Mit tanultam ebben a félévben?

Göbhardter Ádám

Bevezetés

A tantárgy felvételekor kicsit féltem, hogy milyen lesz másokkal együtt dolgozni egy nagy projekten, hiszen eddig ilyet még csak rendes munkahelyeken láttam. Azonban ahogy a munkánk haladt előre, egyre jobban kezdtem hozzászokni/ráérezni, hogyan is kell ezt csinálni. A projekt elkészítése alatt sok mindent tanultam Imre Gábor tanár úrtól, a csapattársaimtól és persze magamtól is. A következőkben összefoglalom azokat a témákat, amikkel kapcsolatban számomra új információkra tettem szert.

Git Workflow

Munkánk során a webalkalmazás egyes állapotainak karbantartásához GitHub-ot használtunk, ami a Git-en alapszik. Ezt a rendszert használtam már korábban is, ám mint kiderült, amatőr szinten.

A projekt kezdeti fázisában Széll Adrián Bence bevezetett minket a git workflow rejtelmeibe. Megtanultam, hogy minden elvégzendő feladatra létre kell hozni egy issue-t, aminek beszédes nevet kell adni. Ezután, ha az issue által megnevezett feladatot meg akarom csinálni, akkor létre kell hozni hozzá egy külön branch-et, aminek a neve meg kell, hogy egyezzen az issue nevével (a későbbi könnyebb azonosítás érdekében). Fontos, hogy ezen a branch-en csak az adott feladatot oldjuk meg és hogy sűrűn commit-oljunk. A branch-en történő utolsó commit leírásának végére be kell szúrni a „Closes #<issue_number>” sort, ezáltal, ha a kódunk merge-elődik, akkor a hozzátartozó issue automatikusan törlődik.

Továbbá megtanultam, hogy nem kell aggódni akkor, ha egy branch jó pár commit-tal le van maradva a fő branch-től, hiszen egy gyors rebase-elés megoldja ezt a problémát, és máris a friss fő branch lesz a saját branch-ünk alapja.

Néhány további git-es fogalom is kitisztult a fejemben, mint például, hogy mi az a „pull request”, „pull origin”, „push to branch” és „checkout”. A projekt készítése közben én végig a GitHub Desktop programot használtam, így már annak funkcióinak használatában is elég gyakorlatot szereztem.

Tesztelés

A tesztelés számomra mindig is egy ilyen elhanyagolható dolog volt, mivel talán eddig egyik másik tantárgy keretein belül sem foglalkoztunk vele komolyabban. Itt most más volt a helyzet. A csapattársaimmal már a projekt elején megegyeztünk abban, hogy csak olyan kódot engedünk be a repository-ba, ami le is van tesztelve. Így nem volt más választásom, mint hogy megtanuljam hogyan is kell helyesen tesztelni. Szerencsére ehhez kaptam jó sok külső segítséget is.

Csapatommal az egyes kódrészletek tesztelését már azelőtt elkezdtük, mielőtt erre a tanár úr felhívta volna a figyelmünket. A projekt létrehozásakor Széll Adrián Bence létrehozott egy

tesztkörnyezetet, amivel én még sohasem találkoztam, és kezdetben csak ezt használtuk. Ez a Testcontainers volt. Érdekes volt számomra ez a technológia, mivel egész integrációs teszteket tudtunk vele készíteni úgy, hogy maga a tényleges adatbázis sem maradt ki a képből. Ebben a környezetben való tesztelésnél két dolgot tanultam meg nagyon: az egyik, hogy a PostgreSQL adatbáziskezelőnek alapból is van egy „user" nevű táblája, így, ha én is létrehozok egy ilyet, akkor csúnya, nehezen értelmezhető hibákat kapok. A másik dolog pedig az, hogy az adatbázisból addig nem tudok törölni egy rekordot, amíg további rekordok hivatkoznak rá. Ezekkel a dolgokkal tesztelés során jól meggyűlt a bajom, és jó sokáig vakartam a fejem, mire rájöttem a hibára.

Ahogy haladt előre a félév, Imre Gábor tanár úr bemutatott nekünk, egy számomra szintén csak hallásból ismert, tesztelő eszközt. Ez a Mockito volt. Kezdetben számomra ez az eszköz teljesen érthetetlen volt. Nem értettem azt, hogy hogyan lehet ezzel igazán tesztelni valamit, ha előre megmondjuk, hogy mit adjon vissza. Kellett egy kis idő, meg az internet böngészése, mire leesett mit is tudunk vele valójában kiküszöbölni. Segítségével megtanultam gyorsabb lefutású teszteket írni, amelyek gyakorlatilag ekvivalensek a Testcontainers-ben megírtakkal. Megtanultam továbbá, hogy mikor és hogyan kel használni a „@Mock", „@InjectMocks" és „when" annotációkat, valamint kulcsszót.

Egy webalkalmazás felépítése

Amikor kiderült, hogy a projektünk során egy webalkalmazást kell csinálnunk, kicsit megijedtem, mivel ez az egész webes téma nem az én asztalom. Azonban azt kell, hogy mondjam, hogy a félév során egészen megkedveltem a témát. Projektünk során megtanultam az egyes lépéseket, amelyeket követve a végeredmény egy jól összerakott webalkalmazás lesz. Ezek a lépések pedig sorban:

- Modellek (azaz DAO-k) létrehozása, melyek szerepe, hogy az eltárolni kívánt adatokat reprezentálják.
- Repository-k létrehozása az egyes modellekhez, amelyek egyszerűbb metódusokat valósítanak meg.
- Service-ek létrehozása, amelyek már a bonyolultabb üzleti logikákat megvalósító függvények implementációit tartalmazzák.
- Controller-ek létrehozása, amelyek már a webrétegben történő kommunikációt valósítják meg GET, POST, PUT és DELETE metódusokkal és map-eléssel.

A controller-ek tesztelését Mockito segítségével csináltam. Megtanultam, hogy az eszköz segítségével akár egy egész MVC-t is tudunk mock-olni, amely segítségével utána tudjuk az egyes metódusokat (GET, POST, PUT és DELETE) imitálni, így nincs szükség a teszteléshez külső eszközre (mint például a Postman).

Összegzés

Sajnos egy félév nem volt elegendő, hogy az egész alkalmazást befejezzük, így a frontend rész teljesen kimaradt. Ezt leszámítva, azt kell, hogy mondjam, hogy rengeteg új ismeretre tettem szert, mind a csapatmunka, mind a programozás terén, amelyeket reményeim szerint tovább tudok majd hasznosítani jövőbeni tanulmányaim, illetve munkám során.

Mit tanultam ebben a félévben?

Bidló András Rudolf

GitHub

A félév során a közös munka során a verziókövetéshez és közös fejlesztés koordinálásához a GitHub-ot használtuk. Sajnos korábbi tárgyaim során nem volt lehetőségem elmélyülni ennek használatában, de szerencsére csapattársaim segítségével sokkal jobban és mélyebben sikerült megértenem a GitHub használatát. Megtanultam, hogy kell issue-kat létrehozni és azokkal dolgozni, azokhoz branch-et létrehozni, hogyan szokás a branch-et elnevezni a hozzá tartozó issue nevével. Megtanultam a pull requests használatát, hogyan kell létrehozni, review, elfogadni egy pull request-et. Ami egy nagyobb nehézséget okozott, hogy hogyan lehet, úgy branchet létrehozni, hogy a megfelelő legyen a kiindulás és a branchet megfelelően lehessen merge-elni a main branch-re. GitHub kapcsolatban még tanultam, hogy hogyan kell megfelelően részletes commit üzenetet írni, hogyan lehet annak segítségével egy issue-t lezárni.

Maven

A fejlesztés megkezdésekor sikerült megismernem a Maven-t, aminek segítésével sokkal könnyebbé tehető a függőségek letöltése és kezelése.

Hot key-k

A félév során a kódoláskor az IntelliJ IDEA-t használtam, amiben sikerült pár billentyű kombinációt megtanulnom, például a kód automatikus formázását, hogyan lehet másik fájlra váltani, hogyan lehet a programot futtatni, hogyan lehet debug módban futtatni.

SpringBoot Framewrok:

A félév során egy jármű bérlő alkalmazást fejlesztettünk SpringBoot segítségével, aminek fejlesztése során sikerült megismernem és megtanulnom, hogyan kell entity-ket létrehozni, azok között függősége definiálni. Beállítani, hogy lehet-e null egy attribútum vagy nem. Hogyan lehet repository-t létrehozni és használni. A repository létrehozásánál sikerült megtanulnom, hogy létezik a JpaRepository ős osztály, aminek segítségével a legtöbb lekérdezést nem kell kézzel megírnunk a repository-ba hanem már létezik, így sokkal gyorsabban tud menni a fejlesztés. A félév során sikerült még megismerkednem a SpringBoot-os Service osztállyal.

Stream

A félév elején még sikerült jobban elmélyednem a stream használatában, aminek segítségével sokkal gyorsabban és kevesebb kódolás segítségével lehet feladatokat elvégezni. Sokkal gyorsabban lehet ennek segítségével egy Repository osztályban keresni vagy műveleteket létre hajtani rajta.

Lombok

A félév során még megismertem a lombok-ot, aminek segítségével automatikusan legenerálhatók az attribútumok getter-ei, setter-ei, amivel így sokkar gyorsabbá válik a fejlesztés, ez számomra az elején kicsit fura, volt, hogy ez szóval megoldjuk az összes getter-t vagy setter-t, de aztán rá jöttem, hogy ez mennyivel megkönnyíti a dolgunkat és így mennyivel gyorsabban megy egy új osztály megírása. Még a lombok-oknál sikerült megismernem, hogy létezik a @RequiredArgsConstructor, ami automatikusan generál egy konstruktort, ami inicializálja a tag változókat.

Mockito test

A félév során még sikerült megismernem a számomra teljesen ismeretlen mockito tesztelés alapjait és annak segítségével egy pár tesztet írnom. A mockito teszt azért jó mivel azzal úgy is tudjuk tesztelni a service függvényünket, hogy nem hozunk létre hozzá adatbázis, így a tesztek megírása sokkal könnyebben megy mintha egy adatbázist kéne hozzá írni, de ennek hátránya, hogy a kapcsolatokat nem tudjuk tesztelni, se a repository-ban lévő hibákat. Ezzel csak az üzleti logika hatékony és gyors tesztelése lehetséges. A mockito teszt megértése először nehéz volt számomra nem értettem, hogy adatbázis elérés nélkül, hogyan szeretnénk tesztelni, de aztán ahogy írtam a teszteket, sikerült átlátnom, hogy mi az egész tesztnek a lényege és hogy hogyan működik maga a teszt, miket lehet vele csinálni

Spring Unit test

A félév során még egy kis mértékben újdonság volt számomra az eddig ismeretlenen SpringBootTest, korábban Unit teszteket használtam már, de így SpringBootTest-el együtt még nem. De jó volt rá jönni, hogy mennyivel könnyebb így tesztelni, meg, hogy a SpringBoot milyen sok segítséget tartalmaz a teszteléshez.

Controller

A félév vége felé még sikerült megtanulnom a Controller-t, aminek segítségivel sokkal könnyebb a webes elérést létrehozni.

Mit tanultam ebben a félévben?

Dongó Rajmund

A félév során sok dolgot tanultam, ezeket szeretném összefoglalni, miközben végig vezetem az olvasót a projekt elkészítésének folyamatán.

A félév első feladata egy könyvrészlet elolvasása, majd prezentálása volt, itt is sok érdekességet hallottam és voltak olyan dolgok, amelyek hosszú idő után tisztázódtak.

A Clean Code elvek ismétlése, új elvek felfedezése nagy öröm volt számomra, hiszen eddig is érdekelt a téma és örültem neki, hogy a témalaboratórium keretein belül is tudok ezzel foglalkozni.

A témalaboratórium első kihívása a projekt kiválasztása volt, mivel a témalaboratórium címe java fejlesztést támogató eszközök, ennek megfelelően próbáltuk összeállítani a követelményeket a csapattal.

Az alkalmazás a témalaborarótrium fejléceivel egységben, Spring Boot-tal készült alkalmazás lett, ahol adatbázisnak PostgreSQL-t használtunk. Ennyi volt az alap koncepció, ebből dolgozott a csapat a munka elején.

Végül egy járműbérlő alkalmazás mellett döntött a csapatom, mivel ki lehet használni a program egyszerűségét, de mégis tetszőlegesen sokáig bonyolítható.

Egy másik nem szakmai kihívás a csapatmunka megoldás volt, ezt mi GitHub segítségével igyekeztük menedzselni. A csapatban már volt tapasztaltabb tagunk így az ő közreműködésével **sikerült elsajátítani a GitHub megfelelő használatát.**

A GitHub-on történő fejlesztés menete az alábbi volt ideális esetben a csapaton belül:

Elsőnek a fejlesztendő funkció/ javítandó hibához rendeltünk egy issue-t, amelyet ideális esetben hozzárendeltünk vagy magunkhoz, vagy a csapat egy tagjához.

A klónolt repository-ban mindig minden issue-hoz egy külön branch-et rendeltünk, így ennek megfelelően tudtunk dolgozni vele, külön a többi fejlesztendő/ félig fejlesztett funkciótól.

Klónolt repsitory-n elvégeztük a változtatásokat, majd ezeket commit-oltuk es push-oltuk. Egy fontos részlet volt a commit-oláson belül, hogy megadjuk melyik issue-t zárja lesz és felpusholtuk.

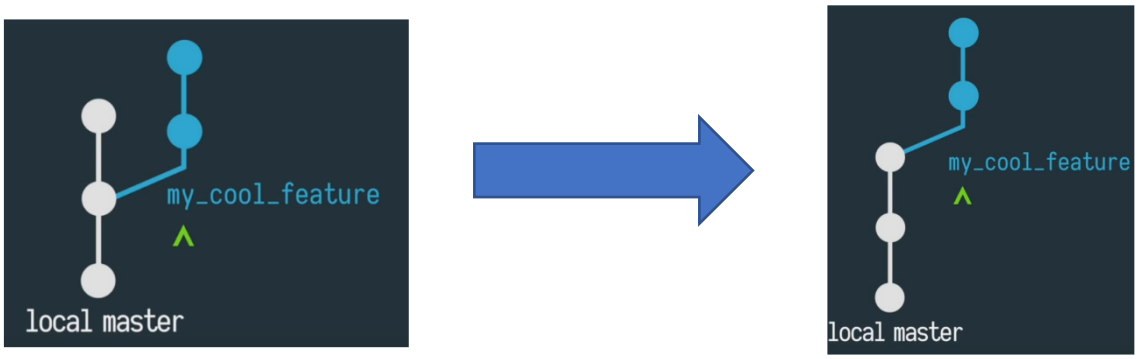
A commit és push után a pull request következett, itt a bevett szokás az volt, hogy a csapat nagyrészének ¾-ének el kellett fogadni a változtatást, tehát minimum 2 db. review szükséges volt a merge/rebase elvégzéséhez.

A Git-es munkavégzés megkönnyítésére a SourceTree szoftvert alkalmazta a csapat, így sokkal átláthatóbb és könnyebb volt menedzselni a jelenlegi „git tree” állását.

A csapatunk a git workflow-ja egy main és a rajta futó developement branch-ekből állt, és nem merge, hanem rebase használatával helyeztük production-be a változtatásainkat.

Habár az elején nem működött könnyen, végül sikerült egészen jól megszoknunk a workflow-t és ez a git history-ból is látszódik.

A gitworkflow-nk az alábbi módon nézett ki:



A projekt specifikációjának a leadása után a feladat a projekt vázának elkészítése volt, ahol az alapvető osztályokat, osztályok közötti összefüggéseket kellett leírnunk.

A következőkben szeretnék kitérni a megvalósítás technikai részleteire.

A program elkészítéséhez a Spring Boot keretrendszert alkalmaztuk, ezzel megkönnyítve a Spring konfigurálását. Itt mivel még nem használtam Spring-et (Spring Boot nélkül), ezért nem éreztem a könnyedséget, amelyet a Spring Boot hozott nekünk, viszont megtanultam ezt is használni, a saját megoldásaival együtt.

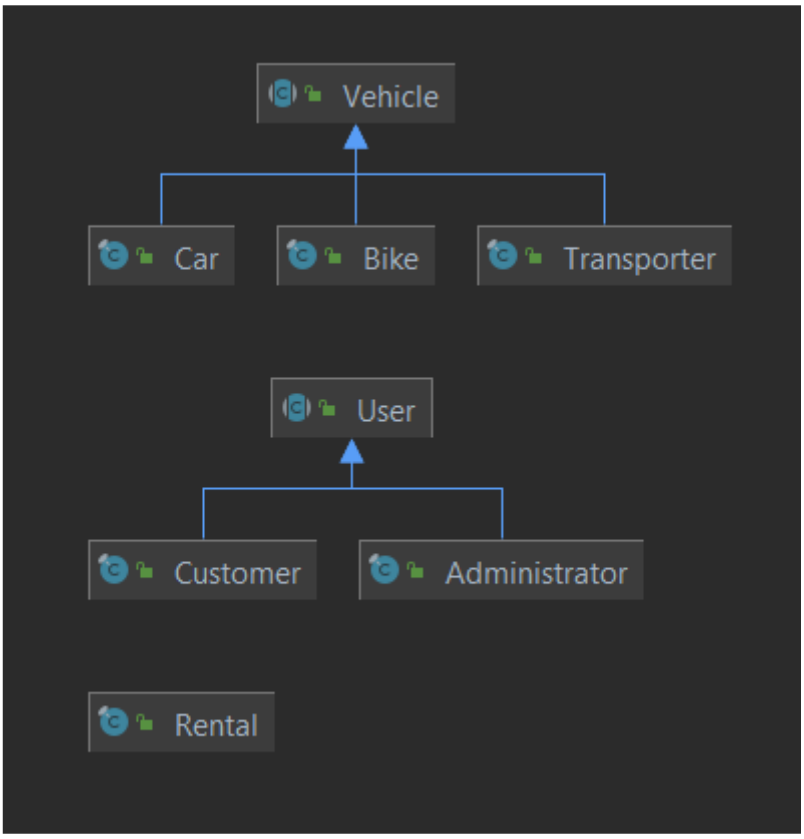
A Spring első meglepő eleme számomra a Maven volt, mivel még nem használtam Maven-t ezért ez is egy új kihívásnak számított nekem, habár rendelkeztem némi Gradle tapasztalattal, de az XML-t kicsit nehezebbnek és kevésbé átláthatónak találtam.

A program elkészítésének első lépése, ahogyan már említettem, az entitások elkészítése, definiálása volt. Mivel járműveket tároltunk ezért létrehoztunk egy *vehicle* osztályt, amely az ősosztálya volt a többi járműnek az adatbázisban.

Emellett létre kellett hoznunk enumerációkat is, hogy az egyes változók lehetséges értékeit megadjuk.

Ezen alap osztályok létrehozása után, jött a neheze: Kapcsolatokat kellett definiálnunk az osztályok között, itt minden osztály között OneToOne vagy ManyToMany, esetleg OneToMany kapcsolatot alkalmaztunk. Ezen kapcsolatokat az adatvezérelt rendszerek tárgyból tanultak alapján sikerült definiálnunk.

A csapatunk az alábbi osztályhierarchiát alkalmazta:



Megismert technológia: @OneToOne, @ManyToOne, @OneToMany

Mivel bizonyos értékek definiálása kötelező, ezt constraint-ként meg is kellett adnunk a program számára, mi ezt a *@Column(nullable = false)* és a kapcsolatokat leíró például ManyToOne után adtuk meg az (optional = false) kényszert.

Megismert technológia: @Column(nullable=false)

Természetesen az adatbázisok alapvető tulajdonságainak is meg kellett felelnie a programnak, ennek megfelelően minden entitás rendelkezik egy azonosítóval, amelyet a Spring maga hoz létre. Mivel ID generálással nem volt még tapasztalatom, mindenképpen kellemes kihívást jelentett számomra, de egyszerűbbnek bizonyult, mint hittem, hogy lesz.

A java-hoz megszokott módon a programunkban is szükséges lett volna konstruktorok definiálása, de a Spring keretrendszer könnyedségei szinte mindenhol meglátszódtak a fejlesztés során, itt is elég volt egy *@NoArgsConstructor* alkalmazása.

Megismert technológia: @NoArgsConstructor, @AllArgsConstructor

A getter-ek és setter-ek sem okoztak problémát a fejlesztés során, ezeket igy egyszerű *@Getter és @Setter* annotáció alkalmazásával el lehetett készíteni.

Megismert technológia: @Getter, @Setter

Talán az adatbázisra történő leképezés legnehezebb része az öröklődés definiálása volt az adatbázis számára, itt kénytelenek voltunk megadni az öröklődés módját is, amely számunkra *table_per_class* lett.

Ez, mint később kiderült azt jelenti, hogy az egyes táblákhoz 1-1 osztályt rendelünk.

Egy speciálisabb osztályunk a UserRole, mivel ez implementálta a GrantedAuthority-t és ennek segítségével tudunk megkülönböztetni jogosultsági szinteket a programon belül.

Megismert technológia: GrantedAuthority

A járműveinkre általánosan igaz, hogy a típusuk egy enumerált változó, hogy meg tudjuk különböztetni a fajtájukat.

Végül az egyes osztályokat, amelyek Entity-knek felelnek meg *@Entity* annotációval láttuk el, és megadtuk, hogy az enumeráció típusa String.

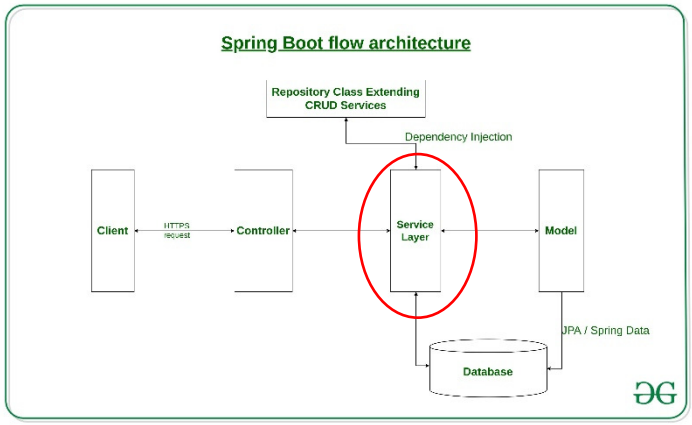
Megismert technológia: @Entity

Természetesen a programunkat el kellett látni adatbázis eléréssel is, ezt mi PostgreSQL használatával tettük meg, amely Docker-en keresztül futott a csapat tagjai számára. A kapcsolat eléréshez pedig definiáltuk az application.yaml-ben.

Megismert technológia: PostgreSQL, adatbázis kapcsolat elérése Spring-gel.

A projekt elkészítésének harmadik fázisa a tesztelésből állt, és az üzleti logika megvalósítása volt a célunk.

A projekt üzleti logikáját a Spring konvenciói szerint a service osztályokban implementáltuk.



Megismert technológia: @Service

Az üzleti logika megvalósításában a Spring-hez köthető újdonság a Repository használata volt, amely segítségével alapvető metódusokat tudtunk definiálni és az adott entitásainkat tudtuk tárolni benne.

Megismert technológia: JpaRepository, Repository használata

A tesztelést a programon belül két részre lehet bontani: Első részben voltak a hagyományos módon elkészített tesztek, mikor a program különböző funkciót teszteltük le, viszont ezekhez persze szükséges volt az adatbázis megléte. Természetesen hátránynak számít, ha szükséges az adatbázis megléte a teszteléshez, így nem csak ezt a módszert, de Mockito használatát is alkalmaztuk a program tesztelése során.

Az első tesztmódszer problémáját viszont sikerült kiküszöbölnünk, mivel @SpringbootTest alkalmazásával elértük, hogy automatikusan hozzon létre PostgreSQL-t az eszközön amennyiben a Docker fut a számítógépen.

Megismert technológia: @SpringbootTest @ActiveProfiles

Felhasznált technológia: JUnit

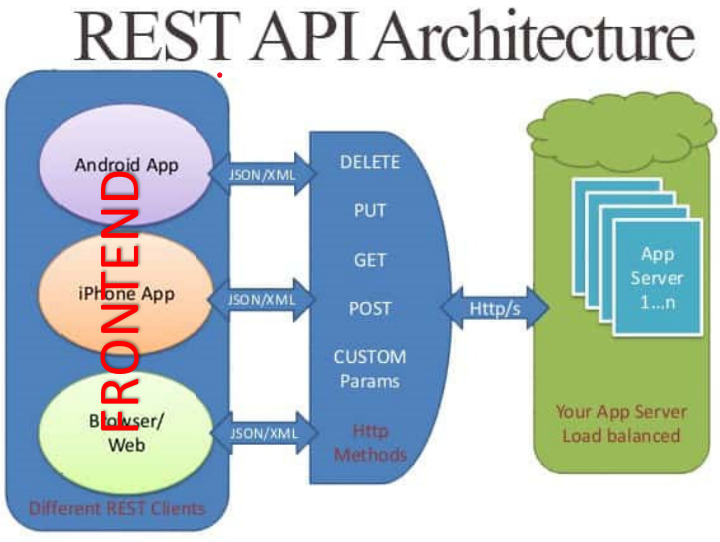
A Mockito használata nehezebb volt kicsivel, viszont ebből is sikerült sokat tanulnom, és egy hasznos skill-t szereztem vele. A Mockito-t tartalmazó osztályokat, nevükkel megjelölve külön java fileokba szerveztük ki, aszerint, hogy melyik osztályra vonatkoznak a tesztjeik.

Megismert technológia: @Mock, @InjectMocks

Végül a program elkészítésének utolsó fejezete a REST API-ok elkészítése, és a frontend elkezdése volt. Sajnos az utóbbira már nem jutott időnk a félév végén.

A REST végpontok elkészítését az alábbi elvek alapján végeztük el: (Szeretném részletezni, miközben bemutatom, hogy mit tanultam a technológiáról)

Elsőként elláttuk az osztályainkat @RestController annotációval, amely jelezte a Spring számára, hogy ez az osztály felelős a különböző request-ek kontrolálásért, és képes ezeket küldeni, majd az eredményeket feldolgozni. A @RestController annotáció a @RequestBody és a @Controller annotációk kombinációja, így az utóbbiak alkalmazásával azonos eredmény érhető el. Egy rest api felépítése az alábbi módon nézett ki:



Megismert technológia:

@PutMapping

Jelzi a Spring számára, hogy a megfelelő fejléccel (URL-ben megadott érték) beérkezett PUT request-eket a megjelölt függvény dolgozza fel. A PUT szerepe a REST API-ok világában az új elemek létrehozása, vagy esetlegesen a már létrehozott elemek frissítése.

@PostMapping

Jelzi a Spring számára, hogy a megfelelő fejléccel (URL-ben megadott érték) beérkezett PUT request-eket a megjelölt függvény dolgozza fel. A POST szerepe a REST API-ok között az új objektumok létrehozása http request-ek segítségével.

@DeleteMapping

Jelzi a Spring számára, hogy a megfelelő fejléccel (URL-ben megadott érték) beérkezett PUT request-eket a megjelölt függvény dolgozza fel. A DELETE szerepe pedig természetesen az egyes létező objektumok törlése.

@RequestBody

A @requestbody annotáció jelzi a Spring számára, hogy azon paraméterhez tartozó objektumot a request-ben találja meg. Persze ez POST és PUT request-ek esetén ideális.

@PathVariable

Segítségével a Spring tudni fogja, hogy az adott változót nem a RequestBody-ban hanem a Request URL-ben kell keresnie. Például Ha

@GetMapping(„/rentals/{id}”) az URL

akkor a @PathVariable UUID id nem lesz más, mint a megadott {id} például localhost:8080/rentals/2 esetén 2 lesz az UUID id változó értéke.

Felparaméterezés

Az egyes típusú request-ek számára mást jelenthet egy-egy paraméter megléte, hiánya. Így ezeket szeretném részletezni, hogy a program szempontjából milyen elvek voltak fontosak.

POST request

Itt csak a /kollekcio-t használtuk, ahol a metódus szerepe az volt, hogy a RequestBody-ban kapott objektumot a megadott kollekcióhoz adja.

GET request

Két fajta implementációt alkalmaztunk, az első a /kollekcio ahol a teljes kollekciót adta vissza a függvény, a második pedig a /kollekcio/{id} volt ahol a megadott id-hez tartozó elemmel tért vissza a függvény a kollekcióból.

DELETE request

Egy fajta delete-et használtunk, itt /kollekcio/{id} lett megvalósítva, a kollekcióból az id azonosítóval rendelkező objektum törlése volt a cél. Persze megvalósítható lenne a /kollekcio a teljes kollekció törlésére, de az iterálás /kollekcio/{id} is megoldja a problémát, habár több http request-et igényel.

A dokumentáció végén pedig szeretném részletezni egyes megtanult technológiák / annotációk funkcióit:

@OneToOne

Az annotáció segítségével megadtuk, hogy az adatbázisra leképezés módját, hogy a két entitás one to one viszonyban áll egymással. Egy entitáshoz egy entitás tartozik.

@OneToMany

Az annotáció segítségével megadtuk, hogy az adatbázisra leképezés módját, hogy a két entitás one to many viszonyban áll egymással. Egy entitáshoz több entitás tartozhat.

@ManyToMany

Az annotáció segítségével megadtuk, hogy az adatbázisra leképezés módját, hogy a két entitás many to many viszonyban áll egymással. Több entitáshoz több entitás tartozhat.

@Mock

Az objektumot tudjuk vele mock-olni, tehát egy dummy példányt létrehozni vele, ezzel is kevesebb hibalehetőséget adni a programnak teszteléskor vagy akár a nemkívánt funkciókat tudjuk vele kikerülni pl. a valós adatbáziskapcsolat mert akkor nem mindenhol lenne tesztelhető.

@SpringBootTest

Létrehozzuk vele a környezetet, amely a Spring Boot alkalmazás teszteléséhez szükséges

@InjectMocks

Létrehozhatunk vele egy objektumot, amelybe a @Mock-olt elemeket tudjuk injektálni.

@TestProfile

Teszt profilt hozhatunk létre, amely production-ben nem lesz aktív.

PostgreSQL

Adatbáziskezelő rendszer, relációs adatbázisokat kezel, application.ymal-ben lehet hozzá adatokat megadni (kapcsolatról)

@Service

Ezzel a taggel látjuk el azokat az elemeket, amelyek üzleti logikát valósítanak meg a Spring-ben.

@Entity

Az adatbázisban tárolandó entitásokat jelöljük meg ezzel a taggel.

GrantedAuthority

Hozzáférést reprezentál, hozzáférések menedzseléséhez használjuk

@NoArgsConstructor

Default konstruktor készít nekünk argumentumok nélkül.

@AllArgsConstructor

Konstruktort állít elő az összes argumentummal.

@Column

Egy oszlop különböző tulajdonságait tudjuk vele megadni, valamint jelezhetjük, hogy ez az adatbázisban egy sorban fog szerepelni.

Végül összefoglalás a tanultakról:

A félév egyik legizgalmasabb feladata volt számomra a témalaboratórium, habár a spring technológiáról hallottam már, és valamit próbálkoztam vele, érteni nem értettem még az. Viszont a félév során sikerült elsajátítani olyan technikákat, nem csak a springre vonatkozóan amelyek nagy értékkel bírnak, ilyen git workflow, a mockolás, és a megfelelő tesztelés, továbbá szereztem még tapasztalatot csapatmunkában is. Az első technológia amely megfogott a témalaboratórium során már a maven használata volt, habár a mavenrepository segítségével könnyebben tudtam már használni a végén. Sosem volt tapasztalatom még database relációk leírásában sem springes környezetben, habár adatvezérelt rendszerekből már tanultuk, így is izgalmas volt kipróbálni egy általunk írt környezetben. Az egyes spring annotációk pedig végre értülmeket nyerték számomra, sikerült megértenem őket, és úgy érzem hogy rengeteget gyorsítják a munkát és egy hatékony eszköztárat szereztem amely sok helyzetben ki tud majd segíteni. A Junit tesztek használata már ismerős volt számomra, azok nem jelentettek nagyobb kihívást, viszont a Mockito annál inkább, én a mockito tesztjeimet elsődlegesen nagyon primitív és kezdetlegesnek éreztem, viszont a RestController tesztelésére úgyérzem elég jól belejöttem a használatába, így ezt is egy sikerként könyvelem el végül. Az üzleti logika nem volt szerintem bonyolult, viszont nem is az volt számomra a tárgy lényege, hogy bonyolult függvényekkel valamilyen adatot varázsoljak elő, hanem hogy valamilyen technológiát amit érdekel megismerjek és elsajátítsam, hogy később akár tovább vigyem Önálló laboratóriumra, esetleg szakdolgozat témának , és ez egy kívánló lehetőség volt ennek a technológiának a tesztelésére amely elnyerte a tetszésemet. A legizgalmasabb része a témalaboratóriumnak számomra a REST apik tervezése és megírása volt , hiszen akkor lehetett látni hogy a programunk mire is képes, és hogyan is működik a backend összekapcsolása a frontenddel habár én ebből inkább a teszteléssel foglalkoztam, a mockito segítségével szimulálhattam az esetleges HTTP requestek beérkezését és az arra adott válaszokat is.

Zárószónak:

A témalaboratórium tárgy számomra elérte a célját és örülök hogy kipróbálhattam egy techonlógiát, megismerhettem az alapjait és mindezt nem csak egyedül de csapatban is tehettem, ezzel is kicsit felkészülve a real life szituációkra.

A spring mint keretrendszer hatalmas eszköztárával egy ideig mindenképpen a Go To lesz számomra, hiszen ennek elsajátítása még egy távoli cél, de az egyetem rengeteg lehetőséget ad ezen technológia megismerésére is.

A tesztelést illetően is sokat fejlődtem , nem csak Junit de Mock teszteket is tudok már írni amely segítségével tudom bővíteni a palettámat.

Végül még kommentelni is elkezdtem amely nem rossz, habár nem szoktam meg, és a Cleancode alapelvek miatt inkább úgy gondolom, hogy a program legyen magától érthetődő és többet mond a kód mint a szó.

A témalaboratórium számomra elérte célját ,örülök, hogy megismerhettem ezt a keretrendszert és a csapatomat is.

Mit tanultam ebben a félévben?

Széll Adrián Bence

Előszó

Számomra a technológiák már ismertek, mivel dolgoztam már velük éles helyzetben. A tárgy alatti új tapasztalat, hogy mar nem meglevő rendszerbe kellet dolgoznom, hanem egy teljesen új projektben, itt volt lehetőségem a környezeti beállításokkal játszani és a Testcontainers Framework-ot felconfigolni (ami mellesleg nagyon tetszik, mert így az integritás tesztek álomszerűen működnek Docker segítségével). Így is rengeteg dokumentációt olvastam miközben próbáltam a felmerülő problémákat megoldani pl. a Spring Security világában. Természetesen ugyanígy a Git verziókezelő által adott kihívások is sokat tanítottak, legfőképp meglevő tudásokat tettek helyre.

Git, mint általunk használt verzió kezelő

A félév során megtanultam a merge es a rebase közti különbséget, próbáltam konvenciót bevezetni a GitHub által adott issue-kal, bíztatni a társaim, hogy micro commitokkal dolgozzunk és legfőképp, hogy sose merge-eljünk csak rebase-eljünk. Megtanultam, hogy kell megoldani komolyabb conflict-okat, és hogy mennyire fejfájós tud lenni, ha a main egy branch-re merge-elődik (nem fordítva). Az elején sok conflict-ot kellet megoldanom, de később társaim is megtanulták ezt a folyamatot és a végére szép egyenes lett az idővonal ☺.

Spring Framework

Spring Framework egy hatalmas eszköztár, szerencsére jó dokumentációkkal. Segítségével gyorsan fel lehet építeni egy production grade web applikációt. Mérnöki megközelítést igényel a Spring Security rész, ahol az adatbázis schema felépítésnél már gondolni kell, hogy az API miket kér, de utána az authentikáció es az authorizáció már könnyedén kezelhető. A JPA egy nagyon könnyen kezelhető interface-t ad az adatbázis műveletekhez és rengeteg mindent elrejt a háttérben, ami nagyban megkönnyíti a perzisztens adatok kezelését. Másrészt a model osztályok @Entity-vel ellátva, számomra érthetőbben leírnak egy entitást, mint tablet és a táblák közti kapcsolat is könnyedén megadható dekorátorokkal. Itt ezen a ponton készítettem egy Dev Enviroment-et, ahol Docker-ben fut egy adatbázis szerver és az IntelliJ IDEA által könnyedén lehet figyelni real time mi történik rajta, ez sokat segített a megfelelő schema létrehozásában. Controller-eknél a customer controller-en belül a save endpoint-nál is találkoztam egy kihívással, mivel a Spring Security által adott UserDetails-ből származtattuk a User osztályunk, és ha csak a bejövő JSON-t akarjuk rábízni a Spring-re, hogy egy User objektumot várjon és deserializálja a JSON-t amit kap, akkor hiányzott neki 2 Time-stamp sorra az isAccountNonExpired, isCredentialsNonExpired (amiket a feladat megkönnyítése miatt felülírtam, hogy ezeket most nem kezeljük le), a framework az ősosztály értekéit is várta amiket nem talált, ezért létrehoztam először egy Custom deserializálót, amely nem szép megoldás, aztán pedig egy CreateCustomer objektumot, amely annyit ír le, hogy milyen JSON-t várunk, majd egy service-en keresztül adtam át a CustomerRepository save metódusának. Végső soron meg szeretném említeni a Builder és SuperBuilder annotációkat, amelyek

biztosítják a SOLID szabályai közül a Single Responsibility megvalósulását factory pattern segítségével.

Java Stream

A megvalósításaim során próbáltam mindent stream-ekkel megcsinálni, mivel van meg ott mit tanulni ezért ezek jó kihívások voltak, és törekedtem a legszebb megoldásokra, ahol tudtam.

Problémák megközelítése TDD módszerrel

A nyáron kezdett munkaviszonyom közben egyik első feladatom volt a Kent Beck: Test-Driven Development című könyvet elolvasni, így nem volt újdonság a módszer és volt szerencsém ezt a látásmódot komplexebb feladatoknál is alkalmazni. Ha jól használja az ember a könyv tanításait, akkor ritkán kell használni az Error Driven Development módszereit ☺.

Végszó

Voltak pontok, ahol a feladatok megoldásai helyett inkább a project egyben tartásával töltöttem sok időt. Végső soron élveztem ezt a tárgyat és jó volt a csapat.