Spring Data JPA könyvtárat használtam. A klienshez az Angular keretrendszert választottam, mert népszerű párosítás Spring-hez. A szebb és reszponzívabb megjelenést Bootstrap4 és ng-bootstrap könyvtárak biztosítják. A fejlesztés alatt a H2 relációs adatbázist használtam, mivel nagyon sok más népszerű relációs adatbázissal kompatibilis, valamint memória rezidens módban sokkal gyorsabban elindul, mint más hasonló alkalmazások, ezen felül még népszerű is.



ábra 3: Architektúra

4.2 A projekt

A projekt egy maven projekt. A projekt pom.xml leíró fájlja a Spring boot alkalmazásokban megtalálható spring-boot-starter-parent leíró file-t hivatkozza be, ez tartalmazza a konkrét maven artifact-ek verzióit. A parent pom fájlból a legújabb verziót adtam meg, hogy minél frissebb modulokkal dolgozhassak. Így végül java 11, maven 3.6.3, Spring boot 2.4.0, Spring Security 5.4.1 verziókat használtam.

Rendhagyó módon, az egyszerűség kedvéért az Angular kliens alkalmazást is maven-nel fordítottam és csomagoltam. A két projektet összevontam, a kliens lefordított kódja fordítás során becsomagolódik a backend statikus fájljai közé, így nem szükséges külön statikus webszerveren szolgáltatni a kliens alkalmazás fájljait.

A kliens maven-nel való buildelésére a maven-front-end-plugin-t [1]-t használtam, ez a plugin letölti a node.js -t, az npm -et és lefordítja az Angular klienst. A végeredmény hogy egy

```
mvn package
```

utasítás kiadás hatására lefordul a back-end, front-end, lefutnak mindkét alkalmazás tesztjei és egy közös csomagolt artifact-et kapunk. Az alkalmazás futtatása például a spring-boot maven plugin használatával lehetséges:

```
mvn spring-boot:run
```

4.3 UI

A felhasználói felület oldalait a funkcionális specifikáció use-case-ei alapján határoztam meg. Ezek Angular komponensekként jelentek meg az alkalmazásban, entitás listák valamint entitás részletező oldalak, amikre a listákról lehet navigálni. A részletező oldalakon Create, Read, Update, Delete (CRUD) műveleteket lehet végrehajtani az entitáson valamint a vele közvetlenül kapcsolatban álló más entitásokkal.

4.4 REST API

A felhasználói felületek gombjai és egyéb vezérlői alapján hajtottam végre a backend HTTP REST API interfészének tervezését. Az interfész szándékom szerint a Richardson Maturiy model [2] kettes szintjén áll, valamint követi a Microsoft REST API Guidelines [3] című dokumentum tervezési ajánlásait.

4.5 Spring komponensek

Minden Spring komponensben konstruktoros függőség injektálást alkalmaztam, a Lombok könyvtár @RequiredArgsConstructor annotációjával generáltam a konstruktorokat. Mivel állapotot csak entitásokon keresztül tárolok a szerver alkalmazásban, minden saját controller, service és repository komponensem singletonként injektálható.

4.6 Controllerek

A REST API megvalósítását a spring-web modul által nyújtott @RestController annotációval ellátott osztályokban végeztem. Az osztályokban engedélyeztem a

localhost:8080, localhost:4200 címet Cross origin kérések forrásának, productionben a végleges kliens címet kéne betölteni konfigurációs fájlból.

A controllerek Service osztályokat használnak, ezektől entitás objektumokat kérnek le, azonban ezeket átalakítják Data Transfer Object (DTO) -té, amivel végül visszatérnek. A json sorosítást a keretrendszer végzi. A controllerek DTO-kat kapnak a http kérések törzsében, ezeket entitássá alakítják és így küldik tovább a Service-eknek.

4.7 Data Transfer Object-ek

Az entitásokat a saját @Service annotációjú osztályokban használom, más helyeken, például a hálózaton keresztüli kéréseknél, DTO -kat használok. A DTO-k konvertálását kézzel oldottam meg, mert fel tudtam mérni, hogy mennyi idő kézzel megírni ezeket, viszont azt nem tudtam, hogy mennyi idő megtanulni egy ilyen mapper könyvtár használatát.

A DTO -k nem tartalmazzák kollekcióként a velük kapcsolatban álló entitások DTO-it. Ennek nagy hátránya, hogy a kliensnek több REST kérésből kell összeraknia a modelleket. Viszont sokkal egyszerűbb lesz tőle az API megértése, ugyanis soha nem kell azzal foglalkozni, hogy mi fog történni a szerverre felküldött objektumok kollekciójában lévő valahol módosult objektummal.

Kevés érdekes dolog történik a DTO-kban. A jelszavakat nem küldjük le a UserDto részeként, a User entitás pontos típusát egy dtype nevű mezőben küldjük le a kliensnek, ugyanis szükség van ott is a többszintű jogosultságkezelésre. A mezők validációja deklaratívan megadott validátorokkal is a DTO osztályokban történik. A measurement-nek kétféle DTO -ja van, az egyik az IoT kliensektől jön, a másik az Angulár klienssel tart kapcsolatot.

4.8 Szolgáltatások

A szolgáltatások a repository-kat használják. Az entitásokon végeznek műveleteket. Az egynél több SQL lekérdezést igénylő metódusokra kitettem az @Transactional annotációt, ezzel kibővítve az egyébbként csak a repository-k metódusára érvényes tranzakciót. Bizonyos service metódusok olyan entitásokkal térnek

vissza amiknél szükség van a lazy módon betöltődő kapcsolódó entitásokra is, ilyenkor általában a proxy kollekció size() metódusának meghívásával töltöm be ezeket. Próbáltam minél hatékonyabban használni az adatelérési réteget. A service réteg hiba esetén futási idejű kivételeket dob.

4.9 Repository-k

A Spring Data JPA által interface alapján generált repository-kat használom. Egy namedquery-t használok, egy User entitás lekérdezésére a Team -jeivel együtt:

```
@Query("SELECT u FROM User u LEFT JOIN FETCH u.teams WHERE u.id = :id")
Optional<User> getOneWithTeamsFetchedEagerly(@Param("id") Long id);
```

4.10 Entitások

Az adatszerkezetet első körben ER diagramon vettem fel, csak a rendszer működésének demonstrálásához szükséges legalapvetőbb tulajdonságokkal. Az ER diagram alapján UML osztálydiagramot készítettem. Az osztálydiagram alapján felvettem a java osztályokat. A java osztályok alapján SQL tábla leírással felvettem az adatbázist, figyelve, arra hogy olyan tábla és oszlopneveket és típusokat használjak, amit a Hibernate alapértelmezetten megtud feleltetni a java osztálynak és mezőinek, hogy kicsit kevesebb konfigurációt keljen megadni. A java osztályban felvettem a leképezést megadó JPA konfigurációkat. Kis SQL és entitás kód változtatásokkal elértem hogy a hibernate séma ellenőrzője elfogadja a leképezést. Próbáltam olyan SQL típusokat használni amik minél több adatbázis szoftverrel kompatibilisek, ugyanis még nincs kiválasztva végleges adatbázis szoftver

Az entitások equals és hashcode függvényét felüldefiniáltam, az equals az entitás generált id -je alapján működik. A hascode konstans számot ad vissza, hogy betartsa az equals es a hashcode contract-jét.

A JPA műveleteket kaszkádosítom mindig a szülő felöl, a gyerekentitás felé.

4.11 Hibakezelés

A hibákat a GlobalControllerExceptionHandler osztályban kezelem, az @ControllerAdvice annotáció segítségével képes elkapni a controllerekben nem elkapott kivételeket. A hibákat logolom és visszatérek a megfelelő http státusz kóddal.

4.12 Biztonság

A Spring Security-t az @Configuration annotációval ellátott SecurityConfig osztályban konfigurálom, ami leszármazik a WebSecurityConfigurarAdapter osztályból, a legtöbb alapértelmezett viselkedést meghagyom, csak kevés dolgon változtatok.

Az @Bean annotációval beteszek a Spring kontextusba egy PasswordEncoder példányt, ezzel fogja kódolni a jelszavakat. Pontosabban egy DelegatingPasswordEncoder-t teszek a kontextusban, ami az alapértelmezés, szóval olyan mintha nem csináltam volna semmit, de legalább látszódik, hogy melyik encodert használja a Spring. Ez az encoder sokféle algoritmust támogat egyszerre, a nyílt szövegű jelszó elején zárójelben lehet megadni az algoritmus paramétereit. A jelszó kódolása a UserService osztály createUser() és modifyUser() metódusokban történik és innentől kezdve mindenhol csak a bcrypt salted hash algoritmussal kódolt jelszó tárolódik, és jó esetben nem kerül ki a szerverről se, ugyanis nem sorosítódik json-be a UserDTO -ból.

Component scant-t kihasználva hozzáadok még a kontextushoz egy UserDetailsService példányt, ezzel felülírva az alapértelmezettet. Ez a JpaUserDetailsService név alapján keres User entitást, tehát az autentikációnál küldött névnek egyeznie kell a User entitás nevével.

A biztonsági konfigurációs osztály configure() metódusában, bekapcsolom a http basic autentikációt, mert ez tűnt a legegyszerűbb autentikációs megoldásnak, amit ráadásul könnyű is tesztelni. Kikapcsolom a CSRF védelmet, mert erre saját megoldást szerettem volna. Kikapcsolom a sütiket, mert ha nincs CSRF védelem akkor nagyon veszélyesek a sütik. Beállítom, hogy az API kéréseknél basic autentikáció legyen, más kéréseknél viszont ne legyen semmilyen autentikáció.

Az autorizációt controller metódusonként ellenőrzi az alkalmazás, az éppen autentikált User példány jogosultságai alapján, illetve a többfelhasználós biztonsághoz az AuthorizationManager nevű közönséges komponens metódusai alapján.

4.13 A Kliens

A kliens teljesen átlagos Angular 11 alkalmazás. HTML formokat tartalmaz, amelyek vezérlői injektált szolgáltatáson keresztül REST kéréseket küldenek. Az oldalak Bootstrap 4 könyvtár segítségével vannak formázva, néhány html elem ng-bootstrap direktívákat tartalmaz az egyszerűbb adatkötések érdekében. A HttpService osztály nyílt szövegben tárolja a bejelentkezett felhasználó adatait, illetve ezeket elmenti/betölti a böngésző local storage-ébe. Kijelentkezéskor a local storage-ből is törlődik a felhasználó adat. Az alkalmazásban route-ok segítségével lehet navigálni, a /login kivételével minden route-ot Authguard véd.

A kliens alkalmazásban is megjelenik a többszintű jogosultságkezelés, minden felhasználó csak azokat a gombokat, felületeket látja, amihez jogosultsága van.

5 Data-collector értékelése

A szoftver működőképes, teljesíti a funkcionális követelményeket, lehet vele szerkeszteni az adatgráfot. A nemfunkcionális követelmények közül a felhasználói felület kinézete rendben van. A kliensben, csak az Authguard mintát nem sikerült megoldani. A jelenlegi megoldás valamennyire működik, de nagyon csúnya. A Spring best practice-eket és anti-patternek-et nem tudom megítélni. A Spring Data JPA-n keresztüli adatelérés közepes hatékonyságú lett. A jelszavak kódolva vannak tárolva. RSA kulcspár és TLS tanúsítvány lett generálva a projekthez, sajnos ingyenes CA aláírást nem adtak ingyenes domain névhez, a saját CA által aláírt tanúsítványhoz használata nehéz, mert mindenhol importálni kell a saját CA tanúsítványát is. A CSRF védelem megvalósult a sütik kikapcsolásával, de a felhasználói adatok local storage-ban tárolása fokozza az XSS sebezhetőséget, mert az oldalon futó javascript hozzáfér ehhez.

6 Továbbfejlesztési lehetőségek

Nagyon sok továbbfejlesztési lehetőség van. Legfontosabb talán az autentikáció lecserélése egy modernebb és jobb megoldásra, beleértve ebbe az Angular kliensben a bejelentkezési állapot tárolását és az Authguardokat. Hatékonyabb Spring Data JPA használatot lehetne elérni nagyobb tudással. Grafikonokra és dashboardokra lenne szükség, hogy ténylegesen adatvizualizációs szoftver legyen. Deployment kitalálása, esetleg Dockeres környezet, CI szerverrel. Adatbázis kiválasztása. Adatbázisindexek felvétele. Reagálni a beküldött adatokra, riasztással, esetleg irányítani beágyazott eszközöket. FTP és MQTT támogatás.

7 Hivatkozások

[1] [Online]. Available: https://github.com/dsyer/spring-boot-angular.

[2] [Online]. Available: https://en.wikipedia.org/wiki/Richardson_Maturity_Model.

[3] [Online]. Available: https://github.com/microsoft/api-guidelines.