SZAKDOLGOZAT

Vaszily Zoltán

2022

Magyar Agrár- és Élettudományi Egyetem

Károly Róbert Campus

Edzéstervező alkalmazás fejlesztése Spring boot keretrendszer használatával

Konzulens

Dr. Pántya Róbert

egyetemi adjunktus

Készítette

Vaszily Zoltán

gazdaságinformatika szak

nappali tagozat

2022

Tartalomjegyzék

Bevezetés

Programozással kapcsolatos tanulmányaimat Java nyelven kezdtem el, azóta az egyetem és a munkámban számos programnyelvet próbálhattam ki, mint a C#, PHP, javascript, phyton. Ha egy újdonság megtanulásáról vagy egy ötlet megvalósításáról van szó, akkor mindig a Java-t választom. Így volt ez a szakdolgozatomnál is.

A fejtörést a „Mit?" kérdés okozta, végül úgy döntöttem, hogy egy edzés tervező alkalmazást fogok fejleszteni. Egy olyan programot, amivel nem csak gyakorlatokat lehet listába szedni, hanem amivel lehet a hobbi sportolókat támogatni abban, hogy jobb eredményeket érjenek el. Én is hobbi sportolok és volt is pár ötletem, hogy a program mire is legyen képes, de ahogy hozzáértőkkel beszéltem egyre gyűltek az ötletek. Mint az edzésmunka számszerűsítése, az egyes izomcsoportok monitorozása, túl-, illetve aluledzés tekintetében, edzéstervek megosztása és lehetőség biztosítása a diskurzusra. Hamar beláttam, hogy túl nagy feladat egy embernek, valamint edző szakember bevonása is szükség lenne egy ilyen alkalmazás fejlesztéséhez. Ezért úgy döntöttem, hogy egy leegyszerűsített változatban elkészítem az edzés tervező részét (későbbiekben WokroutPlanner) az alkalmazásnak és a fejlesztés tapasztalatait a későbbi verziók soron fogom kamatoztatni.

Úgy gondolom, hogy egy ilyen alkalmazásnak az lenne a legjobb, hogy több felületen lehetne elérhető, ezért a felépítését úgy terveztem, hogy az üzleti logikát és az adatbázis kezelését egy szerveren futó Java alkalmazás végzi, majd a felhasználó és a Java alkalmazás között egy böngészőre írt kliens, illetve egy telefonos alkalmazás végzi. A szakdolgozatom csupán a szerverre szánt Java alkalmazásról fog szólni. A fejlesztéshez a Spring boot keretrendszert választottam, amiben van egy kevés fejlesztési tapasztalom. A keretrendszer rendkívül széleskörű és könnyedén használhatóak benne a harmadik fél által készített program könyvtárak is. A szakdolgozatom célja a Spring Boot bemutatása a WorkoutPlanner alkalmazás fejlesztésével, a keretrendszer szolgáltatásai rendkívül széleskörűek egy ezeknek egy töredékét tudom bemutatni, abból viszont látható lesz, hogy a keretrendszerrel egyszerűen lehet létrehozni önálló működésre képes alkalmazást.

Az alkalmazás fejlesztéséhez a git verziókezelő programot használtam, ami támogatva van a szintén fejlesztéshez használt IntelliJ integrált fejlesztői környezet által. A programkódból

a maven csomag menedzselő szoftver segítségével készítek futtatható fájlt, amit szintén biztosít az IntelliJ beépítve. Az adatok tárolásához H2 adatbázis használtam, ami a memóriában tárolja az adatokat, tehát minden újraindítás után elvesznek. A fejlesztéshez nagyszerűen használható, ugyanis biztosit egy böngészőből elérhető konzolt, ahol bármikor ellenőrizhető az adatbázis állapota.

Az elkészült program alkalmas regisztrációra, az edzéstervek összeállítására, visszakeresésére, módosítására, értékelésére, továbbá edzésekre és edzéstervek való feliratkozásra. Azonban bőven akadnak fejlesztési lehetőségek, egyrészt abból fakadóan, hogy egy leegyszerűsített projektről van szó, másrészt a fejlesztésből levont következtetésekből.

1. Technológiai háttér

Az alkalmazás fejlesztéséhez Spring boot-ot használtam, ami leggyakrabban egy mikro szervizek írására használt, nyílt forráskódú keretrendszer. A Spring boot egy másik keretrendszerre épül, ez pedig a Spring. Ez a ráépülés úgy történik, hogy a Spring rengeteg konfigurációs lehetőséget tartalmazó keretrendszer, míg a Spring boot egy alapértelmezett konfigurációval ellátott rendszert készít ebből. Így nem kell minden beállítást a fejlesztőnek elvégezni, csupán a meglévő beállításokat kell felülírnia amennyiben szükséges. Spring fejlesztési nyelve java, az egyik leggyakrabban használt program nyelv. Dolgozatomban bemutatom ezeket a technológiákat, ráépülési sorrendben. A java programnyelvtől kezdően, majd a Spring boot-al bezárólag.

1.1 Java programnyelv bemutatása

A nyelvet James Gosling fejlesztette 1995-ben. Egy osztály alapú objektum orientál nyelv, ami segít az újra használható programkód írásában. A célja az „írd meg egyszer, futtasd bárhol” azért a java alkalmazások bájt kódra fordulnak és egy úgynevezett java virtuális gép futtatja így bármilyen platformon futtatható, ami támogatja a java-t. [https://www.w3schools.com/java/]

Jellemzői:

- Objektum orientált
- Használ primitív adat típusokat
- Bájt kódra fordul, amit a JVM (java virtual machine) futtat, így platform független
- Magas szintű programozási nyelv
- Erősen típusos
- Biztonságos

[www.geeksforgeeks.org/java/]

Az elkészülése óta, már a 19 verzió számú kiadásnál tart a programnyelv, dolgozatomban java 8-at használok. Ebben a kiadásban számos új szolgáltatással gazdagodott a nyelv, valamint a Spring minimum java 8-at követel meg. [https://www.java.com/releases/]

Új szolgáltatások többek között:

- Lambda kifejezések
- Metódus referenciák
- Funkcionális interfészek
- Stream API
- Alapértelmezett metódus
- Date Time API
- Optional osztály

[https://www.interviewbit.com/blog/java-8-features/]

A program implementálása során használt újdonságokat kiemelem és bemutatom, hogy miben segítik a fejlesztők munkáját, illetve azokat a megoldásokat, hogy kellene megvalósítani az új szolgáltatások nélkül.

1.1.1 Stream API

A java funkcionális stílusú program csomagja. A Stream-ek nem tárolnak elemet, egy adatstruktúrából, egy tömbből keresztül viszi egy művelet pipeline-on, anélkül, hogy az adatforrást módosítaná. Ha például az műveletek eredményét kell elmenteni, az egy új Streambe fog történni, ami gyorsabb lefutást eredményez. [https://www.javatpoint.com/java-8-stream]

```
return workoutService.getAll().stream().map(WorkoutResponse::new)
.collect(Collectors.toList());
```

1. ábra: Workout objektumok átalakítása WorkoutResponse objektumokká java stream használatával

Forrás: Saját, WorkotPlanner forráskód

A fenti kódrészletben a workoutService –el segítségével lekérdezem az összes Workout objektumot, amit egy List típusú objektummal tér vissza. Majd ezt stream típusúvá alakítva kezdetét veszi a funkcionális része, ahol meghívhatunk különböző filtereket, átalakítókat, rendezőket a végén mindin egy stream-el tér vissza. Jelen esetben a Workout típousú objektumokat alakítja át WorkoutResponse típusúvá úgy, hogy az rendelkező egy konstruktorral ami Workout paramétert vár. Végül egy kollektor függvény összegyűjti és

egy List objektumba helyezi, amivel visszatér. A Stream API jól alkalmazhatók lambda kifejezésekkel együtt, amit a későbbiekben bemutatok, de az összehasonlíthatóság miatt a fent kód stream nélküli implementálását az alábbi kód tartalmazza.

```
List<Workout> workouts = workoutService.getAll();
List<WorkoutResponse> workoutResponses = new ArrayList<>();
for(Workout w : workouts){
    workoutResponses.add(new WorkoutResponse(w));
}
return workoutResponses;
```

2. Ábra: Workout objektumok átalakítása WorkoutResponse objektumokká java stream használata nélkül.

Forrás: Saját, WorkotPlanner forráskód

1.1.2 Lambda kifejezések

Egy rövid kód blokk aminek lehetnek bemeneti paraméterei és visszatérési értéke. Hasonlóak a metódusokhoz, mert egy törzsben lehet implementálni a működésüket, de nem újra felhasználhatóak. Anonim függvényként és szokták hívni.
[https://www.w3schools.com/java/java_lambda.asp]

```
toDelete.getWorkouts().forEach(workout -> {
    workout.setWorkoutPlans(workout.getWorkoutPlans()
            .stream().filter(wp -> wp.getId()!= toDelete.getId()).collect(Collectors.toList()));
    workoutService.save(workout);
});
```

3. Ábra: Lambda kifejezés használatának illusztrálása egymásba ágyazott for ciklus esetén

Forrás: Saját, WorkotPlanner forráskód

A fenti programkódban egy több a többhöz kapcsolat esetén a törlés előkészítését lehet látni. A toDelete egy WorkoutPlan típusú változó, van egy Set típusú workouts mezője amiben Workout típusú objektumokat tárolok. A Workout típusnak szintén van egy Set típusú mezője, ebben WorkoutPlan típusú objektumokat tárolok. A készrészlet felelős, hogy végig iteráljon az adott WorkoutPlan-hez tartozó Workout objektumokat és a workoutPlans

mezőjének egy olyan Set-et állítson be, ami már nem tartalmazza a törölni kívánt WorkoutPlan objektumot majd a frissítést lementi az adatbázisba. A lenti kódrészletben szemléltetem, hogy valósulna meg lambda kifejezések nélkül.

```
for(Workout workout : toDelete.getWorkouts()){
   Set<WorkoutPlan> updatedWorkoutPlanSet = new HashSet<>();
   for(WorkoutPlan workoutPlan : workout.getWorkoutPlans()){
      if(workoutPlan.getId()==toDelete.getId())
         updatedWorkoutPlanSet.add(workoutPlan);
   }
   workout.setWorkoutPlans(updatedWorkoutPlanSet);
   workoutService.save(workout);
}
```

4. Ábra: Egymásba ágyazott for ciklus lambda kifejezés használata nélkül

Forrás: Saját, WorkotPlanner forráskód

1.1.3 Optional osztály

Egy konténer osztály, amely vagy null értéket vagy egy nem null értéket tartalmaz. Ha az isPresent() függvénye igazzal tér vissza akkor a get() függvénye nem null értékkel fog visszatérni. Akkor használjuk, amikor egy visszakapott érték lehet null. Például amikor egy adatbázisból lekérdezünk egy elemet azonosító alapján. Az alább program részletben is ez történik, ahol az orElseThrow() függvényét hívom meg. Itt, ha null értéket tárol az objektum, a megadott Exception-t dobja az érték helyett, a kívánt üzenettel. [https://docs.oracle.com/javase/8/docs/api/java/util/Optional.html]

1.2 A spring keretrendszer bemutatása

A spring egy könnyűsúlyú keretrendszer. Úgy lehet rá gondolni, mint a keretrendszerek keretrendszere, ugyanis számos keretrendszert támogat, mint például a Hibernate, Logback stb. [https://www.javatpoint.com/spring-tutorial]

A Spring keretrendszer előnyei

- Előre definiált sablonok
- Lazán összekapcsolt
- Könnyű tesztelhetőség
- Könnyűsúlyú
- Gyors fejlesztés
- Erőteljes absztrakció
- Deklaratív támogatás

A Spring keretrendszer hátrányai

- Komplexitás
- Rengeteg XML konfigurációt követel
- Nehezen tanulható
- Paralel mechanizmus: Több megoldási biztosít a fejlesztők számára, ahol rossz választás esetén teljesítmény csökkenést eredményez.

[https://www.educative.io/answers/what-are-java-spring-framework-advantages-and-disadvantages]

A keretrendszer szolgáltatási előtt tisztázni szükséges egy fogalmat. Ami pedig a Java bean (továbbiakban bean). Olyan osztályok gyűjtőneve, ami megfelel a következő kritériumoknak:

- Van olyan konstruktora, ami nem vár argumentumot
- Szerializálható, implementálja a Serializable interfészt
- Rendelkeznie kell metódusokkal az mezők értékének kinyeréséhez és beállításához, vagyis rendelkezzen getter-ekkel és setter-ekkel. Itt fontos azt is megjegyezni, hogy névadási konvenció betartása fontos, mert különböző könyvtárak ezekkel a nevekkel fogják meghívni ezeket a metódusokat.

[https://www.edureka.co/blog/what-is-javabeans/]

1. táblázat: Getter és Setter elnevezésének bemutatása

Mező neve	Getter	Setter
moovieTitle	getMoovieTitle()	setMoovieTitle(title)
displayName	getDisplayName()	setDisplayName(displayName)

Forrás: saját készítés

Inversion of control container és bean-ek

Az inversion of control úgy is ismert, mint függőség injektálás (dependency injection), valamint a bean olyan objektumok gyűjtőneve, amit az inversion of control container képes kezelni. Egy olyan folyamat, ahol az objektumok konstruktor argumentumokon, argumentumokon keresztül előre definiálják függőségeiket egy gyártó metódusnak. Vagy olyan mezők, amik az objektum létrejötte után lesznek beállítva. A konténer ezután injektálja a függőségeket, amikor létrehozza a bean-t.

További szolgáltatások:

- Erőforrások (Resources)
- Validáció, Adat kötés, Típus konverzió
- Spring Expression Language
- Aspektus orientál programozás támogatása
- Null-safety
- Adat bufferek és kodek
- Naplózás (logging)

[https://docs.spring.io/spring-framework/docs/current/reference/html/core.html]

1.3 Spring boot keretrendszer bemutatása

A Spring boot egy olyan Spring alapú keretrendszer, amely segít stand-alone (önállóan futó), használatra kész alkalmazásokat készíteni.

Célkitűzései:

- Gyors és széles körben elérhető „getting-started” élmény a fejlesztőknek
- Kód generálás és XML konfiguráció nélküli fejlesztés
- Indításra kész alapértelmezett konfiguráció
- Nem funkcionális szolgáltatások biztosítása melyek közösek a projekt osztályai számára (beépített szerverek, biztonság)

[https://docs.spring.io/spring-boot/docs/2.7.1/reference/pdf/spring-boot-reference.pdf]

Rendszer követelmények, a program elkészítésekor Spring boot 2.7.1 verzióra vonatkoznak:

- Minimum java 8-as kiadás
- Spring keretrendszer 5.3.21+
- Maven 3.5+ vagy Gradle 6.8.x, 6.9.x, 7.x
- Servlet konténer:
    - Tomcat9.0 verzió 4.0, vagy
    - Jetty9.4 verzió 3.1, vagy
    - Jetty10.0 verzió 4.0, vagy
    - Undertow 2.0 verzió 4.0

[https://docs.spring.io/spring-boot/docs/2.7.1/reference/pdf/spring-boot-reference.pdf]

1.3.1 Spring Application

A SpringApplication osztállyal megjelölhető az alkalmazás main metódusa. Már ezzel rengeteg szolgáltatás válik elérhetővé az alkalmazás számára. Az beszédes üzembe helyezési hibaüzenetek valamint az üzembe helyezés követési segít megérteni az a keretrendszer működését, eseményfigyelőket biztosít, ami az alkalmazás elérhetőségi funkciókkal együtt használva figyelemmel kísérhető az alkalmazás állapota. Hozzáférhetőek az alkalmazás argumentumok, amik akár konfigurálásra is használhatóak, mint például a lista inicializáció bekapcsolása vagy a banner lecserélése, ezek egybefűzhetőek a Spring Fluent Buildere segítségével. Támogatja a webes környezetet, valamint az alkalmazás leállításához egy graceful shutdown implementációt. [https://docs.spring.io/spring-boot/docs/2.7.1/reference/pdf/spring-boot-reference.pdf]

A keretrendszer további alapszolgáltatásai röviden:

- Külső konfiguráció
- Profilok
- Internacionalizáció (I18n)
- JSON
- Feladat végrehajtás és időzítés
- Tesztelés
- Egyéni automatikus konfiguráció

[https://docs.spring.io/spring-boot/docs/2.7.1/reference/pdf/spring-boot-reference.pdf]

1.4 Egyéb csomagok

Az alkalmazás fejlesztéséhez a Spring boot biztosít egy keretrendszert, opcionális csomagokat valamint, be lehet importálni harmadik fél által készített csomagokat. A következőkben röviden ismertetem az opcionális csomagok, illetve a harmadik féltől származó csomagok szerepét.

1.4.1. Spring Data

A Spring keretrendszer széleskörűen támogatást nyújt az SQL adatbázisokkal való munkához. Ilyen például a Hibernate ORM, ami az SQL adatbázis és az Objektum orientált Java között teremti meg a kapcsolatot. A Spring Data ezt tovább egyszerűsíti, méghozzá úgy, hogy a fejlesztőnek egy interfészt kell létrehozni, majd a függvény nevekből visszafejti és megvalósítja az adatbázissal kapcsolatot tartó osztályt. [https://www.baeldung.com/the-persistence-layer-with-spring-data-jpa]

1.4.2. Spring Security

A Spring Security egy erőteljes, egyénre szabható autentikációs és hozzáférés szabályzó keretrendszer, ami egyaránt szolgáltatja az autentikációt és az autorizációt. Ezen védelmet nyúlt a különböző támadások ellen, mint például a munkamenet fixáció vagy a cross site request forgery. [https://spring.io/projects/spring-security]

1.4.3. Spring Web

A Spring boot jól illeszkedik a web alkalmazás fejlesztéshez. A legtöbb alkalmazás (a WorkoutPlanner is) a spring-boot-starter-web modult használja, de építhető reaktív alkalmazás is a spring-boot-starter-webflux modullal. A reaktív programozás egy aszinkron, esemény vezérelt, kevés szálat igénylő programozási irány. A skálázhatóságra inkább a JVM-en belül törekszik. Jelen alkalmazás a Spring Web MVC keretrendszerét használja, abból is a RestController beaneket amik kezelek a beérkező http kéréseket. [https://docs.spring.io/spring-boot/docs/2.7.1/reference/pdf/spring-boot-reference.pdf]

1.4.4. Lombok

A Lombok egy java könyvtár a gyakran ismétlődő kódok csökkentésére kiiktatására annotációk használatával, továbbá fokozza a kód olvashatóságát. Az integrált fejlesztői környezetek letudják generálni ezeket a kódokat, azonban helyet foglalnak és az olvashatóság is romlik. Ezekkel az annotációkkal generálhatóak konstruktorok, állító és hozzáférő metódusok, felülírhatóak az Object osztályból örökölt toString, illetve equals függvények. [https://projectlombok.org/]

1.4.5. H2

A H2 egy beépített, nyílt forráskódú, memóriában működő adatbázis, azaz az adatokat a memóriában tárolja, semmit sem ment háttértárba. Támogatja a tranzakciókat és az adatbázis kezeléséhez biztosít egy böngészőből elérhető felületet. Fejlesztéshez előnyös, hogy nem kell külön adatbázis kezelő telepíteni, ha meg kell vizsgálni az adattáblákat. Majd amikor az alkalmazás produkciós környezetbe fog kerülni, a használt adatbázis könnyedén lecserélhető az adatbázishoz használt konnektor csomag lecserélésével, illetve az adatbázis konfiguráció módosításával. [https://www.h2database.com/html/features.html]

2. Anyag és módszer

Az edzés tervező alkalmazás fejlesztéséhez számos előismeret szükséges egyrészt program fejlesztői ismeretek, egy probléma megoldásához az adott programnyelven, ebben az esetben Java. Másrészt edzői ismeretek az erőnlét fejlesztésének alapjairól, módszertanáról. Az utóbbit, hobbi sportolóként tudom biztosítani a saját fejlődésem érdekében elsajátított ismeretekkel. Úgy vélem, hogy ez az ötlet bemutatására alkalmas program fejlesztéséhez elegendő. További szükséges ismeret, hogy egy életszerű, összetett problémát, hogyan kell leképezni úgy, hogy az objektum orientáltan lefejleszthető legyen, valamint az adatok és adatkapcsolatok megfelelően reprezentálva legyenek egy adatbázisban. Végül a fejlesztést fordítani, üzembe helyezni és üzemeltetni kell.

2.1 Anyag

Az alkalmazás a WorkoutPlanner nevet kapta és a célja az edzésterv összeállítás modul megvalósítása, ami egy nagyobb alkalmazásba épülhet. A WorkoutPlanner feladata egy edzés terv összeállítása, a rögzített gyakorlatokból, ezeket az edzésterveket nyilván tartja, kezeli a frissítésüket, törlésüket, értékelésüket.

2.1.1 Követelmények

Az alkalmazással szemben támasztott követelmények, hogy egy bárki is összetudjon állítani egy edzéstervet, ezzel segíteni az edzéseinek megtervezését, követését.

Edzéstervezés specifikus követelmények:

- Legyen képes gyakorlatok rögzítésére
- Legyen képes több gyakorlat egybefogására, mint például a szuperszettek, triszettek
- Legyen képes az összeállított, gyakorlatokból edzést,
- Az összeállított edzésekből edzéstervet rögzíteni
- Biztosítson böngészési lehetőséget az elkészült tervek között
- Az edzéstervekre lehessen feliratkozni
- Az edzésterveket lehessen értékelni

Felhasználó kezelés specifikus követelmények:

- Az oldal használata regisztráció után legyen lehetséges
- A felhasználó tudjon magához rendelni edzéseket, edzésterveket
- Egy adatot csak a (gyakorlatot, edzést, edzéstervet stb.) csak a szerzője módosíthasson

A jelen verzió elkészítésének célja, nem egy piacképes alkalmazás elkészülése, hanem egy olyan program váz létrejötte, ami képet ad az alkalmazás edzéstervezés specifikus követelményeinek megvalósításáról. Ezen a ponton még alacsony energia befektetéssel lehet módosítani ezen a működésén, megváltoztatni vagy hozzáadni új követelmény elemeket.

2.2 Az alkalmazás felépítése

Egyrészt az alkalmazás felépítése kötött, mivel Spring boot projektről van szó, így egy alap struktúrából dolgozik a fejlesztő. Másrészt a kapott felépítésen belül bármilyen csomag szerkezetet fel lehet építeni. A Workout Planner alkalmazás felépítése során a cél az volt, hogy a változtatásokat, illetve az programkód bővítését minél egyszerűebben meg lehessen valósítani, anélkül, hogy a változás vagy bővítés az egész forráskódon rajta hagyná a nyomát. Ezért a fejlesztés során az azonos szerepet betöltő osztályokat egy csomagba szerveztem ez a csomagba szervezés rétegeknek is megfeleltethetők. [https://medium.com/the-resonant-web/spring-boot-2-0-project-structure-and-best-practices-part-2-7137bdcba7d3]

2.2.1 Model

A model csomagba az entitás osztályok szerepelnek, ezeknek az osztályok feladata az adattáblák reprezentálása a programban, továbbá ezen osztályok feladata, hogy egy azonos típusú példány alapján frissítsék az adataikat. Ennek előnye, hogy a szerviz csomagjai nem híznak egyszerű adatváltoztatásoktól.

2.2.2. Repository

A repository csomagban az entitásokhoz tartozó interfészek szerepelnek. Ezek az interfészek kiterjesztik a Jpa interfészt, amiben számos előre megírt lekérdezést lehet használni. Az interfészekben van definiálva, hogy milyen entitást kezelnek, milyen típusú az entitás azonosítója, valamint ha szükséges lehetséges megfogalmazni a Jpa-nak új lekérdezéseket.

2.2.3. Service

A service csomag felel az üzleti logikáért, a CRUD műveletek végrehajtásáért és minden olyan szolgáltatás használatáért, amit a program használ. Két alcsomagra válik, az entities és az other csomagokra. Az entities csomagban vannak azok az osztályok, amik az entitások kezeléséért felelnek. Itt lekérhetőek, adható hozzá új, frissíthetők vagy törölhetők entitások. Ezek a műveletek a repository interfészekben definiált lekérdések meghívásával történnek, amit a szerviz osztályban történő injektálással lett megvalósítva. Minden entitásnak saját szerviz osztálya van és ezekben az osztályokban csak az entitáshoz tartozó repository interfészeket vannak injektálva. Az összetett entitások miatt egy művelet, más entitásokra is hatással lehet. Ezeknek a hatásoknak a rögzítése a kapcsolt entitás szerviz osztályának injektálásával és a megfelelő metódus meghívásával történik. Minden módosító művelet naplózásra kerül, amit a szerviz osztályban egy LoggerFactory objektum példányosít.

2.2.4. Controller

A controller csomagban szerepelnek a kontroller osztályok, ezek kezelik a http kéréseket Itt is azt a struktúrát, követtem, hogy minden entitásnak saját kontroller osztálya van, amibe az entitást kezelőt szerviz osztály van injektálva. Ezen osztályok metódusai paraméterként vagy azonosítót fogadnak, vagy DTO-kat (Data trasnsfer object) fogadnak, valamint az jogosultság kezeléséhez egy Principal objektumot. Válaszként pedig egy http státuszt, illetve metódustól függően egy DTO-t vagy DTO-okat tartalmazó listát JSON formátumban. JSON (JavaScript Object Notation) egy könnyűsúlyú adatcserélési formátum. A felépítése két elemből áll. Kulcs-érték párok az egyes mezők neveihez való érték rendeléshez, valamint egy listából, amiben az objektumok vannak helyezve.

2.2.5. DTO

A DTO (Data transfer object) feladata, hogy az minél több lényeges információt lehessen visszaküldeni egy kérésre, valamint egy művelet csak a lényeges információk elküldését igényelje. Így az entitásokat mielőtt az alkalmazás kiküldené, DTO-ká alakítja, valamint a kérésekben kapott DTO-kat entitásokká alakítja. Ezeknek az átalakításoknak az elvégzése ezeknek a közvetítő osztályoknak a felelőssége azért, hogy a szerviz osztályok ne legyenek hizlalva a konvertálás kódjával, valamint az entitásokban történt változások minél kisebb hatással legyenek a szerviz osztályokra. Ennek megfelelően a DTO csomag két alcsomagra válik, ami a request és a response. A kérésben lévő adatok a request DTO által értelmeződnek és a válaszokat az alkalmazás response DTO objektummá alakítva küldi el válaszban. [FOWLER, 2002]

2.2.6. Exception

Az exception csomag tartalmazza a hibakezeléssel kapcsolatok osztályokat. Itt vannak létrehozva az egyedi exception osztályok, amik kiterjesztik a RuntimeException osztályt, így amikor ezeket a kivételeket használom, nem muszáj helyben kezelni, hanem a RestExceptionHandler osztály a kivétel eseményt kezeli a számára egy metódusban megfogalmazott módon. Ezek a kivételek egy RestApiError típusú osztályba lesznek konvertálva és JSON formátumba visszaküldve a kérés feladójának.

2.2.7. Configuration

A configuration csomagban tartalmazza azokat az osztályokat, amikkel az alkalmazás alapértelmezett beállításai változtathatók. Jelenleg a WebsecurityConfig osztály található ebben a csomagban, ez az osztály állítja be a használt autentikációs módszert, a bárki által elérhető, illetve a regisztrációhoz kötött végpontokat, a használt titkosítási eljárást és adja hozzá az alapértelmezett felhasználókat.

2.3 Módszer

Egy alkalmazás fejlesztés módszereinek kiválasztásánál rengeteg döntést kell meghozni. Kezdve a nyelv választásával (jelen esetben Java), a keretrendszerrel folytatva, ugyanis amit már mások lefejlesztettek az nem érdemes újra megírni. Ha ezek megvannak, jöhet a build tool, ami összegyűjti a program függőségeit és a lefordított programkódhoz csatolva egy futtatható fájlba csomagolja ezeket. A következő fontos kérdések, hogy milyen adatbázissal fogunk dolgozni, szükséges-e verzió kezelő, majd a fejlesztés megkönnyítése érdekében célszerű egy integrált fejlesztő környezet használata. Az eddigiek szoftveres eszközökkel kapcsolatos kérdések vannak programozás technikai eszközök, amikkel megkönnyíthető a fejlesztés, kiváltképp akkor, ha egy nagy projektről van szó.

2.3.1 Szoftveres eszközök

Mivel a nyelvet és a keretrendszer már bemutattam, így azokról itt nem fogok írni, egyből a fejlesztés egyik kulcs szoftveréhez ugrok. Az alkalmazás fejlesztéséhez az IntelliJ integrált fejlesztői környezet community kiadását használom, a tapasztalataim szerint nagyszerűen támogatja a Java nyelven való fejlesztést, valamint a git verzió követő használatát, mivel a színkódokkal jelöli az egyes fájlok státuszát. További rendelkezik integrált build tool-okkal amit így nem szükséges külön telepíteni, de véleményem szerint érdemes külön is telepíteni, mert ha csak az kell használni sokkal gyorsabb, mint az IntelliJ betöltésére várni. Az alkalmazásból a maven segítségével készítek futtatható állományt, ami egy szoftver projekt menedzsment eszköz. Egy XML fájlban le van képezve a projekt modellje, ami többek között leírja a program nevét, Java verzióját, függőségeit, majd ezen információk alapján a maven elkészíti a futtatható állományt. Az alkalmazás futtatásához szükséges az a JRE (Java Runtime Envirement) méghozzá az a verzió, amiben az alkalmazás is készült. Ez egyedileg nem probléma, de tegyük fel, hogy adott egy szerver, amin fut 20 alkalmazás mindegyik más Java verzióval az 20 különböző JRE verzióz jelent, amit menedzselni kell, továbbá ez egy függőséget jelent. Ennek a megoldására egy konténerizáló programot használok, amivel egy környezetet biztosítok a program számára a futásához szükséges JRE verzióval. Ez a program a Docker, egy leírás alapján elkészít egy docker image-t és ezekből egy futó példány a konténer, ami elszigetelten fut. Ezekkel a konténerekkel a Dockeren keresztül lehet kommunikálni, belépni, portot nyitni a külvilág felé. [https://www.jetbrains.com/idea/features/, https://maven.apache.org/, https://docs.docker.com/desktop/]

2.3.2 Programozás technikai eszközök

A programozás technikai eszközök alatt azt értem, hogyan formázom, építem fel a forráskódot, milyen elnevezéseket használok, valamint, hogy használom ki az objektum orientáltság előnyeit. A formázásban sokat segít az integrált fejlesztői környezet. Automatikusan formáz, és ha valamilyen oknál fogva átrendezzük a kódot, manuálisan meg lehet hívni a formázó funkciót. A programkód felépítést a funkciók szerint szeparálással valósítottam meg, ezt a későbbiekben részletesen bemutatom. Az elnevezések használatában arra törekedtem, hogy a választott név a lehető legrövidebb ugyanakkor beszédes legyen. Ennek több szempontból is fontos célja van. Nem szükséges átböngészni egy függvényt vagy egy osztályt azért, hogy megtudjam mit csinál, hanem már a neve elárulja. Egy mező esetén még fontosabb, hogy milyen neveket használok, mert egy semmit mondó mezőnévből jóval nehezebb és jóval több utána járást kíván kikövetkeztetni, hogy mire való. További előnye a beszédes név választásnak, hogy a programkód olvasmányosabb lesz így könnyebb megérteni a működését. A következő technika szorosan kapcsolódik az elnevezési technikához. Egy függvény, osztály lehetőleg csak egy dolgot csináljon. Ennek is több előnye van, kezdve az egyszerűséggel. Minél kevesebb dolga az adott osztálynak, függvények annál egyszerűbb lesz annak és átláthatóbb lesz annak a kódja. Másik előnye, hogy ezzel a technikával több olyan kód készíthető, amit újra lehet használni. [MARTIN, 2008]

3. Implementáció

A célom az volt, hogy az alkalmazás megvalósítása könnyen érhető legyen, valamint a változásokat a lehető legegyszerűbben lehessem beépíteni, ezért a program különböző részeit különválasztottam mind funkcionálisan, mind az egyes entitásokra vonatkozóan. Majd ezeket a részeket összekapcsoltam olyan módon, hogy az egyes elemeket akadálytalanul ki lehessen cserélni egy másikra.

3.1. Entitások

Az entitás osztályok segítségével kommunikál a program az adatbázissal, ezek az osztályok vannak reprezentálva, mint adattáblák, valamint az osztályok mező, mint az adattáblák oszlopai.

3.1.1. Edzés specifikus entitások

Az entitások implementációjánál az első lépés a közös tulajdonságok összegyűjtése volt, azzal a céllal, hogy ezeket kiszervezzem egy közös ősosztályba, a BaseEntity osztályba. Az alábbi részlettel a BaseEntity osztály mutatom be. Mind az osztály, mind annak egyes mezői különböző annotációkkal vannak ellátva, ezeknek a feladata az adatbázis kezelőnek információt, utasítást adni vagy elkerülni az ismétlődő kódok írását.

```
@MappedSuperclass
@Getter
@Setter
public class BaseEntity {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    protected Long id;
    @CreationTimestamp
    private Date createdAt;
    @UpdateTimestamp
    private Date updatedAt;
    private String createdBy;
}
```

5. Ábra: BaseEntity osztály mezői és annotációi

Forrás: Saját, WorkotPlanner forráskód

A MappedSuperclass jelzi az adatbázis kezelőnek, hogy az ezzel megjelölt osztályok nem entitások, az entitások fogják kiterjeszteni. Az Id annotációval megjelölt mező az entitás azonosítója ez lehet számérték, szöveg vagy egyéb érték. A GeneratedValue jelzi, hogy a megjelölt mező egy generált érték, valamint az átadott paraméter egy azonosító, tehát ezt az értéket az adatbázis fogja generálni. A CreationTimestamp és az UpdateTimestamp annotációval a megjelölt mező is egy automatikusan generált dátum érték, ami a generálást kiváltó esemény idejének az értékét fogja kapni itt az első mentés, illetve frissítés. A Getter/Setter annotáció a gyakran használt kódokat fogja automatikusan generálni a konvencióknak megfelelően. [https://docs.oracle.com/javaee/6/api/, https://docs.jboss.org/hibernate/orm/4.3/javadocs/org/hibernate/annotations/, https://projectlombok.org/features/GetterSetter]

A következő három entitás szorosan összekapcsolódik, ugyanis ezzel a három entitással oldottam meg, hogy különböző gyakorlatok egybefoghatók legyek, egy szuperszettben, triszettben stb. Ezek az Exercise, ExerciseWrapper és a Task.

Az Exercise tartalmazza a gyakorlat nevét (name) és típusát (repeatable), ami jelen megvalósításban egy boolean típusú változó, mely igaz érték esetén egy ismétlésekből álló gyakorlatot jelöl, hamis esetén egy tartást, amit meg kell tartani. Továbbá rendelkezik egy exerciseWrappers listával, ami az adatbázisban nem ennél az entitásnál lesz reprezentálva. Ennek a mezőnek a szerepe, hogy a programban könnyedén hozzáférhetőek legyenek azok a ExerciseWrapper objektumok, amik az Exercise entitás példány csomagolják be.

```
@Getter
@Setter
@Entity
@NoArgsConstructor
public class Exercise extends BaseEntity{
    private String name;
    private Boolean repeatable;
    @OneToMany(mappedBy = "exercise")
    private List<ExerciseWrapper> exerciseWrappers;

}
```

6. Ábra: Exercise osztály mezői és annotációi

Forrás: Saját, WorkotPlanner forráskód

A NoArgsConstructor automatikusan generáltatok egy argumentumok nélküli konstruktor. Ha nincs argumentummal rendelkező konstruktor a Java fordítója ezt automatikusan megcsinálná, de az osztálynak van definiálva egy számértéket váró konstruktora csupán helytakarékossági okokból ez nem ábrázoltam. Majd az entitásnak létrehozok egy OneToMany azaz egy a többhöz kapcsolatot, ami azt jelzi, hogy az entitás egy példányához egy kapcsolt entitásnak több példánya is tartozhat, tehát jelen esetben egy Exercise példányt több ExerciseWrapper példány is használhat. Az átadott paraméter a kapcsolt entitás kapcsolódó mezőjét jelöli. A jelenlegi entitás adattáblájában ez a kapcsolat nem lesz jelölve.

Az Exercise példányok önmagukban nem használhatók. Ezek a felhasználáshoz egy ExerciseWrapper entitásba kerülnek, ami tartalmazza az elvégzendő gyakorlatot (exercise), a sorozatok számát (sets), a használt súly (usedWeight), valamint attól függően, hogy a gyakorlat kitartás vagy egy ismétlésekből álló gyakorlat az időtartamot másodpercben (durationInSecond), illetve az ismétlésszámot (reps). Végül azt a feladatot (Task) amibe tartozik.

```
@Entity
@Getter
@Setter
@NoArgsConstructor
public class ExerciseWrapper extends BaseEntity{
    @ManyToOne
    private Exercise exercise;
    private Integer reps;
    private Integer sets;
    private Integer durationInSecond;
    private Integer usedWeight;
    @ManyToOne
    private Task task;

}
```

7. Ábra: ExerciseWrapper osztály mezői és annotációi

Forrás: Saját, WorkotPlanner forráskód

Létrehoztam egy ManyToOne azaz több az egyhez kapcsolatot az Exercise entitással ez azt jelenti, hogy az entitás osztály több példánya használja a megjelölt mező osztályának egy példányát. Ez a kapcsolat az adattáblában a jelenlegi entitás oldaláról van jelölve.

Végül a Task osztály reprezentálja a tényleges feladatot. Tartalmaz egy listát, amely ExerciseWrapper-eket tartalmaz, komment (comment) is hozzáadható, valamint fantázia névvel is ellátható. Ebben az entitásban van jelölve az is, hogy melyik Workout entitás példányhoz tartozik.

```
@Entity
@Getter
@Setter
@NoArgsConstructor
public class Task extends BaseEntity {
    private String name;
    @OneToMany(mappedBy ="task", fetch = FetchType.EAGER)
    private List<ExerciseWrapper> exerciseWrappers;
    @ManyToOne
    private Workout workout;
    private String comment;

}
```

8. Ábra: Task osztály mezői és annotációi

Forrás: Saját, WorkotPlanner forráskód

A OneToMany viselkedését módosítom az átadott FetchType paraméter, még pedig úgy, hogy a listát mindenképp lekérdezi az adatbázisból, ha egy Task lekérdezésre került. Az edzések a feladatokból állnak, ezeket az edzéseket a Workout osztály reprezentálja, aminek szintén adható név (name), majd tartalmaz egy listát az elvégezendő feladatokról (tasks). Az edzéseknek továbbá ki lehet tűzni cél időpontot az elvégzésre (goalDate) és rögzíteni a tényleges elvégzés időpontját(finishDate), majd egy leírás csatolni hozzá (description), illetve külön rögzíteni a edzés hatását (afterEffect). Végül van kettő több a többhöz kapcsolata a felhasználókkal, ezt egy Account objektumokat tartalmazó lista reprezentálja (accounts) és az edzéstervekkel, ezt egy WorkoutPlan objektumokat tartalmazó lista reprezentálja (workoutPlans).

Létrehozok egy ManyToMany vagyis több a többhöz kapcsolatot a Workout entitással, ami azt, hogy egy Workout entitáshoz példányhoz az Account entitás több példánya tartozhat és fordítva Account entitás egy példányához Workout entitás több példánya tartozhat. Ez az adatbázisban egy kapcsolótáblában van reprezentálva, ami a kapcsolódást az entitás azonosítók párosításával jelzi.

```
@Entity
@Data
@NoArgsConstructor
public class Workout extends BaseEntity{
    private String name;
    @OneToMany(mappedBy ="workout")
    private List<Task> tasks;
    @ManyToMany
    private List<WorkoutPlan> workoutPlans;
    @ManyToMany
    private List<Account> accounts;
    private String description;
    private String afterEffect;
}
```

9. Ábra: Workout osztály mezői és annotációi

Forrás: Saját, WorkotPlanner forráskód

Végül az utolsó edzés specifikus entitás az edzésterv (workoutPlan). Névvel, leírással, kommenttel ellátható (name, description, comment), valamint van értékelése és az újraszámításhoz tárolva van az értékelések száma. Ez az osztály is két több a többhöz kapcsolattal rendelkezik egyrészt az edzésekkel való kapcsolata innen is jelölve van egy Workout objektumokat tartalmazó listával, másrészt a felhasználókkal való kapcsolat is több a többhöz szintén egy listával reprezentálva, ezúttal Account objektumokat tartalmaz.

```
@Entity
@Getter
@Setter
@NoArgsConstructor
public class WorkoutPlan extends BaseEntity {
    private String name;
    @ManyToMany(mappedBy ="workoutPlans")
    private List<Workout> workouts;
    @ManyToMany
    private List<Account> accounts;
    private Double rating;
    private Integer numberOfRating;
    private String description;
    private String comment;
}
```

10. Ábra: WorkoutPlan osztály mezői és annotációi

Forrás: Saját, WorkotPlanner forráskód

3.1.2. Felhasználó specifikus entitások

A felhasználó specifikus entitások alapjai a Spring Security dokumentációjában javasolt entitások. Ezek egy auth csomagban helyezkednek és a felhasználókat (Account) és a jogokat (Authorities) reprezentálják. Ezek az osztályok nem terjesztik ki a BaseEntity osztályt. [https://docs.spring.io/spring-security/site/docs/4.2.x/reference/html/appendix-schema.html]

Az Account osztály tartalmazza a felhasználónevet (username), ami egyben az azonosító is, a jelszót (password) valamint, hogy a felhasználó engedélyezett-e (enabled). Végül a több a többhöz kapcsolatok ezen az oldalt is jelölve vannak két listával amik Workout, illetve WorkoutPlan objektumokat tartalmaznak.

```
@Entity
@Data
@NoArgsConstructor
@Table(name = "users")
public class Account {
    @Id
    @Column(name = "USERNAME")
    private String username;
    private String password;
    private char enabled;
    @ManyToMany(mappedBy = "accounts")
    private List<WorkoutPlan> workoutPlans;
    @ManyToMany(mappedBy = "accounts")
    private List<Workout> workouts;

}
```

11. Ábra: WorkoutPlan osztály mezői és annotációi

Forrás: Saját, WorkotPlanner forráskód

A Table segítségével jelzem az adatbázis kezelő számára, hogy egy osztály az objektum orientált környezetben én Accountnak hívom, de az adatbázisban olyan users néven találja meg vagy hozza létre.

Az Authorities osztály reprezentálja a jogokat, tartalmaz egy azonosítót, a felhasználót, akihez tartozik (username), valamint a jogosultság megnevezését (authority).

```
@Entity
@Getter
@Setter
@Table(name = "authorities",
uniqueConstraints = {@UniqueConstraint(columnNames = {"username", "authority"})})
public class Authorities {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    @ManyToOne
    @JoinColumn(name = "username", columnDefinition = "VARCHAR")
    private Account username;
    private String authority;

}
```

12. Ábra: Authorities osztály mezői és annotációi

Forrás: Saját, WorkotPlanner forráskód

A Table segítségével explicit megmondom az adatbázisnak, hogyan nevezze el az adattáblát, továbbá készítek hozzá egy UniqueConstraint, ami egy egyedi megszorítás lesz. Az adattábla username és authority oszlopainak értékpárosának egyedinek kell lenni, ugyanis szükségtelen egy felhasználóhoz többször ugyan azt a jogosultságot hozzárendelni, valamint potenciális hibaforrásnak tartom. Amennyiben mégis duplikálni próbálnék egy rekordot, az adatbázis hibaüzenetet küldene. Majd a több az egyhez kapcsolatot árnyalom tovább az adatbázis kezelőnek. Megmondom, hogy a jogosultságok, melyik oszloppal kapcsolódik a felhasználókhoz ennek a megfogalmazása látható a JoinColumnban.

3.2. Hibernate és JPA

A Hibernate egy ORM (Object/Relational mapping) megoldás Java környezetre. A feladata, hogy kapcsolatot teremtsen az objektum orientált java és a relációs adatbázisok (SQL) között. Gondoskodik arról, hogy a Java osztályok át legyenek alakítva adatbázis táblákká, az adattípusok SQL adattípusokká. Továbbá biztosítja a lekérdezéseket és visszakereséseket. A Hibernate célja, hogy felgyorsítsa a fejlesztési időt azzal, hogy nem kell a fejlesztőnek megírnia az adatbázis lekérdezéseket. [https://hibernate.org/orm/documentation/6.1/]

A JPA (Java Persistence API) egy interfész, amiben specifikálni lehet, hogy milyen adatbázis műveletet kell végrehajtani. Itt a végrehajtás módja nem fontos, csupán annyit

definiál, hogy mit kell végrehajtani az adatbázisban de, nem valósítja meg. Ami a műveletet meg fogja valósítani az a Hibernate. A JPA számos műveletet előre specifikált, mint például a mentés, törlés, keresés azonosító alapján. Viszont amikor olyan lekérdezés kell, amit nincs előre specifikálva, a fejlesztőnek kell ezt megtennie. Előtte az entitásokat elő kell készíteni, mégpedig rendelkezniük kell az @Entity annotációval, aminek a követelménye, hogy az entitásnak legyen egy azonosítója. Ezt az azonosítót el kell látni egy @Id annotációval, ami jelzi a JPA-nak, hogy az a mező az azonosító, ezt az azonosítót megadhatja a fejlesztő is, ebben az esetben nincs tovább annotáció, amit az azonosítóra kell helyezni. A legtöbb esetben az azonosító egy generált érték, amit az adatbázis tölt fel, ilyenkor egy @GeneratedValue annotációval kell megjelölni és ennek paraméternek átadni a generálás módját.

Az entitások között kapcsolatokat szintén jelölni kell a JPA számára, amit a következő annotációkkal lehet megtenni.

- @OneToOne
- @ManyToOne
- @OneToMany
- @ManyToMany

Ezek a kapcsolatok tovább paraméterezhetőek a mappedBy, fetch és cascade, targetEntity, oprhanRemoval tulajdonságokkal, ezek tovább specifikálják a kapcsolat viselkedését.
Az alkalmazásban a mappedBy paramétert használtam, ami elmondja, hogy a kapcsolt entitás melyik mezőjével kapcsolódik. Valamint a fetch paramétert, ami az adatbázisból kiolvasást specifikálja, ez lehet LAZY (lusta) és EAGER (mohó). A lusta csak akkor kérdezi le a kapcsolt entitást, ha az szükséges, míg a mohó mindenképp lekérdezni.

Miután specifikálva lettek az entitások, létre kell hozni nekik egy interfészt, ami kiterjeszti a JpaRepository<T,ID> interfészt, amit paraméterezni kell a hozzá tartozó entitás osztályával valamint az azonosító típusának az osztályával. Ebben az állapotban rendelkezésre állnak az alapértelmezett lekérdezések, valamint további lekérdezések specifikálhatóak azáltal, hogy az entitás interfészében metódus deklarálásával.

A metódus nevével mondható el, hogy mit csináljon a lekérdezés illetve a paraméterek a lekérdezés változói. Az alábbi lekérdezés például visszatér egy WorkoutPlan objektumokat

tartalmazó listával, azzal a kritériummal, hogy a adatrekord név értéke egyezzen meg a paraméterben kapott névvel.

```
@Repository
public interface WorkoutPlanRepo extends JpaRepository<WorkoutPlan,Long> {
    List<WorkoutPlan> findAllByName(String name);
}
```

13. Ábra: WorkoutPlanRepo interfész definiciója

Forrás: Saját, WorkotPlanner forráskód

Spring IoC szolgáltatásának jelzem, hogy az interfész egy Repository típusú bean. Egy Repository típusú bean adat tárolási, visszakeresési folyamatokat végeznek.

3.3. Service

A Szerviz osztályok végzik az alkalmazás üzleti logikáját, valamint a naplózást. Ezek az osztályok @Service annotációval vannak ellátva, jelezve a Spring IoC szolgáltatásának, hogy az osztály egy Service típusú bean.

3.3.1. Entitás service

Az entitás szerviz osztályokhoz készült egy interfész, ami összegyűjti, hogy mit kell tudnia egy entitás szerviznek. Az interfész paraméterezendő, mégpedig olyan osztályt vár paraméternek, ami kiterjeszti a BaseEntity osztályt.

```
public interface IEntityService<T extends BaseEntity> {
    List<T> getAll();
    T getById(Long Id);
    T save(T toSave);
    T update(Long id, T toUpdate, Principal principal);
    void delete(Long id, Principal principal);
}
```

14. Ábra: IEntityService interfész definíciója

Forrás: Saját, WorkotPlanner forráskód

A szervizek implementálják az IEntityService interfészt, így a benne deklarált metódusokat, amiket kötelességük megvalósítani. Ez úgy hajtják végre, hogy a IoC szolgáltatásnak jelzik, hogy a működésükhöz szükséges egy Repository, a következő formában.

```
private final ExerciseRepo exerciseRepo;
@Autowired
public ExerciseService(ExerciseRepo exerciseRepo){
   this.exerciseRepo = exerciseRepo;
}
```

15. Ábra: Függőség injektálás bemutatása

Forrás: Saját, WorkotPlanner forráskód

Így a program elindulásakor, mielőtt elkészülne az ExerciseService osztály, az IoC látja a függőséget és készít egy ExerciseRepo példányt, majd az ExerciseService konstruktorának átadja, ez a függőség injektálás (dependency injection). Majd következnek a kötelezően megvalósítandó metódusok megvalósítása.

```
@Override
public Exercise save(Exercise toSave) {
   return exerciseRepo.save(toSave);
}
```

16. Ábra: Interfész metódus felülírása

Forrás: Saját, WorkotPlanner forráskód

A fenti kódrészletben a mentéshez az injektált exerciseRepo save metódusát meghívva menti az adatbázisba. Azonban vannak olyan metódusok, amik több adatbázis műveletet tartalmaznak. Ebben, ez esetben valós hibaforrás lehet, hogy egyes adatbázis műveletek kimaradnak, például egy mentés vagy törlés. Az ilyen esetekben használható annotáció a @Transactional ami biztosítja, hogy ha a több művelet közül akár nem egyet sikerül végrehajtani akkor az eddig történt változtatások vissza lesznek állítva. A következő függvényben már egy összetettebb objektumról van szó. Egy Task objektum lementése úgy történik, hogy egy TaskRequest átalakítja magát Task entitássá. Ilyenkor az ExerciseWrapper, amik a példányhoz tartoznak, csak az azonosítójuk tartalmazzák, ezeket lekérdezi az adatbázisból, majd hozzá adja a Task példányhoz. Ezután a Task mentésre kerül. Erre azért van itt szükség, mert ezután fog a Task példány azonosítóval rendelkezni. A mentés után az ExerciseWrapper példányokon végig iterálva beállításra kerül, hogy melyik

Task példányhoz tartoznak, majd mentésre kerülnek. Ha ez megtörtént a metódus visszatér a Task példánnyal.

```
@Override
@Transactional
public Task save(Task toSave) {

toSave.setExerciseWrappers(getExerciseWrappersByDummies(toSave.getExerciseWrappe
rs(), toSave));
    Task toRet = taskRepo.saveAndFlush(toSave);
    for(ExerciseWrapper e : toSave.getExerciseWrappers()) {
        e.setTask(toSave);
        exerciseWrapperService.save(e);
    }
    return toRet;
}
```

17 Ábra: @Transactional használata

Forrás: Saját, WorkotPlanner forráskód

Az IEntityService interfészben deklarált metódusokon kívül a szerviz osztályban kerülnek megfogalmazásra az olyan segédfüggvények, mint például a fenti kódrészletben látható getExerciseWrappersByDummies, ami visszatér tényleges adatokkal feltöltött ExerciseWrapper objektumok listájával. Továbbá azok a függvények, amik entitás specifikusak, mint a felhasználó feliratkozása edzéstervre.

3.3.2. Account service

Az AccountService osztály is adatbázisban tárolja a fejhasználók adatait. Az előbbi szervizekkel ellentétben nem tudja implementálni az IEntityService interfészt, mert az Account osztály azonosítója szöveg típusú. Ezért ennek a szerviz osztálynak saját interfésze van, az IAccountService. A megkövetelt metódusok funkcionálisan ugyan azok, csak az azonosító típusával térnek el. Továbbá az AccountService az egyetlen szerviz osztály, amiben kettő Repository is injektálva van, az egyik felhasználók kezeléséhez, a másik a jogosultságok kezeléséhez.

3.3.3. Jelszó validátor

A jelszó validálást a PasswordValidator szerviz osztály végzi, ami megvalósítja a ConstraintValidator<ValidPassword,String> interfészt. Az interfésznek egy metódusa van, az isValid, ami fogad egy String típust illetve egy ConstraintValidatorContext típusú objektumot. A String reprezentálja a validálandó jelszót, és ha megfelel a kritériumoknak, akkor igaz értékkel tér vissza.

A következő lépésben létrehoztam egy egyedi annotációt, amit mezőkre és paraméterekre lehet használni. Ha a mező, illetve paraméter nem felel meg a kritériumoknak akkor a hibaüzenetet átadja egy Errors objektumnak amit ConstraintValidatorContext-en keresztül.

```
@Target({ElementType.FIELD, ElementType.PARAMETER})
@Retention(RetentionPolicy.RUNTIME)
@Constraint(validatedBy = {PasswordValidator.class})
public @interface ValidPassword {
    String message() default "Password must contain at least an lower case and an upper case character, numeric character, special symbol(@,#,$,%,&) and length should be between 8 and 20!";

}
```

18. Ábra Egyedi annotáció

Forrás: Saját, WorkotPlanner forráskód

Létrehozok egy egyedi annotáció, amit a mezőkön és paramétereken lehet használni (Target). A ValidPassword logikáját a PasswordValidator osztály fogja szolgáltatni, ezt a Constraint és paramétereivel fejezem ki. Miután az annotáció logikája lefut, annak eredménye már futási időben el lesz vetve (Rentention). [https://dzone.com/articles/spring-boot-custom-password-validator-using-passay]

3.4. Rest controller

A @RestController annotációval megjelölt osztályok kezelik a http kéréseket. Értelmezik a kérés fajtáját (GET, POST, PUT, DELETE például) és a kérésben bejövő adatokat értelmezi és elirányít a megfelelő szerviz osztálybeli függvényhez. [https://www.digitalocean.com/community/tutorials/spring-restcontroller]

```
@DeleteMapping("/{id}")
@ResponseStatus(HttpStatus.NO_CONTENT)
public void deleteWorkout(@PathVariable Long id, Principal principal){
    workoutService.delete(id, principal);
}
```

19. Ábra: Kontroller metódus egy végpontra

Forrás: Saját, WorkotPlanner forráskód

A fenti példában egy törlési kérést fogad a kontroller és ehhez vár egy azonosítót, valamint egy Principal típusú objektumot, ami a felhasználónevet tárolja. Majd meghívja az illetékes szerviz osztály törlés metódusát, ami törli az entitást.

3.5. DTO

A DTO rövidítés feloldása Data Transfer Object, egy programozási minta, aminek lényege, hogy az adatok nem az eltárolt formában küldi vissza a program. A célja, hogy a drága metódus hívások számát csökkentse úgy, hogy egy hívásra több adatot szállít. További előnye, hogy a validáció kiszervezhető ezekre az osztályokra, ahelyett, hogy ezt a szerviz vagy az entitás osztályok végeznék. [FOWLER, 2002]

Az alkalmazásban az entitások felvétele és frissítése ezeken az objektumokat keresztül történik. A kontroller a mentési és frissítési kérésekben érkező DTO-t fogadja, majd a DTO a benne definiált módon entitássá alakítja magát és így kerül a szerviz osztályba a frissítés vagy mentés elvégzésére. Ezután a válaszban, ha kell szerepelnie visszaadott adatnak az entitásról, akkor az olyan DTO lesz, aminek a konstruktorában egy entitás argumentum van, majd ez alapján beállítja az értékeit. Tehát az alkalmazásban két típusú DTO van definiálva, az egyik a típusa az kérésekhez használt, a másik a válaszokban használt DTO.

```java
@Getter
@Setter
public class ExerciseWrapperRequest {
    @NotNull
    @Min(value = 1, message = "Invalid exercise value!")
    private Long exercise;
    @NotNull
    @Min(value = 1, message = "Min number of sets is 1!")
    private Integer sets;
    @Min(value = 1, message = "Min value of reps is 1!")
    private Integer reps;
    @Min(value = 1, message = "Min value of duration in second is 1!")
    private Integer durationInSecond;
    @Min(value = 0, message = "Used weight cannot be less than 0!")
    private Integer usedWeight;
}
```

20. Ábra: DTO osztály és validáció

Forrás: Saját, WorkotPlanner forráskód

Az osztályban szereplő mezők validálásánál az volt a cél, hogy a kötelező mezők ne maradhassanak üresen vagy ne legyenek null értékűek erről a NotNull gondoskodik, a többi nem kötelező mezőnél az volt a fontos, hogy mi a minimum érték amit felvehet, ha kap értéket, továbbá a használat megkönnyítésé érdekében egyedi hibaüzenetet hozzáadtam ez látható a Min annotációknál.

3.6. Konfiguráció

Az alkalmazás konfigurációja egyrészt az application.properties fájl alapján történik, másrészt a configuration csomagban a WebSecurityConfig osztály által.

3.6.1. Application.properties

Az application.properties fájlban rengeteg konfigurációs lehetőség van többek között

- Email konfiguráció
- Json konfiguráció
- Adat konfiguráció
- Web/Teszt/Szerver stb. konfiguráció

[https://docs.spring.io/spring-boot/docs/current/reference/html/application-properties.html]

A WorkoutPlanner alkalmazás az adatbázissal kapcsolatos konfigurációkat valamint a naplózással kapcsolatos konfigurációkat tölti be ebből a fájlból.

3.6.2. Konfigurációs osztály

A Spring biztosít lehetőséget az osztály alapú konfigurációra, ezeket a konfigurációs osztályok és @Configuration annotációval vannak ellátva. Ezekkel az osztályokkal, olyan beállításokat is elvégezhetünk, amiket pusztán az application.properties által nem tudnánk megvalósítani, illetve alternatív megvalósítása az XML alapú konfiguráció lenne. Az alkalmazásban egy ilyen osztály van implementálva, amibe injektálva van az egy DataSource objektum ezzel biztosítva kapcsolatot az adatbázissal. Továbbá a rész beállítások bean-ek létrehozásával történik. Ezek a SecurityFilterChain, a UserdetailsManager és a PasswordEncoder. [https://spring.io/blog/2022/02/21/spring-security-without-the-websecurityconfigureradapter]

A SecurityFilterChain felelős azért, hogy a kérések átessenek minden szükséges szűrésen, továbbá meghatározza az autentikáció típusát, a nyitott és autentikáció köteles végpontokat és biztonsági beállításokat. A UserDetailsManager beállítja az adatforrást a felhasználókhoz és hozzáad alapértelmezett felhasználókat, valamint a PasswordEncoder a jelszó kódolás típusának meghatározásáért.

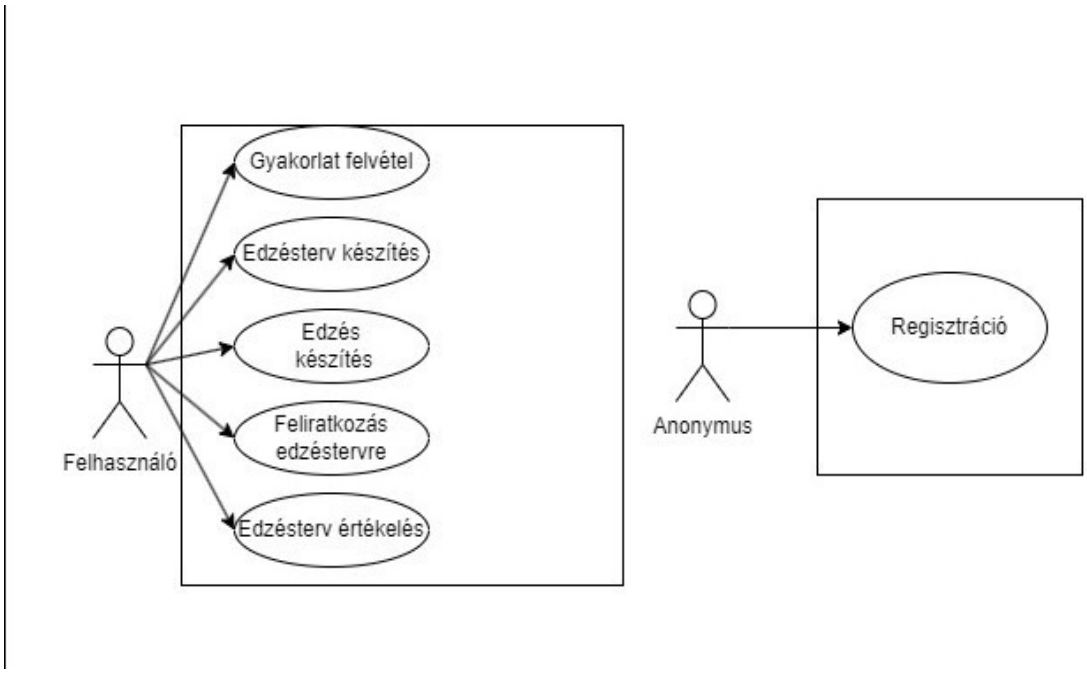
4. Fejlesztés eredménye

A fejlesztés eredményeként létre jött program képes a gyakorlatok rögzítésére. Ezek csupán a gyakorlat definícióját tartalmazzák, azaz a nevüket továbbá, hogy ez egy dinamikus gyakorlat-e vagy statikus, így önmagukban nem használhatóak az edzésterv elkészítéséhez. Ezek a gyakorlatok bekerülnek egy csomagolóba, ekkor már látszik, hogy az adott gyakorlatból mennyit kell csinálni, illetve használandó-e hozzá e súly. Ilyen becsomagolt gyakorlatok kerülnek a feladatokba. Ez biztosítja, hogy a gyakorlatokat egybe lehessen fűzni szuperszettekbe illetve, triszettekbe de egy feladat tartalmazhat csupán egy gyakorlat csomagolót is. Az edzések feladatokból állnak, továbbá az edzéstervek edzésekből. A felhasználó az elejétől kezdve fel tud magának építeni egy edzéstervet vagy az edzések és edzéstervek között böngészve fel tud iratkozni az edzésre, edzéstervre. Lehetőség van továbbá az edzésterv értékelésére, ami az értékelések egyszerű számtani átlaga. Az adható értékelés egy 1-5 skálán terjed.

A program a feltölteni készült adatokat validálja, hogy egyértelműen rossz érték ne kerülhessen az adatbázisba. Azonban a szakmailag rossz vagy kifejezett káros feladatok, edzések, edzéstervek feltölthetőek ezek egy része megelőzhető lenne további fejlesztéssel, illetve szakember bevonásával. Az elkészült alkalmazás nem rendelkezik semmiféle nézettel, a külvilággal való kommunikációja a DTO-k használatával és a végpontok meghívásával történik. A végpontok eredményes meghívásához a kérésben szerepelni kell az autentikációs adatoknak, különben az alkalmazás 401 http hibakódot küld vissza. Továbbá az sem mindegy, hogy az egyik végpontok milyen metódussal vannak meghívva. A kifejezetten adatlekérésre szolgáló végpontokat a GET metódussal kell meghívni. A törzsadatok felvitele, mint például a gyakorlatok, edzések POST metódussal történnek, míg ezeknek az adatoknak a módosítása PUT metódussal történik. Ebben az esetben a nem változtatott mezőket az eredeti állapotban kell elküldeni, különben az új értékkel lesznek felülírva. Egy törzsadatok törlése DELETE metódussal történik, ezt a műveletet illetve a módosítást csak az adat rögzítője tudja megtenni. Az edzéstervre feliratkozás illetve értékelés PATCH metódussal történik. Ha a végpontok nem a megfelelő metódussal vannak meghívva, akkor az alkalmazás 405-ös http státuszkódot küld válaszban.

4.1. Az használati eset diagram

Az alkalmazás használati eset diagramja két aktorral rendelkezik, ezek a felhasználó és az anonymus. Az anonymus aktorok az alkalmazás regisztrációs végpontjához férnek hozzá, majd autentikálás után, mint felhasználó tudja meghívni az alkalmazás végpontjait, amikkel tud gyakorlatot rögzíteni, majd ezekből feladatokat készíteni.



21. Ábra: Az alkalmazás use-case diagramja

Forrás: Saját készítés https://app.diagrams.net/ használatával

Az alkalmazás nem közvetlen felhasználó általi használatra lett tervezve, hanem egy kliens általi használatra így egy edzésterv elkészítése meglehetősen bonyodalmas. A legegyszerűbb végpont a regisztráció, ami bárki számára elérhető és meghívása POST metódussal történik. A kérésnek tartalmaznia kell a kívánt felhasználónevet, aminek egyedinek kell lennie, illetve a jelszót, aminek tartalmaznia kell kisbetűt, nagybetűt, számot és speciális karaktert továbbá minimum 8 karakter, maximum 20 karakter hosszú lehet. A jelszó az adatbázisba mentés előtt kódolva lesz. Ezeket az adatokat a kérés body részébe kerülnek JSON formátumban.

A regisztráció példájából látható, hogy manuális használata a végpontoknak bonyodalmas. Ugyanakkor véleményem szerint egy edzésterv készítés folyamatának leírása nem felesleges, mivel a kliensnek is ezt a folyamatot kell követni, legyen az egy mobil alkalmazás

vagy egy böngészőből használható alkalmazás. Tehát egy edzésterv elkészítésének lépései a következők:

1. Gyakorlat felvétele (opcionális, ha már van rögzítve a gyakorlat)
2. Gyakorlat csomagolók felvétele, gyakorlatra való hivatkozás a gyakorlat azonosítóval
3. Feladatok felvétele, egy feladat több gyakorlat csomagolót tartalmazhat és a hivatkozás a gyakorlat csomagoló azonosítójával
4. Edzések felvétele, egy edzés több feladatot tartalmaz és a hivatkozás a feladat azonosítójával történik
5. Edzésterv felvétele, egy edzésterv több edzést tartalmazhat és a hivatkozás az edzés azonosítójával történik

5. Fejlesztési javaslat

Az alkalmazás egy nagyobb web-alkalmazás részének készült és így is bőven rendelkezik fejlesztési lehetőségekkel. Kezdve az autentikációs eljárással, ami egy elavult basic autentikáció, az adatok validációján át, a szolgáltatások bővítéséig. Ezekből a fejlesztési javaslatokból összegyűjtöttem néhányat, melyek megvalósításával a program jobb lehet. Ezek a javaslatok egyrészt olyan ötletek, amik korábban is felmerültek és az implementáció tervben van, másrészt olyan észrevételekből származnak, amik a fejlesztés közben vagy a fejlesztés végére fogalmazódtak meg.

5.1. Back-end

Az alkalmazás üzleti logikáját a Spring boot back-end végzi. A jelenlegi verzió minimális üzleti logikát tartalmaz, jellemzően a CRUD műveletek kerültek megvalósításra benne. Azonban arra lehetőséget nyújtott, hogy következtetéseket vonjak le a jövőbeli fejlesztésekkel kapcsolatban.

5.1.1. A jelenlegi verzióból levont következtetések

- Az alkalmazás az entitások frissítéséhez PUT kéréseket használtam, ami feleslegesen nagy adatcsomagok fogadását jelenti, mert ilyenkor a teljes objektumot tartalmazza a kérés. A frissítések kezelésére PATCH kéréseket várni és kezelni előnye lenne, hogy nem kell elküldeni a teljes objektumot, hanem csak a mezőt, amit módosítani szükséges. [https://www.geeksforgeeks.org/difference-between-put-and-patch-request/]
- Az alkalmazás entitás felépítése alkalmas komplex edzésterv objektumok tárolására, kezelésére, felhasználóhoz rendelésére. Azonban alkalmatlan arra, hogy egyes felhasználók esetén követni lehessen az edzéstervek teljesítését, kommentálását. Ennek megoldására csomagoló entitások bevezetését tartom célszerűnek, ami tartalmazza az edzéstervet és a haladással kapcsolatos adatokat. Ezek az edzéstervek helyett pedig ezek a csomagoló entitások legyenek a felhasználókhoz kapcsolva.

5.1.2. Fejlesztendő szolgáltatások

- Kommentelés: Az egyes edzésekkel, edzéstervekkel kapcsolatos párbeszéd támogatására olyan komment rendszer kialakítása, amivel egy entitáshoz több kommentet lehet hozzárendelni, továbbá a kommentre közvetlenül lehessen válaszolni.
- Edzésmunka mennyiség követése: Az edzésmunka mennyiségével kapcsolatban két hibát lehet véteni. Az első, hogy alul-edzés történik, vagy az edzésmunka mennyisége nem elegendő a fejlődéshez. A második, hogy az edzésmunka olyan mennyiségű, amiből a test nem képes regenerálódni megfelelően. Ez fenn áll a test egészére és az izmokra is. Ennek követésére egyrészt az edzésmunka összegzését tartom célszerűnek a következő módon. Az edzésmunka mértékegysége kilogramm, egyenlő az ismétlések számának és a használt súly szorzatával. Másrészt a gyakorlatokra kialakítani egy profilt, amiben a terhelt izmok vannak szerepeltetve így izmokra vetítve is követhető az edzésmunka. Ez a szolgáltatás segítséget nyújthat egy lemaradás okának feltárásában, illetve a fáradásos sérülések elkerülésében.
- Testprofil felépítése: A testprofil alatt itt azt értem, hogy a felhasználó teste mekkora edzésmunkát bír elvégezni úgy, hogy az biztonságosan regenerálódni tudjon, mint ezt izomcsoportokra lebontva. Az egyes izomcsoportok különböző mértékű terhelésre reagálnak jobban. Általánoson gondolkodva meg lehet határozni egy intervallumot, amibe a legtöbb izomcsoport tartozik, valamint ennek az intervallumnak az átlagát a beállítani az valamennyi izomra alapértelmezésnek. Majd a felhasználó ezt az alapértelmezést képes felülírni, ezzel pontosítva a testprofilját.

4.2. Kliens

Az alkalmazást célszerű több klienssel fejleszteni, ugyanis az edzés közben az emberek általában telefont tartanak közelben. A telefonos kliensen az edzések követésére, a teljesítés és megjegyzések hozzáfűzésére lenne alkalmasabb, míg ha az edzésmunka követése is megvalósul az egy komplex, sok információt tartalmazó felületet is eredményezhet, ennek kezelése egy böngészőből használható klienssel lenne felhasználó barát megoldás.

Összefoglalás

A szakdolgozatommal betekintést nyújtottam a Spring boot keretrendszer használatába és bemutattam a technológiákat, amire épül. A keretrendszer hatalmas és sokoldalúan használható az alkalmazásom csupán egy részét használja ki a Spring boot nyújtotta lehetőségeknek. Mivel ez egy folyamatosan fejlődő rendszer ezért a fejlesztés során számomra is újdonság volt a biztonsági konfiguráció módja és hosszas utána járást kívánt a megvalósítás.

Az alkalmazás struktúrájának kialakítása során azokat az ajánlásokat követtem, amik a rétegekbe szervezést javasolják. Továbbá a fejlesztés közben törekedtem olyan változó-, metódus-, osztálynevek használatára, melyek elegendően beszédesek ahhoz, hogy abból következtetni lehessen annak szerepére, mégis rövid itt kapcsolódik a következő elv, amelyet igyekeztem követni, hogy egy metódus, osztály lehetőleg egy dologért legyen felelős, tehát egy dolgot csináljon. Tapasztalataim szerint ez a fajta alkalmazás felépítés segíti a fejlesztő munkáját, mivel további funkcionalitás lényegesen könnyebben beilleszthető a programba valamint fejlesztési idő is megtakarít. Ez az időmegtakarítás akkor még jelentősebb, amikor hosszú idő után újra elő kell venni egy program forráskódját egy fejlesztési igény vagy egy hiba miatt.

A fejlesztés eredményeként létre jött program képes a gyakorlatok rögzítésére, módosítására, törlésére. Ezek csupán a gyakorlat definícióját tartalmazzák, azaz a nevüket továbbá, hogy ez egy dinamikus gyakorlat-e vagy statikus, így önmagukban nem használhatóak az edzésterv elkészítéséhez. Ezek a gyakorlatok bekerülnek egy csomagolóba, ekkor már látszik, hogy az adott gyakorlatból mennyit kell csinálni, illetve használandó-e hozzá e súly. Ilyen becsomagolt gyakorlatok kerülnek a feladatokba. Ez biztosítja, hogy a gyakorlatokat egybe lehessen fűzni szuperszettekbe illetve, triszettekbe de egy feladat tartalmazhat csupán egy gyakorlat csomagolót is. Az edzések feladatokból állnak, továbbá az edzéstervek edzésekből. A felhasználó az elejétől kezdve fel tud magának építeni egy edzéstervet vagy az edzések és edzéstervek között böngészve fel tud iratkozni az edzésre, edzéstervre. Lehetőség van továbbá az edzésterv értékelésére, ami az értékelések egyszerű számtani átlaga.

A kommunikáció http kéréseken keresztül történik, az entitásnak/ műveletnek megfelelő végponton és a megfelelő metódussal és paraméterezéssel. A feltölteni kívánt adatokkal kapcsolatban validáció is történik, azonban ez a triviálisan rossz adatokat szűri ki, mint amilyen például a negatív ismétlés szám, azonban a szakmailag téves vagy esetleg egyenesen káros adatokat képtelen kiszűrni. Ennek a téves adatoknak egy része automatikusan kiszűrhetőek lennének további fejlesztésekkel, illetve egy edző szakember bevonásával.
Egy edzésterv elkészítésének lépései a következők:

1. Gyakorlat felvétele (opcionális, ha már van rögzítve a gyakorlat)
2. Gyakorlat csomagolók felvétele, gyakorlatra való hivatkozás a gyakorlat azonosítóval
3. Feladatok felvétele, egy feladat több gyakorlat csomagolót tartalmazhat és a hivatkozás a gyakorlat csomagoló azonosítójával
4. Edzések felvétele, egy edzés több feladatot tartalmaz és a hivatkozás a feladat azonosítójával történik
5. Edzésterv felvétele, egy edzésterv több edzést tartalmazhat és a hivatkozás az edzés azonosítójával történik

Az alkalmazás funkcionalitása javában egy CRUD alkalmazásra emlékeztet, ami autentikálja a kéréseket, majd validálja a beérkező adatokat. Az alapvető műveleteken kívül egy minimális üzleti logikát és megvalósítottam. Ezek a funkciók is az entitásokat módosítják, viszont ezekhez a módosításokhoz nem csupán értékátadások kellenek, hanem számítások is, mint például az edzéstervek értékelése, vagy feliratkozás edzéstervre. A fejlesztésben használt entitások számos kapcsolattal rendelkeznek azonban az edzésterv kezelésére alkalmasak az egyéni haladás követésére nem. Ezzel kapcsolatos fejlesztési javaslat, hogy az edzéseket és edzésterveket egyrészt egy definícióként kell tárolni az adatbázisban, majd a felhasználó az edzésre vagy edzéstervre való feliratkozásával létre jön egy csomagoló entitás. Ez a csomagoló entitás tartalmazza az edzéstervet, továbbá a haladás adatait, egyéni megjegyzéseket.

Az alkalmazás szolgáltatásai tovább bővíthetőek ez edzések adatainak felhasználásával, illetve a felhasználó profilozásával. Az edzések adataiból megállapítható az elvégzett edzésmunka, valamint a felhasználó profilozásával megállítható, hogy az egy edzésmunka mennyiségével a felhasználó milyen eredményt várhat az edzéstervtől. Ugyanis az

edzésmunka mennyisége döntően befolyásolja azt, hogy fejlődünk, stagnálunk vagy épp visszafejlődünk, illetve növeljük a sérülés kockázatát. A funkció hátránya, hogy a felhasználónak egy alapértelmezett profil szolgáltatása lehetséges, amit neki kell pontosítania, hogy egy ilyen funkciót megvalósítani lehessen. Azon felül az önismeret hiányából az is meglehet, hogy profilját hibásan állítja be és így a funkció sem feltétlen támogatja a fejlődését. A probléma részben kiküszöbölhető lenne a felhasználó okos órájának adatainak elemzésével, ugyanis annak adataiból lehetséges következtetni arra, hogy a felhasználó egyéni kapacitásához képest milyen mennyiségű terhelést kapott. Összességében úgy vélem, hogy az alkalmazásban rengeteg fejlesztési lehetőség van, a kérdés az, hogy megéri-e lefejleszteni?

A WorkoutPlanner alkalmazással bemutattam, hogy egy a Spring boot keretrendszer, milyen mértékben megkönnyíti egy alkalmazás fejlesztés azáltal, hogy biztosít az általa szolgáltatott számos szolgáltatáshoz egy alapértelmezett működést és beállítást, amit bármikor felülírhatunk. Véleményem szerint érdemes kipróbálni a keretrendszer, ugyanis számos konfigurációs terhet le vesz a fejlesztő válláról, tartja a lépés az aktuális fejlesztésekkel és új paradigmákkal és elég komplex ahhoz, hogy a legtöbb fejlesztési igényt lefedje és biztosítja a fejlődési lehetőséget azzal, hogy felülírhatjuk az alapértelmezett viselkedéseket, de ahhoz mélyebbre kell ásni a keretrendszerben.

Irodalomjegyzék

1. https://docs.docker.com/desktop [Hozzáférés dátuma: 2022.10.21]
2. https://docs.jboss.org/hibernate/orm/4.3/javadocs/org/hibernate/annotations/ [Hozzáférés dátuma: 2022.10.21]
3. https://docs.oracle.com/javaee/6/api/ [Hozzáférés dátuma: 2022.10.21]
4. https://docs.oracle.com/javase/8/docs/api/java/util/Optional.html [Hozzáférés dátuma: 2022.10.01]
5. https://docs.spring.io/spring-boot/docs/2.7.1/reference/pdf/spring-boot-reference.pdf [Hozzáférés dátuma: 2022.10.01]
6. https://docs.spring.io/spring-boot/docs/current/reference/html/application-properties.html] [Hozzáférés dátuma: 2022.10.28]
7. https://docs.spring.io/spring-framework/docs/current/reference/html/core.html [Hozzáférés dátuma: 2022.10.01]
8. https://docs.spring.io/spring-security/site/docs/4.2.x/reference/html/appendix-schema.html [Hozzáférés dátuma: 2022.10.26]
9. https://dzone.com/articles/spring-boot-custom-password-validator-using-passay [Hozzáférés dátuma: 2022.10.26]
10. https://hibernate.org/orm/documentation/6.1/ [[Hozzáférés dátuma: 2022.10.26
11. https://maven.apache.org [Hozzáférés dátuma: 2022.10.21]
12. https://medium.com/the-resonant-web/spring-boot-2-0-project-structure-and-best-practices-part-2-7137bdcba7d3 [Hozzáférés dátuma: 2022.10.15]
13. https://projectlombok.org/ [Hozzáférés dátuma: 2022.10.15]
14. https://projectlombok.org/features/GetterSetter [Hozzáférés dátuma: 2022.10.26]
15. https://spring.io/blog/2022/02/21/spring-security-without-the-websecurityconfigureradapter [Hozzáférés dátuma: 2022.10.28]
16. https://spring.io/projects/spring-security [Hozzáférés dátuma: 2022.10.15]
17. https://www.baeldung.com/the-persistence-layer-with-spring-data-jpa [Hozzáférés dátuma: 2022.10.01]
18. https://www.digitalocean.com/community/tutorials/spring-restcontroller [Hozzáférés dátuma: 2022.10.26]
19. https://www.educative.io/answers/what-are-java-spring-framework-advantages-and-disadvantages [Hozzáférés dátuma: 2022.10.01]

20. https://www.edureka.co/blog/what-is-javabeans/ [Hozzáférés dátuma: 2022.10.01]
21. https://www.h2database.com/html/features.html [Hozzáférés dátuma: 2022.10.15]
22. https://www.interviewbit.com/blog/java-8-features/ [Hozzáférés dátuma: 2022.09.25]
23. https://www.java.com/releases/ [Hozzáférés dátuma: 2022.09.25]
24. https://www.javatpoint.com/java-8-stream [Hozzáférés dátuma: 2022.09.25]
25. https://www.javatpoint.com/spring-tutorial [Hozzáférés dátuma: 2022.10.01]
26. https://www.jetbrains.com/idea/features/ [Hozzáférés dátuma: 2022.10.21]
27. https://www.w3schools.com/java/ [Hozzáférés dátuma: 2022.09.25]
28. https://www.w3schools.com/java/java_lambda.asp [Hozzáférés dátuma: 2022.09.25]
29. M. FOWLER (2002): Patterns of Enterprise Application Architecture [Hozzáférés dátuma: 2022.10.15]
30. R. C. MARTIN (2008): Clean Code: A Handbook of agile Software Craftsmanship [Hozzáférés dátuma: 2022.10.28]
31. www.geeksforgeeks.org/java/ [Hozzáférés dátuma: 2022.09.25]

Táblázatok és ábrák jegyzéke

Táblázatok jegyzéke

Ábrák jegyzéke

Mellékletek

1. **számú melléklet:** Konzultációs nyilatkozat
2. **számú melléklet:** Eredetiségi nyilatkozat

1. számú melléklet: Konzultációs nyilatkozat

KONZULTÁCIÓS NYILATKOZAT

A **Vaszily Zoltán** (név) (hallgató Neptun azonosítója: **EJB9Z9**) konzulenseként nyilatkozom arról, hogy a szakdolgozatot[1] áttekintettem, a hallgatót az irodalmi források korrekt kezelésének követelményeiről, jogi és etikai szabályairól tájékoztattam.

A szakdolgozatot a záróvizsgán történő védésre javaslom / nem javaslom[2].

A dolgozat állam- vagy szolgálati titkot tartalmaz: igen nem*[3]

Kelt: **2022.** év **október** hó **26.** nap

Belső konzulens

[1] A megfelelő dolgozattípus meghagyása mellett a többi típus törlendő.
[2] A megfelelő aláhúzandó.
[3] A megfelelő aláhúzandó.

2. számú melléklet: Eredetiségi nyilatkozat

NYILATKOZAT

a szakdolgozat nyilvános hozzáféréséről és eredetiségéről

A hallgató neve:	Vaszily Zoltán
A Hallgató Neptun kódja:	EJB9Z9
A dolgozat címe:	Edzéstervező alkalmazás fejlesztése Spring boot keretrendszer használatával
A megjelenés éve:	2022
A konzulens tanszék neve:	Alkalmazott informatika

Kijelentem, hogy az általam benyújtott szakdolgozat egyéni, eredeti jellegű, saját szellemi alkotásom. Azon részeket, melyeket más szerzők munkájából vettem át, egyértelműen megjelöltem, s az irodalomjegyzékben szerepeltettem.

Ha a fenti nyilatkozattal valótlant állítottam, tudomásul veszem, hogy a Záróvizsga-bizottság a záróvizsgából kizár és a záróvizsgát csak új dolgozat készítése után tehetek.

A leadott dolgozat, mely PDF dokumentum, szerkesztését nem, megtekintését és nyomtatását engedélyezem.

Tudomásul veszem, hogy az általam készített dolgozatra, mint szellemi alkotás felhasználására, hasznosítására a Magyar Agrár- és Élettudományi Egyetem mindenkori szellemitulajdon-kezelési szabályzatában megfogalmazottak érvényesek.

Tudomásul veszem, hogy dolgozatom elektronikus változata feltöltésre kerül a Magyar Agrár- és Élettudományi Egyetem könyvtári repozitori rendszerébe.

Kelt: _____2022._____ év _____11._____ hó ___2.___ nap

Vaszily Zoltán

_____

Hallgató aláírása