

# Отчет по лабораторной работе 5

Студент: Зернов Иван и Рябков Владислав

Группа: ПИМ-22

## 1. Постановка задачи

В процессе выполнения лабораторной работы необходимо выполнить следующие задачи:

1. Метод добавления автора
2. Вывод всех книг по ид автора
3. Удаление книги
4. Удаление автора

## 2. Разработка задачи

### 2.1 Структура проекта

Проект разделен на следующие директории:

#### **docs**

Документация

#### **src**

Исходный код лабораторной работы

#### **src/main/java/ru/rsatu/hibernatetutorial/mapper**

Мапперы Dto в Entity и наоборот

#### **src/main/java/ru/rsatu/hibernatetutorial/pojo/entity**

Entity **Book** и **Author**

#### **src/main/java/ru/rsatu/hibernatetutorial/pojo/dto**

DTO для Entity **Book** и **Author**

#### **src/main/java/ru/rsatu/hibernatetutorial/repository**

Репозитории для сущностей **Book** и **Author**

#### **src/main/java/ru/rsatu/hibernatetutorial/resource**

Ресурсы **BookResource** и **AuthorResource**

#### **src/main/java/ru/rsatu/hibernatetutorial/service**

Сервисы для работы с сущностями **Author** и **Book**

## 3. Информация о реализации

### 3.1 Задание 1

Для выполнения первого задания созданы следующие классы:

- AuthorDto - Dto для entity **Author**.
- AuthorMapper - Mapper, для entity **Author** и dto **AuthorDto** преобразующий их между собой
- AuthorsRepository - Репозиторий, обращающийся к базе данных для entity **Author**. Содержащий методы **saveAuthor** и **getAuthors** для создания автора и получения списка всех авторов
- AuthorsService - Сервис, через который взаимодействуют **AuthorResource**, **AuthorsRepository** и **AuthorMapper**, содержащий аналогичные методы.
- AuthorResource - Ресурс, через который можно взаимодействовать с сущностью **Author** по http., обрабатывающий запросы **GET /books/api/v1/author** и **POST /books/api/v1/author**

Листинг 1. AuthorDto

```
package ru.rsatu.hibernate tutorial.pojo.dto;
import lombok.Getter;
import lombok.Setter;
@Getter
@Setter
public class AuthorDto {
    private Long id;
    private String name;
    private String birthDate;
    private String nickName;
}
```

Листинг 2. AuthorMapper

```
package ru.rsatu.hibernate tutorial.mapper;
import org.mapstruct.Mapper;
import org.mapstruct.Mapping;
import ru.rsatu.hibernate tutorial.pojo.dto.AuthorDto;
import ru.rsatu.hibernate tutorial.pojo.entity.Author;
@Mapper(componentModel = "cdi")
public abstract class AuthorMapper {
    @Mapping(target = "id", source = "id")
    @Mapping(target = "name", source = "name")
    @Mapping(target = "birthDate", source = "birthDate", dateFormat = "dd.MM.yyyy")
    @Mapping(target = "nickName", source = "nickName")
    public abstract AuthorDto toAuthorDto(Author author);
    @Mapping(target = "id", source = "id")
    @Mapping(target = "name", source = "name")
    @Mapping(target = "birthDate", source = "birthDate", dateFormat = "dd.MM.yyyy")
}
```

```

    @Mapping(target = "nickName", source = "nickName")
    public abstract Author toAuthor(AuthorDto authorDto);
}

```

### Листинг 3. AuthorsRepository

```

package ru.rsatu.hibernate.tutorial.repository;
import ru.rsatu.hibernate.tutorial.mapper.AuthorMapper;
import ru.rsatu.hibernate.tutorial.pojo.dto.AuthorDto;
import ru.rsatu.hibernate.tutorial.pojo.entity.Author;
import javax.enterprise.context.ApplicationScoped;
import javax.inject.Inject;
import javax.persistence.EntityManager;
import javax.transaction.Transactional;
import java.util.List;
/**
 * Репозиторий для работы с авторами
 */
@ApplicationScoped
public class AuthorsRepository {
    @Inject
    EntityManager entityManager;
    @Inject
    AuthorMapper authorMapper;
    /**
     * Получить всех авторов
     */
    public List<Author> getAuthors() {
        return entityManager.createQuery("select a from Author a", Author.class)
            .getResultList();
    }
    /**
     * Сохранение автора
     */
    @Transactional
    public Author saveAuthor(AuthorDto authorDto) {
        Author author = authorMapper.toAuthor(authorDto);
        if (author.getId() != null) {
            entityManager.merge(author);
        } else {
            entityManager.persist(author);
        }
        entityManager.flush();
        return author;
    }
}

```

### Листинг 4. AuthorsService

```

package ru.rsatu.hibernate.tutorial.service;

```

```

import ru.rsatu.hibernate.tutorial.mapper.AuthorMapper;
import ru.rsatu.hibernate.tutorial.mapper.BookMapper;
import ru.rsatu.hibernate.tutorial.pojo.dto.AuthorDto;
import ru.rsatu.hibernate.tutorial.pojo.dto.BookDto;
import ru.rsatu.hibernate.tutorial.repository.AuthorsRepository;
import ru.rsatu.hibernate.tutorial.repository.BooksRepository;
import javax.enterprise.context.ApplicationScoped;
import javax.inject.Inject;
import java.util.List;
/**
 * Сервис для работы с авторами
 */
@ApplicationScoped
public class AuthorsService {
    @Inject
    AuthorMapper authorMapper;
    @Inject
    AuthorsRepository authorsRepository;
    /**
     * Получить всех авторов
     */
    public List<AuthorDto> getAuthors() {
        return authorsRepository.getAuthors()
            .stream()
            .map(authorMapper::toAuthorDto)
            .toList();
    }
    /**
     * Сохранение автора
     */
    public AuthorDto saveAuthor(AuthorDto authorDto) {
        return authorMapper.toAuthorDto(authorsRepository.saveAuthor(authorDto));
    }
}

```

#### Листинг 5. AuthorResource

```

package ru.rsatu.hibernate.tutorial.resource;
import ru.rsatu.hibernate.tutorial.pojo.dto.AuthorDto;
import ru.rsatu.hibernate.tutorial.service.AuthorsService;
import javax.inject.Inject;
import javax.transaction.Transactional;
import javax.ws.rs.*;
import javax.ws.rs.core.MediaType;
import java.util.List;
@Path("/books/api/v1")
public class AuthorResource {
    @Inject
    AuthorsService authorsService;
    /**

```

```

    * Получение всех авторов
    */
    @GET
    @Path("/author")
    public List<AuthorDto> getAuthors() {
        return authorsService.getAuthors();
    }
    /**
    * Сохранение автора
    */
    @POST
    @Consumes(MediaType.APPLICATION_JSON)
    @Produces(MediaType.APPLICATION_JSON)
    @Path("/author")
    public AuthorDto saveAuthor(AuthorDto authorDto) {
        return authorsService.saveAuthor(authorDto);
    }
}

```

Так же был модифицирован класс `Author`, id авторов сделаны автогенерируемыми:

Листинг 6. `Author`

```

@Getter
@Setter
@Entity
@Table(name = "authors")
public class Author {
    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE, generator = "authors_id_gen")
    @SequenceGenerator(name = "authors_id_gen", sequenceName = "authors_id_gen_seq",
        initialValue = 1, allocationSize = 10)
    private Long id;
    private String name;
    private Date birthDate;
    private String nickName;
}

```

Для проверки работоспособности добавим с помощью Swagger следующих авторов:

Листинг 7. Добавляемые авторы

```

{
  "name": "Author1",
  "birthDate": "01.01.1991",
  "nickName": "author1"
}

```

```

{

```

```
"name": "Author2",
"birthDate": "02.02.1992",
"nickName": "author2"
}
```

И запросим список всех авторов:

*Листинг 8. Результаты выполнения запроса*

```
[
  {
    "id": 1,
    "name": "Author1",
    "birthDate": "01.01.1991",
    "nickName": "author1"
  },
  {
    "id": 2,
    "name": "Author2",
    "birthDate": "02.02.1992",
    "nickName": "author2"
  }
]
```

## 3.2 Задание 2

Для выполнения этого задания модифицируем класс `BookResource` и добавим новый метод:

`GET /book/author/{authorId}`

*Листинг 9. Добавленный метод в `BookResource`*

```
@GET
@Path("/book/author/{authorId}")
public List<BookDto> getBooksByAuthor(@PathParam("authorId") Long authorId) {
    return booksService.getBooksByAuthor(authorId);
}
```

Для работы этого метода доработаем классы `BooksRepository` и `BookService` добавив в них методы `getBooksByAuthor`:

*Листинг 10. Модифицированный `BookRepository`*

```
package ru.rsatu.hibernate.tutorial.repository;
import ru.rsatu.hibernate.tutorial.mapper.BookMapper;
import ru.rsatu.hibernate.tutorial.pojo.dto.BookDto;
import ru.rsatu.hibernate.tutorial.pojo.entity.Author;
import ru.rsatu.hibernate.tutorial.pojo.entity.Book;
import javax.enterprise.context.ApplicationScoped;
```

```

import javax.inject.Inject;
import javax.persistence.EntityManager;
import javax.transaction.Transactional;
import java.util.List;
/**
 * Репозиторий для работы с книгами
 */
@ApplicationScoped
public class BooksRepository {
    @Inject
    EntityManager entityManager;
    @Inject
    BookMapper bookMapper;
    /**
     * Загрузить все книги
     */
    public List<Book> loadBooks() {
        return entityManager.createQuery("select b from Book b", Book.class)
            .getResultList();
    }
    /**
     * Получить книги автора
     */
    public List<Book> getBooksByAuthor(Long authorId) {
        return entityManager.createQuery("select b from Book b where author.id =
:authorId", Book.class).setParameter("authorId", authorId).getResultList();
    }
    /**
     * Сохранение книги
     */
    @Transactional
    public Book saveBook(BookDto bookDto) {
        Book book = bookMapper.toBook(bookDto);
        if (book.getId() != null) {
            entityManager.merge(book);
        } else {
            entityManager.persist(book);
        }
        entityManager.flush();
        return book;
    }
}

```

Листинг 11. Модифицированный BooksService

```

package ru.rsatu.hibernateTutorial.service;
import ru.rsatu.hibernateTutorial.mapper.BookMapper;
import ru.rsatu.hibernateTutorial.pojo.dto.BookDto;
import ru.rsatu.hibernateTutorial.repository.BooksRepository;
import javax.enterprise.context.ApplicationScoped;

```

```

import javax.inject.Inject;
import java.util.List;
/**
 * Сервис для работы с книгами
 */
@ApplicationScoped
public class BooksService {
    @Inject
    BookMapper bookMapper;
    @Inject
    BooksRepository booksRepository;
    /**
     * Загрузить все книги
     */
    public List<BookDto> loadBookList() {
        return booksRepository.loadBooks()
            .stream()
            .map(bookMapper::toBookDto)
            .collect(Collectors.toList());
    }
    /**
     * Загрузить книги конкретного автора
     */
    public List<BookDto> getBooksByAuthor(Long authorId) {
        return booksRepository.getBooksByAuthor(authorId)
            .stream()
            .map(bookMapper::toBookDto)
            .collect(Collectors.toList());
    }
    /**
     * Сохранение книги
     */
    public BookDto saveBook(BookDto bookDto) {
        return bookMapper.toBookDto(booksRepository.saveBook(bookDto));
    }
}

```

Для проверки работоспособности добавим следующие книги с помощью Swagger:

Листинг 12. Добавляемые книги

```

{
  "name": "Book1",
  "authorId": 1
}

```

```

{
  "name": "Book2",
  "authorId": 2
}

```



```
}
```

Запросим все книги автора с id 1:

Листинг 13. Ответ сервера с `authorId = 1`

```
[
  {
    "id": 1,
    "name": "Book1",
    "authorId": 1
  }
]
```

Листинг 14. Ответ сервера с `authorId = 2`

```
[
  {
    "id": 2,
    "name": "Book2",
    "authorId": 2
  }
]
```

### 3.3 Задание 3

Для реализации возможности удалять книги добавим необходимые методы в классы:

- BooksRepository - `public void deleteBook(Long bookId)`
- BooksService - `public void deleteBook(Long bookId)`
- BookResource - `DELETE /books/api/v1/book/{bookId}`

Листинг 15. Изменения в `BooksRepository`:

```
/**
 * Удаление книги
 */
@Transactional
public void deleteBook(Long bookId) {
    Book book = entityManager.find(Book.class, bookId);
    entityManager.remove(book);
}
```

Листинг 16. Изменения в `BooksService`

```
/**
 * Удаление книги
```

```

*/
public void deleteBook(Long bookId) {
    booksRepository.deleteBook(bookId);
}

```

Листинг 17. Изменения в BookResource

```

@DELETE
@Path("/book/{bookId}")
public void deleteBook(@PathParam("bookId") Long bookId) {
    booksService.deleteBook(bookId);
}

```

Для проверки удалим ранее созданную книгу с id = 1:

DELETE /books/api/v1/book1

Ответ сервера:

Code 204 - No Content

Проверим отсутствие книги в списке книг:

Листинг 18. Результат выполнения запроса /loadBookList

```

[
  {
    "id": 2,
    "name": "Book2",
    "authorId": 2
  }
]

```

## 3.4 Задание 4

Для реализации возможности удалять авторов необходимо модифицировать entity Book и добавить каскадное удаление при удалении автора:

Листинг 19. Модифицированный класс Book

```

package ru.rsatu.hibernate4tutorial.pojo.entity;
import lombok.Getter;
import lombok.Setter;
import org.hibernate.annotations.OnDelete;
import org.hibernate.annotations.OnDeleteAction;
import javax.persistence.*;
/**
 * Книга
 */
@Getter

```

```

@Setter
@Entity
@Table(name = "books")
public class Book {
    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE, generator = "books_id_gen")
    @SequenceGenerator(name = "books_id_gen", sequenceName = "books_id_gen_seq",
initialValue = 1, allocationSize = 10)
    private Long id;
    private String name;
    @ManyToOne
    @JoinColumn(name = "author_id")
    @OnDelete(action = OnDeleteAction.CASCADE)
    private Author author;
}

```

Так же потребуется добавить метод `DELETE /books/api/v1/author/{authorId}` в `AuthorResource` и необходимые для его работы методы в `AuthorsService` и `AuthorsRepository`

*Листинг 20. Полученный в результате выполнения лабораторной работы класс `AuthorsRepository`*

```

package ru.rsatu.hibernate.tutorial.repository;
import ru.rsatu.hibernate.tutorial.mapper.AuthorMapper;
import ru.rsatu.hibernate.tutorial.pojo.dto.AuthorDto;
import ru.rsatu.hibernate.tutorial.pojo.entity.Author;
import javax.enterprise.context.ApplicationScoped;
import javax.inject.Inject;
import javax.persistence.EntityManager;
import javax.transaction.Transactional;
import java.util.List;
/**
 * Репозиторий для работы с авторами
 */
@ApplicationScoped
public class AuthorsRepository {
    @Inject
    EntityManager entityManager;
    @Inject
    AuthorMapper authorMapper;
    /**
     * Получить всех авторов
     */
    public List<Author> getAuthors() {
        return entityManager.createQuery("select a from Author a", Author.class)
            .getResultList();
    }
    /**
     * Сохранение автора
     */
    @Transactional

```

```

    public Author saveAuthor(AuthorDto authorDto) {
        Author author = authorMapper.toAuthor(authorDto);
        if (author.getId() != null) {
            entityManager.merge(author);
        } else {
            entityManager.persist(author);
        }
        entityManager.flush();
        return author;
    }
    /**
     * Удаление автора
     */
    @Transactional
    public void deleteAuthor(Long authorId) {
        Author author = entityManager.find(Author.class, authorId);
        entityManager.remove(author);
    }
}

```

Листинг 21. Полученный в результате выполнения лабораторной работы класс *AuthorsService*

```

package ru.rsatu.hibernate.tutorial.service;
import ru.rsatu.hibernate.tutorial.mapper.AuthorMapper;
import ru.rsatu.hibernate.tutorial.mapper.BookMapper;
import ru.rsatu.hibernate.tutorial.pojo.dto.AuthorDto;
import ru.rsatu.hibernate.tutorial.pojo.dto.BookDto;
import ru.rsatu.hibernate.tutorial.repository.AuthorsRepository;
import ru.rsatu.hibernate.tutorial.repository.BooksRepository;
import javax.enterprise.context.ApplicationScoped;
import javax.inject.Inject;
import java.util.List;
/**
 * Сервис для работы с авторами
 */
@ApplicationScoped
public class AuthorsService {
    @Inject
    AuthorMapper authorMapper;
    @Inject
    AuthorsRepository authorsRepository;
    /**
     * Получить всех авторов
     */
    public List<AuthorDto> getAuthors() {
        return authorsRepository.getAuthors()
            .stream()
            .map(authorMapper::toAuthorDto)
            .toList();
    }
}

```

```

/**
 * Сохранение автора
 */
public AuthorDto saveAuthor(AuthorDto authorDto) {
    return authorMapper.toAuthorDto(authorsRepository.saveAuthor(authorDto));
}
/**
 * Удаление автора
 */
public void deleteAuthor(Long authorId) {
    authorsRepository.deleteAuthor(authorId);
}
}

```

Листинг 22. Полученный в результате выполнения лабораторной работы класс *AuthorResource*

```

package ru.rsatu.hibernate.tutorial.resource;
import ru.rsatu.hibernate.tutorial.pojo.dto.AuthorDto;
import ru.rsatu.hibernate.tutorial.service.AuthorsService;
import javax.inject.Inject;
import javax.transaction.Transactional;
import javax.ws.rs.*;
import javax.ws.rs.core.MediaType;
import java.util.List;
@Path("/books/api/v1")
public class AuthorResource {
    @Inject
    AuthorsService authorsService;
    @GET
    @Path("/author")
    public List<AuthorDto> getAuthors() {
        return authorsService.getAuthors();
    }
    @POST
    @Consumes(MediaType.APPLICATION_JSON)
    @Produces(MediaType.APPLICATION_JSON)
    @Path("/author")
    public AuthorDto saveAuthor(AuthorDto authorDto) {
        return authorsService.saveAuthor(authorDto);
    }
    @DELETE
    @Path("/author/{authorId}")
    public void deleteAuthor(@PathParam("authorId") Long authorId) {
        authorsService.deleteAuthor(authorId);
    }
}

```

Для проверки работоспособности удалим автора с id = 2

DELETE /books/api/v1/author/2

Ответ сервера:

204 - No Content

Проверим список авторов:

Листинг 23. Список авторов, полученный с помощью запроса `GET /books/api/v1/author`

```
[
  {
    "id": 1,
    "name": "Author1",
    "birthDate": "01.01.1991",
    "nickName": "author1"
  }
]
```

Автор с id = 2 успешно удален

Так же проверим что удалились книги, относящиеся к данному автору:

Листинг 24. Список книг, полученный с сервера

```
[]
```

Книги так же успешно удалены

## 4. Результаты выполнения

В результате выполнения лабораторной работы разработан back-end сервер обрабатывающий следующие запросы:

- `GET /books/api/v1/author` - получение списка всех авторов
- `POST /books/api/v1/author` - создание нового автора
- `DELETE /books/api/v1/author/{authorId}` - удаление автора с указанным id
- `GET /books/api/v1/book/author/{authorId}` - получение книг по заданному автору
- `DELETE /books/api/v1/book/{bookId}` - удаление книги по заданному id
- `GET /books/api/v1/loadBookList` - получение списка всех книг
- `POST /books/api/v1/saveBook` - создание новой книги

Результат открытия Swagger-ui

[swagger] | *swagger.png*

## 5. Вывод

В результате выполнения лабораторной работы получены навыки по созданию backend серверов с помощью hibernate и resteasy. А также по работе со swagger и rest-api.