**Témalabor 1. dokumentáció**

# Feladat: Egyszerű Instagram-szerű weboldal készítése Java Spring technológiával

# Készítették: Kálmán Bence (B06FMG)

# Kovács Dániel (E0UTKP)

# Dátum: 2016.11.22.

# Ezen dokumentáció célja az, hogy leírjuk a tapasztalatainkat, és lényeges ismereteinket amit eddig szereztünk a Java Spring Framework-ről, illetve azon eszközökről, amit a projekt megválósításához felhasználni fogunk., valamint hogy tájékoztatást adjunk, hogy mit csináltunk.

# A projekt első szakasza

# Az első közös konzultáción azt a célt jelöltük ki az első hónapra, hogy mindenki ismerkedjen meg azokkal a technológiákkal, vagy programokkal, amelyeket nem ismert, de szükséges a feladat végrehajtásához.

# Eszközök, technológiák:

# IntelliJ IDEA 2016.2.4.

# Gradle

# Java Spring Framework

# Thymeleaf

# Spring Boot

# Cloudinary

# GitHub

# Rövid ismertető és tapasztalatok az eszközökről, technológiákról:

# IntelliJ IDEA 2016.2.4.:

# Az IntelliJ IDEA egy Java IDE a JetBrains-től. Támogatja a Java-n kívűl a (csak a legfontosabbakat, és elterjedtebbeket említve) HTML-t, XHTML-t, CSS-t, JavaScript-et, Python-t, SQL-t, XML-t vagy különböző pluginokon keresztül például a PHP-t is. Támogatott technológiák és framework-ök között található többek között a JSP, EJB, Spring, és az Android is. A Git integrálva lett az IntelliJ-be, amely megkönnyítí a fejlesztőcsapat dolgát a verziókövetés terén (,ami elengedhetetlen a csapatmunkához), hiszen nem kell külön batch vagy grafikus felülettel rendelkező git-et használni a fejlesztés során a programozóknak, hanem minden verziókövető funkció elérhető az IDEA-ból.

# Eddig java projektek fejlesztésére Eclipse-t használtunk fejlesztőkörnyezetnek, de a témalabor során IntelliJ-re váltunk. Szerintünk az IntelliJ szélesebb körben támogatja a fejlesztő munkáját. Külön kiemelnénk, hogy a debug funkció nagyon fejlesztő barát, segítségével a hibák sokkal gyorsabban megtalálhatóak, mint az Eclipse-s debuggerrel.

# Gradle:

A Gradle egy általános célú build management rendszer ami támogatja a függőségek és más könyvtárak automatikus letöltését és konfigurálását.

Egy Gradle-t használó projectben mindig megtalálható egy build.gradle fájl ami leírja a felépítését. Ennek a fájlnak a nyelve Domain Specific Language (DSL) és a project főkönyvtárában található. Ezek mellet több Gradle plug-in is létezik a fejlesztés megkönnyítése érdekében, például a Java plug-in ami lehetővé teszi Java forráskód fordítását, unit tesztek futtatását és futtatható JAR fájl készítését vagy a Spring Boot plug-in amit mi is használni fogunk. Ezeket az alább látható formában kell megadni.

apply plugin: ‘plugin név’

A repositories{} szegmensbe kell megadni azokat a távoli repositorikat ahol a Gardle a függőségeket keresni fogja. A Gardle támogatja a Maven és Ivy repositorikat is.

repositories {  
 mavenCentral()  
}

A dependencies{} szegmensben hozzá adhatunk JAR fájlokat, könyvtárakat és más projecteket is az alkalmazásunk build path-jéhez. Ennek hatására a project fordításakor a Gradle letölti a megadott forrásokat beleértve azok függőségeit is.

dependencies {  
 compile("org.springframework.boot:spring-boot-starter-thymeleaf")  
 testCompile("org.springframework.boot:spring-boot-starter-test")  
}

A project kezdetekor a Maven használatán is gondolkoztunk. A Maven nagyon hasonló, működésében csak kisseb eltérések vannak (pl. a Maven nyelve az XML). Végülis a Gradle mellett döntöttünk, mert többeknek is van múltbeli tapasztalat vele, illetve más tárgy keretében is ezt használjuk ebben a félévben.

# Java Spring Framework:

# A Spring egy nyílt forráskodú rendszer, ami létezik Java és .Net platformon is. Mi a projektben Java platformon fogjuk használni. A Spring rétegelt kialakítású, ami azt jelenti, hogy az egyes feladatkörök külön modulba tartoznak. A keretrendszer nem kényszeríti ki egyetlen egy pattern, vagy szabvány követését se, ezeket a fejlesztőre bízza.

# A Spring Framework fő feladata, hogy menedzselje az objektumokat, és ha egy objektumnak szüksége van valamire, amit egy másik objektum birtokol, akkor szól a keretrendszernek, a keretrendszer pedig a kérést delegálja az objektumok felé.

# A Spring magja az Inversion of Control (IoC) konténer, ami konfigurálja az alkalmazás kontroljait és Java objektumait, illetve menedzseli azokat életciklusuk során.

# IoC felelőssége:

# Objektumok létrehozása

# Objektumok értékének másolása

# Inicializáló metódusok hívása

# Objektumok konfigurálása

# Modulok között az objketumok megosztása

# A konténer által létrehozott objektumok hívhatnak további menedzselt objektumokat vagy beaneket . A konténer konfigurálása egy XML betöltésével és értelmezésével történik. Ebben az XML fájlban vannak a beanek definíciói, ez alapján kell a beaneket létrehozni.

# Az objektumok függőség injektálással (röviden: DI) érhetőek el. A DI konfigurálható XML-ben, de sokkal egyszerűbb Java osztálybeli annotációval megoldani. A @Autowired annotációval kell megjelölni az injektálási pontokat. A használni kívánt osztályt is annotálni kell @Component-tel. Példa:

@Component  
public class CommandService {  
 @Autowired  
 private SettinsService settingsService;  
.......  
}

# Bean az az objektum, amit a keretrendszer a DI-n kereszül menedzsel. A beanek konfigurálhatók kódból és XML fájlból is.

# A DI-nél fontos fogalmak a függő, függőség és injektáló fogalma. A függő, akinek szüksége van valamilyen szolgáltatásra vagy információra, amit egy másik objektum birtokol. A függőség olyan objektum, akitől a függő megkapja a kért szolgáltatást vagy információt. Egy függőnek több függősége is lehet, ami fordítva is igaz. A függőségek gráfként ábrázolhatóak.

# Az injektáló az, aki biztosítja a kapcsolatot a függő és függőségei között, illetve ő menedzseli az objektumokat is. Ha egy objektumnak szüksége van valamire, amit egy másik objektum birtokol, akkor szól a keretrendszernek, a keretrendszer pedig a kérést delegálja az objektumok felé.

# Thymeleaf:

# A Thymeleaf egy Java könyvtár ami transzformációkat hajt végre a megadott template fájlokon annak érdekben hogy az alkalmazás által előállított adatokat és szöveget megjelenítse. Fő célja ,hogy egy egyszerű módot biztosítson ezen templatek elkésztésére. Ahhoz, hogy ezt elérje XML atribútumokhoz hasonlóan jelöli ki a végre hajtandó logikát ahelyett, hogy azt a templetbe bele írnánk. Ezzel sokkal átláthatóbb a működés bár az elején kicsit bonyolultnak tűnt és megérteni sem volt könnyű. Főleg XHTML és HTML5 webalkalmazásokhoz használatos de XML fájl feldolgozására is képes. A Thymeleaf támogatja a Spring MVC-t ezért a Spring webalkalmazás view rétegének elkészítéséhez ez az egyik legegyszerűbb eszköz.(A Spring Boot-os tutorialokban alapértelmezetten a Thymeleaf-et használja)

# Spring Boot:

# A célja az, hogy leegyszerűsítse a Spring alapú alkalmazásfejlesztést. A legtöbb Spring Boot applikációnak nagyon kicsi Spring konfigurációra van szüksége. Előre elkészített Gradle konfigurációt tartalmaz. Automatikus konfigurálja a Springet, ha lehetséges. Nagyon gyakori problémákhoz nyújt megoldásokat előre elkészített template-kel.

# A Spring Framework teljes körű megismerése nagyon sok időt venne igénybe. A Spring Boot egy egyszerű eszköz arra, hogy a Spring Framework képességeit kihasználjuk annak alaposabb ismerete nélkül. A hivatalos oldalon rengeteg tutorial található amelyekbe jól elmagyarázzák a működést is.

# MVC:

# Az MVC (model-view-controller) egy tervezési minta. A lényege, hogy elválasztja a modelt és a nézetet, és ezeket a kontrollerrel köti össze. Mivel a view függ a modeltől, és nem fordítva, ezért könnyű új nézetet adni az alkalmazáshoz. Ha a felhasználó a UI-on valami adatmódosítást végez, akkor azt a kontroller átvezeti a modelbe. A Spring MVC view rétege a Thymeleaf, amely HTML template-ket ad Java alkalmazásokhoz.

# Cloudinary:

# A Cloudinary egy felhő alapú szolgáltatás. Alapvetően képfeltöltésre való, de lehet videókat is. A projektben mi a kép feltöltő funkcióját fogjuk használni. Támogatja az alapvető kép típusokat. Biztosít egy API-t a képek feltöltéséhez, tárolásához, manipulálásához. Az API-jához a honlapjukon van egy dokumentáció, amiből meg lehet tanulni az API használatát.

# A következő Java kóddal lehet képet feltölteni a felhőbe:

import com.cloudinary.Cloudinary;  
Cloudinary cloudinary = new Cloudinary();  
cloudinary.upload(fileRef, ObjectUtils.emptyMap());

# Egy a helyi gépen lévő image.jpg képet például így lehet feltölteni:

File file=new File(„my\_image.jpg”);  
Map uploadResult= cloudinary.uploader().upload(file, ObjectUtils.emptyMap());

# A Cloudinary függvény könyvtárat Gradle függőségként kell megadni.

# GitHub:

# A GitHub egy verziókezelő program, ami elengedhetetlen lesz a csapatban való fejlesztéshez. A GitHub segítségével egy program verzióit lehet követni. Lehetőséget biztosít a fejlesztő csapat tagjainak, hogy követni tudják ki mit módosított a kódon. A régebbi verziók visszaállíthatóak.

# A lényege az, ha egy programozó elkezd fejleszteni, akkor szinkronizálja a projektét, vagyis töltse le a legfrissebb verzióját a projektnek, és ha bármi módosítást csinál, amit szeretne, hogy a csapat más tagjainál is látszódjon, akkor commit-oljon. A commit-oknak van nevük, és egy leírásuk, ami nagy segítség abban, hogy meg tudjuk milyen változások történtek.

# A Git az IntelliJ-be is integrálva van, ami lehetővé teszi a fejlesztőknek, hogy ne kelljen a Git batch vagy grafikus felületét használni. Az IntelliJ-ben kényelmes helyen megtalálhatóak a Git szolgáltatásai.

# Amit eddig csináltunk:

# Megismerkedtünk az IntelliJ IDEA-val. Már elég sokat fejlesztettünk benne ilyen rövid idő alatt nem csak a témalabor kapcsán, hanem egyéb más egyetemi java projektekben (android fejlesztésre is ezt használjuk).

# A Gradle-el is megismerkedtünk. Android fejlesztésre elég sokat kell használni (pl. külső könyvtárak használata), így már szereztünk tapasztalatot benne ebben a félévben.

# A Java Spring Framework kapcsán megcsináltunk három tutorial-t:

# egy egyszerű helloword weboldal, ami a http get kéréseit tudja kezelni

# a http get és post kéréseit is kezelő oldal:

# fájl feltöltő alkalmazás (mi ezt nem fogjuk használni)

# Igyekeztünk minden dolognak utána olvasni. Egy Spring Web MVC-ről szakdolgozatot is elolvastunk, a konzulesünktől kapott Spring-ről szóló diasort áttanulmányoztunk, valamint több más internetes forrásból tájékozódtunk.

# A csapattal tartottunk közös konzultációt is. Itt tapasztalatot cseréltünk, segítettünk egymásnak a környezeteket fel konfigurálni.

# A projekt második szakasza

# A félév elején tartott közös konzultáción a projekt második szakaszának a végére azt a célt jelöltük ki, hogy legyen készen már egy olyan weblap, amin keresztül lehet képeket feltölteni, és ezeket a képeket ki lehessen listázni.

# Kezdtünk egy Gradle projektet, amibe készítettünk egy GreetingController java osztályt, és egy index.html fájlt. A GreetingController kezeli az index.html-re érkező http get és post kéréseket. A get kérést az index nevű metódus végzi, amit a @GetMapping(’’/index’’) annotációval látttunk el. Az oldalra érkező get kérés esetén nem történik semmi csak visszadjuk a kezdőoldalunk nevét (az indexet), tehát maradunk az index.html oldalon.

# A post kérés esetén elvégezzük a kép feltöltést. Ezt az uploadingSubmit metódus végzi, amit egy @PostMapping(’’/index’’) annotációval láttunk el. A képeket a Cloudinary nevű felhő alapú tárhelyre töltöttük fel. A Cloudinary biztosít egy API-t, amivel a képek feltöltése viszonylag könnyen megtehető. Ehhez szükség van egy Cloudinary nevű mezőre, amely egy kapcsolatot reprezentál a cloudinary felhővel. Ennek az objektumnak meg kell adni a tárhelyünk nevét, api key-t és a api secret-t, amely információkhoz a cloudinary.com-ra való bejelentkezés után tudhatunk meg. Az api key és az api sercet azért szükséges, hogy kapcsolódni tudjunk a Cloudinary API-hoz, valamint hogy beregisztráljuk a kéréseinket a tárhely irányába. A feltöltött képnek adni kell egy egyedi nevet. Mivel a felhasználó nem biztos, hogy egyedi nevet ad meg neki, ezért a névütközést elkerülve a felhasználó által megadott névhez hozzáfűzzük a feltöltés dátumát és a feltöltés idejét másodperc pontossággal.

# A feltöltésnél a klasszikus módszernél maradtunk. Ki kell tallózni a képet, amit szeretnék feltölteni, annak egy beviteli mezőben nevet lehet adni, és egy submit gombbal el lehet végezni a feltöltést. Megjegyeznénk, hogy volt egy másik megvalósításunk is, amely a Cloudinary API egy nagyon elegáns képfeltöltési lehetőségét valósította meg. Ennek a módszernek a neve az, hogy upload widget, és az a lényege, hogy a lapon volt egy téglalap alakú rész ahova, be lehet húzni tetszőleges mennyiségű képet, és az azonnal automatikusan feltöltődött a felhőbe. Ez nagyon elegáns, felhasználóbarát, és gyors megoldás lehetett volna, amely kiküszöbölte volna azt a problémát, hogy mi van ha a felhasználó nemcsak egy-két képet akar felölteni, mivel így a rengeteg tallózást, és a submit gombra való kattintást megspórolta volna. A megoldás problémája alapvetően az volt, hogy így a képek nevének a megadását nem tudtuk megoldani, hiszen ha behúztuk a képeket a feltöltés automatikusan megindult. Valamint az sem jött össze, hogy a képekről referenciát szerezzünk, így ezt az upload widget-t lényegében kipróbáltuk, de maradtunk az eredeti tervnél. Magával az upload widge-t a honlapba illeszteni elég sok időbe telt, mivel híján voltunk a css és javascript ismereteknek. Nem tudtuk egész pontosan, hogy a cloudinary dokumentációban található css és javascript fájlokat hova kell tenni, és azokat, hogyan kell importálni, hogy használhatóak legyenek. A cloudinary dokumentációjának tanulmányozása, és jó pár próbálkozást után azonban sikerült megoldani a problémát, és megjelent az upload widge-t, ami első próbálzásra is már jól működött. Az eredmény az alábbi képen látható:

# 

# Bár végül az upload widgetet nem használjuk érdekes volt látni, hogy a cloudinary API milyen széleskörben támogatja a fejlesztő munkáját, és hogy nagyon bonyolultnak tűnő dolgokat egy kis css, html és javascript ismerettel milyen gyorsan és szépen meg lehet oldani az API-jukkal. Megjegyeznénk, hogy az upload widget csak egy a sok lehetőség közül, amit a cloudinary API biztosít.

# Miután már tudtunk képet feltölteni elkezdtük megvalósítani a képek kilistázását. Ehhez el kell tárolni a képek ID-ját, és nevét egy adatbázisban, hogy a cloudinary felhtőtől az ID alapján el tudjuk kérni a megfelelő képeket. Ehhez létrehoztunk egy Picture java osztályt, ami egy POJO osztály, és lényegében ennek az osztálynak az objektumait sorosítjuk. Ezt az osztályt @Entity annotációval láttuk el. Van egy ID és egy name mezője, utóbbi az adatbázis kulcsa, ezért @Id annotációval láttuk el.