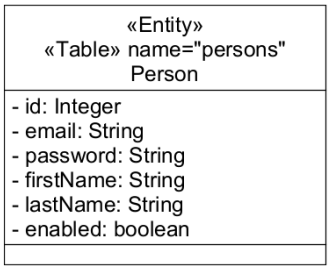
Egyszerű manager alkalmazás Spring és BootStrap segítségével

1. Készítsünk egy új projektet Spring Initilaizer-rel. A szükséges függőségek:
    - Lombok
    - Spring Web
    - Thymeleaf
    - H2 Database
    - Spring Data JPA
    - Spring Boot Dev Tools
2. Teszt entitásként hozzunk létre egy Person osztályt egy új model elnevezésű csomagban az alábbiak szerint:



3. Az osztályhoz hozzunk létre Spring Jpa repositoryt.
    - A application.properties állományban adjuk meg a következőket:

```
spring.h2.console.enabled=true
spring.jpa.hibernate.ddl-auto=create
spring.datasource.url=jdbc:h2:file:~/testdb22
```

    - Indítsuk el legalább egyszer az alkalmazást és vegyük észre, hogy a h2-console-ban (localhost:8080/h2-console) megjelenik a persons tábla.
4. Töltsük fel az adatbázisunkat néhány teszt-adattal
    - Állítsuk le az alkalmazást, ha még fut.
    - Módosítsuk az application.properties fájlban a `create` beállítást `validate` -re
    - A korábban tenultaknak megfelelően hozzunk létre egy olyan egységtesztet, amivel adatokat töltünk fel az adatbázisba.
    - A teszt osztály előtt ne felejtsük el használni az alábbi annotációkat:

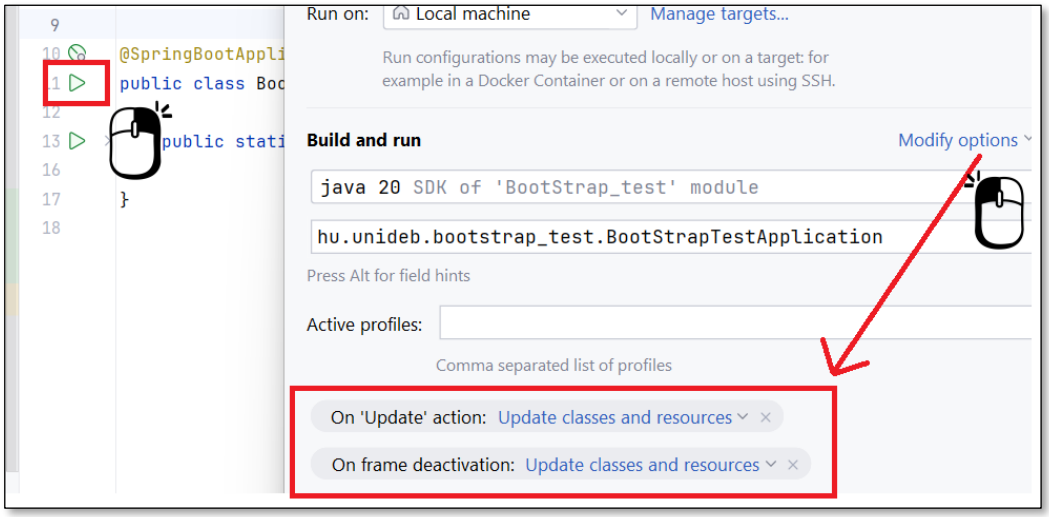
```
@DataJpaTest
@AutoConfigureTestDatabase(replace =
            AutoConfigureTestDatabase.Replace.NONE)
@Rollback(value = false)
```

    - Indítsuk el újra az alkalmazást és vegyük győződjünk meg róla, hogy nem üres adatbázissal dolgozunk tovább.
5. Készítsük el első webes végpontunkat.
    - Hozzunk létre egy MainController nevű Java osztályt egy új controller nevű csomagon belül.
    - Az osztályon használjunk @Controller annotációt és adjunk hozzá egy új metódust az alábbiak szerint:

```
@GetMapping("")
public String showHomepage(){
    return "index";
}
```

- A végpontok előtt használható mapping-ekről bővebben: https://docs.spring.io/spring-framework/reference/web/webmvc/mvc-controller/ann-requestmapping.html
- A projekt resources.templates csomagjában hozzunk létre egy index.html fájlt (A fájl nevének illeszkednie kell a végpontot leíró metódus által visszaadott Stringre.

6. Módosítsuk a html fájlt és vegyük észre, hogy a módosítások érvényre juttatásához újra kell indítanunk az alkalmazást. Mivel ez elég időigényes feladat lenne minden módosítás alkalmával, használjuk a spring dev tools lehetőségeit.
   - Győződjünk meg róla, hogy a `spring-boot-devtools` szerepel a pom.xml fájlban függőségként.
   - A projektünk beállításához válasszuk a File->Settings->Build, execution, deployment ->Compiler->Build project automatically lehetőséget
   - A main metódusunkat indító zöld háromszögre kattintva jobb egérgombbal, módosítsuk a futtatási beállításokat az alábbiak szerint:



   - Futtassuk újra az alkalmazást és néhány módosításon keresztül figyeljük meg, hogy már nem szükséges minden változás után újraindítani a projektet.

7. Kössük össze projektünk backend és frontend komponenseit és használjunk előre definiált html komponenseket
   - Használjuk html fájlunk stílusfájljaként a BootStrap alap css fájlját. Adjuk az index.html fájlt <head> eleméhez az alábbit:

```
<link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.1/css/bootstrap.min.css">
```

   - Figyeljük meg a böngészőben megjelenő index.html fájl változásait
   - A BootStrap-ről bővebben: https://www.hostinger.com/tutorials/w hat-is-bootstrap/
   - Ugyanezt a hatást érhetjük el úgy, is ha a betöltést a ThymeLeaf Template Engine-re bízzuk. (https://www.javatpoint.com/spring-boot-thymeleaf-view)
   - Adjuk hozzá projektünk pom.xml fájljához a BootStrap függőséget https://mvnrepository.com/artifact/org.webjars/bootstrap
   - Szintén szükséges a webjars-locator aktuális verziójának hozzáadása https://mvnrepository.com/artifact/org.webjars/webjars-locator

- Az index.html fájl <html> tag-jéhez adjuk hozzá:

```
<html lang="en" xmlns:th="http://www.thymeleaf.org">
```

- A css stílusfájl betöltéséhez pedig használjuk az alábbi sort:

```
<link rel="stylesheet" type="text/css"
th:href="@{/webjars/bootstrap/css/bootstrap.min.css}"/>
```

8. Táblázatban jelenítsük meg a tárolt Person entitásokat
   - Az index.html fájlhoz adjuk egy linket, ami az entitások kezelőfelületére vezet:

```
<a class="h3" th:href="@{/persons}">Manage persons</a>
```

   - Hozzunk létre egy új controller osztályt PersonController néven és definiáljuk benne a /persons végpontot az alábbiak szerint:

```
@GetMapping("/persons")
String getAllPersons(Model model){
    List<Person> personsList = personRepository.findAll();
    model.addAttribute("personsList", personsList);
    return "persons";
}
```

     A model paraméter a Spring MVC Model komponensében elérhető objektumként fog szerepelni, így ki tudjuk majd olvasni a benne tárolt listát a View komponensben, azaz a html fájlunkban.
   - Hozzuk létre a persons.html fájlt az index.html fájl alapján.
   - A persons.html fájlban hozzunk létre egy egyszerű html táblázatot az alábbi formában:

First name	Last name	e-mail	password

     HTML táblázatok: https://htmlreference.io/tables/
   - Alakítsuk át a táblát BootStrap stílusbeállításokkal

```
<table class="table table-bordered">
    <thead class="table-dark">
```

   - Thymeleaf segítségével iteráljunk végig a backendről kapott entitásokon:

```
<th:block th:each="person: ${personsList}">
    <tr>...</tr>
</th:block>
```

   - Az egy sorban kiírandó adatokat `<td th:text="${person.firstName}"/>` formában adhatjuk meg.
9. Egy új végpont segítségével tegyük lehetővé, hogy a felhasználó új személyt adjon az adatbázishoz.
   - A persons.html fájlban a táblázat fölé helyezzünk el egy linket, ami a `"@{'/persons/new'}"` helyre vezet
   - A korábbiaknak megfelelően készítsünk a PersonControllerben új végpontot a fenti link alapján

```
@GetMapping("/persons/new")
String newPerson(Model model){
    model.addAttribute("newPerson", new Person());
    return "newPersonForm";
}
```

- Készítsük el a newPersonForm.html állományt és inicializáljuk a korábban már elkészült két másik fájl alapján.
- Hozzunk létre BootStrap segítségével egy formot, melyben egy leendő Person objektum minden attribútumát bekérjük. Segítség: https://getbootstrap.com/docs/4.0/components/forms/
- A form végponti műveletét a PersonController-ben majd a save matódus látja el, adjuk ezért a formhoz a `th:action="@{/persons/save}"` beállítást
- A művelet POST http metódust küld a PersonController-nek, ezért szintén használjuk a `method=post` beállítást
- A form mezőinél opcionálisan használjunk html validálást
- Hozzuk létre a PersonControllerben a /persons/save POST alapú végpontot

```
@PostMapping
String savePerson(Person person){
    personRepository.save(person);
    return "redirect:/persons";
}
```

- A paraméterkét szereplő person objektumot a formában fogjuk létrehozni. Adjuk hozzá a form attribútumaihoz az azt leíró html fájlban: `th:object="${newPerson}"`
- Illetve a form inputjaiban használjuk a `th:field="*{lastName}"` attribútumot mindig a megfelelő mezővel

10. Egészítsük ki az alkalmazást törlési lehetőséggel
    - Egészítsük ki a persons.html fájl táblázatát egy új oszloppal. A fejléc legyen üres, a cellák tartalma pedig minden sorban egy „Delete” tartalmú link
    - A link az alábbi végpontra mutasson:
      `th:href="@{'/persons/delete/' + ${person.id}}"`
    - Hozzuk létre a fenti végpontot a PersonControllerben. Az id hozzáadásához használjunk @PathVariable annotációt

```
@GetMapping("/persons/delete/{id}")
String deletePerson(@PathVariable Integer id){
    Person personToDelete = personRepository.findById(id).get();
    if (personToDelete!=null)
        personRepository.delete(personToDelete);
    return "redirect:/persons";
}
```

Egy nagyon hasznos csatorna:
https://www.youtube.com/playlist?list=PLR2yPNIFMlL9MDXaZETXMhI7E-Uf-spZU