Dokumentáció

Galéria szervlet

Írta: Farkas János (FAJTAAP.PTE), Farkas Péter (FAPVABP.PTE)

2015

Koncepció

Egy olyan weboldal felépítését választottuk projektmunkának, ami különböző felhasználók képeit tárolja. A felhasználó tud regisztrálni, bejelentkezni, képet feltölteni, azokhoz címet és leírást megadni, valamint az oldalra összes eddigi feltöltött képet megnézni felhasználókra lebontva. Az oldal felépítését NetBeans-ben végeztük, alkalmazásszervernek Apache Tomcat-et, adatbázisszervernek MySQL-t használtunk (utóbbi kettő megtalálható a népszerű XAMPP csomagban). Az alábbiakban a projektmunkánk működését írjuk le.

(Farkas Péter)

Adatkapcsolati réteg

A felhasználói adatok, illetve a képek adatainak tárolásához MySQL szervert választottunk. Elsőként Apache Derby-vel próbálkoztunk, de számos nehézség és a változások ellenőrzésének nehézsége miatt maradtunk a MySQL-nél. Adatainkat igyekeztünk perzisztens módon tárolni, mégpedig RedHat Hibernate használatával. Természetesen az ehhez szükséges függőségeket Maven segítségével szereztük be.

Mindössze két adattáblánk van, egy `users` és egy `pictures`, melyeket egy idegenkulcs-megszorítás köt össze. Ezeket nem mi írtuk, hanem élve a *Code first* szemlélettel és a JPA tágas eszközkészletével, a Hibernate-tel generáltattuk le entitás osztályokból. Az osztályokhoz nem készült külön mapping fájl, inkább az annotációkkal való konfigurálást választottuk. A *User* osztályban helyet kapott a felhasználó-azonosító, mely természetesen `@Id`, és `@GeneratedValue` annotációkat kapott, továbbá még itt szerepel a felhasználónév, a jelszó (MD5 hash formában), az e-mail cím, mely - ahogy azt illik - egyedi lehet csak, valamint a felhasználó kereszt- és vezetékneve. Szerepel még itt egy *Picture* típusú *Collection* is, melyet `@OneToMany` annotációval kapcsoltunk össze a *Picture* osztállyal. Ennek segítségével tud a Hibernate idegenkulcsos kapcsolatot létesíteni a két tábla közt, mégpedig egy a többhöz módon, ahol a *User* az egyes oldal, *Picture* pedig a többes. A *Picture* osztály ennél egyszerűbb, ott csak a kép azonosítóját, címet, leírást és a kép URL-címét rögzítjük, illetve van még benne egy *User* típusú mező is `@ManyToOne` annotációval, ez mentéskor egy idegenkulccsá alakul, és értéke egy egész szám lesz, ami mutatja, hogy melyik felhasználó töltötte fel.

Az adatok rögzítését vagy módosítását végző eljárások egy külön osztályba kerültek, melynek neve *Controller* lett. Itt egyszerű metódusok szerepelnek, melyek az összes képet vagy felhasználót képesek visszaadni szűrés nélkül, mint például a `queryUsers()` és `queryPictures()`. Utóbbinak van

overloaded verziója is, ami képes `userName` vagy `user` objektum alapján szűkíteni a lekérdezést. Az összes lekérdezés HQL nyelven íródott paraméterek használatával, így akadályozva az SQL injekciót. Szerepelnek még itt képek és felhasználók felvitelére alkalmas függvények is (`newUser()`, `newPicture()`). Bár a valódi fájlfeltöltés nem itt történik, csak az adatok és az URL rögzítése. A rekordrögzítő metódusok mind egy mintára készültek: ha sikeres volt a tranzakció, igaz értékkel térnek vissza, ha valami okból nem volt sikeres, például a felhasználó már szerepel a rendszerben, akkor hamis értéket ad. Akadnak még itt olyan metódusok is, amik azt hivatottak ellenőrizni, hogy egy rekord már létezik-e a rendszerben (`isExistingUser()`, `isExistingEmail()`), ezek is boolean értékkel térnek vissza. Teszteléshez bekerültek még kép- és felhasználótörlő metódusok is (`deleteUser()`, `deletePicture()`). Egy dolog közös az összes korábban felsorolt függvényben, mégpedig az, hogy kezdéskor mindegyik elkér a *HibernateUtil* osztálytól egy *Session*-t (Hasonló az *EclipseLink EntityManager*-éhez), amit ezután meggynyit és használ, majd be is zár, ha adatmódosítást végez, akkor még kezdeményez egy tranzakciónyitást, amit egy commit-tal zár. Az összes itt szereplő metódus statikus, így az osztály példányosítása nélkül is könnyen elérhetőek bárhonnan.

JUnit tesztek

A tesztelés kisebb kihívást jelentett, mivel - mint később kiderült - a teszek lefutási sorrendje nem garantált, így az adatbázis véletlenszerű írása azt eredményezheti, hogy néhány teszt nem képes sikeresen lefutni. Mivel az adatbázis nem enged redundáns rekordokat, így azt sem lehet elkövetni, hogy mindenhova ugyanazt a tesztrekordot szúrjuk be. Nem beszélve arról, hogy az adatok tényleges rögzítéséről csak úgy tudnánk bizonyosságot nyerni, hogy saját magunk által írt metódusokkal ellenőrizzük az adatbázis rekordjait. Így a megoldás az lett, hogy a „rögzítő” metódusok boolean visszatérési értéket kaptak, hogy a működésük valamelyest ellenőrizhető legyen. Az összes teszt elején létre kellett hozni random tesztadatokat, majd azokat a teszt végeztével egyenként törölni. (Itt jöttek kapóra a törlő metódusok). Arra is nagy figyelmet kellett fordítani, hogy a megfelelő sorrendbe törlődjenek az adatok, mivel egy gondatlan törlés idegenkulcs megszorítást sérthet. Az utóbbiakra figyelve végül kivitelezhető volt a tesztelés is.

(Farkas János)

JSP oldalak és szervletek

A felhasználó a JSP-oldalakon keresztül kommunikál a szerverrel, és az adatkapcsolati réteggel, amikor regisztrál, be- vagy kijelentkezik, vagy feltölt egy képet, esetleg lekéri a feltöltött képeket. Eme feladatok majdnem mindegyikére jutott egy JSP-szervlet páros, kezdjük tehát a felsorolást!

index.jsp

Az alkalmazás kezdőoldala, ahol a felhasználó csupán pár figyelemfelkeltő üzenetet lát. A fejlécben látható egy *Képek*, egy *Regisztráció*, és egy *Bejelentkezés* menüpont, bejelentkezés után az utolsó kettő helyett megjelenik a felhasználó nevét kijelző menüpont, egy *Feltöltés*, és természetesen egy *Kijelentkezés* menüpont. Először a regisztráció folyamatát írom le.

registration.jsp/RegistrationServlet.java

A felhasználó a megjelenő form-on kitölti a szükséges adatokat (a jelszót kétszer, nehogy véletlenül félregépeljen valamit), majd a *Regisztráció* gombbal továbbküldi a szervletnek, aminek az elérési útvonala a form `action="RegistrationServlet"` részében lett beállítva. A formról POST metódussal adódnak át a szervletre az adatok, így a szervlet `doPost()` metódusa hívódik meg. A szervlet lekéri a formban elmentett paramétereket, és a különböző hibalehetőségeket ellenőrizve (hiányzó adat, létező felhasználónév/e-mail cím, nem egyező jelszavak) RequestDispatcher-rel továbbít (illetve visszaad) egy hibaüzenet-paramétert a JSP-fájlnak, vagy új felhasználót regisztrál. A felhasználó adatainak az adatbázisba való felvitelét, valamint a felhasználónév-ellenőrzést a Controller osztályban megírt metódusok biztosítják. A JSP lefutásakor ellenőrzi, hogy van-e hibaüzenet, illetve sikeres regisztráció esetén visszajelzés, ha van, akkor ezeket kijelzi. Hiba esetén az eddig kitöltött adatokat (a jelszavakat kivéve) is megkapja a JSP-fájl.

login.jsp/LoginServlet.java/LogoutServlet.java

A felhasználó bejelentkezéskor megadja a felhasználónevét és a jelszavát, az adatok továbbítódnak a LoginServlet-re. Ekkor lefut a Controller osztály `submitLogin()` metódusa, amely megkeresi a passzoló felhasználót. Ha a metódus null-t ad vissza, vagyis a megadott felhasználónév és/vagy a jelszó rossz, akkor a fenti módon hibaüzenetet küld. Ha nincs hiba, akkor létrehoz egy HttpSession-t, amibe elment a user objektumot, majd átirányít az index.jsp-re, amely ellenőrzi, hogy van-e felhasználó-objektum, és ennek megfelelően írja ki a böngésző a menüpontokat. A *Kijelentkezés* menüpontra kattintva elindul a LogoutServlet, ami meghívja a `session.invalidate()` metódust, így kijelentkezve az oldalról.

users.jsp/pictures.jsp

A *Képek* menüpontra kattintva betölt a users.jsp, ami a felhasználók felsorolását tartalmazza, ezt a `Controller.queryUsers()` metódus meghívásával értük el. Bármely felhasználónévre kattintva a pictures.jsp tölt be, ami GET metódussal kapja meg a felhasználónév paramétert. Nem létező felhasználónév esetén a "Hiba" szót írja ki. Itt a `Controller.queryPictures(username)` metódussal kapjuk meg az adott felhasználó képeit, ráadásul a bejelentkezett felhasználó neve mellett felbukkan egy *Feltöltés* gomb is.

upload.jsp/UploadServlet.java

A feltöltés már trükkösebb feladat volt, hiszen egy sima form nem elegendő a feltöltéshez, meg is kellett adni neki az `enctype="multipart/form-data"`paramétert, ezzel is jelezve, hogy egy fájlt készülünk feltölteni. A feltöltést az UploadServlet végzi az *Apache Commons* FileUpload csomagjának metódusai segítségével, három tutorial alapján (első, második, harmadik) sikerült megvalósítani a jelenleg működő modellt.

A servlet ellenőrzi, hogy úgynevezett multipart tartalmat adtunk-e át, vagyis fájlt akarunk-e feltölteni. Miután ez megvan, ellenőrzi a form mezőinek tartalmát: ha fájl, feltölti a szerverre, és beállítja a képobjektum relatív elérési útját; ha szöveges tartalom, akkor beállítja a címet, és a leírást. Ezután lefut a `Controller.newPicture(pic, user)` metódus, ami logikai értéket ad vissza. Ha sikeresen

lefutott, a szervlet visszajelzést küld, ha hiba van (pl. nem lett kiválasztva fájl), hibaüzenetet ír ki, és a feltöltött fájlt megpróbálja törölni, hiszen nem lehet az adatbázisban két egyforma fájlnevű kép.

Design

Az oldal külseje a Bootstrap nevű keretrendszer segítségével lett kialakítva, az ikonokat a Font Awesome nevű speciális webes betűtípus segítségével lettek kirajzolva, a stílustulajdonságok a LESS nevű CSS-preprocesszorban lettek megírva.

(Farkas Péter)