Mini JIRA

munka- és feladatszervezési üzleti alkalmazás

fejlesztői dokumentáció
Java alapú webes keretrendszerek (VIAUBV18)

2022. 05. 19.
Piller Trisztán WHKJZX

Technológiák

Spring Boot alapokkal indított alkalmazás, amely alkalmazza még a következő Spring technológiákat:

- Spring Security
- Spring MVC
- Thymeleaf + Extras for Spring Security
- Spring Data JPA
- Spring Boot Validation
- Beépített Tomcat webszerver

Függ még az alkalmazás a következő technológiáktól és Maven könyvtáraiktól:

- PostgreSQL: RDBMS
- Lombok: Könyvtár, amely annotációkon keresztül képes generálni osztályokon segédfüggvényeket (Builder és Data class minta megvalósítása)

A fejlesztéshez/futtatáshoz szükséges:

- PostgreSQL adatbázis létrehozása
- a megfelelő konfigurációk beállítása (adatbázis elérési útvonala, adatbázis authentikációs adatok felülírása) a application.properties fájlban
- JDK legalább 11-es verziója
- ajánlott: Jetbrains IntelliJ IDEA fejlesztőkörnyezet

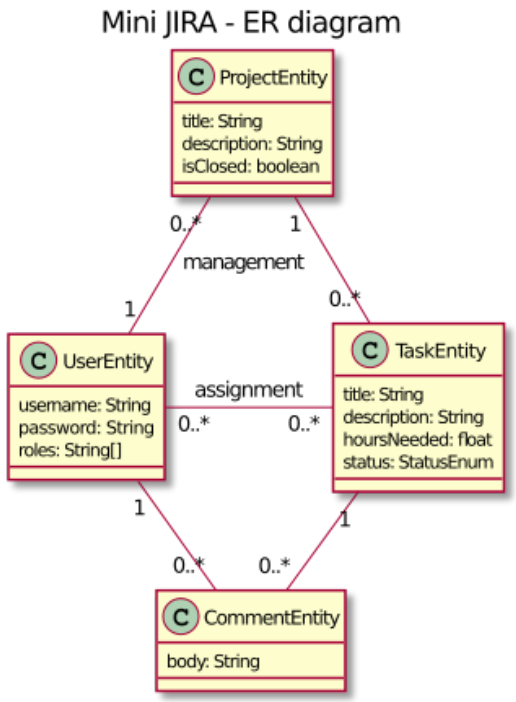
Alap funkcionalitások

A következő lehetőségek adottak: regisztráció → ekkor a felhasználó fejlesztőként kap hozzáférést a rendszerhez. A felhasználó manuális adatbázis tábla manipuláció segítségével léptethető elő manager pozícióba.

Bejelentkezést követően a fejlesztő minden erőforráson olvasási joggal rendelkezik, azonban kommentet szabadon fűzhet akármelyik taszkhoz, a saját kommentjei felett írási joggal is rendelkezik.

Bejelentkezést követően a manager minden erőforrás felett olvasási és írási joggal is rendelkezik.

ER diagram



Az asszociációk nagyrészt @ManyToOne annotáció révén a gyerek osztályok részéről vannak tárolva, a szülőosztályoknál @OneToMany annotációval is jelölve van az asszociáció, azonban lazy loadinggal megoldva.

Az *assignment* asszociáció a TaskEntityben kerül jelzésre *assignedUsers* mező által, @ManyToMany kapcsolat révén pedig egyébként külön táblában tárolódik JPA szabványának megfelelően (Hibernate megoldásával).

Komponensek

A kód alapvetően a többrétegű webalkalmazások mintáinak megfelelően rétegződik:

- View réteg: Thymeleaf sablonokba renderelődik ki a tartalom, ViewModelben kapja meg a Controllertől az adatokat.
- Controller réteg: Controllerekben van megvalósítva a bejövő és kimenő forgalom irányítása, rendereli a Viewokat, a Servicekre építve teszi meg az adatbeszerzést. Kezeli a validációs hibák visszaadását.
- Input osztályok: A bemenő adatokat zárja egységbe, a validációra ad felületet.
- DTO (Data Transfer Object) osztályok: A REST endpointokon kimenő adatokat zárja egységbe, a szerializációra ad felületet.
- Service réteg: Az adatok manipulációjáért (CRUD), a megfelelő üzleti logikáért felelős osztályok, a Repositorykat veszik igénybe.

- Repository réteg: Az adathozzáférési réteget képviseli, JPA alapú interfészeket gyűjt.
- Kézzel írt JPQL, SQL-re és Criteria API-ra nincs egyelőre szükség.

A validációt a beépített Spring Boot könyvtár oldja meg a javax.validation package interfészeit/annotációit implementálva, a fejlesztőnek csupán az Input osztályokat kell megfelelően felannotálni és a Controller bemenetein jelezni a @Valid annotációval.

Elkülönítésre kerül így három formátuma az adatoknak: Input formátum, DTO formátum és adatbázisbeli formátum (entitás).

Spring Security

A fenti bejelentkezés utáni jogok szerepkör szerinti megvalósítása a Spring Security feladata.

Az authorizációs rendszer jelenleg endpointokra pontosan fel van konfigurálva, hogy mely erőforrásokhoz milyen jogköröknek van hozzáférése.

A bejelentkeztetés és regisztráció saját lokális authentikációs megoldással van megoldva: A felhasználó egyedi felhasználónevet és hozzátartozó jelszót kell megadjon.

A jelszó BCrypt-tel van hash-elve, amely elrejti a belső saltozás mechanizmusát, a Spring Security is ezt használja az authentikációra. A Spring Security rendszerében be van még az konfigurálva, mely oldalakat használja beléptetésre és sikeres/sikertelen beléptetések során.

A WebMvc-ben Interceptor beve megoldott, hogy a beléptetett felhasználó a register és login oldalakról a **/projects** oldalra legyen átvezetve.

Kaszkádosított törlés

A szülőentitásokon jelölt @OneToMany lazy loadolt asszociációkon van jelölve a kaszkádosított törlés. Ez kompozíciót valósít meg, azaz ha a szülő törlésre kerül, akkor a gyermekei is törlésre fognak kerülni az adatbázisban. Így biztosítva van az adatbázis szemét elkerülése.

Sablonmotor: Thymeleaf

A Thymleaf motor rendereli ki a választott Viewokat, feltölti adatokkal a ViewModelből.

Ez nem Javascript könyvtár/keretrendszer, így a dinamikus lekérések vanilla Javascript-es AJAX kérések által kerülnek kivitelezésre (törlések és a dinamikusan feltöltött modalok: projekt todo listája, taszkhoz hozzárendelt felhasználók és kommentek listázása).

Endpointok

JSON endpointok

- GET **/api/projects/{id}/todos**: TaskDTO tömbben a projekt todo-i
- GET **/api/tasks/{id}/assignees**: UserDTO tömbben a taszk assignee-jai
- GET **/api/tasks/{id}/comments**: CommentDTO tömbben a taszk kommentjei

Authorizációs/profil endpointok

- GET **/**: beléptetési oldal renderelése
- GET **/profile**: átvezet a /users/{id} oldalra, ahol az id a beléptetett felhasználó id-je
- GET **/users/{id}**: felhasználó adatainak megjelenítése és hozzárendelt taszkok
- GET **/auth/login**: átvezet a / oldalra, azaz a beléptetési oldalra
- GET **/auth/register**: regisztrációs oldal renderelése
- POST **/auth/register**: regisztrációs adatok felküldési formja

Projekt erőforrás manipulálására és lekérésére endpointok

- GET **/projects/**: összes projekt Viewja
- GET **/projects/{id}**: egy projekt jellemzőinek, taszktáblájának Viewja
- GET **/projects/new**: új projekt létrehozási View
- GET **/projects/{id}/edit**: projekt szerkesztési View
- POST **/projects/new**: új projekt létrehozási form leküldése
- POST **/projects/{id}/edit**: szerkesztési form felküldése
- DELETE **/projects/{id}**: projekt törlésére szolgáló parancs

Taszk erőforrás manipulálására és lekérésére endpointok

- GET **/tasks/{id}**: taszk részleteinek, kommentjeinek és hozzárendelt felhasználóinak Viewja
- GET **/tasks/new/{projectId}**: új taszk létrehozási View, ahol projectId a szülőprojekt id-je
- GET **/tasks/{id}/edit**: taszk szerkesztési View
- POST **/tasks/new/{projectId}**: új taszk létrehozási form felküldése, ahol projectId a szülőprojekt id-je
- POST **/tasks/{id}/edit**: szerkesztési form felküldése
- POST **/tasks/{id}/assign**: taszkhoz rendelt felhasználó (assigneeId) felküldése formban
- POST **/tasks/{id}/unassign/{assignId}**: felhasználó hozzárendelésének megszüntetésére szolgáló parancs
- DELETE **/tasks/{id}**: taszk törlésére szolgáló parancs

Komment erőforrás manipulálására és lekérésére endpointok

- GET **/comments/{id}/edit**: komment szerkesztési View
- POST **/comments/new/{taskId}**: új komment létrehozási form felküldése, ahol taskId a szülőtaszk id-je
- POST **/comments/{id}/edit**: szerkesztési form felküldése
- DELETE **/comments/{id}**: komment törlésére szolgáló parancs

Service layer

A Service osztályok tipikus CRUD függvényeket implementálnak, különleges logika csak a nullkezelésre van, így némely függvény IllegalArgumentException kivételt dobhat (kódban annotálva van, ahol ilyen lehet).