Web applikáció készítése Spring boot keretrendszer segítségével

Hollay Norbert

2017 Szeged

Tartalomjegyzék

[Új projekt létrehozása: 3](#_Toc501051287)

[Bejelentkezés megvalósítása 5](#_Toc501051288)

[Adatbázis modell létrehozása 8](#_Toc501051289)

[Adatbázis elérése, szerkesztése 9](#_Toc501051290)

[Új User feltöltése az adatbázisba 10](#_Toc501051291)

[Userek kilistázása 11](#_Toc501051292)

[Userek kilistázása Thymeleaf segítségével 11](#_Toc501051293)

[Új User hozzáadása, frissítése, törlése: 12](#_Toc501051294)

[User hozzáadása 13](#_Toc501051295)

[User törlése 13](#_Toc501051296)

[User módosítása 13](#_Toc501051297)

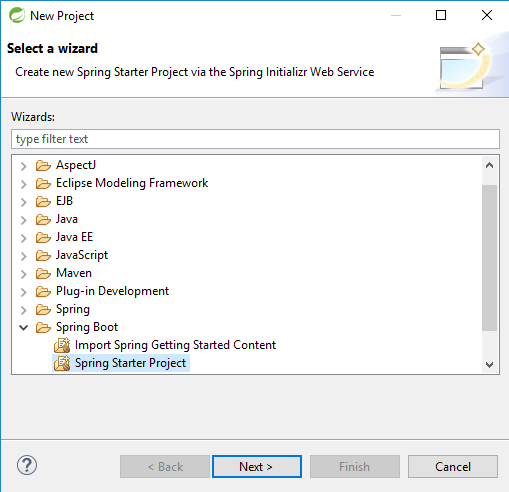
## Új projekt létrehozása:

A projekthez Spring Tool Suite IDE-t használtam!

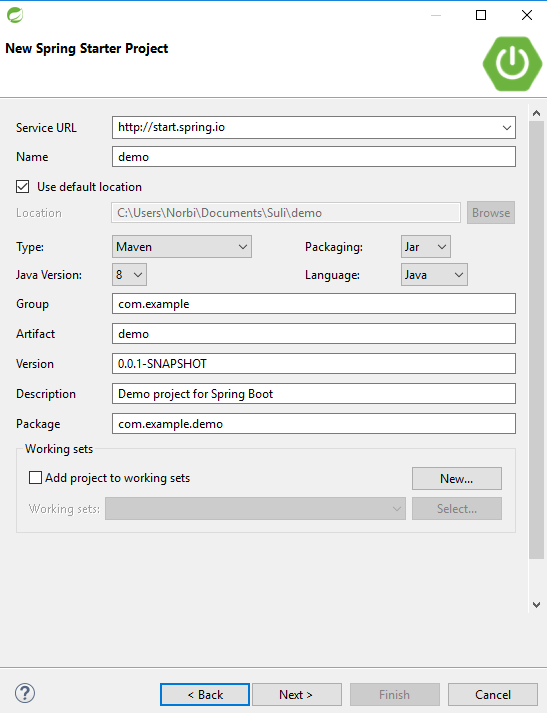
STS letöltése <https://spring.io/tools>

1. Kiválasztjuk a Spring Starter Project-et.

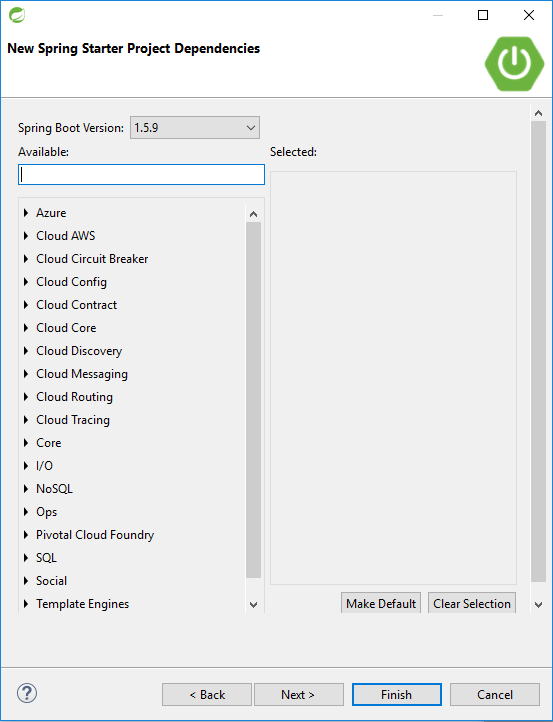
(alternatíva <https://start.spring.io/> - itt online le lehet generálni a projektet)



1. Beállítjuk a projekt adatait.

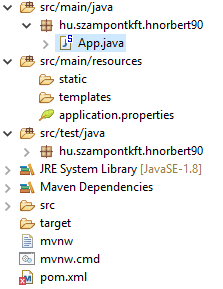


1. Kiválasztjuk a függőségeket, a használni kívánt modulokat, pl: Web, jpa, h2, thymeleaf



„Finish”re kattintva létre hozza a kezdőprojektet

felépítése:



App.java a program belépési pontja, tartalmazza a main metódus, és elindítja a SpringApplication-t.

src/main/resources – mappa fogja tartalmaznia statikus tartalmakat (css,js,képek stb), dinamikus tartalmakat a templates mappa(fragmentek,bejelentkezési képernyő stb.).

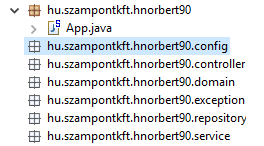
application.properties tartalmazza az elsődleges szerver konfigurációt, ez a konfigurációs fájl a legnagyobb precedenciájú.

src/test/java/ -mappa tartalmazza a unit teszteket.

Maven dependecies-be kerülnek a letöltött függőségek.

pom.xml –ben vannak definiálva a függőségek.

## Bejelentkezés megvalósítása

Létrehozom a package-ket melyek logikailag szeparálják az applikáció osztályait. 

* config - ad helyet a konfigurációs osztályoknak.
* controller - bekerülnek azok az, osztályok melyek a forgalomi rányitásért felelősek.
* domain - ben lesznek létrehozva az adatbázis modellek.
* exception - hibakezelésre létrehozott osztályok
* repository -ba kerül, a logika mely elvégzi az adatbázis műveleteket pl: Create Read, Update, Delete
* service -bekerül az üzleti logikát megvalósító osztályok

Packagek létrehozása után a Config package -ba létrehozok egy SecurityConf nevű osztályt, melyben definiálom a felhasználókat, és az engedélyeket.



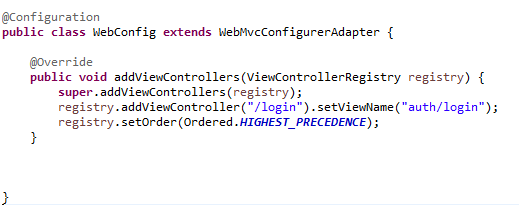
@Configuration annotáció segítségével jelöljük, hogy az osztály a konfigurációért felel.

Az osztály el van látva még @EnableGlobalMethodSecurity annotációval, mellyel azt állítjuk be, hogy melyik controller melyik függvényéhez milyen szerepkör szükséges.

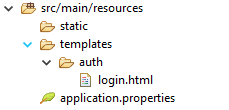
Az osztályt kiterjesztettük a WebSercurityConfigureAdapter-el, segítségével felülírhatjuk a viselkedését.

configureAuth metódussal definiálom a felhasználokat. configure metódussal az endpointok elérhetőségét korlátozom a szerepkörökre.

Továbbá létrehozok egy WebConf nevű osztályt, amely hasonló feladatokat lát el, mint a controller, átirányítja a felhasználót a kért nézethez.

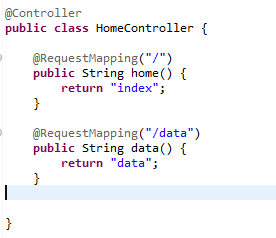


Login view-t létrehozom a templates auth mappában.



login.html kötelező elemei: egy form felhasználónév, jelszó, gomb. A gomb megnyomásával a „/login” endpoint felé intézünk kérést. A spring security a felhasználónév/jelszó alapján authentikálja a felhasználót. Ha az authentikáció sikertelen, a Security error-al jelzi. Thymeleaf segítségével visszajelzést küldünk a felhasználónak, hogy hibás adatokat adott meg.

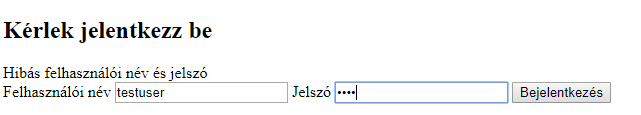
Létrehozok a Controller packageba egy HomeController nevű osztályt, melyben az endpointokat definiálom



Application propertiesbe megváltoztattam a szerverportot 9059-re:

server.port = 9059

Most ha elindítjuk a Spring boot szervert, és felmegyünk a localhost:9059/ oldalra a következő képernyő vár minket:



Bejelentkezés után whitelabel Errort kapunk, mert még nincs létrehozva a kezdőképernyő.

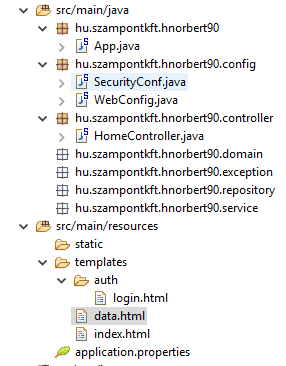
A konfigurációs fájlba két felhasználót hoztunk létre: testuser,testadmin

rendre User, és Admin szerepkörökkel.

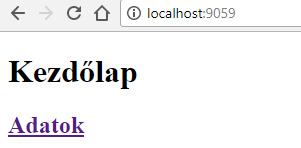
„/” gyökér-re „User” jogosultság szükséges, valamint a „/admin/\*” hez „Admin” jogosultság ezt a SecurityConf osztályban állítottam be.

Ha „user”-ként próbáljuk elérni az „/admin/” endpointot, nem arról kapunk hibát, hogy az erőforrás nem létezik, hanem hogy a hozzáférés megtagadva. Ez azért van, mert a Security először ellenőrzi, hogy a felhasználó jogosult a kért tartalom megtekintésére, ha nem akkor visszautasítja a kérést 403-s hibával.

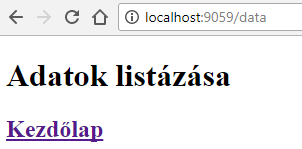
Létrehozom, a kezdőlapot egy linkel a „/data” endpoint elérésére. A html fájlokat a templates mappába rakom.



Szerver újra indítása utána bejelentkezés utána a kezdőlapra kerülünk:



„/data” view

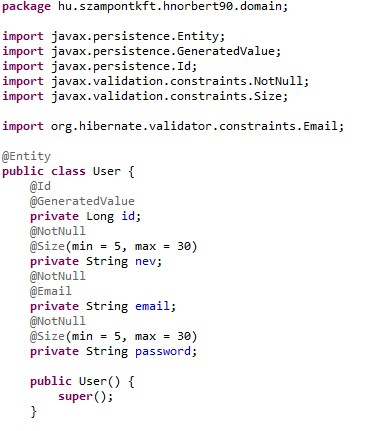


## Adatbázis modell létrehozása

User tábla

Mezők: id, Név, email, password

Létrehoztam a domain packageba egy User osztályt.



Az osztályt @Entity annotációval jelöltem ez jelzi a JPA-nak, hogy a modellből létre kell hoznia egy táblát az adatbázisba.

Az id-t @Id , és a @GeneratedValue annotációkkal láttam el, ezzel jelezve, hogy egyedi azonosítóról van szó, és hogy az értékét a JPA generálja.

Névnél, és a jelszónál bealítottam, hogy legalább 5 max 30 karakteres szöveget várjon. Az emailnél jeleztem, hogy érvényes emailt várunk.

A JPA –nak szüksége van egy paraméter nélküli konstruktorra a működéshez.

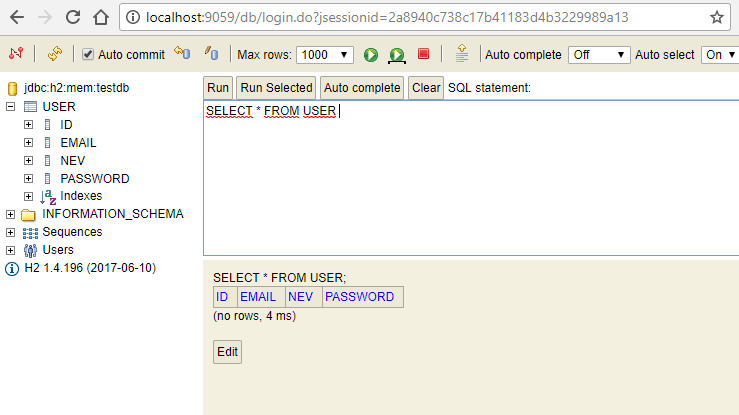
Továbbá legeneráltam a mezőkhöz a getter/setter metódusokat, valamint egy konstruktort, ami nevet,emailt,illetve passwordot vár.

Kiegészítettem az application.properties-t a következő két sorral:

spring.h2.console.enabled=true

spring.h2.console.path= /db

A szervert újra indítva localhost:9059/db címen elérhetjük az adatbázis grafikus felületét.



## Adatbázis elérése, szerkesztése

Létrehoztam az UserRepository interfacet a repository packageban.

**package** hu.szampontkft.hnorbert90.repository;

**import** java.util.List;

**import** org.springframework.data.repository.CrudRepository;

**import** org.springframework.stereotype.Repository;

**import** hu.szampontkft.hnorbert90.domain.User;

@Repository

**public** **interface** UserRepository **extends** CrudRepository<User,Long> {

List<User> findAll();

}

@Repository annotációval jeleztem, hogy az interface egy repository

Az interfacet kiterjesztettem a CrudRepository interfaceval melytől örökli az alapműveleteket.

Jeleztem, hogy User típusú adat lesz Long típusú id-val a repository tartalma.

findAll() metódus visszaadja az összes User-t a repositoryból egy listaként.

## Új User feltöltése az adatbázisba

Service packageban létrehozom az UserService osztályt

@Service

@Validated

**public** **class** UserService {

@Autowired

UserRepository userRepository;

**public** **void** newUser(@NotNull String username, @NotNull String email,@NotNull String password ) {

User user = **new** User(username, email, password);

userRepository.save(user);

}

**public** **void** deleteUser(@NotNull Long id) {

userRepository.delete(id);

}

**public** **void** updateUser(@NotNull Long id,@NotNull String nev, @NotNull String email,@NotNull String password ) {

User user = userRepository.findOne(id);

user.setEmail(email);

user.setNev(nev);

user.setPassword(password);

userRepository.save(user);

}

}

@Service annotációval jelzem, hogy egy servicet hoztam létre, valamint a @Validated annotációval, azt jelzem, hogy valid adatokkal dolgozik az osztály.

@Autowired annotációval automatikusan beinjektálja a függőséget a spring az alatta lévő változóba.

newUser metódus várja az User adatait, létrehoz egy új Usert, és lementi a repositoryba.

deleteUser törli a megadott id alapján felhasználót

## Userek kilistázása

@Controller

**public** **class** HomeController {

@Autowired

UserRepository userRepository;

@RequestMapping("/")

**public** String home() {

**return** "index";

}

@RequestMapping("/data")

**public** String data(Model model) {

model.addAttribute("user",userRepository.findAll());

**return** "data";

}

}

Kibővítettem az osztályt, UserRepository referenciájával, valamint Model attribútumot adtam a data metódushoz, ebbe a modellbe rakom bele az Usereket mint objektumokat. Ez alapján tudja majd a thymeleaf kilistázni a felhasználókat.

## Userek kilistázása Thymeleaf segítségével

<body>

<h1>Adatok listázása</h1>

<h2><a href=*"/"*>Kezdőlap</a></h2>

<table><tr>

<th>ID</th>

<th>NEV</th>

<th>EMAIL</th>

</tr>

<tr th:each=*"users : ${users}"*>

<td th:text=*"${user.id}"*>...</td>

<td th:text=*"${user.nev}"*>...</td>

<td th:text=*"${user.email}"*>...</td>

</tr>

</table>

</body>

Mivel üres még az adatbázis ezért feltöltöm pár teszt adattal egy ideiglenes osztály segítségével. A repository packageba lérehozom az UserPopulator osztályt.

@Component

**public** **class** UserPopulator {

@Autowired

UserRepository userRepository;

@PostConstruct

**public** **void** createTestUsers() {

**for**(**int** i=0; i<10; i++) {

userRepository.save(**new** User("user"+i,"email@email.hu"+i,"passoword"+i));

}

}}

@Component annotációval jelzem, hogy az osztály része a Spring applikációnak.

@PostConstruct-al beállítom, hogy fusson le az alatta lévő kódrészlet a szerver indulásakor.

Szervert újra indítva rámegyünk a „/data” endpointra és az eredmény:



## Új User hozzáadása, frissítése, törlése:

Létrehoztam egy UserController nevű osztályt a controller packageba, és a „/admin” endpointot figyeli az osztály. Műveletekhez admin szintű jogosultság szükséges.

@Controller

@RequestMapping(value = "/admin")

**public** **class** UserController {

@Autowired

UserRepository userRepository;

@Autowired

UserService userService;

@RequestMapping("/newuser")

**public** String newUser(@RequestParam("nev") String nev,@RequestParam("email") String email,@RequestParam("password") String password) {

userService.newUser(nev, email, password);

**return** "redirect:/";

}

@RequestMapping("/updateuser")

**public** String updateUser(@RequestParam("id") Long id,@RequestParam("nev") String nev,@RequestParam("email") String email,@RequestParam("password") String password) {

userService.updateUser(id,nev, email, password);

**return** "redirect:/";

}

@RequestMapping("/updateuser")

**public** String deleteUser(@RequestParam("id") Long id) {

userService.deleteUser(id);

**return** "redirect:/";

}

}

### User hozzáadása

Ehhez a következő formot használtam

<form action=*"/admin/newuser"* method=*"post"* autocomplete=*"off"*>

<input type=*"text"* name=*"nev"* placeholder=*"nev"*/>

<input type=*"email"* name=*"email"* placeholder=*"email"*/>

<input type=*"password"* name=*"password"* placeholder=*"password"*/>

<button type=*"submit"* name=*"submit"*>Add</button>

</form>

### User törlése

Ehhez generálhatunk törlő linket a thymleafal

<table><tr>

<th>ID</th>

<th>NEV</th>

<th>EMAIL</th>

</tr>

<tr th:each=*"user : ${users}"*>

<td th:text=*"${user.id}"*>...</td>

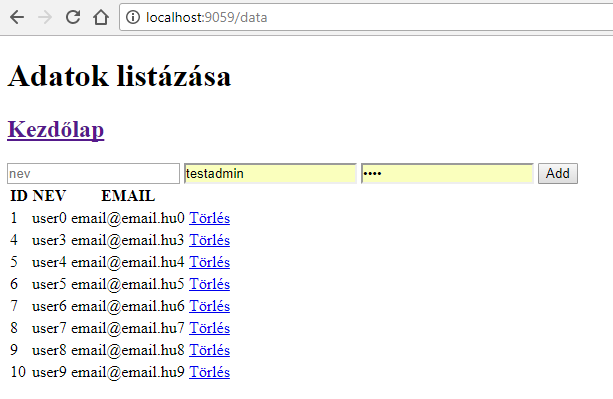
<td th:text=*"${user.nev}"*>...</td>

<td th:text=*"${user.email}"*>...</td>

<td><a th:href=*"'/admin/deleteuser?id='+${user.id}"*>Törlés</a></td>

</tr>

</table>



### User módosítása

Szintén létre hozzunk egy űrlapot. És a „/admin/updateuser” endpoint lehet használni a módosítások felküldésére.