Rendszertervezés laboratórium 2.

Java backend labor (AUT1)

Felkészülés a laborra

Beugró nincs, de felkészülés hiányában nem fogod tudni kielégítő eredménnyel teljesíteni a labort. Az „Alkalmazásfejlesztési környezetek” tantárgy diáit és példaprogramjait nézd át, különösen ezeket:

Előadások:

102 Spring DI és MVC

103 JPA és tranzakciók

104 Spring Boot és REST

105 JPA további lehetőségei és Spring Data

Példaprogramok:

102 Spring DI példák

103 JPA Spring MVC alatt

104 Spring Boot és REST példák

Jelenlét

A laboratóriumon szükséges a részvétel, ezért megjelenés nélkül, csupán emailben elküldött jegyzőkönyvet nem áll módunkban elfogadni. A feladatok teljesíthetők előzetesen otthon, de a bemutatáshoz meg kell jelenni a foglalkozáson, és a vezetett rész után lehet a kész megoldást bemutatni. Pótolni a beosztástól eltérő időben is lehetséges, ha a laborvezető emailben vagy a labor elején szóban engedélyezi. Az ilyen részvétel azonban pótlásnak minősül, tehát ezután a pótlási héten másik labor pótlására már nem lesz lehetőség.

Értékelés

A laborra kapott jegyet a sorban elkészített feladatok száma adja. (Feladatot átugrani nem lehet.) A beadásnak két módja van. Az első a vezetett bevezető utáni személyes bemutatás a laborvezetőnek a labor végéig. Ebben az esetben nem szükséges jegyzőkönyvet készíteni. A másik lehetőség a részvétel után 7 napon belül jegyzőkönyv beküldésével lehet. A jegyzőkönyvnek tartalmaznia kell a hallgató nevét, Neptun-kódját és minden feladathoz a megvalósítás kódját, illetve egy működést szemléltető screenshotot. A jegyzőkönyvet **PDF-formátumban** kell elküldeni a kovesdan.gabor@aut.bme.hu emailre. Más formátumban történő küldés esetén -1 jegy levonás jár.

A labor témája

A labor során egy apróhirdetési portál backendjét készítjük el, amely REST API-n képes fogadni a kéréseket webalkalmazától, mobil apptól vagy akár vastagkliens-alkalmazástól. Kliensalkalmazást azonban most nem készítünk, csupán a Postman programmal teszteljük a backendet.

A feladatok

0. feldadat (vezetett)

- Töltsük le, és csomagoljuk ki a projekt vázát a tantárgy weboldaláról.
- Indítsuk el az IntelliJ IDEA Community Editiont.
- Importáljuk a projektvázat Maven projektjént.
- A projektváz egy Spring Boot alkalmazás, amely induláskor beágyazott HSQL relációs adatbázist indít a memóriában. Az alkalmazásban konfigurálva van a JPA, és van egy Note entitásunk, valamint REST endpointok, hogy szöveges feljegyzéseket vigyünk fel, és tároljunk el az adatbázisban. Ezek majd később törölhetők, egyelőre tanulási célokat szolgálnak, és a későbbi feladathoz szolgálnak támpontul.
- Indítsuk el az alkalmazást!
- Indítsuk el a Postmant a start menüből! A laborvezetővel együtt vizsgáljuk meg a REST elvű kommunikáció működését!

A következő feladatok már önállóak. Használhatod a példakódot, illetve az ALF tárgy előadásdiáit és példaprogramjait is. **A hallgatótársaidtól történő közvetlen másolás plágiumnak számít, de a diákról, internetes fórumról másolhatsz, és adaptálhatsz kódot!**

1. feladat

- Készítsd el a hirdetések kezeléséhez az Ad entitást! Tartalmazza az alábbi adatokat:
  - Generált azonosító
  - Cím
  - Leírás
  - Ár (int)
  - Létrehozás ideje – ez a backendben töltődjön ki
- Készíts egy AdRepository osztályt, és ebben valósítdk meg a mentés műveletét!
- Készíts egy AdController osztályt, és ebben a POST –kérést kezelő metódust, amellyel majd a mentés REST hívással elvégezhető!
- A POST-kérés adja vissza a mentés utáni állapotot, a kitöltött azonosítóval és létrehozási idővel!
- Teszteld Postmannel a hirdetésfeladást!

2. feladat

- Készítsd el a keresés adatbázisműveletet az AdRepository osztályban, amely keresőszó, minimális ár és maximális ár alapján adja vissza a találatokat. A keresőszó a címben keres.
- Készítsd el az AdControllerben a GET-kérést kezelő metódust! A paramétereket request-paraméterekként vegyük át. A keresőszó megadása legyen kötelező, az árak pedig opcionálisak. A minimális ár alapértelmezett érteke 0, a maximálisé pedig 10 000 000.
- Teszteld Postmannel a keresést!

3. feladat

- A változtatást fogjuk megvalósítani, de bejelentkezés hiányában nem tudjuk, hogy ki adta fel a hirdetést, tehát ki módosíthatja. Ezért először egy „titkos” kódot kell generálnunk a hirdetések feladásakor. A hirdető ezt visszakapja, és ez alapján lesz majd képes később a hirdetést módosítani.

- Vegyél fel egy új tagváltozót a kódnak az Ad entitásba!
- Mentéskor generálj egy hat karakter hosszú kódot, és a mentéskor ezt add vissza. Használhatod az alábbi algoritmust:
- ```
  public class SecretGenerator {
      private static final char[] CHARS =
  "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789".toCharArray()
  ;
      private static final Random RND = new Random();

      public static String generate() {
          StringBuffer sb = new StringBuffer();
          for(int i = 0; i < 6; i++)
              sb.append(CHARS[RND.nextInt(CHARS.length)]);
          return sb.toString();
      }
  }
  ```
- Módosítsd a keresést úgy, hogy az AdControllerben az eredményhalmaz visszaadása előtt, minden találat kódját nullázd ki, hogy ne adjuk vissza.
- Készíts egy metódust az AdRepository-ban, amely a módosított hirdetést várja úgy, hogy a titkos kód is ki van benne töltve. A metódus kérdezze le ID alapján a hirdetést, hasonlítsa össze a titkos kódot, és csak akkor végezze el a módosítást, ha egyezik a kód. Visszatérési értékkel vagy kivételdobással jelezheted, ha nem sikeres a módosítás.
- Az AdController metódusa PUT hívást fogadjon a módosított állapotot pedig a törzsben vegye át. Ha sikeres a módosítás, akkor OK, ha nem jó a kód, akkor FORBIDDEN legyen a visszaadott HTTP státusz!
- Teszteld Postmannel a módosítást!

4. feladat

- Szeretnénk, ha egy hirdetés változásaira fel tudnánk iratkozni, és értesítést kapnánk róluk. Pl. így értesülhetnénk arról is, ha olcsóbb lett a termék. Ehhez ebben a lépésben még csak a feliratkozások eltárolását fogjuk megvalósítani.
- Hozz létre egy Subscription entitást. Ebben legyen generált azonosító, email cím (String), és az entitás kapcsolódjon a megfelelő hirdetéshez! Gondold át milyen multiplicitású a kapcsolat, és milyen annotációkat kell használnod! A kapcsolat legyen kétirányú!
- Készíts egy SubscriptionRepository osztályt, és benne egy metódust a feliratkozás elmentéséhez!
- Készíts egy SubscriptionController osztályt és egy POST endpointot a feliratkozáshoz. A hirdetés azonosítója az útvonalba legyen kódolva (pl. /subscriptions/15), a törzsben pedig az email menjen csak, ehhez hozz létre egy EmailDto-osztályt csupán erre a célra.
- Teszteld Postmannel a feliratkozást!

5. feladat

- Most a tényleges értesítésküldést fogjuk elkészíteni. Ez egy összetettebb feladat, de ezért több segítséget is adunk hozzá.
- Először a levélküldést kell konfigurálni a Springben. Ehhez egy új függőség is tartozik, amelyet a pom.xml-be kell felvenni a többi alá:
  ```

- ```
  <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-mail</artifactId>
  </dependency>
  ```
- Spring Bootban nagyon egyszerű a levélküldés konfigurációja, elég az application.yml fájlban elvégeznünk néhány beállítást:

  ```
  spring:
    mail:
      host: smpt.gmail.com
      port: 587
      username: username@gmail.com
      password: password
      properties:
        mail:
          smtp:
            auth: true
            connectiontimeout: 5000
            timeout: 5000
            writetimeout: 5000
            starttls:
              enable: true
  ```
- A következő lépés, hogy kiegészítsük a módosítás logikáját. Amikor megtörtént a mentés az AdControllerben, végig kell menni a kapcsolódó feliratkozásokon, és mindnek elküldeni egy szöveges üzenetet levélben. A feliratkozások lekérdezése átgondolandó. Mivel az Ad oldaláról többes kapcsolat, ezért lusta betöltésű. Célszerű a SubscriptionRepository osztályban készíteni egy metódust, ami Ad id alapján visszaadja a feliratkozásokat. Ha ezt egy @Transactional annotációval ellátott metóduson belül kérdezzük le, akkor menni fog.
- Most következik a tényleges levélküldés. Ehhez az osztályba injektálnunk kell egy JavaMailSender objektumot:

  ```
  @Autowired
  private JavaMailSender javaMailSender;
  ```

- Ezzel az alábbi módon küldhetünk levelet:

  ```
  SimpleMailMessage msg = new SimpleMailMessage();
  msg.setFrom("username@gmail.com");
  msg.setTo("foo@bar.com");
  msg.setSubject("Subject");
  msg.setText("body");
  javaMailSender.send(msg);
  ```

- A feladó emailje bármi lehet, de a Google SMTP szervere úgyis át fogja írni a saját címünkre, amellyel bejelentkeztünk. A beállításainktól függően elképzelhető, hogy az első sikeres próbálkozáskor levelet kapunk arról, hogy kevéssé biztonságos alkalmazás próbált a fiókunkhoz hozzáférni. Mondjuk, hogy mi voltunk, és utána lehet engedélyezni a hozzáférést. A tesztelés után újra blokkolhatjuk.
  ```

Opcionális extra feladatok

- HTML formátumú elegáns levél küldése. Éles alkalmazásban ilyen lenne a levél, tartalmazná az oldal logóját stb. A Thymeleaf template engine igazából szerveroldali renderelésű webalkalmazásokhoz készült elsősorban, de remekül használható erre a célra is. Konfigurálni kell először a Springben a template engine-t, template-et készíteni, majd a template engine-nel feldolgoztatni a template-et, és a végeredményt tenni a levél törzsébe.
- Lejárati dátum a hirdetéseknek. A felvitelkor kell megadni, és periodikusan (pl. 5 percenként) ellenőrizzük, és töröljük a lejárt hirdetéseket. Az érdekesség a feladatban az időzített taszk elkészítése.
- Spring Data. Cseréljük le a kézzel írt repository osztályokat Spring Data repository-kra!