Міністерство освіти і науки України

Черкаський державний технологічний університет

Кафедра комп’ютерних наук та системного аналізу

Звіт з лабораторної роботи №4

з дисципліни «Розробка Web-додатків на Java з використанням фреймворку Spring»

Перевірив: Виконав:

асистент кафедри ПЗАС студент групи КН-2001

Немов Р.Г. Заєць Володимир

Черкаси 2023

Тема: Бази даних.

Мета: Навчитися створювати бази даних, зв’язок між базою даних та програмою, а також утворювати між таблицями зв’язки типу один до одного, один до багатьох, багато до багатьох.

Завдання:

1. Встановити MySQL або іншу б.д.

2. Підключити java додаток розроблений у попередніх л.р. до бд.

3. Створити таблиці до ваших класів.

4. Створити звязок між вашими класами і таблицями у бд.

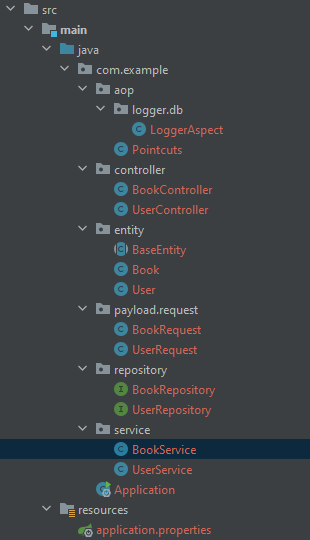
5. Реалізувати збереження, отримання, зміну і видалення (CRUD) java обєктів у бд.

6. Реалізувати у вашому додатку відношення між таблицями: один до одного, один до багатьох, багато до багатьох.

7. Реалізувати тип загрузки eager і lazy.

Хід роботи:

Для початку створимо моделі для таблиць, їх репозиторії, сервіси, аспекти та інше:



application.properties

spring.jpa.hibernate.ddl-auto=update  
spring.datasource.driver-class-name=${DB\_DRIVER:org.postgresql.Driver}  
spring.datasource.url=${DB\_URL:jdbc:postgresql://localhost:5432/spring-backend}  
spring.datasource.username=${DB\_USERNAME:postgres}  
spring.datasource.password=${DB\_PASSWORD:postgres}  
  
server.port=8080  
  
spring.mvc.servlet.load-on-startup=1  
logging.level.org.apache=ERROR

BaseEntity.java

package com.example.entity;  
  
import jakarta.persistence.GeneratedValue;  
import jakarta.persistence.GenerationType;  
import jakarta.persistence.Id;  
import jakarta.persistence.MappedSuperclass;  
import lombok.Getter;  
import lombok.Setter;  
  
@Getter  
@Setter  
@MappedSuperclass  
abstract class BaseEntity {  
 @Id  
 @GeneratedValue(strategy = GenerationType.*IDENTITY*)  
 public Long id;  
}

User.java

package com.example.entity;  
  
import jakarta.persistence.\*;  
import lombok.AccessLevel;  
import lombok.Getter;  
import lombok.NoArgsConstructor;  
import lombok.Setter;  
  
import java.util.Set;  
import java.util.UUID;  
  
@Getter  
@Setter  
@NoArgsConstructor  
@Entity  
@Table(name = "users")  
public class User extends BaseEntity {  
  
 @Setter(AccessLevel.*NONE*)  
 @Column(name = "uuid")  
 private String uuid;  
  
 @Column(name = "username")  
 private String username;  
  
 @Column(name = "email")  
 private String email;  
  
 @Column(name = "password")  
 private String password;  
  
 @ManyToMany(fetch = FetchType.*LAZY*)  
 @Column(name = "books")  
 @JoinTable( name = "user\_books",  
 joinColumns = @JoinColumn(name = "user\_uuid"),  
 inverseJoinColumns = @JoinColumn(name = "books\_uuid"))  
 private Set<Book> books;  
  
 public User(String username, String email, String password) {  
 this.username = username;  
 this.email = email;  
 this.password = password;  
 }  
  
 public void addBooks(Set<Book> books) {  
 this.books = books;  
 }  
  
 @PrePersist  
 public void onCreate() {  
 uuid = UUID.*randomUUID*().toString();  
 }  
}

Book.java

package com.example.entity;  
  
import jakarta.persistence.\*;  
import lombok.AccessLevel;  
import lombok.NoArgsConstructor;  
import lombok.Getter;  
import lombok.Setter;  
  
import java.util.Set;  
import java.util.UUID;  
  
@Getter  
@Setter  
@NoArgsConstructor  
@Entity  
@Table(name = "books")  
public class Book extends BaseEntity {  
  
 @Setter(AccessLevel.*NONE*)  
 @Column(name = "uuid")  
 private String uuid;  
  
 @Column(name = "name")  
 private String name;  
  
 @ManyToMany(fetch = FetchType.*EAGER*)  
 @Column(name = "author")  
 @JoinTable( name = "book\_authors",  
 joinColumns = @JoinColumn(name = "book\_uuid"),  
 inverseJoinColumns = @JoinColumn(name = "authors\_uuid"))  
 private Set<User> authors;  
  
 public Book(String name, Set<User> authors) {  
 this.name = name;  
 this.authors = authors;  
 }  
  
 @PrePersist  
 public void onCreate() {  
 uuid = UUID.*randomUUID*().toString();  
 }  
}

UserRequest.java

package com.example.payload.request;  
  
import jakarta.validation.constraints.NotBlank;  
import lombok.Getter;  
import lombok.Setter;  
  
@Getter  
@Setter  
public class UserRequest {  
 @NotBlank  
 private String username;  
  
 @NotBlank  
 private String email;  
  
 @NotBlank  
 private String password;  
}

BookRequest

package com.example.payload.request;  
  
import jakarta.validation.constraints.NotBlank;  
import lombok.Getter;  
import lombok.Setter;  
  
import java.util.Set;  
  
@Getter  
@Setter  
public class BookRequest {  
 @NotBlank  
 private String name;  
  
 @NotBlank  
 private Set<String> authors;  
}

UserRepository.java

package com.example.repository;  
  
import com.example.entity.User;  
import jakarta.validation.constraints.NotNull;  
import org.springframework.data.jpa.repository.JpaRepository;  
  
import java.util.List;  
  
public interface UserRepository extends JpaRepository<User, Long> {  
  
 boolean existsByUsername(String username);  
  
 boolean existsByEmail(String email);  
  
 boolean existsByUuid(String uuid);  
  
 @NotNull  
 List<User> findAll();  
  
 User findByUuid(String uuid);  
  
 void deleteByUuid(String uuid);  
  
 @NotNull  
 <S extends User> S save(@NotNull S entity);  
  
 @NotNull  
 <S extends User> List<S> saveAll(@NotNull Iterable<S> entities);  
}

BookRepository.java

package com.example.repository;  
  
import com.example.entity.Book;  
import jakarta.validation.constraints.NotNull;  
import org.springframework.data.jpa.repository.JpaRepository;  
  
import java.util.List;  
  
public interface BookRepository extends JpaRepository<Book, Long> {  
  
 boolean existsByName(String name);  
  
 boolean existsByUuid(String uuid);  
  
 @NotNull  
 List<Book> findAll();  
  
 Book findByUuid(String uuid);  
  
 void deleteByUuid(String uuid);  
  
 @NotNull  
 <S extends Book> S save(@NotNull S entity);  
  
 @NotNull  
 <S extends Book> List<S> saveAll(@NotNull Iterable<S> entities);  
}

UserService.java

package com.example.service;  
  
import com.example.entity.Book;  
import com.example.entity.User;  
import com.example.payload.request.UserRequest;  
import com.example.repository.BookRepository;  
import com.example.repository.UserRepository;  
import lombok.AllArgsConstructor;  
import org.springframework.stereotype.Service;  
import org.springframework.transaction.annotation.Transactional;  
  
import java.util.HashSet;  
import java.util.List;  
import java.util.Objects;  
import java.util.Set;  
import java.util.concurrent.atomic.AtomicReference;  
import java.util.stream.Collectors;  
  
@Service  
@AllArgsConstructor  
public class UserService {  
  
 private final UserRepository userRepository;  
  
 private final BookRepository bookRepository;  
  
 public String add(UserRequest userRequest) {  
 String response;  
 if (!userRepository.existsByUsername(userRequest.getUsername())  
 && !userRepository.existsByEmail(userRequest.getEmail())) {  
 User user = new User(userRequest.getUsername(),  
 userRequest.getEmail(),  
 userRequest.getPassword());  
 user = userRepository.save(user);  
 response = "Saved user with username: "  
 + user.getUsername()  
 + "(UUID = " + user.getUuid() + ")";  
 } else {  
 response = "User with the same username or email already exists";  
 }  
 return response;  
 }  
  
 public User getByUuid(String uuid) {  
 return userRepository.findByUuid(uuid);  
 }  
  
 @Transactional  
 public String deleteByUuid(String uuid) {  
 String response;  
 if (userRepository.existsByUuid(uuid)) {  
 userRepository.deleteByUuid(uuid);  
 response = "Deleted user with uuid: " + uuid;  
 } else {  
 response = "Can't find user with uuid: " + uuid;  
 }  
 return response;  
 }  
  
 public String addAll(List<UserRequest> userRequests) {  
 String response;  
 AtomicReference<String> message = new AtomicReference<>("");  
 List<User> users = userRequests.stream()  
 .map(userRequest -> {  
 if (!userRepository.existsByUsername(userRequest.getUsername())  
 && !userRepository.existsByEmail(userRequest.getEmail())) {  
 return new User(userRequest.getUsername(),  
 userRequest.getEmail(),  
 userRequest.getPassword());  
 } else {  
 return null;  
 }  
 })  
 .filter(user -> {  
 if (user == null) {  
 message.set("Some users already exist by username or email!\n");  
 }  
 return Objects.*nonNull*(user);  
 })  
 .collect(Collectors.*toList*());  
 if (!users.isEmpty()) {  
 response = message + "Saved users with usernames: " + userRepository.saveAll(users)  
 .stream()  
 .map(user -> user.getUsername()  
 + "(UUID = " + user.getUuid() + ")")  
 .collect(Collectors.*joining*(", "));  
 } else {  
 response = "All users with the same usernames or emails already exist";  
 }  
 return response;  
 }  
  
 public List<User> getAll() {  
 return userRepository.findAll();  
 }  
  
 public String addBooks(String user\_uuid, Set<String> book\_uuids) {  
 String response;  
 if (userRepository.existsByUuid(user\_uuid)) {  
 User user = getByUuid(user\_uuid);  
 Set<Book> books = new HashSet<>();  
 for (String uuid: book\_uuids) {  
 books.add(bookRepository.findByUuid(uuid));  
 }  
 user.addBooks(books);  
 userRepository.save(user);  
 response = "Added books with names:"  
 + books.stream()  
 .map(book -> book.getName()  
 + "(UUID = " + book.getUuid() + ")")  
 .collect(Collectors.*joining*(", "));  
 } else {  
 response = "Can't find user with uuid: " + user\_uuid;  
 }  
 return response;  
 }  
}

BookService.java

package com.example.service;  
  
import com.example.entity.Book;  
import com.example.entity.User;  
import com.example.payload.request.BookRequest;  
import com.example.repository.BookRepository;  
import com.example.repository.UserRepository;  
import lombok.AllArgsConstructor;  
import org.springframework.stereotype.Service;  
import org.springframework.transaction.annotation.Transactional;  
  
import java.util.HashSet;  
import java.util.List;  
import java.util.Objects;  
import java.util.Set;  
import java.util.concurrent.atomic.AtomicReference;  
import java.util.stream.Collectors;  
  
@Service  
@AllArgsConstructor  
public class BookService {  
  
 private final BookRepository bookRepository;  
  
 private final UserRepository userRepository;  
  
 public String add(BookRequest bookRequest) {  
 String response;  
 if (!bookRepository.existsByName(bookRequest.getName())) {  
 Set<User> authors = new HashSet<>();  
 for (String uuid: bookRequest.getAuthors()) {  
 authors.add(userRepository.findByUuid(uuid));  
 }  
 Book book = new Book(bookRequest.getName(), authors);  
 book = bookRepository.save(book);  
 response = "Saved book with name: "  
 + book.getName()  
 + "(UUID = " + book.getUuid() + ")";  
 } else {  
 response = "Book with the same name already exists";  
 }  
 return response;  
 }  
  
 public Book getByUuid(String uuid) {  
 return bookRepository.findByUuid(uuid);  
 }  
  
 @Transactional  
 public String deleteByUuid(String uuid) {  
 String response;  
 if (bookRepository.existsByUuid(uuid)) {  
 bookRepository.deleteByUuid(uuid);  
 response = "Deleted book with uuid: " + uuid;  
 } else {  
 response = "Can't find book with uuid: " + uuid;  
 }  
 return response;  
 }  
  
 public String addAll(List<BookRequest> bookRequests) {  
 String response;  
 AtomicReference<String> message = new AtomicReference<>("");  
 List<Book> books = bookRequests.stream()  
 .map(bookRequest -> {  
 if (!bookRepository.existsByName(bookRequest.getName())) {  
 Set<User> authors = new HashSet<>();  
 for (String uuid: bookRequest.getAuthors()) {  
 authors.add(userRepository.findByUuid(uuid));  
 }  
 return new Book(bookRequest.getName(), authors);  
 } else {  
 return null;  
 }  
 })  
 .filter(book -> {  
 if (book == null) {  
 message.set("Some books already exist by name!\n");  
 }  
 return Objects.*nonNull*(book);  
 })  
 .collect(Collectors.*toList*());  
 if (!books.isEmpty()) {  
 response = message + "Saved books with names: " + bookRepository.saveAll(books)  
 .stream()  
 .map(book -> book.getName()  
 + "(UUID = " + book.getUuid() + ")")  
 .collect(Collectors.*joining*(", "));  
 } else {  
 response = "All books with the same names already exist";  
 }  
 return response;  
 }  
  
 public List<Book> getAll() {  
 return bookRepository.findAll();  
 }  
}

UserController.java

package com.example.controller;  
  
import com.example.entity.User;  
import com.example.payload.request.UserRequest;  
import com.example.service.UserService;  
import jakarta.validation.constraints.NotNull;  
import lombok.RequiredArgsConstructor;  
import org.springframework.http.ResponseEntity;  
import org.springframework.web.bind.annotation.\*;  
  
import java.util.List;  
import java.util.Set;  
  
@RestController  
@RequiredArgsConstructor  
public class UserController {  
  
 private final UserService userService;  
  
 @PostMapping("/user")  
 public ResponseEntity<?> addUser(@RequestBody UserRequest userRequest) {  
 return ResponseEntity.*ok*(userService.add(userRequest));  
 }  
  
 @GetMapping("/user/{uuid}")  
 public User getUserByUuid(@PathVariable @NotNull String uuid) {  
 return userService.getByUuid(uuid);  
 }  
  
 @DeleteMapping("/user/{uuid}")  
 public ResponseEntity<?> deleteUser(@PathVariable @NotNull String uuid) {  
 return ResponseEntity.*ok*(userService.deleteByUuid(uuid));  
 }  
  
 @PostMapping("/users")  
 public ResponseEntity<?> addAllUsers(@RequestBody List<UserRequest> userRequests) {  
 return ResponseEntity.*ok*(userService.addAll(userRequests));  
 }  
  
 @GetMapping("/users")  
 public List<User> getAllUsers() {  
 return userService.getAll();  
 }  
  
 @PostMapping("/user/{user\_uuid}")  
 public ResponseEntity<?> addBooksToUser(@PathVariable @NotNull String user\_uuid,  
 @RequestBody Set<String> book\_uuids) {  
 return ResponseEntity.*ok*(userService.addBooks(user\_uuid, book\_uuids));  
 }  
}

BookController.java

package com.example.controller;  
  
import com.example.entity.Book;  
import com.example.payload.request.BookRequest;  
import com.example.service.BookService;  
import jakarta.validation.constraints.NotNull;  
import lombok.RequiredArgsConstructor;  
import org.springframework.http.ResponseEntity;  
import org.springframework.web.bind.annotation.\*;  
  
import java.util.List;  
  
@RestController  
@RequiredArgsConstructor  
public class BookController {  
  
 private final BookService bookService;  
  
 @PostMapping("/book")  
 public ResponseEntity<?> addBook(@RequestBody BookRequest bookRequest) {  
 return ResponseEntity.*ok*(bookService.add(bookRequest));  
 }  
  
 @GetMapping("/book/{uuid}")  
 public Book getBookByUuid(@PathVariable @NotNull String uuid) {  
 return bookService.getByUuid(uuid);  
 }  
  
 @DeleteMapping("/book/{uuid}")  
 public ResponseEntity<?> deleteBook(@PathVariable @NotNull String uuid) {  
 return ResponseEntity.*ok*(bookService.deleteByUuid(uuid));  
 }  
  
 @PostMapping("/books")  
 public ResponseEntity<?> addAllBooks(@RequestBody List<BookRequest> bookRequests) {  
 return ResponseEntity.*ok*(bookService.addAll(bookRequests));  
 }  
  
 @GetMapping("/books")  
 public List<Book> getAllBooks() {  
 return bookService.getAll();  
 }  
}

Pointcuts.java

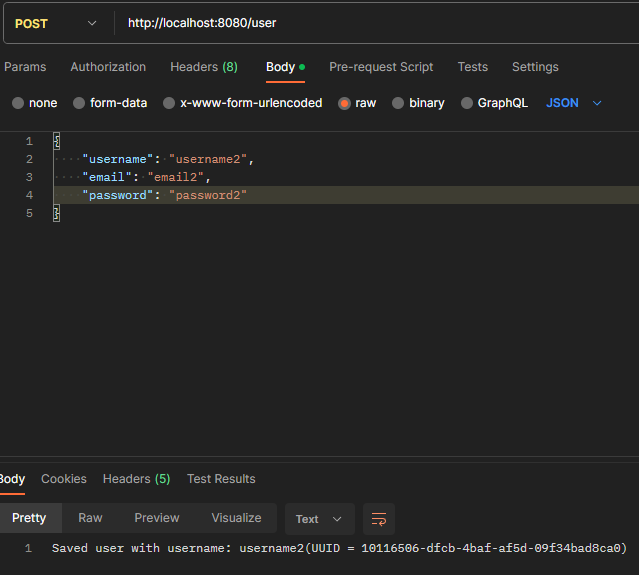
package com.example.aop;  
  
import org.aspectj.lang.annotation.Pointcut;  
  
public class Pointcuts {  
  
 @Pointcut("execution(\* com.example.repository.\*.\*(..))")  
 public void onRepositoryMethod() {  
 }  
  
 @Pointcut("execution(\* com.example.service.\*.\*(..))")  
 public void onServiceMethod() {  
 }  
}

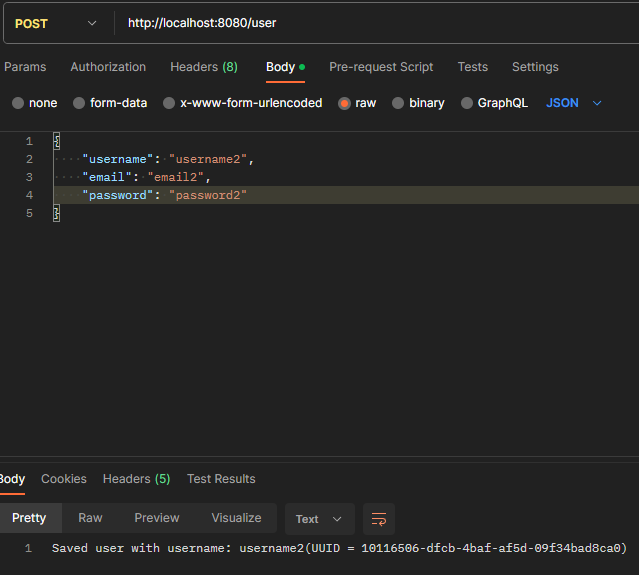
LoggerAspect.java

package com.example.aop.logger.db;  
  
import lombok.SneakyThrows;  
import lombok.extern.slf4j.Slf4j;  
import org.aspectj.lang.ProceedingJoinPoint;  
import org.aspectj.lang.annotation.Around;  
import org.aspectj.lang.annotation.Aspect;  
import org.springframework.stereotype.Component;  
  
import java.util.Arrays;  
import java.util.List;  
  
@Component  
@Aspect  
@Slf4j  
public class LoggerAspect {  
  
 @SneakyThrows  
 @Around("com.example.aop.Pointcuts.onServiceMethod()")  
 public Object onServiceMethod(ProceedingJoinPoint joinPoint) {  
 String className = joinPoint.getSignature().getDeclaringType().getSimpleName();  
 String methodName = joinPoint.getSignature().getName();  
 List<String> args = Arrays.*stream*(joinPoint.getArgs()).map(Object::toString).toList();  
  
 LogInfo logInfo = new LogInfo(className, methodName, args);  
  
 *log*.info("Service: Class: " + logInfo.className  
 + ", method: " + logInfo.methodName  
 + ", args: " + args);  
  
 return joinPoint.proceed();  
 }  
  
 @SneakyThrows  
 @Around("com.example.aop.Pointcuts.onRepositoryMethod()")  
 public Object onRepositoryMethod(ProceedingJoinPoint joinPoint) {  
 String className = joinPoint.getSignature().getDeclaringType().getSimpleName();  
 String methodName = joinPoint.getSignature().getName();  
 List<String> args = Arrays.*stream*(joinPoint.getArgs()).map(Object::toString).toList();  
  
 LogInfo logInfo = new LogInfo(className, methodName, args);  
  
 *log*.info("Repository: Class: " + logInfo.className  
 + ", method: " + logInfo.methodName  
 + ", args: " + args);  
  
 return joinPoint.proceed();  
 }  
  
 record LogInfo(String className, String methodName, List<String> args) {  
 }  
}

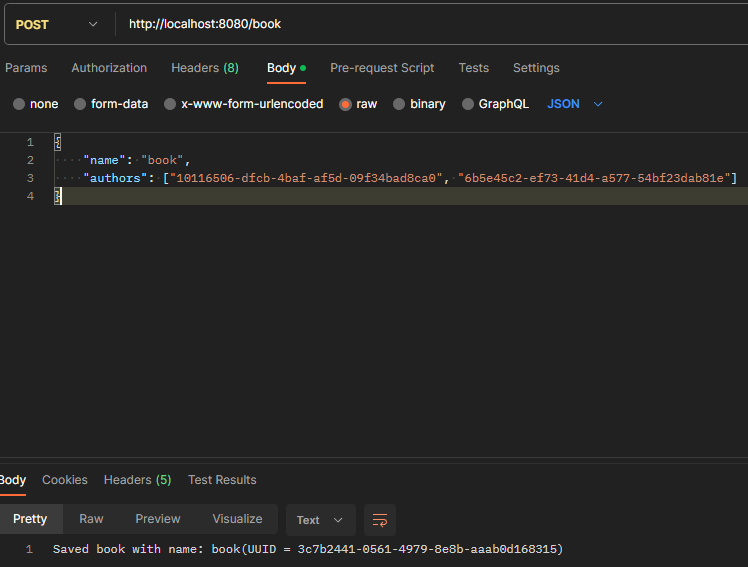
Тепер протестуємо працездатність розвобленого додатку за допомогою Postman.

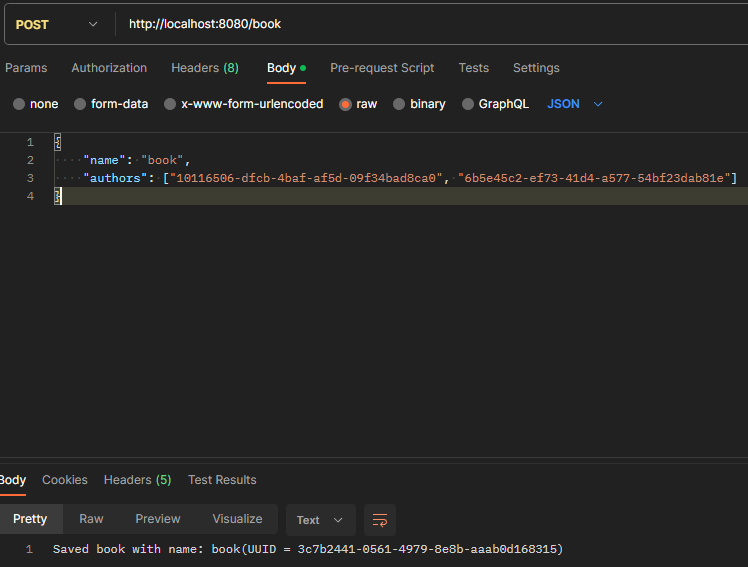
Приклад створення нового користувача:



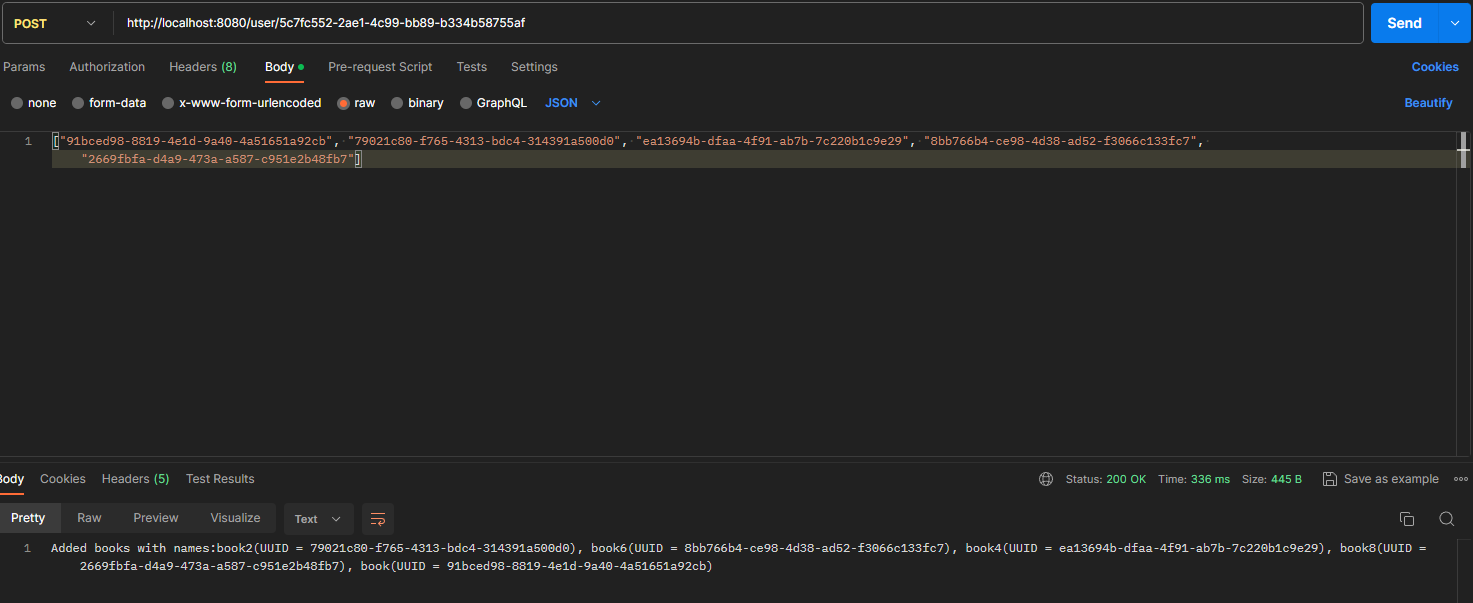


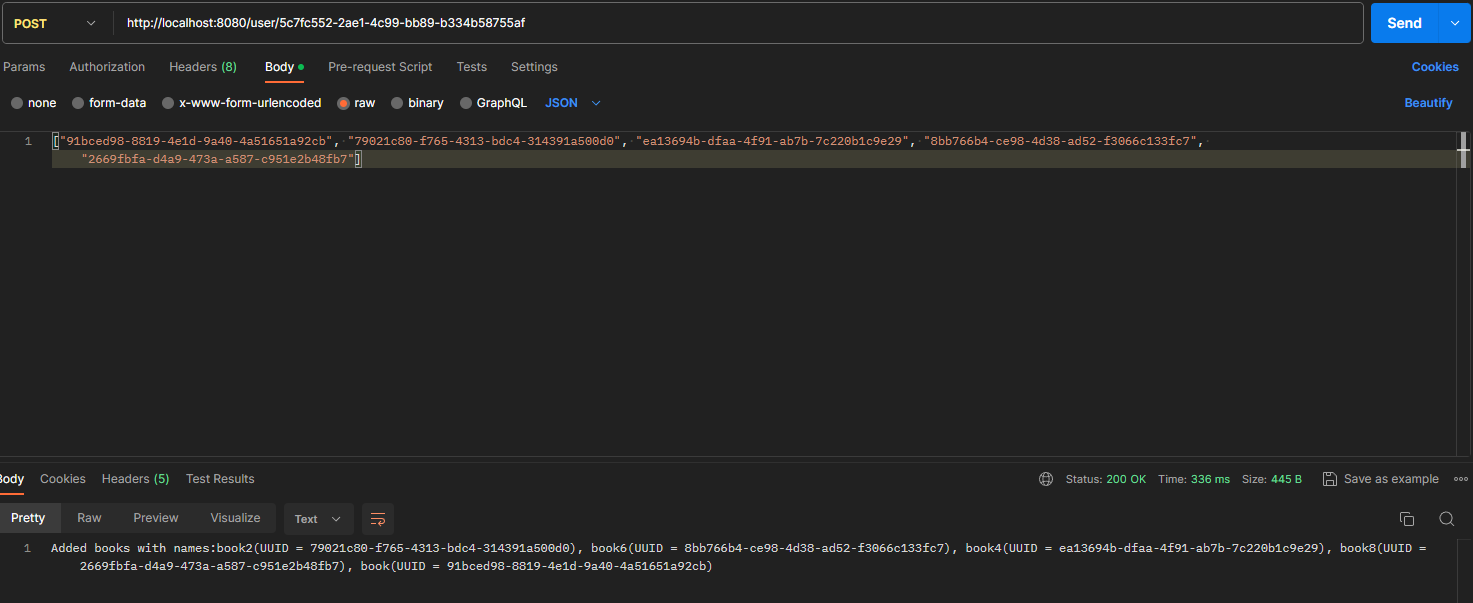
Приклад створення нової книги:



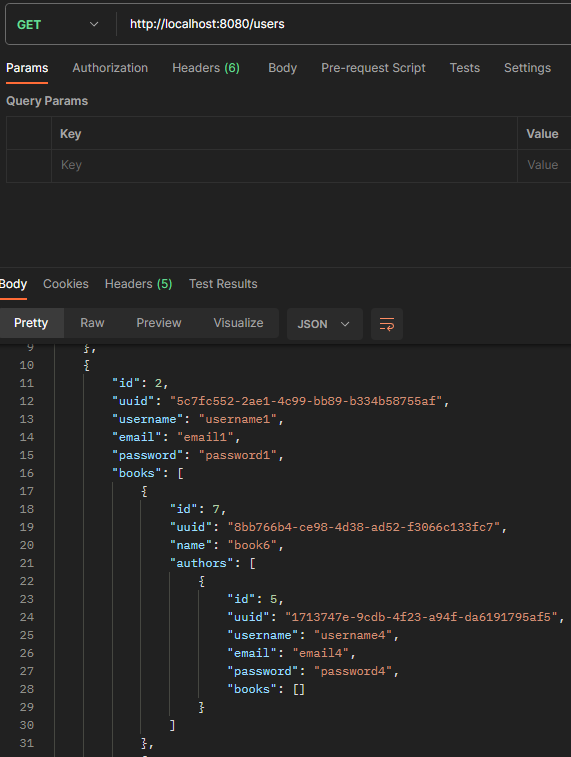


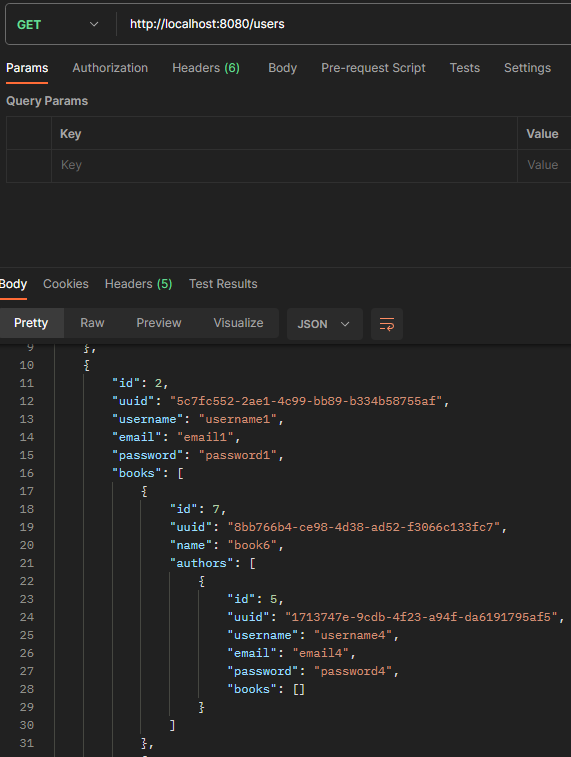
Приклад додавання користувачу нової книги:





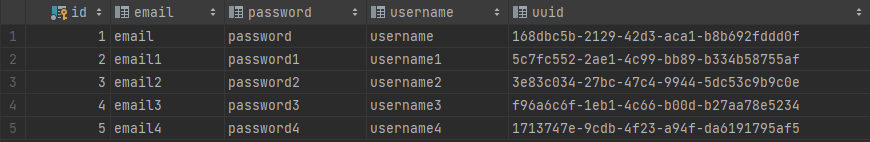
Приклад отримання інформації про користувачів, з загрузкою книг по Lazy, та загрузкою авторів книг по Eager



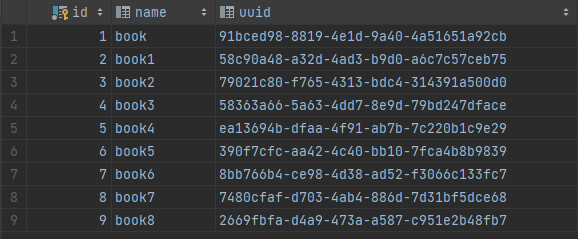


Результат заповнення бази даних інформацією:

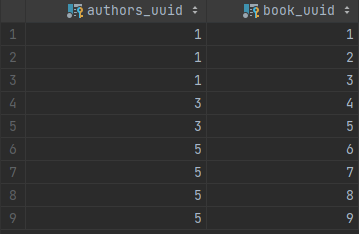
users



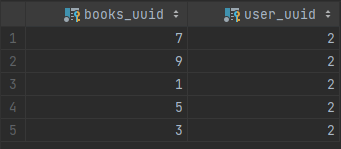
books



book\_authors



user\_books



Посилання на код на GitHub:

<https://github.com/volodymyr-zaiets/Java-for-Web-Spring/tree/main/Lab4>

Висновок: В результаті виконання лабораторної роботи навчився створювати бази даних, зв’язок між базою даних та програмою, а також утворювати між таблицями зв’язки типу один до одного, один до багатьох, багато до багатьох.