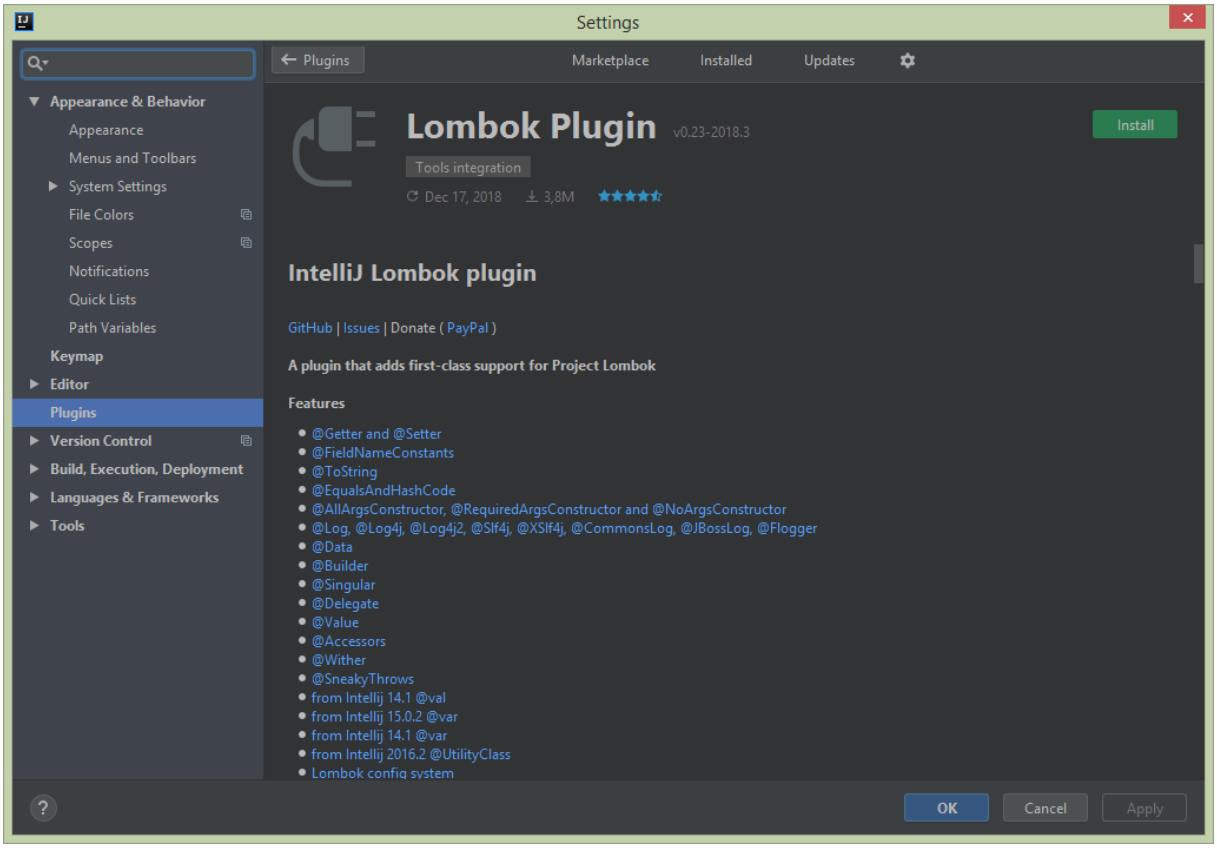
Laborator 4: Operații CRUD (create, read, update, delete)

1. Importați proiectul laborator_4.

2. Adăugați din File-Settings plugin-ul Lombok și restartați IntelliJ.



Lombok Spicing up your Java

Folosește adnotări pentru generare de cod.
Suportat în IntelliJ, Eclipse, Netbeans prin plug-inuri. Fără utilizarea plugin-urilor se vor obține erori de compilare.

@Getter generează metode getter pentru toate proprietățile.
@Setter generează metode setter pentru toate propritățile care nu sunt declarate **final.**
@ToString generează metoda toString, poate include în string-ul returnat de toString numele proprietăților și valoarea returnată de metoda toString a superclasei.

@EqualsAndHashCode generează metode equals și hashCode pornind de la valorile proprietăților ne-statice, non-transient, se pot specifica proprităţile a căror valoare va fi folosită.
@NoArgsConstructor generează constructor fără argumente.
@RequiredArgsConstructor generează constructor adăugând argumente pentru toate proprietățile adnotate @NonNull
@Data include @Getter @Setter @ToString @RequiredArgsConstructor, nu include construtori dacă nu există constructori declarați explicit.

@Value asemănător cu @Data dar include private și final pentru toate proprităţile.

@Builder implementează un pattern de tip builder care poate fi utilizat ca în exemplul urmptor: Film.builder().titlu("title").cast(castList)

@Log and @Slf4j

Dependența Maven:

```
<dependency>
        <groupId>org.projectlombok</groupId>
        <artifactId>lombok</artifactId>
</dependency>
```

3. Adăugați adnotarea @Data clasei apbdoo.laboratorul4.Cast utilizând din meniu Refactor – Lombok. Este opțiunea activă și pentru clasa apbdoo.laboratorul4.Film? Comentați metoda setInfo, refactorizați utilizând Lombok și decomentați metoda setInfo.

4. Adăugați adnotarea @Slf4j în clasa GenerareFilme

```
@Slf4j
public class GenerareFilme implements
ApplicationListener<ContextRefreshedEvent> {
…
    @Override
    public void onApplicationEvent(ContextRefreshedEvent
contextRefreshedEvent) {
        log.debug("context refreshed");
        filmRepository.saveAll(getFilme());
    }
```

5. Includeți în lista de filme un buton care să navigheze către pagina de crearea a unui film nou. În filme.html adăugați:

```
<a href="#" th:href="@{'/film/new'}">
    <button class="btn btn-primary">New</button>
</a>
```

6. Modificați redirectarea care se face după adăugarea unui film nou, astfel încât pagina afișată să fie filme.html.

```
@PostMapping("film")
public String saveOrUpdate(@ModelAttribute Film film){
        Film savedCommand = filmService.saveFilm(film);

        //return "redirect:/film/info/" + film.getId();
        return "redirect:/filme" ;
}
```

7. Adăugați în info.html un buton care să permită vizualizarea distribuției unui film.

```
<div class="col-md-1">
  <a class="btn btn-default" href="#" th:href="@{'/film/' +
${film.id} + '/cast'}" role="button">Detalii</a>
</div>
```

Atenție la atributele *bootstrap* . col-md-10 va specifica procentul ocupat de coloană (10 din 12 diviziuni), *md* se referă la dimensiunea ecranului. Alte opțiuni sunt: *xs*, *sm*, *lg*.

Bootstrap este un framework pentru dezvoltarea *front-end* a aplicațiilor web. Conține template-uri html, css și componente JavaScript. Pentru a se putea folosi într-un template tymeleaf, bootstrap se va include ca dependență web-jar.
WebJars sunt jar-uri disponibile în repositoriuri Maven.
Spre exemplu putem include în fișierul pom:

```
<dependency>
        <groupId>org.webjars</groupId>
        <artifactId>bootstrap</artifactId>
        <version>3.3.7-1</version>
</dependency>
```

Puteți consulta lista *webjar*-urilor disponibile la url-ul:
https://www.webjars.org/

8. Creați un controller care să servească un request pentru distribuția unui film. Se va folosi template-ul resources/templates/cast/list.html. Cum va fi inițializat atributul filmService?

```
@GetMapping("/film/{filmId}/cast")
public String listCast(@PathVariable String filmId, Model model){
        model.addAttribute("film",
filmService.findById(Long.valueOf(filmId)));
        return "cast/list";
    }
```

9. Creați o interfața CastService cu următoarele metode:

```
public interface CastService {
    Cast findByFilmIdAndCastId(Long filmId, Long castId);
    Cast saveCast(Cast cast);
    void deleteById(Long filmId, Long castId);
}
```

10. Implementați interfața CastService, creați o clasă de tip service cu următoarele atribute: FilmRepository filmRepository și InfoActorRepository infoActorRepository. Cum vor fi inițializate atributele filmRepository si infoActorRepository? Ce metode trebuie implementate?

```
@Service
public class CastServiceImpl implements CastService{
    FilmRepository filmRepository;
    InfoActorRepository infoActorRepository;
}
```

11. Implementați metoda public Cast findByFilmIdAndCastId(Long filmId, Long castId). Afișați un mesaj de tip *log* în cazul în care nu există nici un film cu *id*-ul specificat.

```
@Override
public Cast findByFilmIdAndCastId(Long filmId, Long castId) {
        Optional<Film> filmOptional =
filmRepository.findById(filmId);
        Film film;
        Cast cast = null;
        if (filmOptional.isPresent()) {
            film = filmOptional.get();
            for (Cast c : film.getCast()) {
                if (c.getId() == castId)
                    cast = c;
            }
        }
        else{

        }
        return cast;
    }
```

12. Adaugati în CastController:

```
@GetMapping("film/{filmId}/cast/{id}/show")
public String showFilmCast(@PathVariable String filmId,
                           @PathVariable String id, Model model){
        model.addAttribute("cast",
castService.findByFilmIdAndCastId(Long.valueOf(filmId),
Long.valueOf(id)));
        return "cast/show";
}
```

@Controller Un controller are rolul de a trata request-urile http, endpointurile fiind specificate prin adnotarea **@RequestMapping**. Cererile http sunt trimise de **DispatcherServlet** controlerului care executa o metoda si trimite un raspuns, **@ResponseBody,** intr-un anumit format.

@RequestParam asociaza parametrul unei metode unui parametru al cererii http.
@PathVariable asociaza parametrul unei metode unui element al URI-ului, dat sub forma unui template.

13. Implementați metoda Cast saveCast(Cast cast) a clasei CastServiceImpl. Metoda va fi apelată atât pentru crearea unui nou rol asociat unui film, cât și pentru adăugarea unui rol nou.

```
Optional<Cast> castOptional = film.getCast().stream()
                    .filter(c -> c.getId().equals(cast.getId()))
                    .findFirst();
```

```
if(castOptional.isPresent()){
    Cast castToSave = castOptional.get();
    castToSave.setRol(cast.getRol());
    castToSave.setInfoActor(infoActorRepository
    .findById(cast.getInfoActor().getId())
    .orElseThrow(() -> new RuntimeException("ACTOR NOT FOUND")));
            }
else {
                Cast castToSave = new Cast();
                castToSave.setFilm(film);
                castToSave.setInfoActor(infoActorRepository
                        .findById(cast.getInfoActor().getId())
                        .orElseThrow(() -> new
RuntimeException("ACTOR NOT FOUND")));
                castToSave.setRol(cast.getRol());
                film.addCast(castToSave);
            }

 Film savedFilm = filmRepository.save(film);
```

14. Adăugați interfața InfoActorService

```
public interface InfoActorService {
    Set<InfoActor> listAllActors();
}
```

15. Implementați metoda listAllActors luând ca model implementarea metodei getFilme()

```
@Override
public Set<Film> getFilme() {
        Set<Film> filme = new HashSet<Film>();

filmRepository.findAll().iterator().forEachRemaining(filme::add);
        return filme;
    }
```

16. Adăugati în CastController metoda care va fi apelată pentru adăugarea unui nou rol:

```
@GetMapping("film/{filmId}/cast/new")
    public String newCast(@PathVariable String filmId, Model
model){
    Film film = filmService.findById(Long.valueOf(filmId));
    Cast cast = new Cast();
    cast.setFilm(film);
    model.addAttribute("cast", cast);
    cast.setInfoActor(new InfoActor());
    model.addAttribute("infoActorList",
infoActorService.listAllActors());
        return "cast/castform";
    }
```

17. Adăugati în CastController metoda care va trata un request de tip POST care va fi apelată pentru înregistrarea unui nou rol:

```
@PostMapping("film/{filmId}/cast")
    public String saveOrUpdate(@ModelAttribute Cast cast){
        Cast savedCast = castService.saveCast(cast);
        return "redirect:/film/" + savedCast.getFilm().getId() +
"/cast/" + savedCast.getId() + "/show";
    }
```

18. Implementați metoda deleteById a interfeței CastService

```
@Override
public void deleteById(Long filmId, Long castId) {
        Optional<Film> filmOptional =
filmRepository.findById(filmId);
```

19. Adăugați în CastController metoda care va fi apelată pentru ștergerea unui rol.

```
@GetMapping("film/{filmId}/cast/{id}/delete")
public String deleteCast(@PathVariable String filmId,
                         @PathVariable String id){
      castService.deleteById(Long.valueOf(filmId),
Long.valueOf(id));

        return "redirect:/film/" + filmId + "/cast";
}
```

20. Adăugați interfața ImageService și o implementare a acesteia:

```
public interface ImageService {
    void saveImageFile(Long filmId, MultipartFile file);
}
```

```
package apbdoo.laboratorul5.services;
import apbdoo.laboratorul5.domain.Film;
import org.springframework.web.multipart.MultipartFile;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;
import java.io.IOException;
@Service
public class ImageServiceImpl implements ImageService{
    @Override
    @Transactional
    public void saveImageFile(Long filmId, MultipartFile file) {
        try {
            Film film = filmRepository.findById(filmId).get();
            Byte[] byteObjects = new
Byte[file.getBytes().length];

            int i = 0;
            for (byte b : file.getBytes()){
                byteObjects[i++] = b;
            }
            film.setImagine(byteObjects);
            filmRepository.save(film);
        } catch (IOException e) {
```

```
        }
    }
}
```

21. Includeți în pagina care conține informații despre un film un buton care să navigheze către pagina de upload a unei imagini. În info.html adăugați:

```
<div class="row">
   <div class="col-md-10">
      <h1 class="panel-title" th:text="${film.titlu}">Titlu</h1>
   </div>
   <div class="col-md-1">
        <a class="btn btn-default" href="#" role="button"
th:href="@{'/film/' + ${film.id} + '/image'}">Imagine</a>
   </div>
</div>
```

22. Creați o clasă de tip controller care să servească un request pentru înregistrarea imaginii unui film.

```
@Controller
public class ImageController {
    private final ImageService imageService;
    private final FilmService filmService;

    public ImageController(ImageService imageService, FilmService
filmService) {
        this.imageService = imageService;
        this.filmService = filmService;
    }

@GetMapping("film/{id}/image")
public String showUploadForm(@PathVariable String id, Model
model){
        model.addAttribute("film",
filmService.findById(Long.valueOf(id)));
        return "imageform";
    }

@PostMapping("film/{id}/image")
public String handleImagePost(@PathVariable String id,
@RequestParam("imagefile") MultipartFile file){

        imageService.saveImageFile(Long.valueOf(id), file);

        return "redirect:/film/info/" + id;
    }
}
```

23. Adăugați în info.html un tag de tip imagine.

```
<div class="col-md-6">
      <img th:src="@{'/film/' + ${film.id} + '/filmimage'}"
                                width="200" height="200">
</div>
```

24. Adăugați în clasa ImageController o metodă care să încarce imaginea asociată unui film.

```
@GetMapping("film/{id}/filmimage")
    public void downloadImage(@PathVariable String id,
HttpServletResponse response) throws IOException {
        Film film = filmService.findById(Long.valueOf(id));

        if (film.getImagine() != null) {
            byte[] byteArray = new
byte[film.getImagine().length];
            int i = 0;

            for (Byte wrappedByte : film.getImagine()){
                byteArray[i++] = wrappedByte;
            }

            response.setContentType("image/jpeg");
            InputStream is = new ByteArrayInputStream(byteArray);
            try {
                IOUtils.copy(is, response.getOutputStream());
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
```

HttpServletResponse și HttpServletRequest sunt obiecte care încapsulează, alături de conținutul mesajelor http trimise/primite de la server alte informații care trebuie incluse în *header*-ul requestului sau în *cookie*-uri. Se poate specifica de asemenea tipul conținutului mesaj trimis (json, imagine, text, etc.) și *status*-ul mesajului. (SC_OK, SC_UNAUTHORIZED, SC_FORBIDDEN etc.)