Laborator 6: Tymeleaf, teste, validări

MODEL OBJECTS

Model Attributes

În tymeleaf *model attributes* se numesc *context variables* și pot fi accesate cu următoarea sintaxă din Spring EL (Spring Expression Language):

${attributeName}

O proprietate a unui model attribute se poate accesa astfel:

${attributeName.property} sau *{property}

```
@RequestMapping("/film/new")
 public String newFilm(Model model){
      model.addAttribute("film", new Film());
      return "new";
 }

@PostMapping("film")
public String saveOrUpdate(@ModelAttribute Film film){
      Film savedFilm= filmService.saveFilm(film);
      return "redirect:/filme" ;
}
```

1. Improtați proiectul LAB6_START.
2. Adugați pachetul apbdoo.laborator6.commands
3. Adaugați un obiect de tip command CategorieCommand.
4. Adnotați clasa CategorieCommand astfel încânt să fie generate metode de tip setter și getter și un constructor fără argumente.

```
package apbdoo.laboratorul6.commands;

public class CategorieCommand {
    private Long id;
    private String denumire;
}
```

Command objects sunt obiectele expuse în web-forms. Nu este recomandată expunerea directă a *domain objects*
Converter (org.springframework.core.convert.converter.Converter) este o interfață ce va fi utilizată pentru următoarele conversii:

CommandObject → **Converter** → Entity → Database
Database → Entity → **Converter** → CommandObject

5. Creați pachetul apbdoo.laboratorul6.converters și clasa CategorieToCommandConverter.

```
@Component
public class CategorieToCommandConverter
        implements Converter<Categorie, CategorieCommand> {
    @Nullable
    @Override
    public CategorieCommand convert(Categorie categorie) {
        if (categorie == null)
            return null;
        …
        return categorieCommand;
    }
}
```

6. Creați în pachetul test.java.apbdoo.laboratorul6.converters teste pentru conversie. Folosiți adnotările @Before, @Test, @Slf4j și *metodele assertNotNull, assertNull, assertEquals.* Se va testa dacă după conversie toate câmpurile obiectului de tip command sunt inițializate corect și dacă nu se aruncă NPE în cazul în care se încearcă conversia unui obiect null.

```
public class CategorieToCommandConverterTest {
    public static final String denumire = "denumire categorie";
    public static final Long id = 1L;
    CategorieToCommandConverter categorieToCommandConverter;

    @Before
    public void initTest(){
        …
    }

    @Test
    public void convertNull(){
        …
    }

    @Test
    public void convert() {
       …
    }
}
```

Adnotări junit:

@Before metoda este executată înainte de fiecare test
@After metoda este executată după fiecare test
@BeforeClass metoda este executată o singură dată înainte de efectuarea testelor
@AfterClass metoda este executată o singură dată, după efectuarea testelor

7. Similar creați clasa CommandToCategorieConverter.
8. Utilizați formularul newctegorie.html pentru a crea o nouă categorie. Adaugați sau modificați serviciul, repository-ule și controller-le necesare. Spre exemplu adăugați în CategorieService metoda saveCategorieCommand:

```
public interface CategorieService {
       CategorieCommand saveCategorieCommand(CategorieCommand
categorieCommand);
}
```

9. Adăugați un test pentru serviciul CategorieService in pachetul test.java.apbdoo.laboratorul6.services

```
@RunWith(SpringRunner.class)
@SpringBootTest
public class CategorieServiceTest {
    public static final String DENUMIRE = "denumire";

    @Autowired
    CategorieService categorieService;

    @Autowired
    CategorieRepository categorieRepository;

    @Autowired
    CategorieToCommandConverter categorieToCommandConverter;

    @Autowired
    CommandToCategorieConverter commandToCategorieConverter;

    @Test
    public void testSaveCategorie(){
    }
}
```

Adnotări clase pentru teste Spring:

@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration(classes=MyApp.class, loader=SpringApplicationContextLoader.class)

@RunWith(SpringJUnit4ClassRunner.class)
@SpringApplicationConfiguration(MyApp.class)

@RunWith(SpringRunner.class)
@SpringBootTest

VALIDARI

Java bean validation API implemented in Hibernate
http://hibernate.org/validator/

Independentă de nivelul web sau persistence
Se pot valida:

- ❑ Lungimea @Size
- ❑ Numere @Min @Max
- ❑ Expresii regulate @Pattern
- ❑ Required @NotNull

Pași

- ❑ Se definește regula de validare în clasa din domain/repository
- ❑ Se afișează pagina de eroare
- ❑ Se face validarea de către Controller

Dependențe de adăugat (incluse in **spring-boot-starter-web**):

```
<dependency>
   <groupId>org.hibernate.validator</groupId>
   <artifactId>hibernate-validator</artifactId>
   <version>6.0.2.Final</version>
</dependency>
<dependency>
   <groupId>org.hibernate.validator</groupId>
   <artifactId>hibernate-validator-annotation-processor</artifactId>
   <version>6.0.2.Final</version>
</dependency>
```

10. Adnotați atributul denumire al clasei categorie astfel încât acesta să fie obligatoriu.

```
@NotNull(message = "required field")
@Size(min=1, message = "required")
private String denumire;
```

11. Modificați CategorieController astfel încât să fie efectuate validări:

```
@PostMapping("categorie")
public String saveCategorie(@Valid @ModelAttribute
CategorieCommand categorieCommand,
BindingResult bindingResult){
        if (bindingResult.hasErrors()){
        //    return "error";
              return "newcategorie";
        }
        else {

categorieService.saveCategorieCommand(categorieCommand);
            return "redirect:/filme";
        }
    }
```

12. Adnotați atributul minute din clasa Film astfel încât să fie permise doar valori între 0 și 200.

```
@Min(value=0, message = "min value 0")
@Max(value=200, message = "max value 200")
private Integer minute;
```