

МЕСТО ДЛЯ ТИТУЛЬНИКА

СОДЕРЖАНИЕ

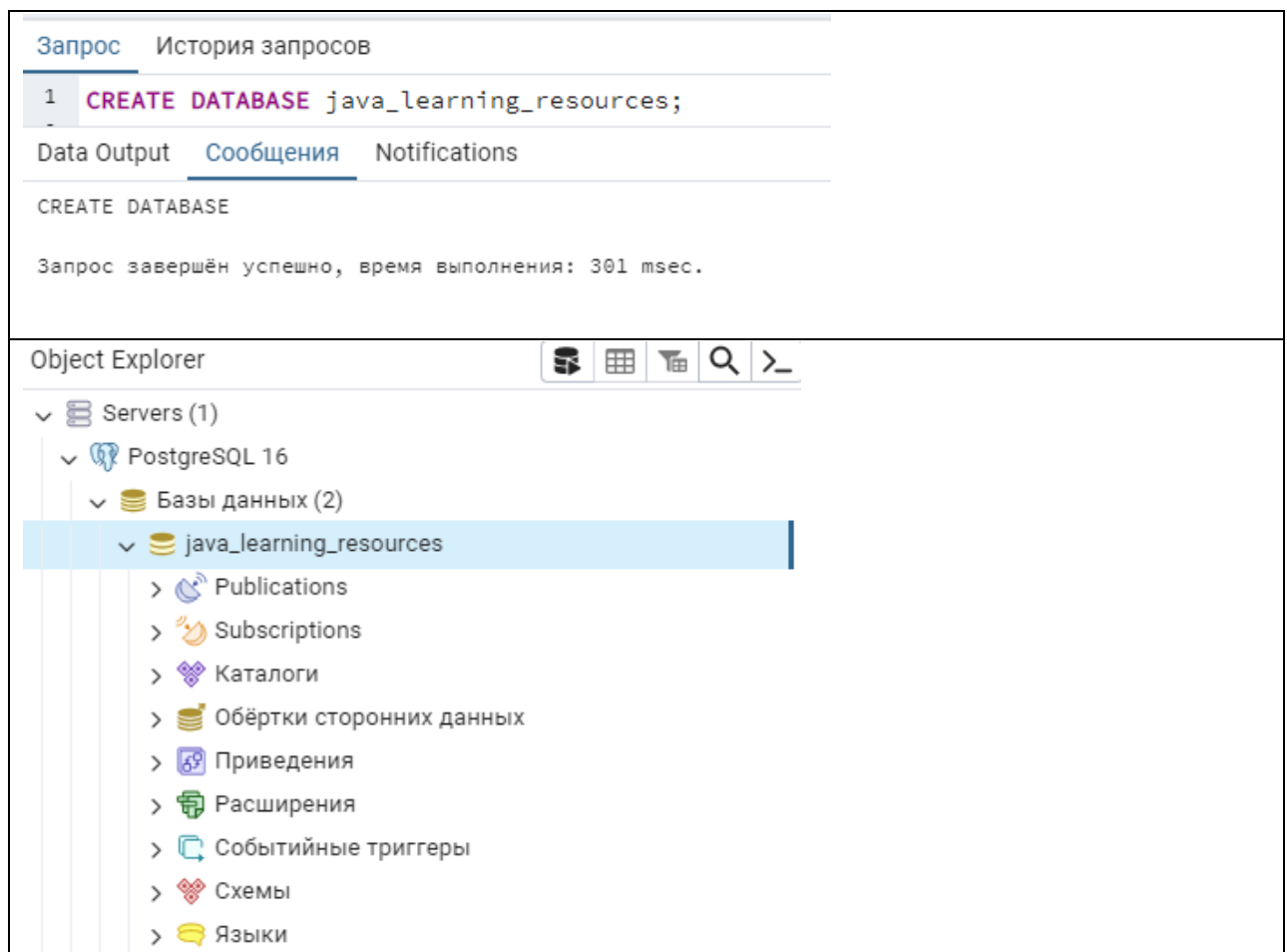
| | |
|--|----|
| 1. Реализация задания «Модуль 3»..... | 3 |
| 1.1.Создание базы данных..... | 3 |
| 1.2.Определение таблиц и их структуры | 4 |
| 1.3.Создание связей между таблицами | 5 |
| 2. Подготовка проекта в IntelliJ IDEA | 7 |
| 2.1. Создание/Импорт проекта | 7 |
| 2.2.Настройка подключения к базе данных | 8 |
| 2.3.Настройка файла application.properties | 10 |
| 3. Разработка приложения | 11 |
| 3.1.Создание моделей | 11 |
| 3. Выгрузка проекта на репозиторий GitHub..... | 37 |
| 3.1.Создание структуры папок с Readme.md..... | 37 |

1. Реализация задания «Модуль 3»

1.1. Создание базы данных

Чтобы создать базу данных в PostgreSQL для приложения "*Библиотека ресурсов для изучения языка программирования Java*", мы пройдем через следующие шаги:

1. **Создание базы данных:** Сначала создадим саму базу данных, к которой будет подключаться ваше приложение.
2. **Определение таблиц и их структуры:** Затем определим структуру таблиц, соответствующих вашим сущностям: Ресурсы, Категории, Пользователи, Избранные ресурсы.
3. **Создание связей между таблицами:** Определим внешние ключи и другие связи между таблицами для обеспечения целостности данных.



1.2. Определение таблиц и их структуры

Далее, используем SQL для создания таблиц. Вот примерные команды для создания каждой из таблиц согласно определенным атрибутам:

```
CREATE TABLE categories (  
id SERIAL PRIMARY KEY,  
name VARCHAR(255) NOT NULL,  
description TEXT  
);
```

```
CREATE TABLE resources (  
id SERIAL PRIMARY KEY,  
title VARCHAR(255) NOT NULL,  
description TEXT,  
type VARCHAR(50),  
url VARCHAR(255),  
category_id INTEGER,  
FOREIGN KEY (category_id) REFERENCES categories (id)  
);
```

```
CREATE TABLE users (  
id SERIAL PRIMARY KEY,  
username VARCHAR(255) NOT NULL UNIQUE,  
email VARCHAR(255) NOT NULL UNIQUE,  
password VARCHAR(255) NOT NULL  
);
```

```
CREATE TABLE favorites (  
id SERIAL PRIMARY KEY,  
user_id INTEGER NOT NULL,  
resource_id INTEGER NOT NULL,  
FOREIGN KEY (user_id) REFERENCES users (id),  
FOREIGN KEY (resource_id) REFERENCES resources (id),  
UNIQUE (user_id, resource_id)  
);
```

Результат обработки SQL запросов на создание таблиц, а также связей к ним.

▼ Таблицы (4)

> categories

> favorites

> resources

> users

Data Output

Сообщения

Notifications

CREATE TABLE

Запрос завершён успешно, время выполнения: 65 msec.

1.3. Создание связей между таблицами

В предыдущем шаге мы уже определили основные связи, создавая внешние ключи в таблицах **resources** и **favorites**. Эти внешние ключи гарантируют, что данные останутся консистентными и правильно связанными между таблицами.

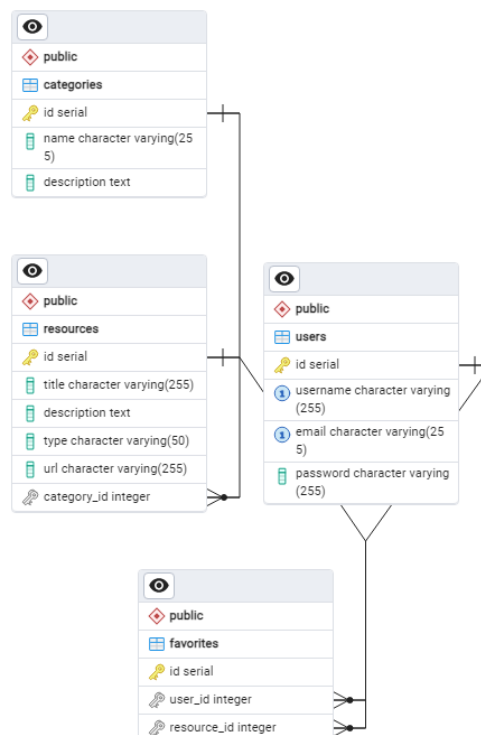


Схема БД «Модуль 3, Библиотека ресурсов для изучения языка программирования Java»

Завершение

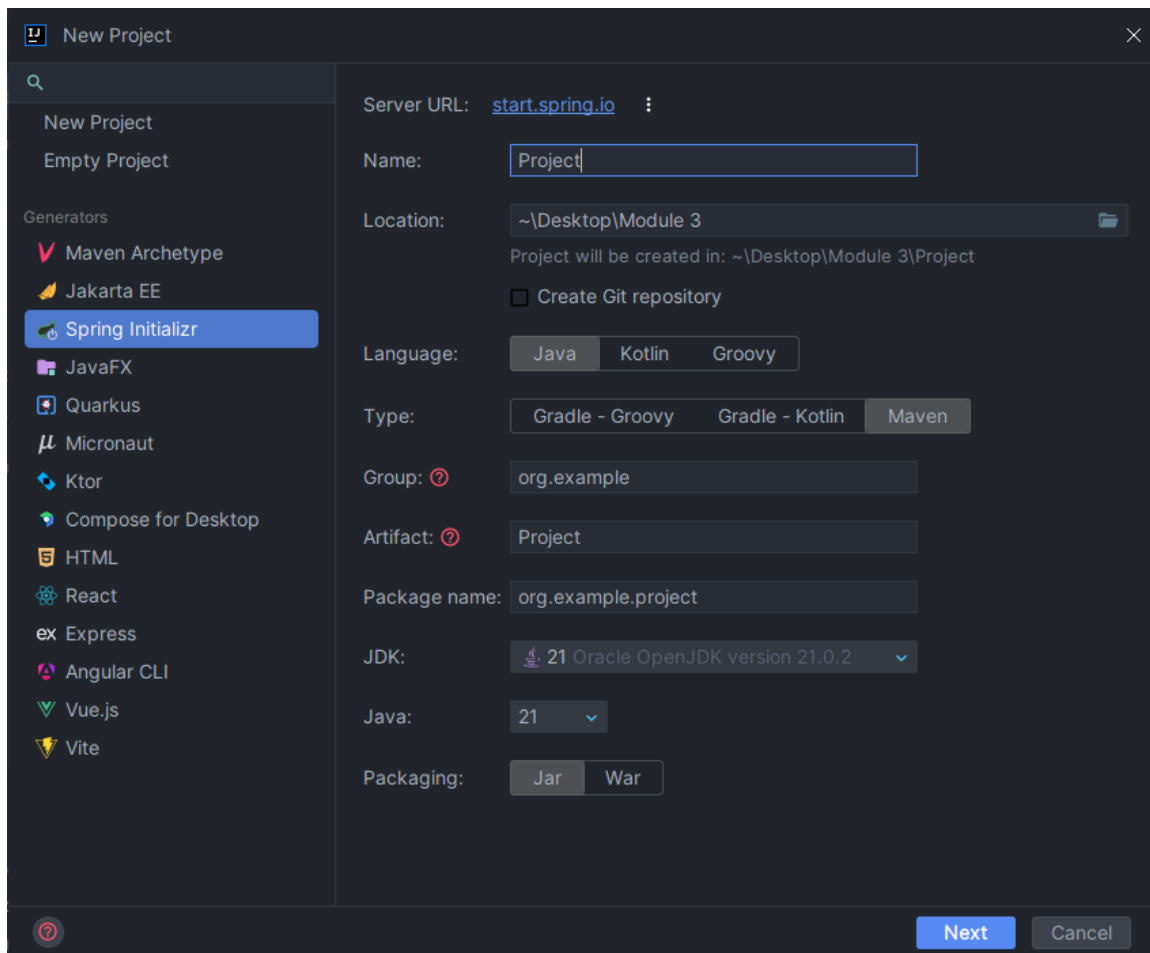
После выполнения этих SQL команд база данных будет готова к использованию в моем приложении, можем начать добавлять первоначальные данные в таблицы **categories** для категоризации ресурсов или продолжить с разработкой Spring Boot приложения для взаимодействия с этой базой данных.

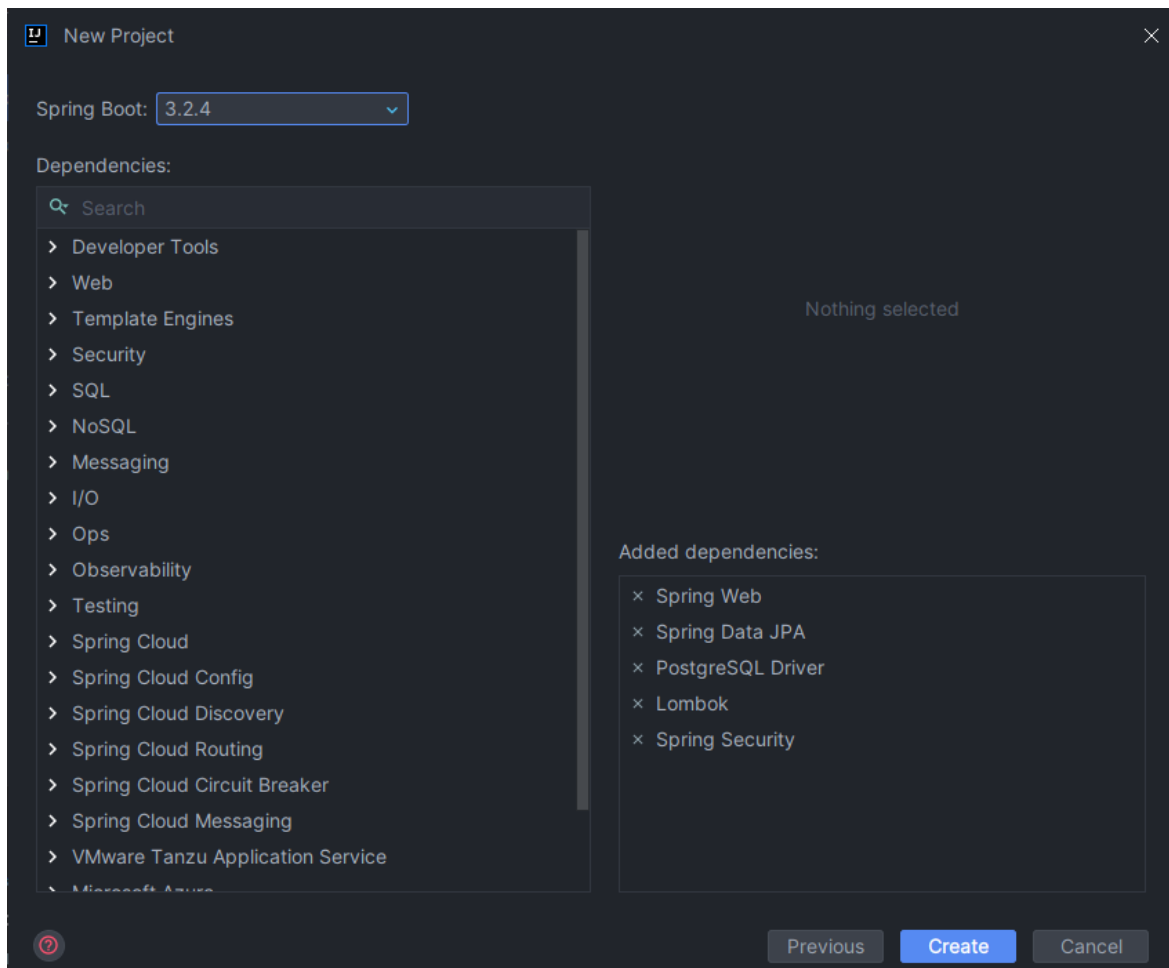
2. Подготовка проекта в IntelliJ IDEA

2.1. Создание/Импорт проекта

1. Запустил IntelliJ IDEA и выберите File > New > Project....
2. В левой колонке выбрал Spring Initializr.
3. Заполнил форму проекта (Project SDK должен быть Java 11 или выше, выбирал версию Spring Boot и зависимости, как указано выше).
4. Нажал Next, задал имя и расположение проекта, затем снова нажал Next.
5. Выбрал нужные мне зависимости (Spring Web, Spring Data JPA, PostgreSQL Driver, Lombok, Spring Security если необходимо) и завершил создание проекта.

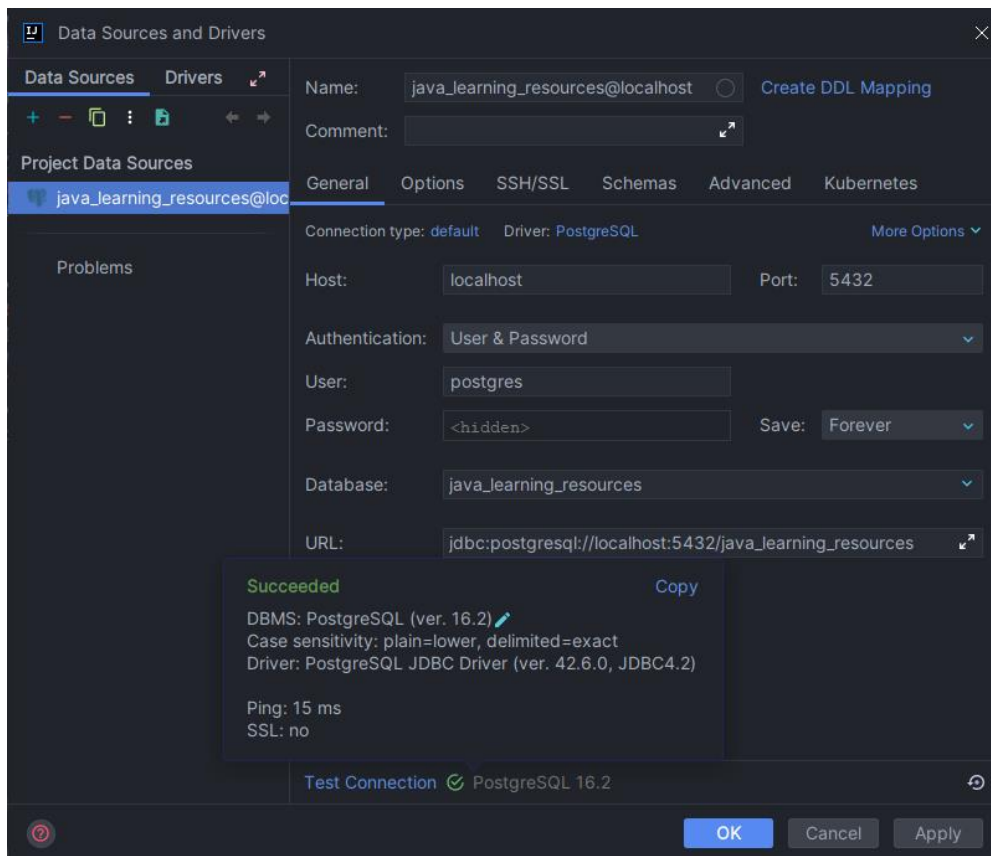
Ниже прикреплён скриншот!



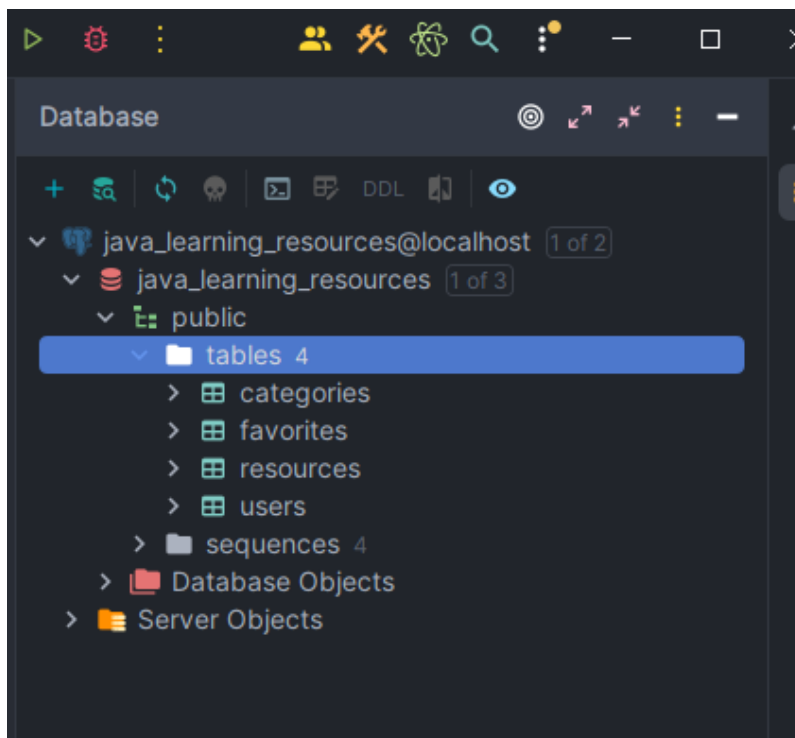


2.2. Настройка подключения к базе данных

1. Открыл View > Tool Windows > Database.
2. Нажал на + и выбрал Data Source > PostgreSQL.
3. Указал параметры подключения к базе данных (hostname, порт, имя базы данных, имя пользователя и пароль).
4. Проверил подключение, нажав **Test Connection**, и если все в порядке, сохранил его.



Test Connection PostgreSQL 16.2 >>> Статус подключения к БД через IDE IntelliJ IDEA.

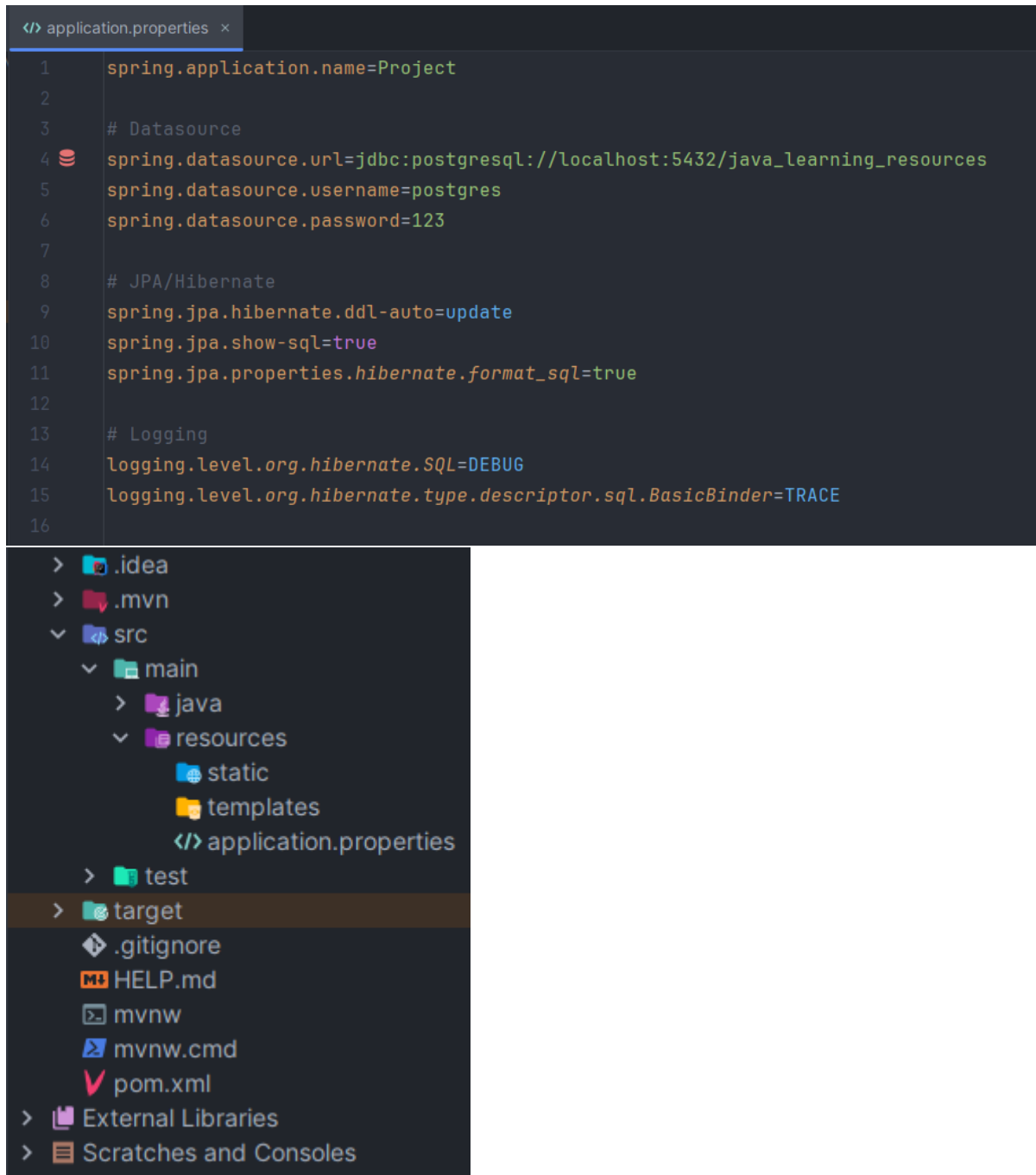


Разметка структуры приложения. Создание набора папок в соответствии с паттерном MVC, Repository;

2.3. Настройка файла application.properties

Перешел к src/main/resources/application.properties.

Добавил и изменил строки подключения к базе данных и другие свойства Spring, как было описано ранее. **Как и показано на скриншоте ниже.**



3. Разработка приложения

3.1. Создание моделей

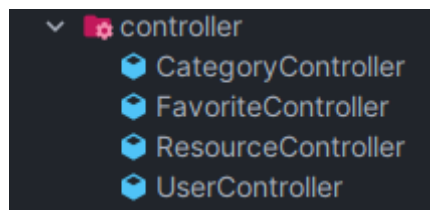
Создаю классы сущностей в пакете **model**. Для каждой таблицы в моей базе данных будет соответствующий класс сущности.

Исходя из структуры моего проекта, вот описание каждого класса и компонента:

Основные Пакеты

Controller: Содержит контроллеры, которые являются точками входа для веб-запросов и отвечают за обработку HTTP-запросов.

| |
|--|
| - CategoryController: Управляет запросами к категориям ресурсов. |
| - FavoriteController: обрабатывает запросы к избранным ресурсам пользователей. |
| - ResourceController: управляет запросами, связанными с учебными ресурсами. |
| - UserController: управляет запросами, связанными с пользователями системы. |



Структура MVC папки в проекте

Листинг класса: CategoryController

```
package controller;

import model.Category;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;
import service.CategoryService;

import java.util.List;

@RestController
@RequestMapping("/api/categories")
public class CategoryController {

    private final CategoryService categoryService;

    @Autowired
```

```

    public CategoryController(CategoryService categoryService) {
        this.categoryService = categoryService;
    }

    @GetMapping
    public ResponseEntity<List<Category>> getAllCategories() {
        return
ResponseEntity.ok(categoryService.getAllCategories());
    }

    @GetMapping("/{id}")
    public ResponseEntity<Category>
getCategoryById(@PathVariable Long id) {
        return categoryService.getCategoryById(id)
            .map(ResponseEntity::ok)
            .orElseGet(() ->
ResponseEntity.notFound().build());
    }

    @PostMapping
    public ResponseEntity<Category> createCategory(@RequestBody
Category category) {
        return
ResponseEntity.ok(categoryService.createCategory(category));
    }

    @PutMapping("/{id}")
    public ResponseEntity<Category> updateCategory(@PathVariable
Long id, @RequestBody Category category) {
        return
ResponseEntity.ok(categoryService.updateCategory(id, category));
    }

    @DeleteMapping("/{id}")
    public ResponseEntity<Void> deleteCategory(@PathVariable
Long id) {
        categoryService.deleteCategory(id);
        return ResponseEntity.ok().build();
    }
}

```

Листинг класса: **FavoriteController**

```

package controller;

import model.Favorite;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;
import service.FavoriteService;

import java.util.List;

```

```

@RestController
@RequestMapping("/api/favorites")
public class FavoriteController {

    private final FavoriteService favoriteService;

    @Autowired
    public FavoriteController(FavoriteService favoriteService) {
        this.favoriteService = favoriteService;
    }

    @GetMapping
    public ResponseEntity<List<Favorite>> getAllFavorites() {
        return
ResponseEntity.ok(favoriteService.getAllFavorites());
    }

    @GetMapping("/{id}")
    public ResponseEntity<Favorite>
getFavoriteById(@PathVariable Long id) {
        return favoriteService.getFavoriteById(id)
            .map(ResponseEntity::ok)
            .orElseGet(() ->
ResponseEntity.notFound().build());
    }

    @PostMapping
    public ResponseEntity<Favorite> addFavorite(@RequestBody
Favorite favorite) {
        return
ResponseEntity.ok(favoriteService.addFavorite(favorite));
    }

    @DeleteMapping("/{id}")
    public ResponseEntity<Void> removeFavorite(@PathVariable
Long id) {
        favoriteService.removeFavorite(id);
        return ResponseEntity.ok().build();
    }
}

```

Листинг класса: **ResourceController**

```

package controller;

import model.Resource;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;
import service.ResourceService;

```

```

import java.util.List;

@RestController
@RequestMapping("/api/resources")
public class ResourceController {

    private final ResourceService resourceService;

    @Autowired
    public ResourceController(ResourceService resourceService) {
        this.resourceService = resourceService;
    }

    @GetMapping
    public ResponseEntity<List<Resource>> getAllResources() {
        return
ResponseEntity.ok(resourceService.getAllResources());
    }

    @GetMapping("/{id}")
    public ResponseEntity<Resource>
getResourceById(@PathVariable Long id) {
        return resourceService.getResourceById(id)
            .map(ResponseEntity::ok)
            .orElseGet(() ->
ResponseEntity.notFound().build());
    }

    @PostMapping
    public ResponseEntity<Resource> createResource(@RequestBody
Resource resource) {
        return
ResponseEntity.ok(resourceService.createResource(resource));
    }

    @PutMapping("/{id}")
    public ResponseEntity<Resource> updateResource(@PathVariable
Long id, @RequestBody Resource resource) {
        return
ResponseEntity.ok(resourceService.updateResource(id, resource));
    }

    @DeleteMapping("/{id}")
    public ResponseEntity<Void> deleteResource(@PathVariable
Long id) {
        resourceService.deleteResource(id);
        return ResponseEntity.ok().build();
    }
}

```

Листинг класса: UserController

```
package controller;

import model.User;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;
import service.UserService;

import java.util.List;

@RestController
@RequestMapping("/api/users")
public class UserController {

    private final UserService userService;

    @Autowired
    public UserController(UserService userService) {
        this.userService = userService;
    }

    @GetMapping
    public ResponseEntity<List<User>> getAllUsers() {
        return ResponseEntity.ok(userService.getAllUsers());
    }

    @GetMapping("/{id}")
    public ResponseEntity<User> getUserById(@PathVariable Long
id) {
        return userService.getUserById(id)
            .map(ResponseEntity::ok)
            .orElseGet(() ->
ResponseEntity.notFound().build());
    }

    @PostMapping
    public ResponseEntity<User> createUser(@RequestBody User
user) {
        return ResponseEntity.ok(userService.createUser(user));
    }

    @PutMapping("/{id}")
    public ResponseEntity<User> updateUser(@PathVariable Long
id, @RequestBody User user) {
        return ResponseEntity.ok(userService.updateUser(id,
user));
    }

    @DeleteMapping("/{id}")
    public ResponseEntity<Void> deleteUser(@PathVariable Long
id) {
```

```

        userService.deleteUser(id);
        return ResponseEntity.ok().build();
    }
}

```

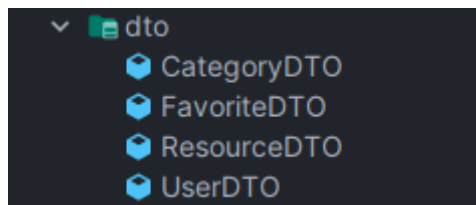
- **dto**: данный пакет содержит DTO (Data Transfer Objects), которые предназначены для передачи данных между клиентом и сервером.

- CategoryDTO: представляет данные категории для передачи.

- FavoriteDTO: представляет данные избранных ресурсов пользователя.

- ResourceDTO: представляет данные учебных ресурсов.

- UserDTO: представляет данные пользователей.



Структура MVC папки в проекте

Листинг класса: **CategoryDTO**

```

package dto;

public class CategoryDTO {

    private Long id;
    private String name;
    private String description;

    // Конструктор по умолчанию
    public CategoryDTO() {
    }

    // Конструктор со всеми полями
    public CategoryDTO(Long id, String name, String description)
    {
        this.id = id;
        this.name = name;
        this.description = description;
    }

    // Геттеры и сеттеры
    public Long getId() {
        return id;
    }
}

```



```

    public void setId(Long id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getDescription() {
        return description;
    }

    public void setDescription(String description) {
        this.description = description;
    }
}

```

Листинг класса: **FavoriteDTO**

```

package dto;

public class FavoriteDTO {
    private Long id;
    private Long userId;
    private Long resourceId;

    // Конструктор без аргументов
    public FavoriteDTO() {
    }

    // Конструктор со всеми полями
    public FavoriteDTO(Long id, Long userId, Long resourceId) {
        this.id = id;
        this.userId = userId;
        this.resourceId = resourceId;
    }

    // Геттеры и сеттеры
    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public Long getUserId() {

```

```

        return userId;
    }

    public void setUserId(Long userId) {
        this.userId = userId;
    }

    public Long getResourceId() {
        return resourceId;
    }

    public void setResourceId(Long resourceId) {
        this.resourceId = resourceId;
    }
}

```

Листинг класса: **ResourceDTO**

```

package dto;

public class ResourceDTO {
    private Long id;
    private String title;
    private String description;
    private String type;
    private String url;
    private Long categoryId; // Принадлежность к категории по ID

    // Конструктор без аргументов
    public ResourceDTO() {
    }

    // Конструктор со всеми полями
    public ResourceDTO(Long id, String title, String
description, String type, String url, Long categoryId) {
        this.id = id;
        this.title = title;
        this.description = description;
        this.type = type;
        this.url = url;
        this.categoryId = categoryId;
    }

    // Геттеры и сеттеры
    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }
}

```

```

    public String getTitle() {
        return title;
    }

    public void setTitle(String title) {
        this.title = title;
    }

    public String getDescription() {
        return description;
    }

    public void setDescription(String description) {
        this.description = description;
    }

    public String getType() {
        return type;
    }

    public void setType(String type) {
        this.type = type;
    }

    public String getUrl() {
        return url;
    }

    public void setUrl(String url) {
        this.url = url;
    }

    public Long getCategoryId() {
        return categoryId;
    }

    public void setCategoryId(Long categoryId) {
        this.categoryId = categoryId;
    }
}

```

Листинг класса: **UserDTO**

```

package dto;

public class UserDTO {
    private Long id;
    private String username;
    private String email;
    // Пароль обычно не включается в DTO, если не
    предполагается, что он передаётся от клиента к серверу,
    например, при регистрации.
}

```

```

    private String password; // Используйте его осторожно,
    только если нужно.

    // Конструктор без аргументов
    public UserDTO() {
    }

    // Конструктор со всеми полями
    public UserDTO(Long id, String username, String email,
String password) {
        this.id = id;
        this.username = username;
        this.email = email;
        this.password = password; // Опять же, будьте осторожны
с паролем.
    }

    // Геттеры и сеттеры
    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public String getUsername() {
        return username;
    }

    public void setUsername(String username) {
        this.username = username;
    }

    public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
        this.email = email;
    }

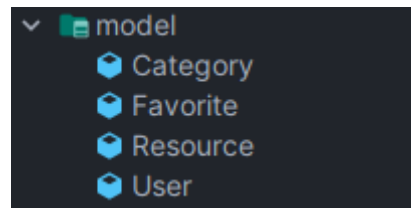
    public String getPassword() {
        return password;
    }

    // Установка пароля должна выполняться с особой
    осторожностью и включать шифрование.
    public void setPassword(String password) {
        this.password = password;
    }
}

```

- **model:** определяет сущности предметной области, которые отображаются на таблицы базы данных.

| |
|---|
| - Category: Отображает категорию учебных ресурсов. |
| - Favorite: Отображает избранное пользователя (какие ресурсы пользователь отметил как избранные). |
| - Resource: Отображает учебный ресурс. |
| - User: Отображает пользователя системы. |



Структура MVC папки в проекте

Листинг класса: **Category**

```
package model;

import jakarta.persistence.*;
import lombok.Getter;
import lombok.Setter;
import lombok.NoArgsConstructor;
import lombok.AllArgsConstructor;

@Entity
@Table(name = "categories")
@Getter
@Setter
@NoArgsConstructor
@AllArgsConstructor
public class Category {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(nullable = false, length = 255)
    private String name;

    @Column(length = 500)
    private String description;

    @Column(name = "deleted")
    private Boolean deleted = false;
}
```

Листинг класса: **Favorite**

```
package model;

import jakarta.persistence.*;
import lombok.Getter;
import lombok.Setter;
import lombok.NoArgsConstructor;
import lombok.AllArgsConstructor;

@Entity
@Table(name = "favorites")
@Getter
@Setter
@NoArgsConstructor
@AllArgsConstructor
public class Favorite {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "user_id", nullable = false)
    private User user;

    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "resource_id", nullable = false)
    private Resource resource;
}
```

Листинг класса: **Resource**

```
package model;

import jakarta.persistence.*;
import lombok.Getter;
import lombok.Setter;
import lombok.NoArgsConstructor;
import lombok.AllArgsConstructor;

@Entity
@Table(name = "resources")
@Getter
@Setter
@NoArgsConstructor
@AllArgsConstructor
public class Resource {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
```

```

    @Column(name = "title", nullable = false, length = 255)
    private String title;

    @Column(length = 1000) // или тип TEXT в PostgreSQL
    private String description;

    @Column(nullable = false, length = 50)
    private String type;

    @Column(nullable = false, length = 255)
    private String url;

    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "category_id")
    private Category category;
}

```

Листинг класса: **User**

```

package model;

import jakarta.persistence.*;
import lombok.Getter;
import lombok.Setter;
import lombok.NoArgsConstructor;
import lombok.AllArgsConstructor;

@Entity
@Table(name = "users")
@Getter
@Setter
@NoArgsConstructor
@AllArgsConstructor
public class User {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(nullable = false, unique = true, length = 255)
    private String username;

    @Column(nullable = false, unique = true, length = 255)
    private String email;

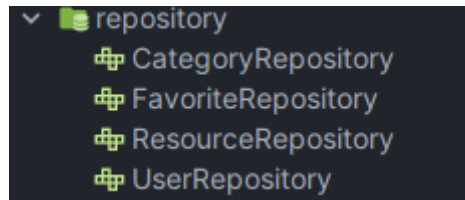
    @Column(nullable = false, length = 255)
    private String password;

    // Lombok берёт на себя создание геттеров и сеттеров.
}

```

- **repository**: Пакет содержит интерфейсы Spring Data JPA репозиторий для взаимодействия с базой данных.

- | |
|--|
| - CategoryRepository: Предоставляет JPA операции для таблицы категорий. |
| - FavoriteRepository: Предоставляет JPA операции для таблицы избранного. |
| - ResourceRepository: Предоставляет JPA операции для таблицы ресурсов. |
| - UserRepository: Предоставляет JPA операции для таблицы пользователей. |



Структура MVC папки в проекте

Листинг класса: **CategoryRepository**

```
package repository;

import model.Category;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

import java.util.List;

@Repository
public interface CategoryRepository extends
    JpaRepository<Category, Long> {
    // Методы для поиска категорий, если они вам нужны,
    // например, по имени
    Category findByName(String name);

    List<Category> findAllByDeletedFalse();
}
```

Листинг класса: **FavoriteRepository**

```
package repository;

import java.util.List;
import model.Favorite;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

@Repository
public interface FavoriteRepository extends
    JpaRepository<Favorite, Long> {
    // Методы для работы с избранными записями, например, поиск
```



```
всех избранных для пользователя
    List<Favorite> findByUserId(Long userId);
    List<Favorite> findByResourceId(Long resourceId);
}
```

Листинг класса: **ResourceRepository**

```
package repository;

import model.Resource;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

import java.util.List;

@Repository
public interface ResourceRepository extends
    JpaRepository<Resource, Long> {
    // Методы для поиска ресурсов, например, по типу или
    категории
    List<Resource> findByType(String type);
    List<Resource> findByCategoryId(Long categoryId);
}
```

Листинг класса: **UserRepository**

```
package repository;

import model.User;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

@Repository
public interface UserRepository extends JpaRepository<User,
    Long> {
    // Методы для поиска пользователей, например, по имени
    пользователя или email
    User findByUsername(String username);
    User findByEmail(String email);
}
```

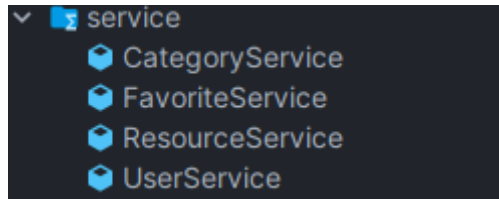
- **service:** Содержит классы сервисов, которые включают бизнес-логику и служат прослойкой между контроллерами и репозиториями.

- **CategoryService:** Содержит бизнес-логику для работы с категориями ресурсов.

- **FavoriteService:** Содержит бизнес-логику для работы с избранными ресурсами пользователя.

- **ResourceService:** Содержит бизнес-логику для работы с учебными ресурсами.

- **UserService:** Содержит бизнес-логику для работы с пользователями.



Структура MVC папки в проекте

Листинг класса: **CategoryService**

```
package service;

import model.Category;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import repository.CategoryRepository;
import java.util.List;
import java.util.Optional;

@Service
public class CategoryService {

    private final CategoryRepository categoryRepository;

    @Autowired
    public CategoryService(CategoryRepository categoryRepository) {
        this.categoryRepository = categoryRepository;
    }

    public List<Category> getAllCategories() {
        // Изменено на использование метода, который возвращает
        // только неудаленные категории
        return categoryRepository.findAllByDeletedFalse();
    }

    public Optional<Category> getCategoryById(Long id) {
        // Добавлена проверка, что категория не помечена как
        // удаленная
        return categoryRepository.findById(id)
    }
```

```

        .filter(category -> !category.getDeleted());
    }

    public Category createCategory(Category category) {
        return categoryRepository.save(category);
    }

    public Category updateCategory(Long id, Category
categoryDetails) {
        Category category = getCategoryById(id)
            .orElseThrow(() -> new
RuntimeException("Category not found with id " + id));
        category.setName(categoryDetails.getName());

        category.setDescription(categoryDetails.getDescription());
        // Здесь нет необходимости менять логику, так как
        обновление уже работает с существующими записями
        return categoryRepository.save(category);
    }

    public void deleteCategory(Long id) {
        // Изменено на установку флага deleted в true вместо
        физического удаления
        Category category = getCategoryById(id)
            .orElseThrow(() -> new
RuntimeException("Category not found with id " + id));
        category.setDeleted(true);
        categoryRepository.save(category);
    }
}

```

Листинг класса: **FavoriteService**

```

package service;

import model.Favorite;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import repository.FavoriteRepository;
import java.util.List;
import java.util.Optional;

@Service
public class FavoriteService {

    private final FavoriteRepository favoriteRepository;

    @Autowired
    public FavoriteService(FavoriteRepository
favoriteRepository) {
        this.favoriteRepository = favoriteRepository;
    }
}

```

```

    public List<Favorite> getAllFavorites() {
        return favoriteRepository.findAll();
    }

    public Optional<Favorite> getFavoriteById(Long id) {
        return favoriteRepository.findById(id);
    }

    public Favorite addFavorite(Favorite favorite) {
        // Здесь может быть логика для предотвращения
дублирования
        return favoriteRepository.save(favorite);
    }

    public void removeFavorite(Long id) {
        Favorite favorite = favoriteRepository.findById(id)
            .orElseThrow(() -> new
RuntimeException("Favorite not found with id " + id));
        favoriteRepository.delete(favorite);
    }

    // Дополнительные методы
}

```

Листинг класса: **ResourceService**

```

package service;

import model.Resource;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import repository.ResourceRepository;
import java.util.List;
import java.util.Optional;

@Service
public class ResourceService {

    private final ResourceRepository resourceRepository;

    @Autowired
    public ResourceService(ResourceRepository
resourceRepository) {
        this.resourceRepository = resourceRepository;
    }

    public List<Resource> getAllResources() {
        return resourceRepository.findAll();
    }

    public Optional<Resource> getResourceById(Long id) {

```

```

        return resourceRepository.findById(id);
    }

    public Resource createResource(Resource resource) {
        return resourceRepository.save(resource);
    }

    public Resource updateResource(Long id, Resource
resourceDetails) {
        Resource resource = resourceRepository.findById(id)
            .orElseThrow(() -> new
RuntimeException("Resource not found with id " + id));
        resource.setTitle(resourceDetails.getTitle());

        resource.setDescription(resourceDetails.getDescription());
        resource.setType(resourceDetails.getType());
        resource.setUrl(resourceDetails.getUrl());
        // Обновление связанных категорий, если это необходимо:
        // resource.setCategory(resourceDetails.getCategory());
        return resourceRepository.save(resource);
    }

    public void deleteResource(Long id) {
        Resource resource = resourceRepository.findById(id)
            .orElseThrow(() -> new
RuntimeException("Resource not found with id " + id));
        resourceRepository.delete(resource);
    }
}

```

Листинг класса: UserService

```

package service;

import model.User;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import repository.UserRepository;
import java.util.List;
import java.util.Optional;

@Service
public class UserService {

    private final UserRepository userRepository;

    @Autowired
    public UserService(UserRepository userRepository) {
        this.userRepository = userRepository;
    }
}

```

```

public List<User> getAllUsers() {
    return userRepository.findAll();
}

public Optional<User> getUserById(Long id) {
    return userRepository.findById(id);
}

public User createUser(User user) {
    // Дополнительная логика, например, шифрование пароля
    return userRepository.save(user);
}

public User updateUser(Long id, User userDetails) {
    User user = userRepository.findById(id)
        .orElseThrow(() -> new RuntimeException("User
not found with id " + id));
    user.setUsername(userDetails.getUsername());
    user.setEmail(userDetails.getEmail());
    // Не забудьте добавить логику для обновления пароля с
шифрованием
    return userRepository.save(user);
}

public void deleteUser(Long id) {
    User user = userRepository.findById(id)
        .orElseThrow(() -> new RuntimeException("User
not found with id " + id));
    userRepository.delete(user);
}

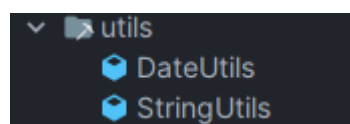
// Дополнительные методы
}

```

- **utils:** Пакет содержит утилитные классы, которые предоставляют общие функции, используемые в различных частях приложения.

- DateUtils: Утилитный класс для работы с датами.

- StringUtils: Утилитный класс для работы со строками.



Структура MVC папки в проекте

Листинг класса: **DateUtils**

```

package utils;

import java.time.LocalDate;
import java.time.format.DateTimeFormatter;

```

```

public class DateUtils {

    public static LocalDate addDaysToDate(LocalDate date, int
daysToAdd) {
        return date.plusDays(daysToAdd);
    }

    private DateUtils() {
        // ЗАКРЫТЫЙ КОНСТРУКТОР
    }

    public static String formatDate(LocalDate date) {
        DateTimeFormatter formatter =
DateTimeFormatter.ofPattern("dd-MM-yyyy");
        return date.format(formatter);
    }
}

```

Листинг класса: **StringUtils**

```

package utils;

public class StringUtils {

    private StringUtils() {
        // ЗАКРЫТЫЙ КОНСТРУКТОР для предотвращения создания
экземпляра утилитного класса
    }

    public static boolean isNullOrEmpty(String str) {
        return str == null || str.trim().isEmpty();
    }
}

```

Дополнительные Компоненты

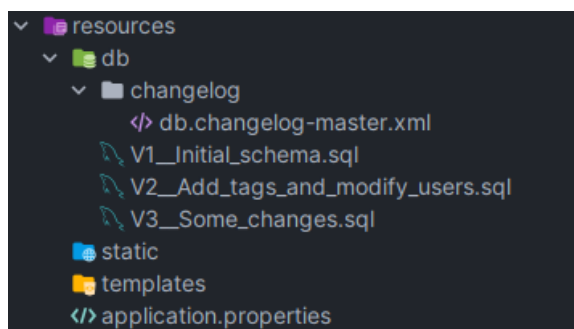
- **resources:** Содержит ресурсы приложения, такие как файлы конфигурации, HTML шаблоны и другие ресурсы.

- **db:** Может содержать SQL-скрипты для инициализации базы данных.

- **static:** Хранит статические ресурсы, такие как CSS, JavaScript, изображения.

- **templates:** Содержит шаблоны для веб-страниц (например, Thymeleaf шаблоны).

- **application.properties:** Файл свойств для конфигурации приложения.



Структура MVC папки в проекте

Листинг настроек: **application.properties**

```
spring.application.name=Project

# Datasource
spring.datasource.url=jdbc:postgresql://localhost:5432/java_learning_resources
spring.datasource.username=postgres
spring.datasource.password=1447900

# JPA/Hibernate
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
spring.jpa.properties.hibernate.format_sql=true

# Logging
logging.level.org.hibernate.SQL=DEBUG
logging.level.org.hibernate.type.descriptor.sql.BasicBinder=TRACE

# Migration Data Base
spring.flyway.baseline-on-migrate=true
spring.flyway.schemas=public
spring.flyway.locations=classpath:db/migration
```



```
spring.liquibase.change-  
log=classpath:/db/changelog/db.changelog-master.xml
```

Листинг конфигурации: **pom.xml**

```
<?xml version="1.0" encoding="UTF-8"?>  
<project xmlns="http://maven.apache.org/POM/4.0.0"  
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0  
https://maven.apache.org/xsd/maven-4.0.0.xsd">  
  <modelVersion>4.0.0</modelVersion>  
  <parent>  
    <groupId>org.springframework.boot</groupId>  
    <artifactId>spring-boot-starter-parent</artifactId>  
    <version>3.2.4</version>  
    <relativePath/> <!-- lookup parent from repository -->  
  </parent>  
  <groupId>org.example</groupId>  
  <artifactId>Project</artifactId>  
  <version>0.0.1-SNAPSHOT</version>  
  <name>Project</name>  
  <description>Project</description>  
  <properties>  
    <java.version>21</java.version>  
  </properties>  
  <dependencies>  
    <dependency>  
      <groupId>org.springframework.boot</groupId>  
      <artifactId>spring-boot-starter-data-  
jpa</artifactId>  
    </dependency>  
    <dependency>  
      <groupId>org.springframework.boot</groupId>  
      <artifactId>spring-boot-starter-  
security</artifactId>  
    </dependency>  
    <dependency>  
      <groupId>org.springframework.boot</groupId>  
      <artifactId>spring-boot-starter-web</artifactId>  
    </dependency>  
  
    <dependency>  
      <groupId>org.postgresql</groupId>  
      <artifactId>postgresql</artifactId>  
      <scope>runtime</scope>  
    </dependency>  
    <dependency>  
      <groupId>org.projectlombok</groupId>  
      <artifactId>lombok</artifactId>  
      <optional>true</optional>  
    </dependency>
```

```

    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-test</artifactId>
      <scope>test</scope>
    </dependency>
    <dependency>
      <groupId>org.springframework.security</groupId>
      <artifactId>spring-security-test</artifactId>
      <scope>test</scope>
    </dependency>
    <dependency>
      <groupId>org.flywaydb</groupId>
      <artifactId>flyway-core</artifactId>
    </dependency>
    <dependency>
      <groupId>org.liquibase</groupId>
      <artifactId>liquibase-core</artifactId>
    </dependency>
    <dependency>
      <groupId>org.testng</groupId>
      <artifactId>testng</artifactId>
      <version>RELEASE</version>
      <scope>compile</scope>
    </dependency>

    <dependency>
      <groupId>org.mockito</groupId>
      <artifactId>mockito-core</artifactId>
      <version>5.11.0</version>
      <scope>test</scope>
    </dependency>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-test-
autoconfigure</artifactId>
      <version>3.2.4</version>
    </dependency>

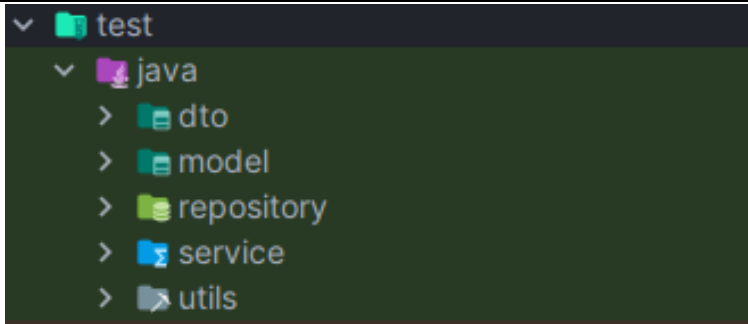
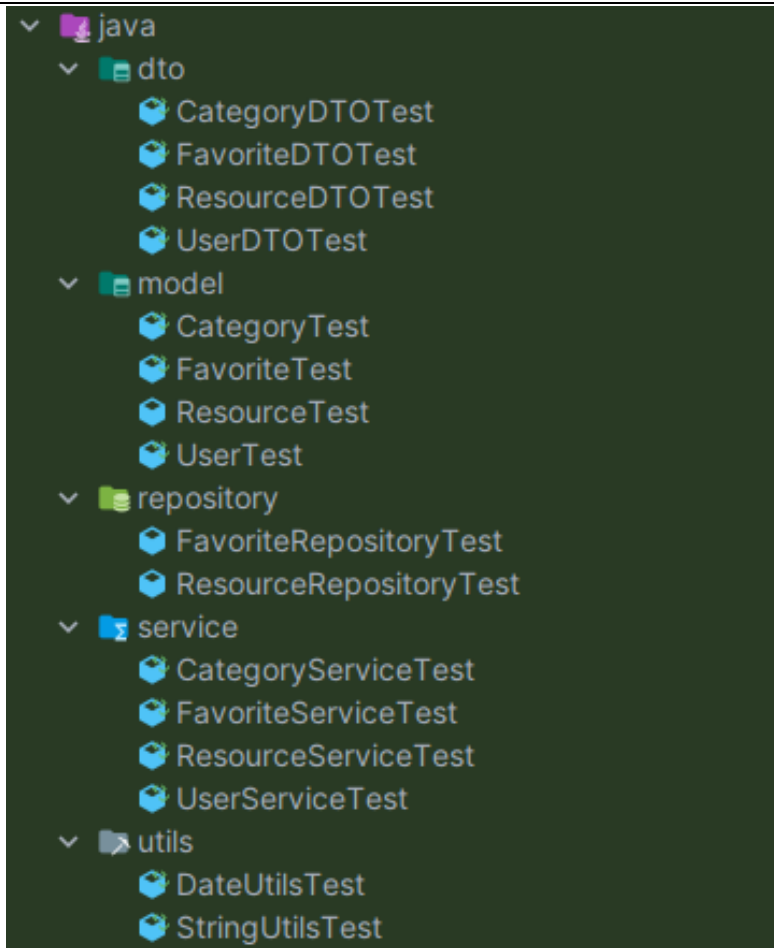
  </dependencies>

  <build>
    <plugins>
      <plugin>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-maven-
plugin</artifactId>
      </plugin>
    </plugins>
  </build>
</project>

```

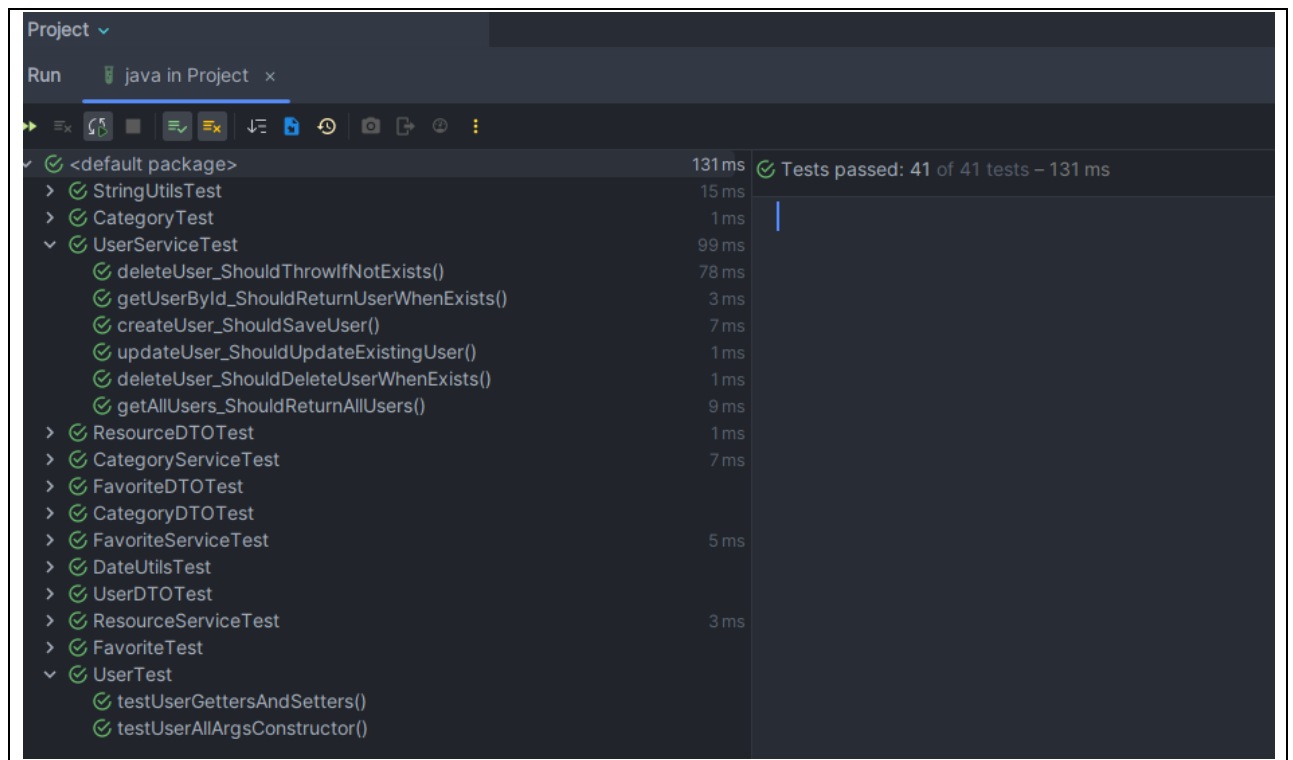
Тестовые Компоненты

| |
|--|
| - test: Пакет содержит тесты для приложения. |
| - dto: Тесты для DTO классов. |
| - model: Тесты для модельных классов. |
| - repository: Тесты для репозиториев. |

| | |
|---|--|
|  | |
|  | |

Структура MVC папки в проекте

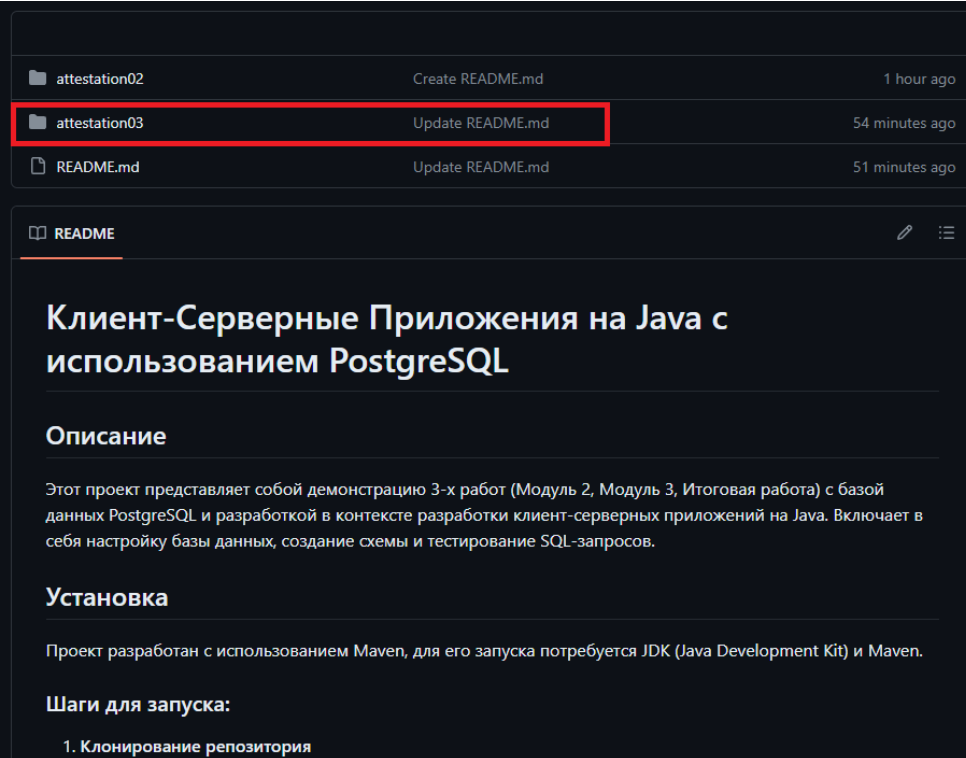
Эта структура отражает типичный Spring Boot проект, где четко разграничены слои представления, бизнес-логики, доступа к данным и утилитные функции.



Результат тестов серверного приложения.

3. Выгрузка проекта на репозиторий GitHub

3.1. Создание структуры папок с Readme.md



attestation02 Create README.md 1 hour ago

attestation03 Update README.md 54 minutes ago

README.md Update README.md 51 minutes ago

Клиент-Серверные Приложения на Java с использованием PostgreSQL

Описание

Этот проект представляет собой демонстрацию 3-х работ (Модуль 2, Модуль 3, Итоговая работа) с базой данных PostgreSQL и разработкой в контексте разработки клиент-серверных приложений на Java. Включает в себя настройку базы данных, создание схемы и тестирование SQL-запросов.

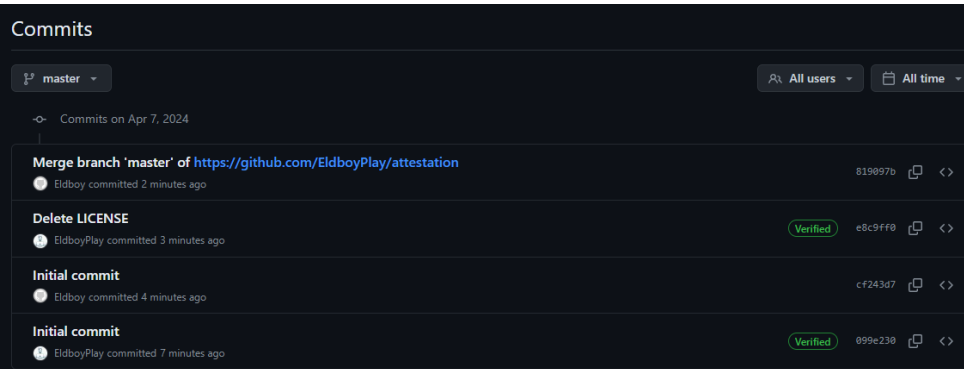
Установка

Проект разработан с использованием Maven, для его запуска потребуется JDK (Java Development Kit) и Maven.

Шаги для запуска:

1. Клонирование репозитория

| Name | Last commit message | Last commit date |
|--------------|---------------------|------------------|
| .. | | |
| .mvn/wrapper | Initial commit | 4 minutes ago |
| src | Initial commit | 4 minutes ago |
| .gitignore | Initial commit | 4 minutes ago |
| README.md | Initial commit | 4 minutes ago |
| mvnw | Initial commit | 4 minutes ago |
| mvnw.cmd | Initial commit | 4 minutes ago |
| pom.xml | Initial commit | 4 minutes ago |



Commits

master

Commits on Apr 7, 2024

- Merge branch 'master' of <https://github.com/EldboyPlay/attestation>
Eldboy committed 2 minutes ago 819897b
- Delete LICENSE
EldboyPlay committed 3 minutes ago e8c9ff8
- Initial commit
Eldboy committed 4 minutes ago cf243d7
- Initial commit
EldboyPlay committed 7 minutes ago 099e230