**SZAKDOLGOZAT**

**Jurás Bence**

**Debrecen  
2019**

Debrecen Egyetem

Informatikai Kar

Java webalkalmazás fejlesztése

|  |  |
| --- | --- |
| Témavezető | Készítette |
| Dr. Jeszenszky Péter adjunktus | Jurás Bence  programtervező informatikus Bsc |

Debrecen  
2019

**Tartalomjegyzék**

[Köszönetnyilvánítás 4](#_Toc4997122)

[Bevezetés 5](#_Toc4997123)

[1. fejezet 7](#_Toc4997124)

[A felhasznált technológiák 7](#_Toc4997125)

[1.1 Spring Boot 7](#_Toc4997126)

[1.1.1. Történelme 7](#_Toc4997127)

[1.2 JPA 7](#_Toc4997128)

[1.3 Hibernate 8](#_Toc4997129)

# Köszönetnyilvánítás

Szeretnék köszönetet mondani témavezetőmnek Dr. Jeszenszky Péternek, amiért a tanított tárgyai által sikerült olyan szakmai alapot elsajátítanom amért, annak idején a debreceni egyetem informatika karát választottam.

Továbbá szeretnék köszönetet mondani Fekete Gyulának, amiért cégüknél végezhetem el szakmai gyakorlatomat valamint, hogy a projektmunkák által tovább mélyíthettem tudásomat.

Végül pedig páromnak, Jenei Alexandrának, amiért türelmes volt velem, az egyetemen töltött éveim alatt, valamint a projektmunkák, illetve a szakdolgozat írása alatt is türelemmel és figyelemmel kísérte végig munkámat.

# Bevezetés

A dolgozatom témájának kiválasztásnál azt tartottam legfontosabbnak, hogy egy olyan tudást szerezzek, amivel később munkába állhatok.

A java nyelvet választottam, amit az egyetemi oktatás során nagyon megkedveltem.

A dolgozatom egy online vásárlással foglalkozó oldal elkészítésére irányult. Elkészítésére célirányosan a modern, valamint az iparban is használt technológiákat választottam.  
A dolgozat első része e technológiák bemutatására fog irányulni.  
 A programmal sikerült minden olyan célt elérni, amit még az elején kitűztem magamnak.  
Főbb célom e technológiák megértése, és felhasználása közel valós környezetben.

Az objektum-relációs leképezés, a web api-k megértése, a hatékony tiszta kód megvalósítása mind céljaim voltak.

A program maga nem teljesített minden célt, amit még az elején kitűztem magamnak az idő hiánya miatt, ám a főbb funkciók elkészültek.

Az oldalra regisztrálást követően tudunk termékeket megtekinteni, termékekre keresni.   
A keresést tudjuk szűkíteni, valamint a kereséshez egy olyan filterező megoldást képzeltem el, amiben attribútumokat, kategóriákat adhatunk át.

Maguk a kategóriák Many-To-Many kapcsolatban állnak a termékekkel.

A kategóriákon belül vannak további kategóriák, amik közül a vásárló tud böngészni.

Az attribútumok jelképezik a termékek tulajdonságait, melyekkel a vásárló ki tudja választani a számára legideálisabb terméket.   
A kategóriák és az attribútumok együttesen írják le a termékeket.  
A termékekre lehet licitálni, illetve ha fix áras, akkor azt megvásárolni.

A termékekre tudunk kommenteket küldeni, továbbá tudunk képeket is csatolni hozzájuk.

Elkészült még egy kezdetleges biztonsági rendszer amely [OAuth 2](https://oauth.net/2/)

-őt használ.

Két faktoros bejelentkezés elkészítésére sajnos már nem maradt idő.

A backendhez Spring Boot 1.5x-et használtam, míg a frontendhez Angular 6-ot.

Az adatbázishoz Mysql-t használtam.

# 1. fejezet

## A felhasznált technológiák

Ebben a fejezetben szeretnék kitérni a fejlesztéshez felhasznált technológiákra.

### Spring Boot

A Springboot király dolog.

#### Történelme

Az igény egy könnyebben használható Spring Framework-re 2012 októberében merült fel Mike Youngstrom‏ által.  
A könnyítésnek lehetőségét abban látta, hogy amennyire csak lehet, absztrahálni kell a Spring keretrendszert. Szerinte a servlet konténer használata az új fejlesztők számára egy olyan tanulási görbét von maga után, amit nem biztos, hogy szükséges volna rájuk helyezni.

A tanulási görbe alatt értendő pl. a web.xml és egyéb servlet konfiguráció, .war mappa szerkezet, konténer implementációk (pl. portok, szálkészlet,..), komplex osztálybetöltő hierarchia stb.

A felsoroltak nem rendelkeznek egy egységes konfigurációval, helyette, egy nem egységes, inkonzisztens módon adhatók meg.

Szerinte egyszerűsíthető volna a Spring keretrendszer, ha egybeágyaznák és egységesítenék a gyakori web konténereket egy közös Spring konténerben.

#### Kiindulás

1. ábra Az alap mappaszerkezet a Spring Intializer használta után

A projekt alapját előállíthatjuk a Spring Intializer segítségével.  
A projektemben Maven-t használtam, így itt is azt fogom körül járni.

A Spring Boot, a Spring által elérhető nagyobb funkcionalitásokat néhány függőségbe gyűjtötte. Ezek hozzáadását követően sok dolgunk már nem akad.

Néhány főbb függőség:

|  |  |
| --- | --- |
| spring-boot-starter-parent | Szülő pom ami függőségek és a bővítmények kezelését szolgáltatja. |
| spring-boot-starter-web | Web fejlesztéshez. Tartalmazza a RESTful architektúrát, továbbá a Spring MVC architektúrát. Tomcat-et használ, mint beépített server. |
| spring-boot-starter-test | Teszteléshez használható jar fájlokat tartalmazza, úgy, mint a JUnit, Hamcrest, Mocikto. |
| spring-boot-starter-data-jpa | Spring Data JPA, Hibernate |

Az Intializer által előállított projektben, létrejön az src/main/java/..package/ mappában a main metódus, amin @SpringBootApplication annotáció szerepel.  
Ez tulajdonképpen három nagyobb annotációt olvaszt magába:

|  |  |
| --- | --- |
| @Configuration | Az osztályon, amelyiken alkalmazzák, ott ez az annotáció ezt az osztályt a Spring Java alapú konfigurációs osztállyá nevezi ki. |
| @ComponentScan | Komponens keresést indít el, így bármelyik web kontroller vagy bármelyik másik komponens megtalálásra kerül. |
| @EnableAutoConfiguration | Itt a Spring Boot egyik „varázslata”, ez az automata konfigurációt teszi lehetővé. Segítségével nem kell több oldalnyi konfigurációs kódot írni. |

#### Függőségek befecskendezése

#### @Autowired: A függőségek befecskendezéséhez használjuk. Azokat az osztályokat, amiket ellátunk olyan annotációkkal, amelyeket a @ComponentScan megtalálhat, azokat itt befecskendezhetjük. Maga a fecskendezés három módon történhet meg: Mezőn, konstruktoron, és getter függvényen.

#### Példa: public class Osztályom {     //1. lehetőség    @Autowired    private Komponens komponens;     /2. lehetőség    @Autowired    public Osztályom(Komponens komponens)   {    this.komponens = komponens;   } //3, lehetőség @Autowired public Komponens GetKomponens() {    return komponens; } }

#### Főbb szkópok

Bab szkópok segítségével adhatjuk meg az osztályok példányosításának módját.

A Spring IOC konténert használhatjuk a bab szkópok általi példányosításához.

Több fajta létezik, úgy, mint a singleton, prototype, request, session és global session.

###### 1.1.4.1 Singleton

Amikor egy bab singleton akkor pusztán csak egy példánya fog használatban lenni. Bármikor, amikor kérés irányul a Spring IOC konténer felé egy singleton bab eléréséhez, akkor mindig ugyanaz a példány kerül átadásra. Ezért aztán ha módosításra kerül egy singleton bab, akkor bármilyen új hivatkozás erre a babra, az új változtatásokkal ellátott babot kapja, hiszen csupán egy példánya létezik.

Példa:

@Bean  
@Scope ("singleton")  
public class SingletonClass  
{  
...  
}

###### 1.1.4.2 Prototype

A prototípus szkóppal ellátott babok minden új kérés esetén új példányt hoz létre.

Singleton szkópot állapot nélküli babokra teszünk, míg az állapottal rendelkezőkre prototípust teszünk. Úgy használhatjuk, hogy a @Scope annotációnak a „prototype” karakterláncot adjuk át.

###### 1.1.4.2 Request

A request szkóp új babot csinál minden egyes HTTP kérésnél egy session-ön belül. Még akkor is, ha 2 lekérés érkezik egy session-ön belül szimultán módon. Minden kérés egy eltérő memória címet hivatkozik. Minden kérés függetlenül kezelt, és nincsenek egymásra hatással.

###### 1.1.4.3 Session

Ebben az esetben egy HTTP kérésen belül 2 ugyanolyan típusú babra való hivatkozás a singleton-ban megismert viselkedést fogja tanúsítani, azaz nem fog új helyet lefoglalni az új hivatkozásra a babnak a kérésen belül, hanem a már meglévőt fogja használni.

### Spring Data

Enterprise applikációk az adatfeldolgozást a lehető legegyszerűbben szeretnék megvalósítani. Az egyik legelőrehaladottabb megoldást Spring Data Project biztosítja. Képes automatikusan generálni repozitorikat az általunk átadott funkcionális interfészből.

Az absztrakció alapja a Repository interfész. Ebből származnak le olyan interfészek, mint pl. JpaRepository, CrudRepository, MongoRepository.

##### Annotáló konfiguráció

Használni úgy, tudjuk, hogy a repozitorinkat injektáljuk. Az injektálás történhet pl. mezőn annotálva, vagy konstruktoron annotálva.

Pl.

@Autowired  
private PersonRepository personRepository;

Illetve:

@Autowired  
public Osztályom (private PersonRepository personRepository)  
{  
 this.personRepository = personRepository;  
}

##### Lekérdező metódusok

Standard, illetve összetett lekérdező műveletek végrehajtására repozitorikat használunk. Egy interfészre van csak szükség, ami fontos, hogy kiterjesszen egy al interfészét a Repository interfész gyermek interfészei közül, vagy magát a Repository interfészt.

Egy példa egy személy interfész létrehozására:  
interface PersonRepository extends Repository <Person, Long> { … }

A generikus paraméterei egy személy entitás, míg a másik a személy entitás egyedi azonosítójának a típusa.

Ezt követően egy lekérdező műveletet a következőképpen készíthetünk el:

interface PersonRepository extends Repository <Person, Long> {

List <Person> findByLastname (String lastname);

 }

Magát a lekérdezést a Spring Data absztrakciói mögött elrejtve oldja meg.

A függvényünk egy személyekből álló listát fog vissza adni, leszűrve őket a vezetéknevükre.

Ehhez hasonló findyBy műveleteket könnyen létre tudunk hozni. Pusztán figyelni kell, hogy mi az a paraméter, amire szűrni szeretnénk.

Pl. életkorra a következő képpen nézne ki:  
List <Person> findByAge (int age);

A leszűrt elemekre lehet használni a distinct műveletet is:  
List <Person> findDistinctPeopleByLastname (String lastname);

Rendezni is lehet a következőképpen:  
 List <Person> findByLastnameOrderByFirstnameAsc (String lastname);  
A névben szerepelni kell ilyenkor az OrderBy-nak, aztán a rendező property neve majd, hogy növekvő, vagy csökkenő sorrendben szeretnénk látni a leszűrt elemeket.

Előfordulhat, hogy 1-nél több propertire szeretnénk szűrni, ilyenkor egy összekapcsoló logikai névre van szükség: Or/And.

Pl.:

List <Person> findByAgeOrHeight (int age, int height);

##### Lekérdezés készítés

Korábban már írtam a lekérdezések használatáról, illetve megadásáról, ám most jobban belemegyek.

Felépítése egy ilyen query-nek a következő módon alakul:  
{visszatérési típus} findyBy{Entitás valamely propertije}[and | or [{entitás valamely propertije}[and | or]..] ]([ [típus property név],[típus property név]..]);

Fontos, hogy a parméter lista elemszáma meg kell, egyezzen a query-ben átadott property nevek számával és a típusainak is meg kell egyeznie.

Felhasználható kulcsszavak:  
And, Or, Is, Equals, Between, LessThan, LessThanEqual, GreaterThan, GreaterThanEqual, After, Before, IsNull, IsNotNull, NotNull, Like, NotLike, StartingWith, EndingWith, Containing, OrderBy, Not, In, NotIn, True, False, IgnoreCase.

Ezekre néhány példa:

|  |  |
| --- | --- |
| GreaterThan | findByAgeGreaterThan |
| Before | findByStartDateBefore |
| Containing | findByFirstnameContaining |
| Not | findByLastnameNot |
| NotIn | findByAgeNotIn (Collection <Age> ages) |
| True | findByActiveTrue () |

##### @Query használata

Amennyiben saját query-ket szeretnénk írni azt, megtehetjük a @Query annotáció használatával. Itt a függvényünk felett egy JPQL lekérdezést adhatunk meg.

Példa:

  @Query ("select u from User u where u.emailAddress = ?1")  
  User findByEmailAddress (String emailAddress);

Ebben a példában a sztringben megjelenő „u” egy a User entitás egy példányát képviseli. Az „u” segítségéével tudunk hivatkozni a benne található property egyikére.

Alkalmazható a Like kifejezés is a következő módon:  
@Query ("select u from User u where u.firstname like %?1")  
  List<User> findByFirstnameEndsWith(String firstname);

A % karaktert a Spring Data kiszedve, egy valid JPQL lekérdezést hoz létre.

A lekérdezésben szerepeltetett paramétereket eddig csak pozíció szerint adtuk át, ám lehetséges ezt név szerint kötni is.  
Példa:  
@Query ("select u from User u where u.firstname = :firstname or u.lastname = :lastname")  
 User findByLastnameOrFirstname(@Param("lastname") String lastname, @Param("firstname") String firstname);

### Hibernate

A Hibernate a JPA-nak az egyik legnépszerűbb megvalósítása.  
A Hibernate egy objektum-relációs le kepézést megvalósító programkönyvtár, vagy ORM.

##### ORM

Hibernate segítséget nyújt az alkalmazásunknak abban, hogy az adatokat perzisztállyuk. Maga a perzisztencia nem más, mint az a vágy, hogy az alkalmazás által kezelt adatok túléljék az alkalmazást. Tehát azt szeretnénk, hogy néhány objektumaink állapotai éljenek a JVM hatáskörén kívül is, így ez az állapot később még visszakapható.

A táblák leképezése POJO osztályokra xml vagy annotációs konfigurációval implementálható.  
Képes kezelni az egy az egyhez, több a többhöz relációkat az osztályok között.  
Támogatja az egyedi érték típusok le kepézését is. Továbbá lehetőség van a következőkre:

* Felülírni az az alapértelemzett SQL típust, amikor egy oszlopot képezünk le egy propertire.
* Leképezni Java Enumokat akárcsak a szokásos típusokat
* Az önálló értéket leképezni több oszlopra

##### HQL

Hibernate szolgáltat egy SQL szerű nyelvet, a Hibernate Query Language-et.

Segítségével SQL szerű lekérdezéseket írhatunk a Hibernate adat objektumaira.

##### Annotációk

A javax.persistance csomagot használva érhetjük el őket.

Az annotációk segítségével tudjuk konfigurálni az entitásokat, és a kapcsolatot közöttük. Helyette használható az xml konfiguráció.

Néhány annotáció:

|  |  |
| --- | --- |
| @Entity | Az entitás babjainkat ezzel az annotációval kell, elássuk. |
| @Table (name=táblanév) | Az átadott táblanév alapján köti az adatbázisban megtalálható táblát ezzel az entitással. |
| @Id | Jelezzük, hogy a property az egy egyedi azonosító. |
| @Column (name=oszlopnév) | Az átadott oszlopnév alapján köti az adatbázisban megtalálható tábla oszlopát ezen entitás propertijén. |
| @Lob | Nagyméretű objektumok használata esetén. |
| @OneToOne | Egy az egyhez relációhoz használandó. |
| @OneToMany | Egy az többhöz relációhoz használandó. |
| @ManyToOne | Több az egyhez relációhoz használandó. |
| @ManyToMany | Több az többhöz relációhoz használandó. |

### Spring Rest API

### Spring Security

<https://dzone.com/articles/magic-of-spring-data>

https://docs.spring.io/spring-data/jpa/docs/current/reference/html/

https://github.com/spring-projects/spring-framework/issues/14521

<https://www.quickprogrammingtips.com/spring-boot/history-of-spring-framework-and-spring-boot.html>

<https://www.tutorialspoint.com/spring_boot/>

<https://docs.spring.io/spring/docs/3.0.0.M3/reference/html/ch04s04.html>

<https://www.baeldung.com/spring-bean-scopes>

<https://docs.oracle.com/cd/E23095_01/Platform.93/ATGProgGuide/html/s0205requestscope01.html>

<http://hibernate.org/orm/what-is-an-orm/>

<https://en.wikipedia.org/wiki/Hibernate_(framework)>

http://www.techferry.com/articles/hibernate-jpa-annotations.html