OTH-Regensburg
Übungen zur Vorlesung
Softwareentwicklung

Übung Nr. 7
Pagination with Bootstrap and Thymeleaf Dialect

Aufgabe 1 – Erstellen eines neuen Projekts und der Abhängigkeiten

• Importieren Sie die folgenden Abhängigkeiten, um Spring Security und JPA zu verwenden:

```
<!-- https://mvnrepository.com/artifact/org.springframework.data/spring-data-jpa -->
                            <dependency>
                                          <groupId>org.springframework.boot</groupId>
                                          <artifactId>spring-boot-starter-data-jpa</artifactId>
                            </dependency>

                            <dependency>
                              <groupId>com.h2database</groupId>
                              <artifactId>h2</artifactId>
                              <scope>runtime</scope>
                            </dependency>

                            <!-- Spring security -->
                            <dependency>
                                           <groupId>org.springframework.security</groupId>
                            <artifactId>spring-security-web</artifactId>
                            </dependency>

                            <dependency>
                            <groupId>org.springframework.security</groupId>
                            <artifactId>spring-security-config</artifactId>
```

Stellen Sie sicher, dass Sie auch die Abhängigkeiten für Thymeleaf, Thymeleaf-Dialekt, Spring Web, Bootstrap, Jquery usw. haben (kopieren Sie sie hier aus der POM-Datei in diesem Projekt: https://xxx)

Fügen Sie der Datei application.properties diese Werte hinzu:

```
spring.main.allow-bean-definition-overriding=true

spring.thymeleaf.cache=false
spring.thymeleaf.prefix=classpath:/templates/
spring.thymeleaf.suffix=.html
spring.thymeleaf.mode=HTML
spring.thymeleaf.encoding=UTF-8

spring.messages.basename=messages
server.error.include-binding-errors=always

#DB
spring.datasource.url=jdbc:h2:mem:testdb;MODE=MySQL;NON_KEYWORDS=USER;
spring.datasource.driverClassName=org.h2.Driver
spring.datasource.username=sa
spring.datasource.password=password

spring.jpa.database-platform=org.hibernate.dialect.H2Dialect
spring.jpa.hibernate.ddl-auto=create-drop

#spring.data.jpa.repositories.bootstrap-mode=default

spring.jpa.defer-datasource-initialization=true

spring.h2.console.enabled=true
spring.h2.console.path=/h2-console
spring.jpa.show-sql=true
#hibernate.auto_quote_keyword=true
```

- Erstellen Sie einen HomeController im Paket-Controller, der ein @Getmapping „/home“ zur Ansicht „home.html“ hat.

- Führen Sie das Projekt aus. Versuchen Sie, auf „/home“ zuzugreifen, und prüfen Sie, ob Sie das von Spring Security bereitgestellte Login-View sehen. Sie können versuchen, sich mit „user“ und dem von Spring Security generierten Passwort im Terminal anzumelden.

Aufgabe 2- User, Authority, Role und zugehörige Klassen, die kopiert werden sollen (nicht Schwerpunkt der Paginierungsübung)

Kopieren Sie die folgenden Dateien aus unserem Projekt in Elo:

package model: User, Role and Authority, Course, Address
package config: MyUserDetails, SecurityConfig.
package repository: MyBaseRepository, CouseRepositoryI and UserRepositoryI.
package repository.impl; CouseRepositoryImpl, UserRepositoryImpl.
package service: CourseServiceI.
package service.impl: CourseServiceImpl.
package utils: GenderEnum.
Aus dem Ordner /resources: das ganze Verzeichnis “/fragments” and the views “/students/student-all”, “home” und “layout”.

Aufgabe 3 : Erstellen Sie den Klassen Student

```
@Entity
@Table(name="student")
public class Student extends User implements Serializable{

        private static final long serialVersionUID = 1L;
        @Id
        Long id;
        @NotBlank(message = "Name is mandatory")
        private String name;

        @Enumerated(EnumType.STRING)
        private GenderEnum gender;

        @OneToOne(cascade = CascadeType.ALL)
        @JoinColumn(name = "address_id", referencedColumnName = "id")
        private Address address;

        @ManyToOne(cascade = CascadeType.PERSIST)
        @JoinColumn(name = "course_id", referencedColumnName = "id")
        private Course course;

        public Student() {
                Address address= new Address();
                this.setAddress(address);
                this.setId((long) -1);

                Course course = new Course();
                this.setCourse(course);

        }

//getters and setters
```

Aufgabe 4 – Erstellen Sie StudentRepositoryI und StudentRepositoryImp

- • Dieses Mal verwendet das Repository die Klasse Page von Spring JPA als Rückgabetyp und auch ein PageableObject als Parameter.
- • Die Methode „findByNameContainingIgnoreCase“ verwendet die Texteingabe von dem student-all View.

```
public interface StudentRepositoryI extends MyBaseRepository<Student, Long>{

        List<Student> findByNameContainingIgnoreCase (String name);
        Page <Student> findAll(Pageable pageable);
        Page <Student> findByNameContainingIgnoreCase (String name, Pageable pageable);


}
```

• Die Implementierung von StudentRepository erweitert die Schnittstelle PagingAndSortingRepository von JPA, die es uns ermöglicht, mit Seiten und Paginierung zu arbeiten.

• Vergessen Sie nicht die Annotation @Repository unten:

```
@Repository
public interface StudentRepositoryImp extends  StudentRepositoryI,
PagingAndSortingRepository<Student, Long>{

        List<Student> findByNameContainingIgnoreCase (String name);
        Page<Student>  findAll(Pageable pageable);
        Page <Student> findByNameContainingIgnoreCase (String name, Pageable pageable);

}
```

Aufgabe 5 -Erstellen Sie den StudentServiceI und StudentServiceImpl

```
public interface StudentServiceI {

        Page<Student> getAllStudents(String name, Pageable pageable);

        List<Student> findStudentsByName(String name);

        Student saveStudent(Student student);

        Student getStudentById(Long id);

        Student updateStudent(Student student);

        void delete(Student student);


}
```

• Die Methode getAllStudents unten erhält einen String-Namen und das Pageable-Objekt als Parameter.
• Das Pageable-Objekt wird vom StudentController basierend auf den Parametern „page“ und „size“ erstellt, die in dem Student-all-View gesendet werden.
• Der Name-Parameter wird ebenfalls vom Controller bereitgestellt und ursprünglich von dem Student-all View gesendet.

```
@Service
public class StudentServiceImpl implements StudentServiceI{

		private StudentRepositoryI  studentRepository;
		private CourseRepositoryI  courseRepository;

		public StudentServiceImpl(StudentRepositoryI studentRepository, CourseRepositoryI  courseRepository) {
				super();
				this.studentRepository = studentRepository;
				this.courseRepository = courseRepository;
		}

		@Override
		public Page<Student> getAllStudents(String name, Pageable pageable) {
				// TODO Auto-generated method stub
				Page <Student> pageStudents;
				if (name == null) {
				  pageStudents = studentRepository.findAll(pageable);
				} else {
				  pageStudents = studentRepository.findByNameContainingIgnoreCase(name, pageable);

				}
				return  pageStudents;
		}

		@Override
		public Student saveStudent(Student student) {
				// TODO Auto-generated method stub

				return studentRepository.save(student);
		}

		@Override
		public Student getStudentById(Long id) {
				// TODO Auto-generated method stub
				return studentRepository.findById(id).get();
		}

		@Override
		public Student updateStudent(Student student) {
				// TODO Auto-generated method stub
				System.out.println(student.getGender()+"***");
				return studentRepository.save(student);
		}

		@Override
		public void delete(Student student) {
				// TODO Auto-generated method stub
				studentRepository.delete(student);
		}

		@Override
		public List<Student> findStudentsByName(String name) {
				// TODO Auto-generated method stub
				return studentRepository.findByNameContainingIgnoreCase(name);
		}

}
```

Aufgabe 6 -Erstellen Sie den StudentController

Wir konzentrieren uns hier auf die Studentenlistenansicht. Create, update und delete liegen außerhalb dieses Kontexts. Wir haben sie bereits in früheren Übungen gesehen.

```
@RequestMapping(value = "/student")
@Controller
public class StudentController {

        private StudentServiceI studentService;
        private CourseServiceI courseService;

        public StudentController(StudentServiceI studentService,
                        CourseServiceI courseService) {
                super();
                this.studentService = studentService;
                this.courseService= courseService;
        }

        @GetMapping(value = {"", "/all"})
         public String showUserList(Model model, @RequestParam(required = false) String keyword,
                @RequestParam(required = false, defaultValue = "1") int page, @RequestParam(required = false,
                defaultValue = "5") int size) {

          try {
                   List<Student> students = new ArrayList<Student>();

                  //the first page is 1 for the user, 0 for the database.
                   Pageable paging = PageRequest.of(page - 1, size);
                   Page<Student> pageStudents;
                   //getting the page from the database....
                   pageStudents = studentService.getAllStudents(keyword, paging);

                    model.addAttribute("keyword", keyword);

                    students = pageStudents.getContent();
                    model.addAttribute("students", students);
                   //here are the variables for the paginator in the student-all view
                    model.addAttribute("entitytype", "student");
                    model.addAttribute("currentPage", pageStudents.getNumber() + 1);
                   model.addAttribute("totalItems", pageStudents.getTotalElements());
                   model.addAttribute("totalPages", pageStudents.getTotalPages());
                   model.addAttribute("pageSize", size);

                  } catch (Exception e) {
                        model.addAttribute("message", e.getMessage());
           }

         return "/students/student-all";
      }

//....other methods for create, update and delete a student, select  a course etc....
}
```

• Stellen Sie sicher, dass Sie verstehen, was oben passiert, insbesondere die Einführung der Werte, die der Paginator benötigt, um die Paginierungskomponente anzuzeigen.

Aufgabe 7 -Erstellen Sie das student-all View

```
<!DOCTYPE html>
<html layout:decorate="~{layout}">
<head>
<meta charset="ISO-8859-1">
<title>Academic Management System - Add Student</title>
<script type="text/javascript">
                $(document).ready(function () {
                window.setTimeout(function() {
                   $(".alert").fadeTo(1000, 0).slideUp(1000, function(){
                      $(this).remove();
                   });
                }, 5000);
                });
 </script>
</head>
<body>

<section class="layout-content" layout:fragment="mybody">
                      <div class="page-header">
                         <nav class="navbar navbar-expand-md navbar-top bg-light">
                            <div class="collapse navbar-collapse" id="navbarsExampleDefault">
                               <ul class="navbar-nav">
                                  <li class="nav-item active">
                                     <i class="oi oi-caret-right"></i>
                                     <span>List Students</span>
                                  </li>
                               </ul>
                            </div>

                         </nav>
                      </div>

     <div class="container" id="studentmessages">

                <div th:replace="~{fragments/alert::alert}">
                </div>
                <div class = "row">
                                <div class = "col-lg-3">
                                                <a th:href = "@{/student/add}" class = "btn btn-primary btn-sm mb-3"> Add Student</a>
                                </div>
                </div>
                <div th:unless="${students.size() > 0}" style="width: 90%">
                                <span>No students found!</span>
                </div>

<!-- 2o part: search form -->

   <div>
                 <form th:action="@{/student}" id="searchForm" method="get">
                   <div class="row d-flex">
                     <div class="col-md-6 mt-2">
                       <div class="search">
                         <i class="fa fa-search"></i>
                         Name: <input id="keyword" type="search" name="keyword" th:value="${keyword}" required class="form-control"
                           placeholder="Enter keyword">

                       </div>
                     </div>

                     <div class="col-md-3 input-group mt-2">
                       <div class="input-group-prepend">
                         <label class="input-group-text" for="pageSize">Items per page:</label>
                       </div>
                       <select form="searchForm" name="size" th:value="${pageSize}" onchange="changePageSize()" class="size-select"
                         id="pageSize">
                         <option th:each="s : ${ {3, 6, 9} }" th:value="${s}" th:text="${s}" th:selected="${s == pageSize}"></option>
                       </select>
                     </div>

                     <div><button type="submit" class="btn btn-secondary">Search</button></div>

                   </div>
                 </form>
    </div>
```

Cont.

```
<!-- 3o part: the table with the list of students, by default, it has the first 5 students  -->

        <br><br>

        <table class = "table table-striped table-bordered" th:unless="${students.size()<1}" style="width: 90%">
                <thead class = "table-dark">
                        <tr>
                                        <th> Name</th>
                                        <th> Gender</th>
                                        <th> Email </th>
                                        <th> Course </th>
                                        <th> ZPL </th>
                                        <th> Actions </th>
                                </tr>
                        </thead>

                        <tbody>
                                <tr th:each = "student: ${students}">
                                        <td th:text = "${student.name}"></td>
                                        <td th:text = "${student.gender}"></td>
                                        <td th:text = "${student.email}"></td>
                                        <td th:text = "${student.course.description}"></td>
                                        <td th:text = "${student.address.ZPL}"></td>
                                        <td>
                                                <a th:href = "@{/student/update/{id}(id=${student.id})}"
                                                                                                 class = "btn btn-
                                                primary"><svg xmlns="http://www.w3.org/2000/svg" width="16" height="16"
                                                fill="currentColor" class="bi bi-pen" viewBox="0 0 16 16">
                                                                                                        <path
                                                d="m13.498.795.149-.149a1.207 1.207 0 1 1 1.707 1.708l-.149.148a1.5 1.5 0 0 1-
                                                .059 2.059L4.854 14.854a.5.5 0 0 1-.233.131l-4 1a.5.5 0 0 1-.606-.606l1-4a.5.5 0 0 1
                                                .131-.232l9.642-9.642a.5.5 0 0 0-.642.056L6.854 4.854a.5.5 0 1 1-.708-
                                                .708L9.44.854A1.5 1.5 0 0 1 11.5.796a1.5 1.5 0 0 1 1.998-.001zm-.644.766a.5.5 0 0
                                                0-.707 0L1.95 11.756l-.764 3.057 3.057-.764L14.44 3.854a.5.5 0 0 0 0-.708l-1.585-
                                                1.585z"/>
                                                </svg></a>

                                                <a th:href = "@{/student/delete/{id}(id=${student.id})}"
                                                                                                 class = "btn btn-
                                                danger"><svg xmlns="http://www.w3.org/2000/svg" width="16" height="16"
                                                fill="currentColor" class="bi bi-trash" viewBox="0 0 16 16">
                                                                                                 <path d="M5.5
                                                5.5A.5.5 0 0 1 6 6v6a.5.5 0 0 1-1 0V6a.5.5 0 0 1 .5-.5zm2.5 0a.5.5 0 0 1 .5.5v6a.5.5 0
                                                0 1-1 0V6a.5.5 0 0 1 .5-.5zm3 .5a.5.5 0 0 0-1 0v6a.5.5 0 0 0 1 0V6z"/>
                                                                                                 <path fill-
                                                rule="evenodd" d="M14.5 3a1 1 0 0 1-1 1H13v9a2 2 0 0 1-2 2H5a2 2 0 0 1-2-2V4h-
                                                .5a1 1 0 0 1-1-1V2a1 1 0 0 1 1-1H6a1 1 0 0 1 1-1h2a1 1 0 0 1 1 1h3.5a1 1 0 0 1 1
                                                1v1zM4.118 4 4 4.059V13a1 1 0 0 0 1 1h6a1 1 0 0 0 1-1V4.059L11.882
                                                4H4.118zM2.5 3V2h11v1h-11z"/>
                                                </svg></a>

                                        </td>
                                </tr>
                        </tbody>
                </table>
```

Cont…

```
<!-- 4th part : the  paginator component -->

                    <nav aria-label="Pagination" th:if="${totalPages > 0}">
                     <ul class="pagination justify-content-center">
                      <li class="page-item" th:classappend="${currentPage == 1} ? 'disabled'">
                       <a th:replace="~{fragments/paging :: paging(1, '<<', 'First Page')}"></a>

                      </li>
                      <li class="page-item font-weight-bold" th:classappend="${currentPage == 1} ? 'disabled'">
                       <a th:replace="~{fragments/paging :: paging(${currentPage - 1}, 'Prev', 'Previous Page')}"></a>
                      </li>
                      <li class="page-item disabled" th:if="${currentPage - 2 > 1}">
                       <a class="page-link" href="#">...</a>
                      </li>
                      <li class="page-item" th:classappend="${page == currentPage} ? 'active'"
                       th:each="page : ${#numbers.sequence(currentPage > 2 ? currentPage - 2 : 1, currentPage + 2 < totalPages ?
                       currentPage + 2 : totalPages)}">
                       <a th:replace="~{fragments/paging :: paging(${page}, ${page}, 'Page ' + ${page})}"></a>
                      </li>
                      <li class="page-item disabled" th:if="${currentPage + 2 < totalPages}">
                       <a class="page-link" href="#">...</a>
                      </li>
                      <li class="page-item font-weight-bold" th:classappend="${currentPage == totalPages} ? 'disabled'">
                       <a th:replace="~{fragments/paging :: paging(${currentPage + 1},'Next', 'Next Page')}"></a>
                      </li>
                      <li class="page-item" th:classappend="${currentPage == totalPages} ? 'disabled'">
                       <a th:replace="~{fragments/paging :: paging(${totalPages}, '>>', 'Last Page')}"></a>
                      </li>
                     </ul>
</nav>

                    <br><br>

          </div> <!--container-->
</section>

<!—End of Table and Pagination Bar -->

<script type="text/javascript">
 function changePageSize() {
  $("#searchForm").submit();
 }
</script>

</body>
</html>
```

- Versuchen Sie zu verstehen, was die Bedingungen der obigen Paginator-Komponente bedeuten.

- Der obige Paginator-Code verwendet das unten stehende Fragment „paging“, um die URLs zu generieren:

```
<a th:fragment="paging(pageNum, label, tooltip)" class="page-link"
  th:href="@{'/'+ ${entitytype} +'?' + ${keyword!=null && keyword!=''? 'keyword=' +
keyword + '&' : ''} + 'page=' + ${pageNum} + '&size=' + ${pageSize}}"
  th:title="${tooltip}" rel="tooltip">
  [[${label}]]
</a>
```

Aufgabe 8 -Erstellen der Testdaten in der Datenbank

- Jetzt ist es an der Zeit, die Datenbank mit einigen Benutzern und Berechtigungen zu füllen. Erstellen Sie die folgende Datei data.sql:

```
INSERT INTO USER (email, password, login, active) VALUES ( 'thomas@gmail.com',
$2a$12$8K4uC9YPl659Qnz6NUqy9e35xsoQ/OIsaVhIWRJP913VpsulQGZNy', 'thomas',
1);
INSERT INTO USER (email, password, login, active) VALUES ( 'anja@gmail.com',
'$2a$12$8K4uC9YPl659Qnz6NUqy9e35xsoQ/OIsaVhIWRJP913VpsulQGZNy', 'anja', 1);

INSERT INTO AUTHORITY (description) VALUES ( 'ADMIN');
INSERT INTO AUTHORITY (description) VALUES ( 'STUDENT');

INSERT INTO USERAUTHORITY (IDUSER, IDAUTHORITY) VALUES ( 1, 1);
INSERT INTO USERAUTHORITY (IDUSER, IDAUTHORITY) VALUES ( 2, 2);
```

Aufgabe 9 – Testen Sie die Paginierung

- localhost:8080/student/all
- User: thomas, password: 123456.

Gute Arbeit!