Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования «Российский химико-технологический университет имени Д.И. Менделеева»

Факультет цифровых технологий и химического инжиниринга

Кафедра информационных компьютерных технологий

ОТЧЕТ ПО КУРСОВОЙ РАБОТЕ ПО КУРСУ

«ТЕХНОЛОГИИ ПРОГРАММИРОВАНИЯ»

НА ТЕМУ

«ВЕБ-ПЛАТФОРМА ДЛЯ ПОИСКА СТАЖИРОВОК»

Ведущий преподаватель

Васецкий А.М.

Выполнил:

Студент группы КС-24

Кошовец Н.А

Москва

2025г

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Актуальность темы

Современные цифровые платформы для поиска стажировок не всегда удовлетворяют требованиям студентов и начинающих специалистов. Основные проблемы включают перегруженность интерфейсов, отсутствие систем избранного и низкую структурированность описаний. Это приводит к потере релевантных предложений и усложнению процесса выбора. В условиях растущей конкуренции на рынке труда необходимо решение, ориентированное на простоту, точность поиска и удобство управления стажировками.

Целью данной курсовой работы является разработка веб-приложения "Internship Management System" — платформы для поиска и администрирования стажировок с чётко структурированным представлением информации и гибкой системой фильтрации.

Задачи работы

- Сформулировать функциональные и нефункциональные требования к системе.
- Спроектировать архитектуру веб-приложения.
- Разработать реляционную модель базы данных.
- Реализовать backend с использованием Java и фреймворка Spring Boot.
- Разработать интерфейс пользователя с помощью HTML, CSS и JavaScript.
- Провести анализ результатов.

Объект и предмет исследования

Объектом исследования является процесс взаимодействия пользователей с системой поиска и управления стажировками. Предметом исследования — проектирование и реализация веб-приложения, обеспечивающего эффективную работу с данными стажировок.

Методы и средства реализации

Разработка велась с применением современных методов программной инженерии: клиент-серверной архитектуры, REST API. В качестве технологий использовались Java 21, Spring Boot, Spring Security, Spring Data JPA, Spring MVC для серверной части, PostgreSQL для хранения данных, а также HTML, CSS и JavaScript для клиентской части.

Структура работы

Структура работы включает проектирование и реализацию приложения, а также оценку достигнутых результатов и выводы по эффективности предложенного решения.

1. ПОСТАНОВКА ЗАДАЧИ

Функциональные требования

1. Регистрация и авторизация пользователей с разграничением прав доступа (пользователь/администратор).
2. Возможность поиска стажировок по фильтрам: совпадение по названию/описанию, категория, тип оплаты, местоположение, дата создания/обновления.
3. Просмотр детальной информации о стажировках.
4. Добавление стажировок в избранное и управление списком избранного.
5. Для администратора — возможность выполнения CRUD-операций (создание, просмотр, редактирование, удаление) над стажировками и категориями.
6. Загрузка изображений стажировок, их отображение в пользовательском интерфейсе.

Нефункциональные требования

1. Безопасность: авторизация пользователей с использованием Spring Security, защита от несанкционированного доступа к административному функционалу.
2. Надёжность хранения данных: использование реляционной базы данных PostgreSQL.
3. Удобство использования: интуитивный и адаптивный интерфейс на стороне клиента.
4. Разделение ответственности: соблюдение принципов многослойной архитектуры и модульности кода.
5. Работа с файлами: изображения стажировок хранятся на диске, ссылки на них — в базе данных; сервер раздаёт изображения как статические ресурсы.

2. ОПИСАНИЕ АРХИТЕКТУРЫ

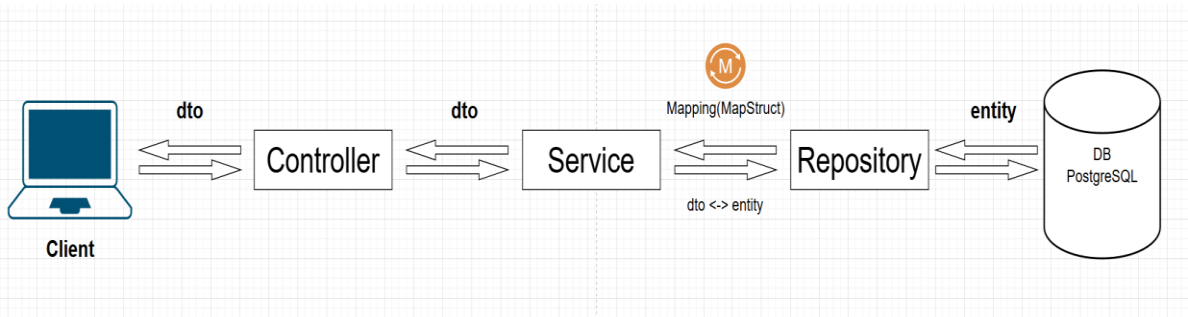


Рис.2.1 Архитектура взаимодействия компонентов веб-приложения

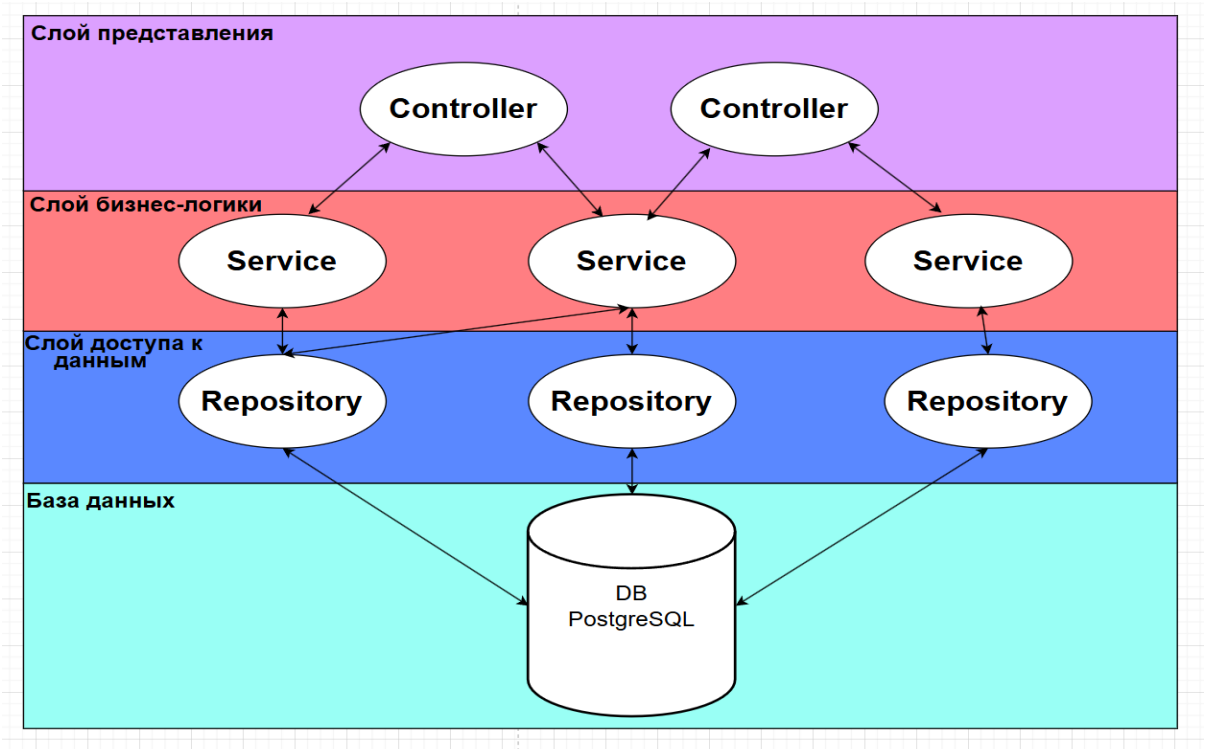


Рис.2.2 Многоуровневая архитектура веб-приложения

Приложение реализовано с использованием классической многослойной архитектуры, которая включает четыре основных уровня:

1. **Presentation Layer (представление)** — отвечает за обработку HTTP-запросов и формирование ответов. Представлен контроллерами (Controller), которые принимают входящие данные от клиента в виде DTO (Data Transfer Objects - объект передачи данных) и возвращают DTO-ответы. На этом уровне реализовано взаимодействие с пользователем через REST API.

2. **Business Layer (бизнес-логика)** — содержит сервисы (Service), которые реализуют основную бизнес-логику приложения. Сервисы обрабатывают данные, полученные от контроллеров, и взаимодействуют с уровнем доступа к данным. Здесь же происходит маппинг DTO<->ENTITY объекты.
3. **Persistence Layer (доступ к данным)** — реализован с использованием репозиториев (Repository), основанных на Spring Data JPA. Репозитории взаимодействуют напрямую с базой данных через Entity-объекты, выполняя операции сохранения, извлечения, обновления и удаления данных.
4. **Database Layer (база данных)** — PostgreSQL используется как система управления реляционной базой данных. Все данные о стажировках, пользователях, избранных стажировках и категориях хранятся в таблицах, структура которых была спроектирована в соответствии с реляционной моделью.

Для преобразования данных между слоями бизнес-логики и доступа к данным используется библиотека MapStruct, обеспечивающая автоматическое преобразование между DTO и Entity.

Дополнительно реализована работа с файловым хранилищем: изображения стажировок сохраняются на жёсткий диск сервера, а в базе данных хранятся только пути к этим файлам. Backend раздаёт изображения как статические ресурсы по этим путям.

Такая архитектура обеспечивает модульность, масштабируемость и упрощает сопровождение приложения. Каждый уровень изолирован от других, что способствует более чистому коду и соблюдению принципов SOLID.

Примечания:

DTO (Data Transfer Object) — это шаблон проектирования, предназначенный для безопасной и эффективной передачи данных между слоями приложения или между клиентом и сервером. В отличие от сущностей (Entity), которые представляют собой модели базы данных, DTO содержит только те поля, которые действительно необходимы в конкретном контексте передачи.

3. ПРОЕКТИРОВАНИЕ БАЗЫ ДАННЫХ

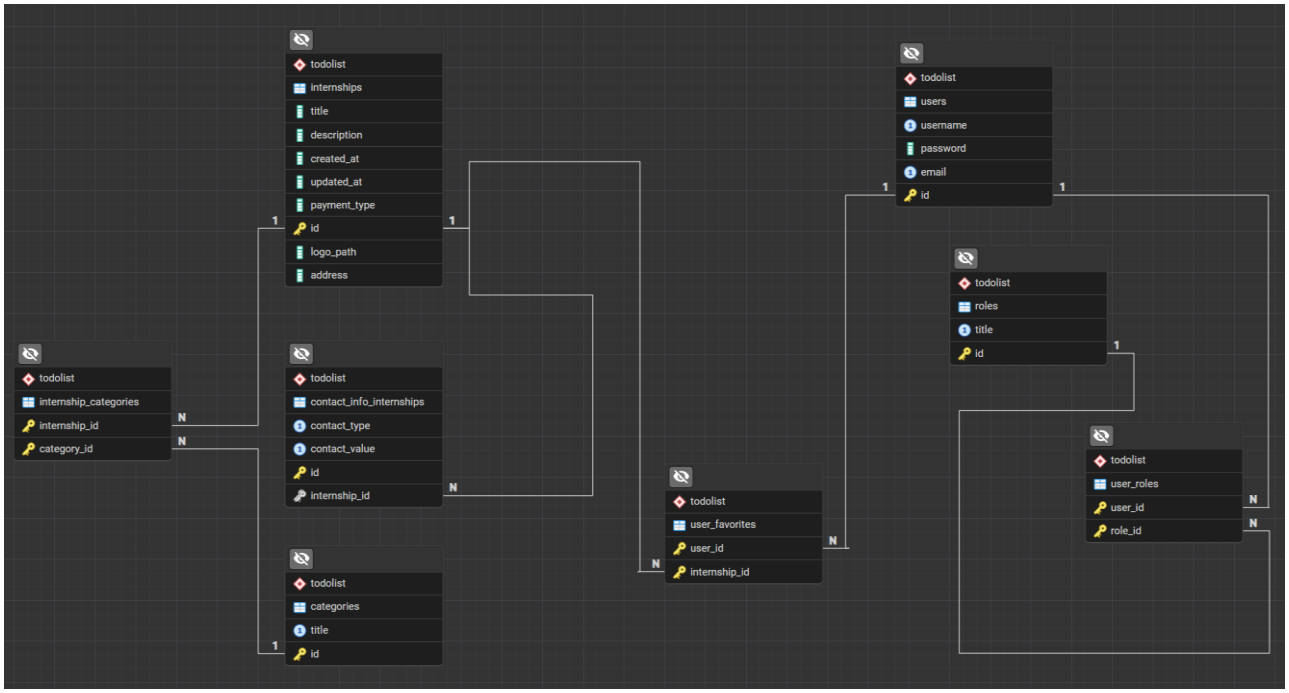


Рис 3.1 ERD

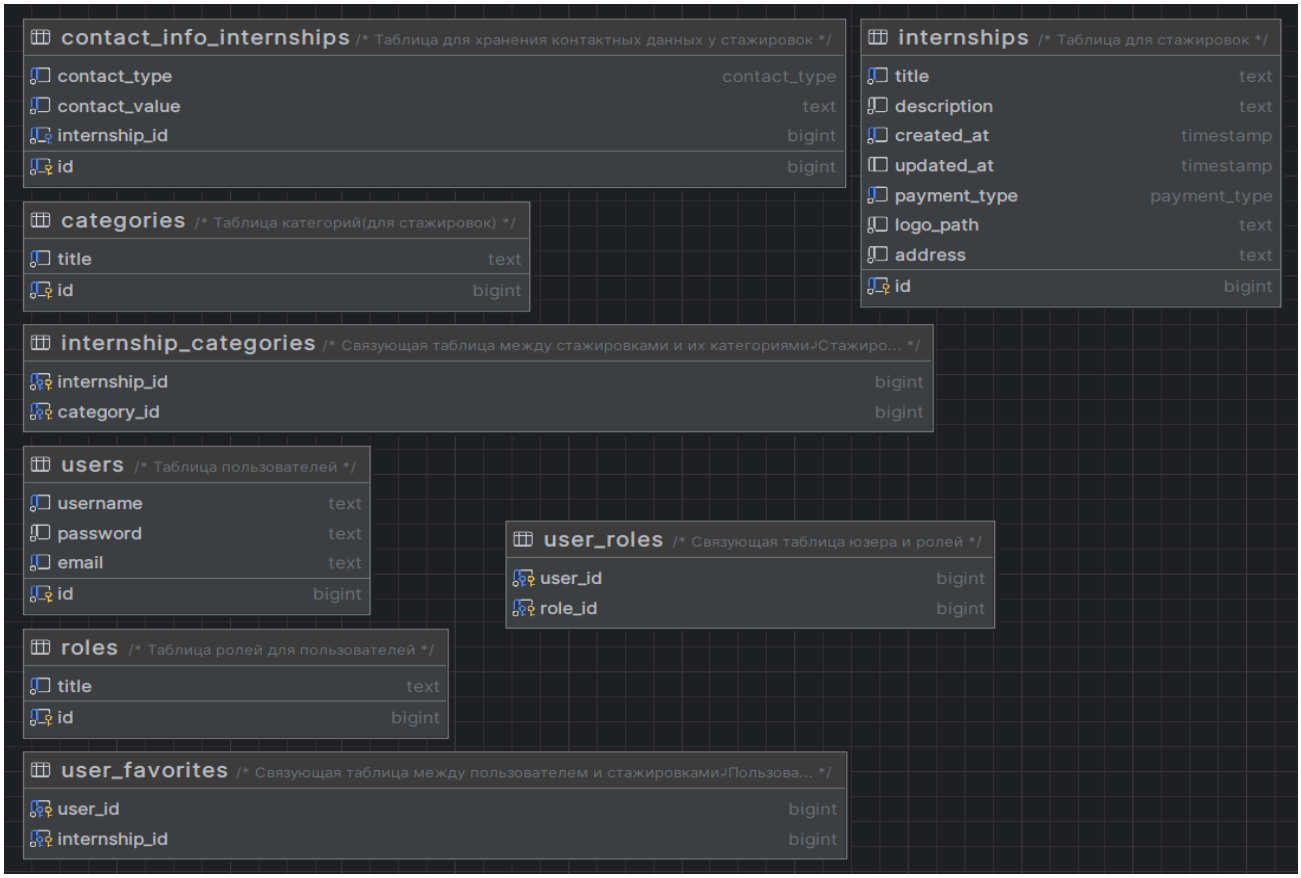


Рис 3.2 Схема таблиц

*ERD (Entity-Relationship-Diagram) — графическая схема, которая показывает, какие данные есть в системе и как они связаны.

Связи между таблицами и их обоснование

В данной базе данных реализовано несколько видов связей между таблицами, направленных на обеспечение целостности данных и отражение логических зависимостей между сущностями системы.

users – roles

Связь: многие ко многим через связующую таблицу user_roles.
Обоснование: Один пользователь может иметь несколько ролей (например: "USER", "ADMIN"), и одна роль может принадлежать нескольким пользователям. Такая структура обеспечивает гибкость при управлении доступами и ролями в системе. Использование связывающей таблицы user_roles позволяет нормализовать данные и избежать дублирования.

users – internships

Связь: многие ко многим через связующую таблицу user_favorites.
Обоснование: Один пользователь может добавить в избранное множество стажировок, и одна стажировка может быть в избранном у разных пользователей. Это реализовано через таблицу user_favorites, которая обеспечивает удобное хранение пользовательских предпочтений и упрощает реализацию функционала избранного.

internships - contact_info_internships

Связь: один ко многим
Обоснование: Каждая стажировка может иметь несколько контактных способов (например, email, телефон, Telegram). Это реализуется через таблицу contact_info_internships, где каждая запись указывает тип контакта и его значение. Такое разделение позволяет удобно масштабировать типы контактов и избегать повторяющихся столбцов в основной таблице стажировок.

internships - categories

Связь: многие ко многим через связующую таблицу internship_categories
Обоснование: Стажировка может относиться к нескольким категориям (например: IT, HR), а одна категория может включать множество стажировок. Таблица-связка internship_categories позволяет реализовать эту гибкость и при этом сохраняет структурную чёткость данных

Использование B-tree индексов

В рамках проектирования базы данных вручную были созданы следующие B-tree индексы:

1. ***contact_info_internships_internship_id_idx***
   (таблица: contact_info_internships, поле: internship_id)
   Назначение: индекс ускоряет выполнение операций выборки и соединения (JOIN) по полю internship_id, особенно при получении контактной информации, связанной с конкретной стажировкой.
2. ***internship_categories_category_id_idx***
   (таблица: internship_categories, поле: category_id)
   Назначение: индекс обеспечивает быструю выборку стажировок по определённой категории. Также ускоряет JOIN-операции между таблицами internship_categories и categories.
3. ***internship_categories_internship_id_idx***
   (таблица: internship_categories, поле: internship_id)
   Назначение: индекс повышает производительность при получении всех категорий, связанных с конкретной стажировкой, и при удалении стажировок (где требуется каскадное удаление связей).
4. ***user_favorites_user_id_idx***
   (таблица: user_favorites, поле: user_id)
   Назначение: позволяет эффективно извлекать все стажировки, добавленные в избранное конкретным пользователем. Особенно важно при отображении пользовательского списка избранных стажировок.
5. ***user_roles_user_id_idx***
   (таблица: user_roles, поле: user_id)
   Назначение: ускоряет выборку ролей, привязанных к конкретному пользователю, что критично при проверке авторизации и отображении прав доступа.

Эти индексы:

- Ускоряют операции соединения таблиц (JOIN).
- Повышают производительность при фильтрации по внешним ключам.
- Обеспечивают быструю проверку при удалении и обновлении записей.

4. РАЗРАБОТКА BACKEND ЧАСТИ

В качестве серверной части приложения используется фреймворк Spring Boot, который обеспечивает быструю настройку, автоматическую конфигурацию и встроенный веб-сервер. Благодаря Spring Boot, разработка REST API сводится к минимальному количеству шаблонного кода, а интеграция с другими модулями осуществляется через аннотации и встроенные механизмы Spring.

Помимо основного фреймворка использовались следующие библиотеки для автоматизации: MapStruct (для маппинга dto←→entity), Lombok (для сокращения шаблонного кода через аннотации), Liquibase(для миграций бд), JJWT-api (для работы с JSON Web Tokens), Swagger (для ручного тестирования api).

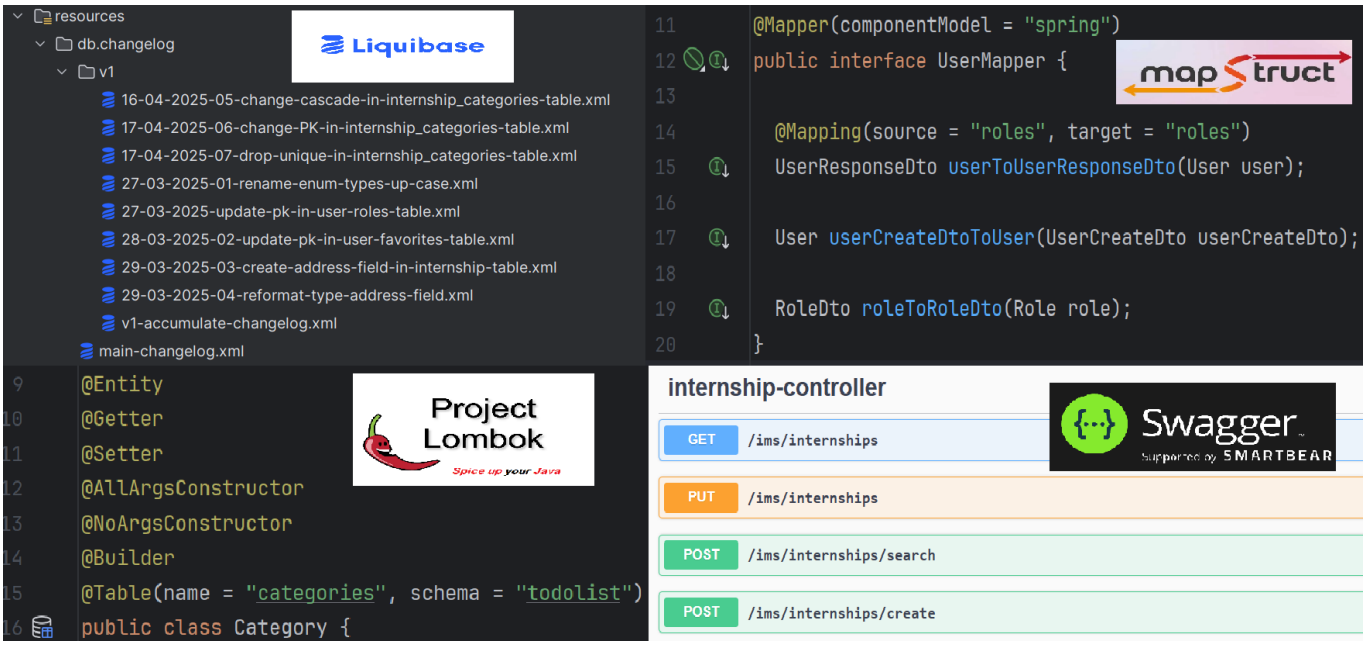


Рис.4.1 Демонстрация работы перечисленных библиотек

Структура проекта организована по многослойной архитектуре. Каждый модуль (controller, service, repository, dto, entity…) вынесен в отдельный пакет, что способствует модульности, читаемости и упрощает сопровождение (см. рис. 4.2). Такая структура облегчает тестирование и позволяет разделить зоны ответственности между компонентами приложения.

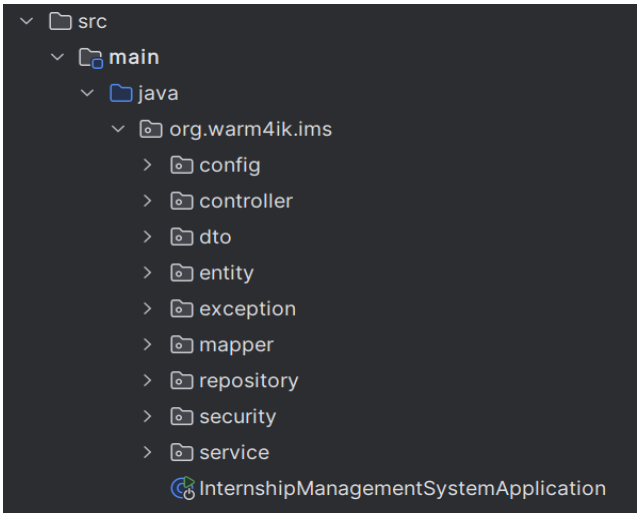


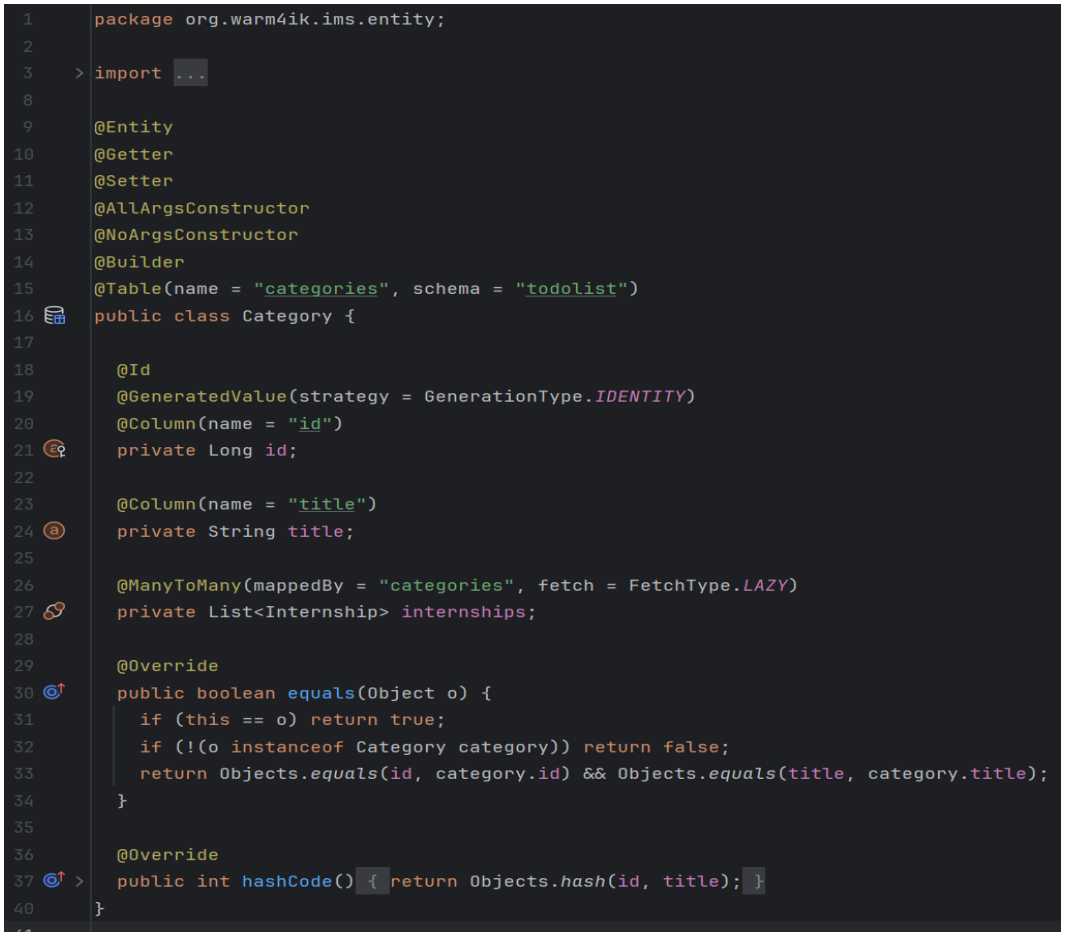
Рис.4.2 Структура проекта

Для взаимодействия с базой данных используется Spring Data JPA — надстройка над JPA (спецификация Java для управления реляционными данными в приложениях), которая позволяет работать с репозиториями без необходимости вручную писать SQL-запросы. Достаточно описать интерфейс, и Spring сам реализует нужные методы (например: findByEmail - см. рис. 4.3). Это существенно ускоряет процесс разработки и упрощает поддержку кода.

```
1  package org.warm4ik.ims.repository;
2
3  import ...
9
10 @Repository
11 public interface UserRepository extends CrudRepository<User, Long> {
12
13   @EntityGraph(attributePaths = {"roles"})
14   Optional<User> findByEmail(String email);
15
16   boolean existsUserByEmail(String email);
17
18   boolean existsUserByUsername(String username);
19 }
```

Рис.4.3 Код репозитория UserRepository.

Также используется Hibernate — реализация ORM (Object-Relational Mapping), которая позволяет описывать таблицы базы данных в виде Java-классов (Entity, см. рис.4.4). Это делает взаимодействие с БД более наглядным, безопасным и удобным: операции CRUD реализуются через методы JPA-репозиториев, а Hibernate автоматически преобразует Java-объекты в SQL-запросы и обратно.



```java
package org.warm4ik.ims.entity;

import ...

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

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

  @Column(name = "title")
  private String title;

  @ManyToMany(mappedBy = "categories", fetch = FetchType.LAZY)
  private List<Internship> internships;

  @Override
  public boolean equals(Object o) {
    if (this == o) return true;
    if (!(o instanceof Category category)) return false;
    return Objects.equals(id, category.id) && Objects.equals(title, category.title);
  }

  @Override
  public int hashCode() { return Objects.hash(id, title); }
}
```

Рис.4.4 Сущность – Category.

В системе реализована безопасность на основе JWT (JSON Web Token). При входе пользователю выдаются два токена: токен доступа, используемый для авторизации в последующих запросах, и токен обновления, позволяющий обновлять токен доступа по истечении его срока действия. Регистрация и аутентификация реализованы через контроллеры (см. рис. 4.6), а проверка токена и авторизация маршрутов — через фильтры Spring Security (см. рис. 4.7). Для хеширования паролей применяется алгоритм BCrypt, что предотвращает хранение паролей в открытом виде и защищает от прямого доступа в случае утечки данных (см. рис.4.5)

	username	password
1	Egorik123	$2a$07$8Q8rfAa0swZWmwEbxcCgRetxFHhJqVI7S4VzGp8/03rh8zCVqxNPK
2	testPass	$2a$07$rG.AkVUZIYPK647jwcnSReZQUqTjFhqFFEfB2Bvu28L/YrlBDQcA2
3	Nikitia	$2a$07$k1dT/HF8b9jxo0Egz8mtsuobGWajTEwOxYyPqJsV3E8DBnqscm5rW

Рис.4.5 Демонстрация хэширования пароля

```java
@RequiredArgsConstructor
@RestController
@RequestMapping("/auth")
public class AuthController {

  private final UserService userServiceImpl;
  private final AuthService authServiceImpl;

  @PostMapping("/sign-in")
  public ResponseEntity<JwtAuthenticationDto> signIn(
      @RequestBody @Valid UserCredentialDto userCredentialDto) {

    JwtAuthenticationDto jwtAuthenticationDto = authServiceImpl.signIn(userCredentialDto);

    return ResponseEntity.ok(jwtAuthenticationDto);
  }

  @PostMapping("/refresh")
  public JwtAuthenticationDto refresh(@RequestBody @Valid RefreshTokenDto refreshTokenDto) {
    return authServiceImpl.refreshToken(refreshTokenDto);
  }

  @PostMapping("/registration")
  public ResponseEntity<UserResponseDto> createUser(@RequestBody @Valid UserCreateDto userCreateDto) {
    return new ResponseEntity<>(userServiceImpl.addUser(userCreateDto), HttpStatus.CREATED);
  }

  @PostMapping("/logout")
  public ResponseEntity<Void> logout(@RequestBody RefreshTokenDto dtoRT){
    authServiceImpl.revokeRefreshToken(dtoRT.refreshToken());
    return ResponseEntity.noContent().build();
  }

}
```

Рис.4.6 Демонстрация контроллера – AuthController.

```java
@RequiredArgsConstructor
@Configuration
@EnableWebSecurity
@EnableMethodSecurity
@EnableScheduling
public class SecurityConfig {

  private final JwtFilter jwtFilter;

  @Bean
  public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {

    http.httpBasic(AbstractHttpConfigurer::disable)
        .csrf(AbstractHttpConfigurer::disable)
        .authorizeHttpRequests(
            auth ->
                auth.requestMatchers(
                        "/v3/api-docs/**",
                        "/swagger-ui/**",
                        "/auth/**",
                        "/main/**",
                        "/favorite/**",
                        "/avatars/**",
                        "/utils/**",
                        "/img/**",
                        "/category/*",
                        "/favicon.ico") AuthorizedUrl
                    .permitAll() AuthorizationManagerRequestMat...
                    .anyRequest() AuthorizedUrl
                    .authenticated())
        .sessionManagement(
            session -> session.sessionCreationPolicy(SessionCreationPolicy.STATELESS))
        .addFilterBefore(jwtFilter, UsernamePasswordAuthenticationFilter.class);

    return http.build();
  }
```

Рис.4.7 Демонстрация конфигурации фильтров SpringSecurity.

Работа с изображениями (логотипы стажировок) реализована через файловую систему сервера. При загрузке изображения оно сохраняется в локальную директорию, а в бд записывается путь до файла. В дальнейшем изображения раздаются как статические ресурсы (см. рис. 4.5), что позволяет эффективно обслуживать медиа-контент без дополнительной нагрузки на контроллеры.

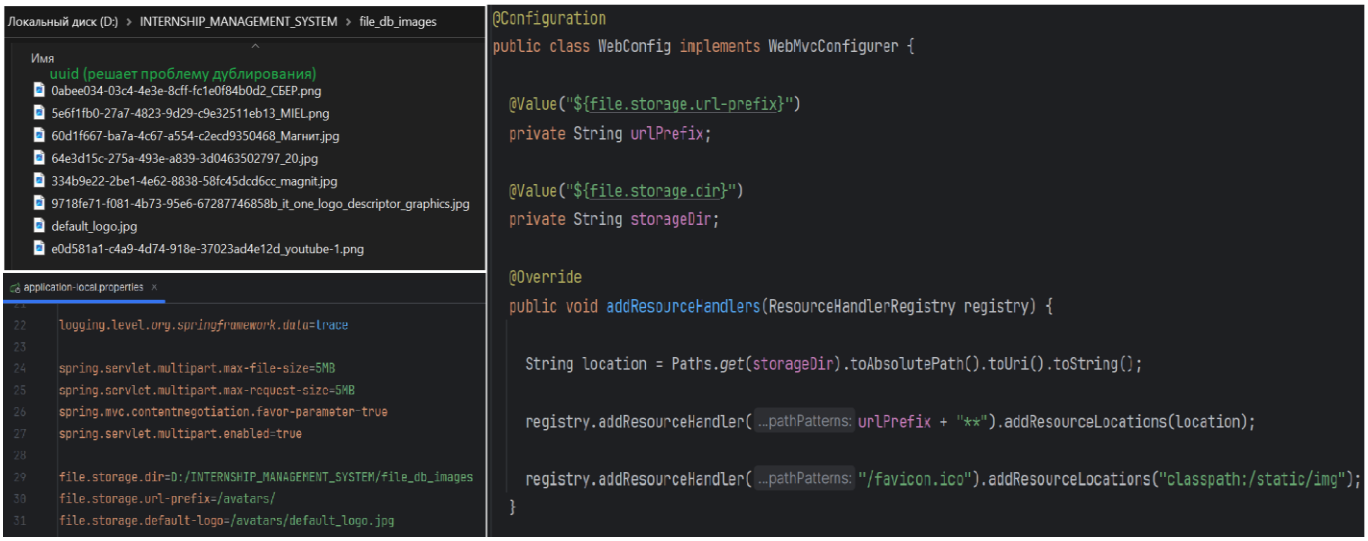


Рис.4.5 Демонстрация статической раздачи ресурсов через WebMvcConfiguer

Таким образом, backend приложения реализован с учётом современных требований к масштабируемости, безопасности и поддерживаемости. Использование Spring Boot, Spring Data JPA и JWT позволяет быстро разрабатывать и надёжно защищать веб-сервис.

5. РАЗРАБОТКА FRONTEND ЧАСТИ

В рамках курсовой работы была реализована клиентская часть веб-приложения с использованием стандартного стека веб-технологий: **HTML**, **CSS** и **JavaScript**. Разработка велась без использования фреймворков (таких как React или Vue), что позволило сконцентрироваться на чистой реализации логики и адаптации под структуру backend-сервиса, реализованного на Spring Boot.

Фронтенд-часть проекта размещена в директории resources/static и разделена на логически обособленные модули и страницы:

- main/ — главная страница приложения с отображением всех стажировок.
- auth/ — страница авторизации/регистрации пользователей.
- favorite/ — страница отображения избранных стажировок.
- utils/ — вспомогательные JavaScript-модули (работа с токенами, фильтрация, отображение модалок и пр.)
- img/ — изображения и иконки.

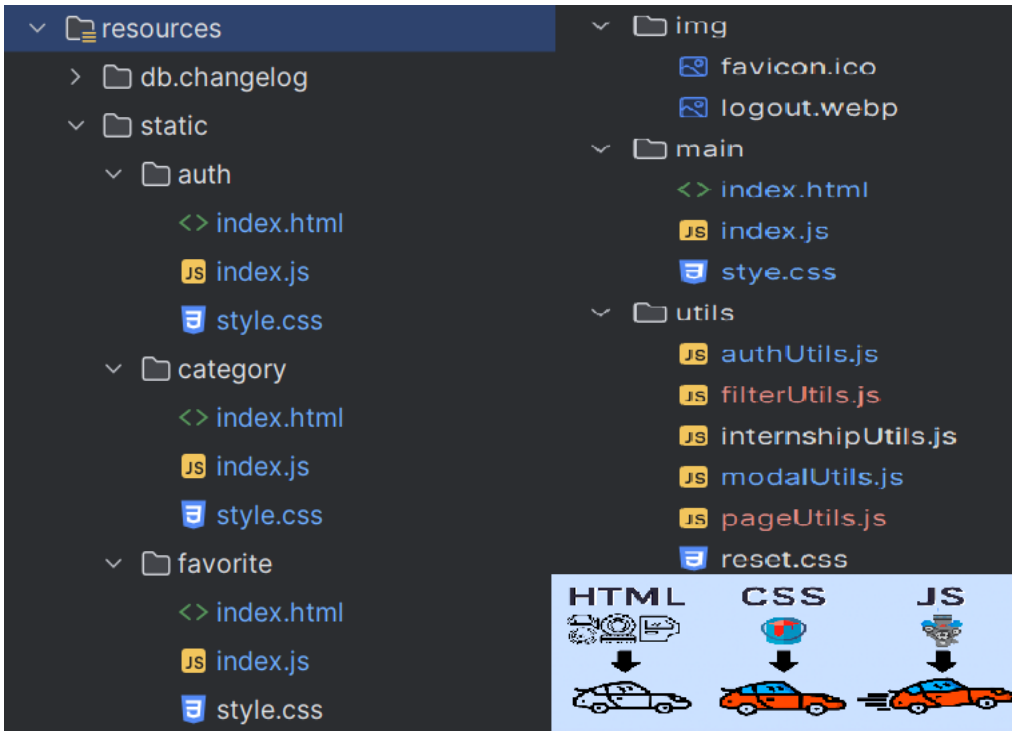


Рис.5.1. Структура фронтенд части проекта

Интерфейс реализованного веб-приложения:

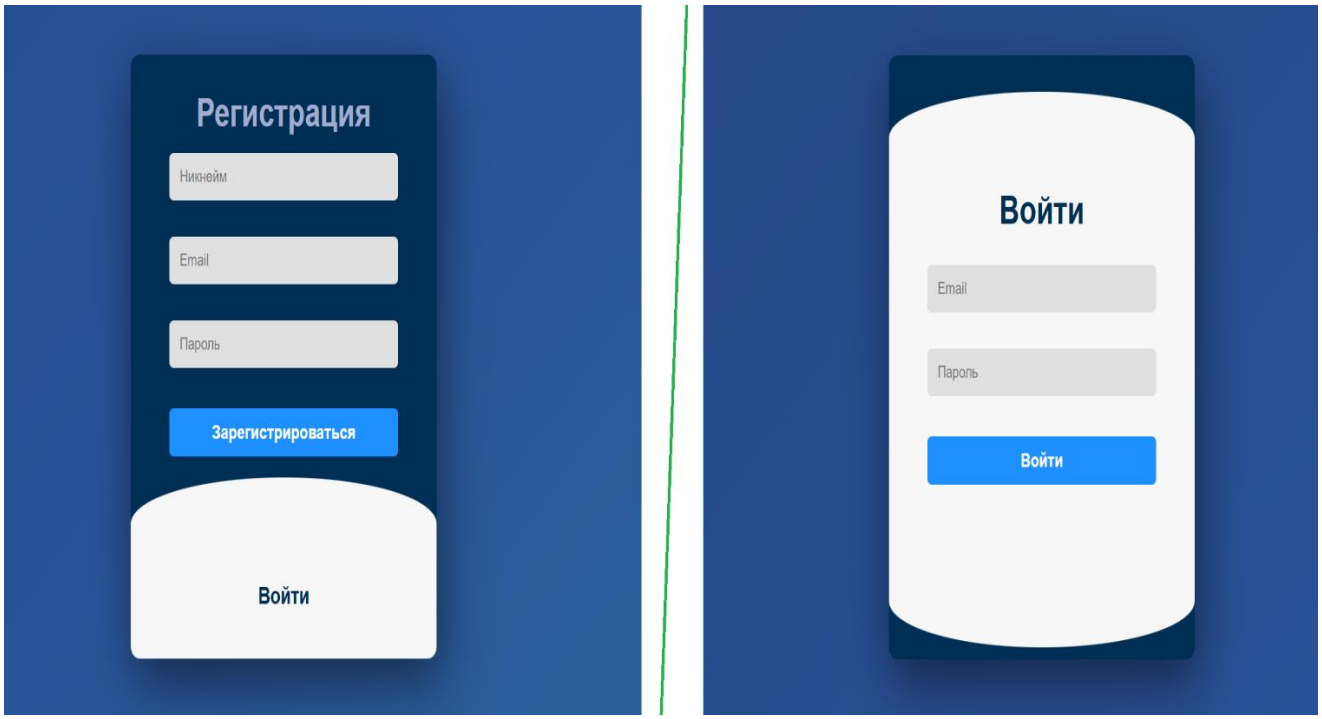


Рис.5.2 Страница регистрации/авторизации.

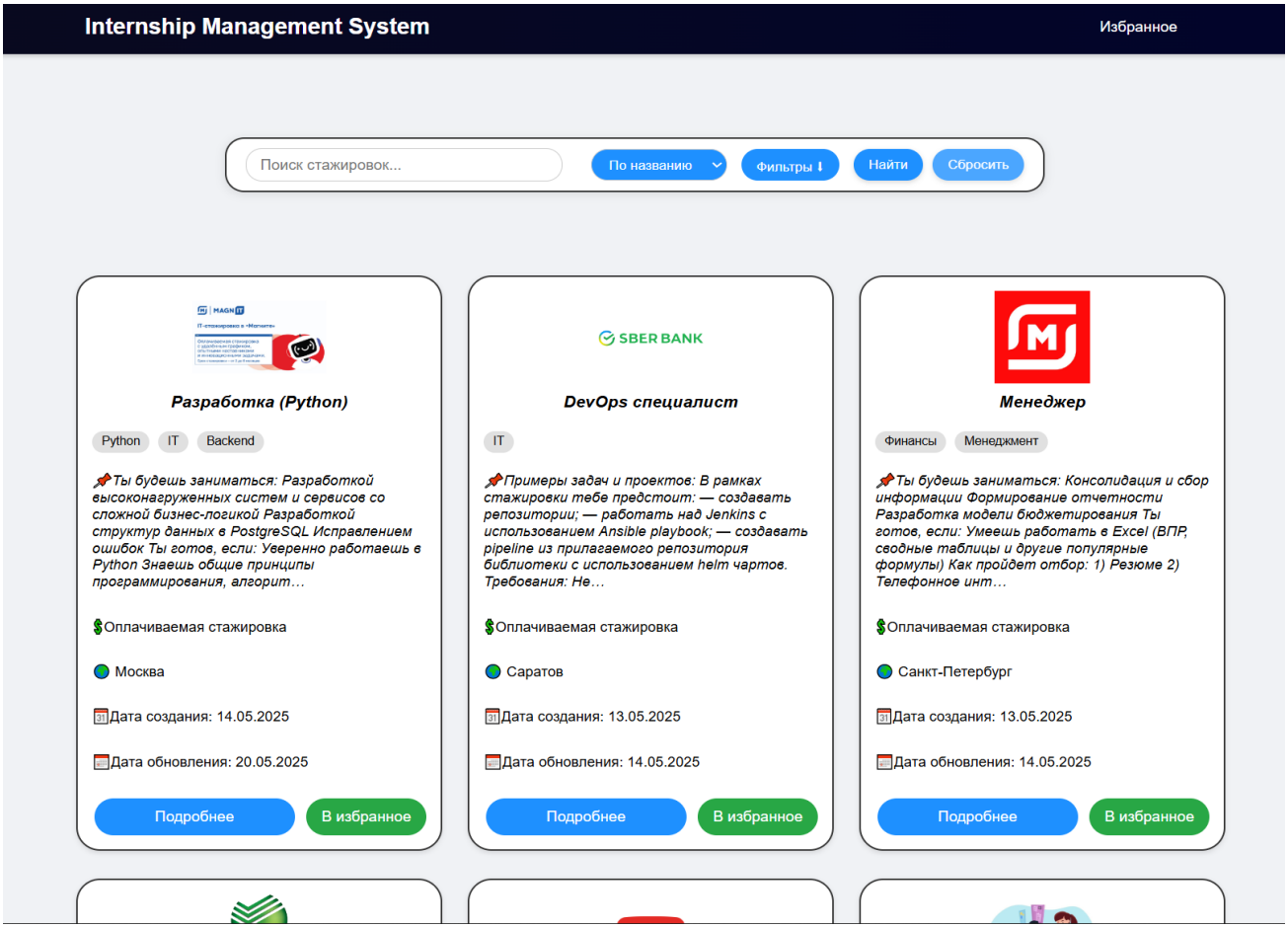


Рис.5.3 Главная страница (вид пользователя).

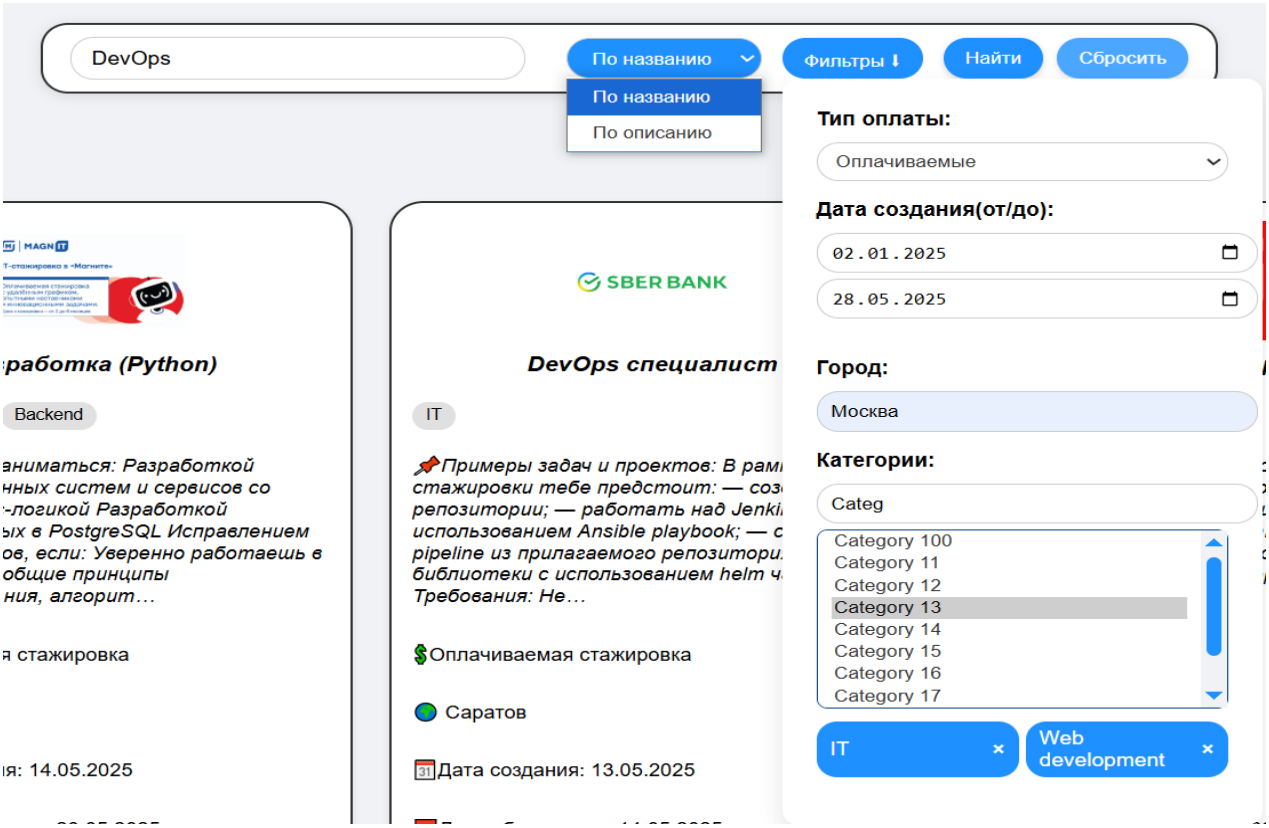


Рис.5.4 Поисковая панель.

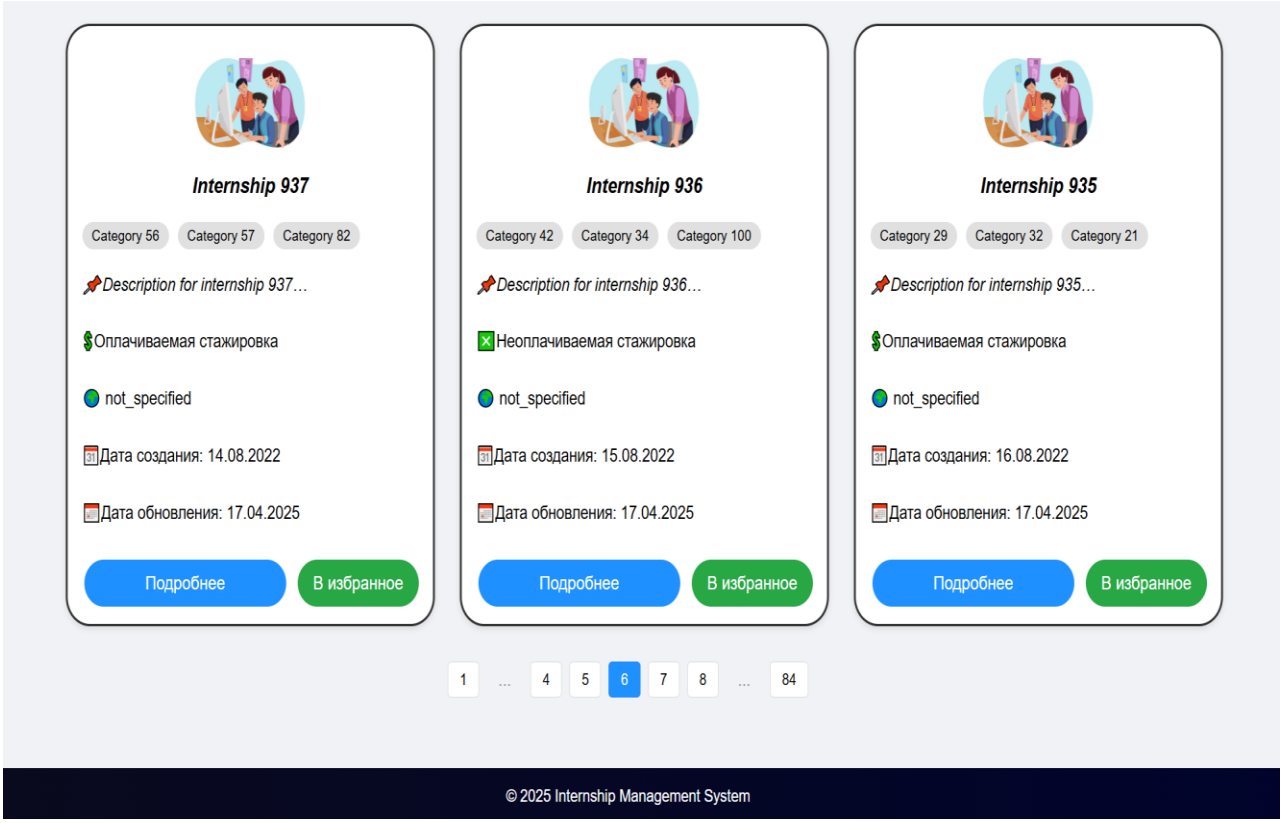


Рис.5.5 Пагинация в нижней части главной страницы

Рис.5.6. Форма подробнее для стажировки (вид пользователя).

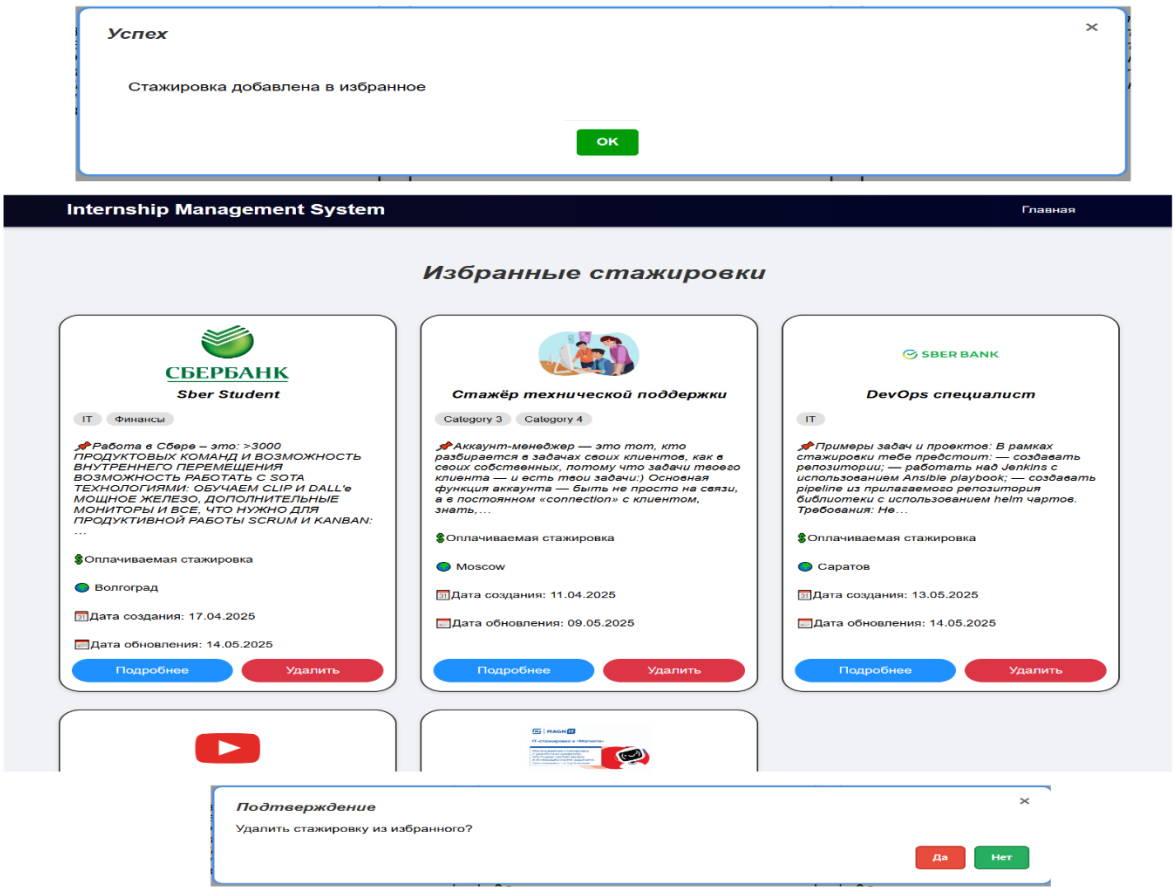


Рис.5.7. Страница избранного

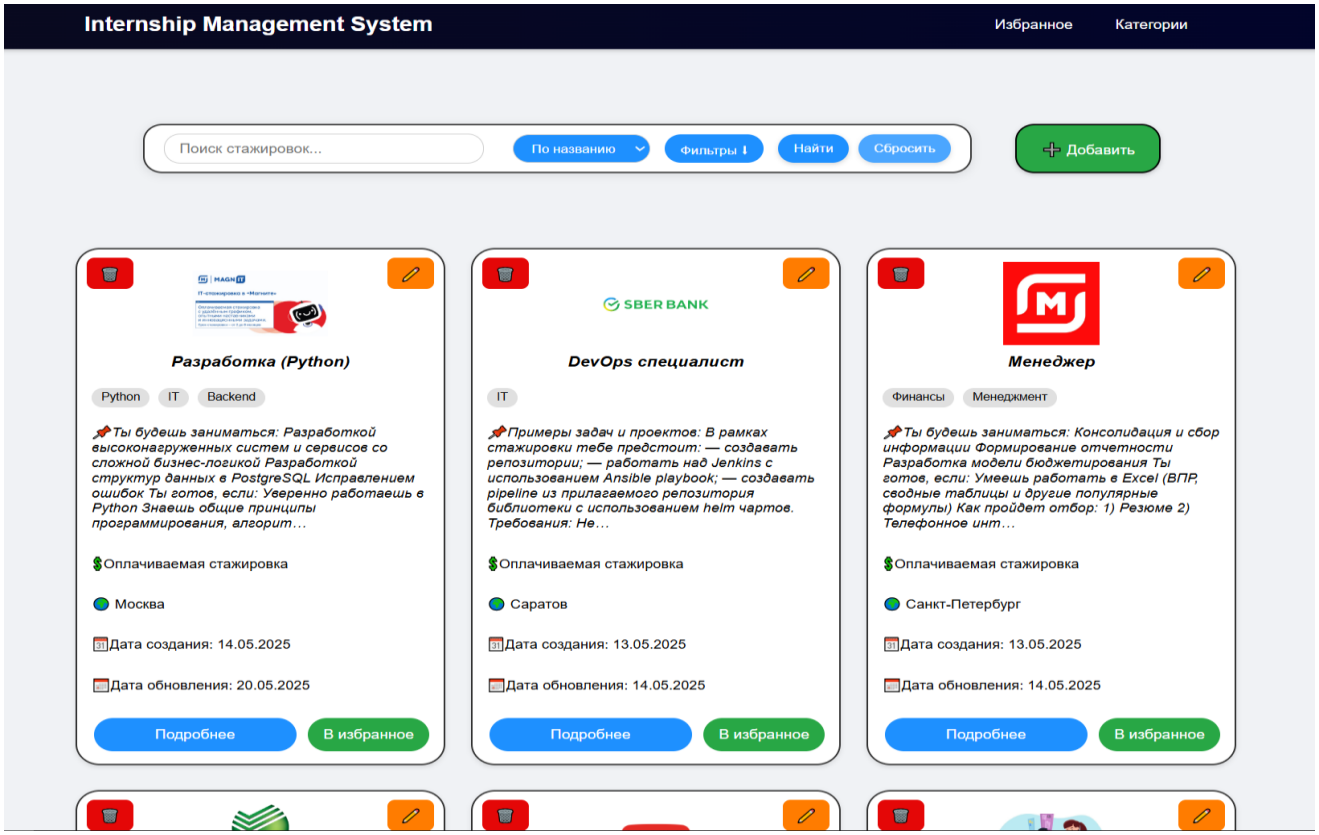


Рис.5.8. Главная страница (вид администратора).

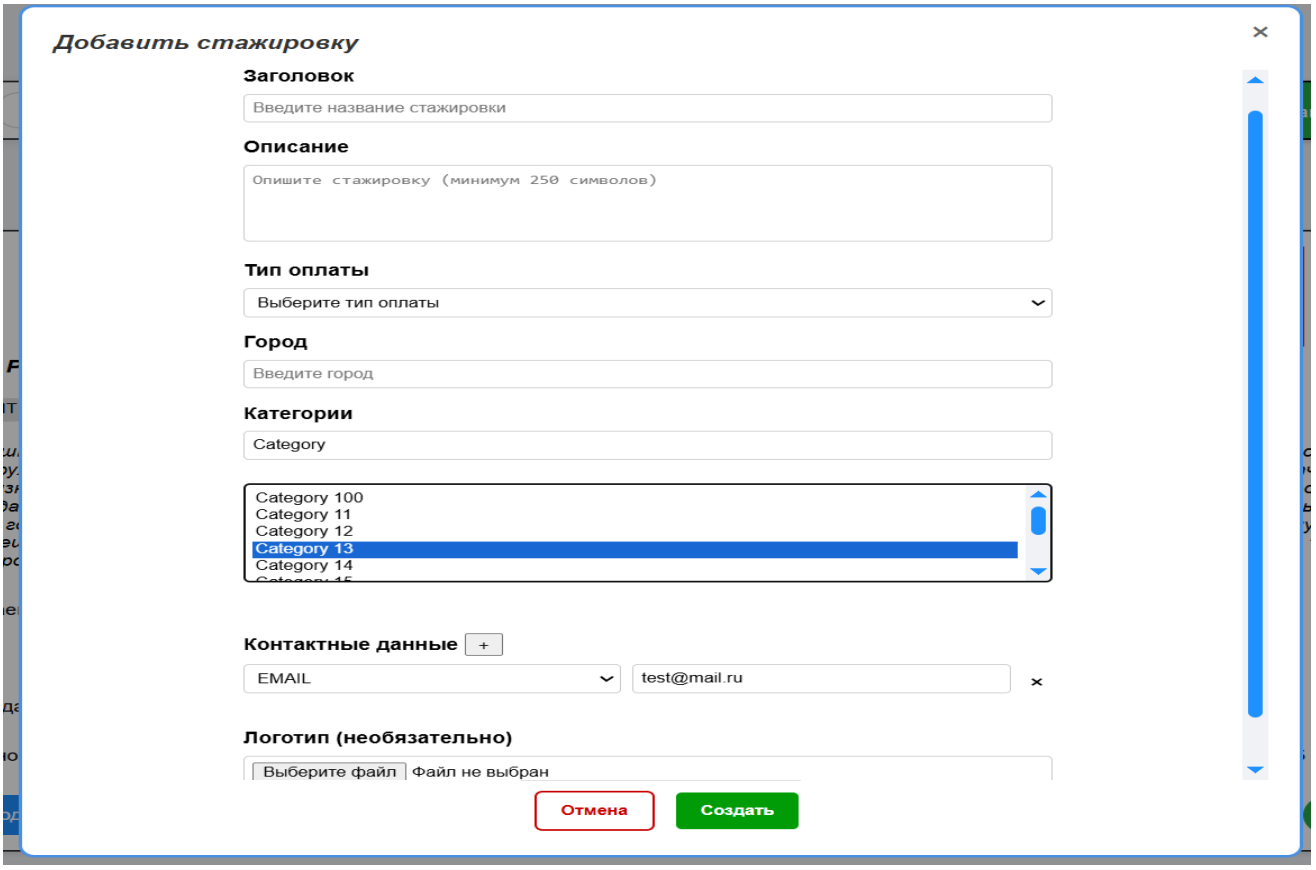


Рис.5.9 Форма создания стажировки (только у администратора).
Для редактирования – аналогично.

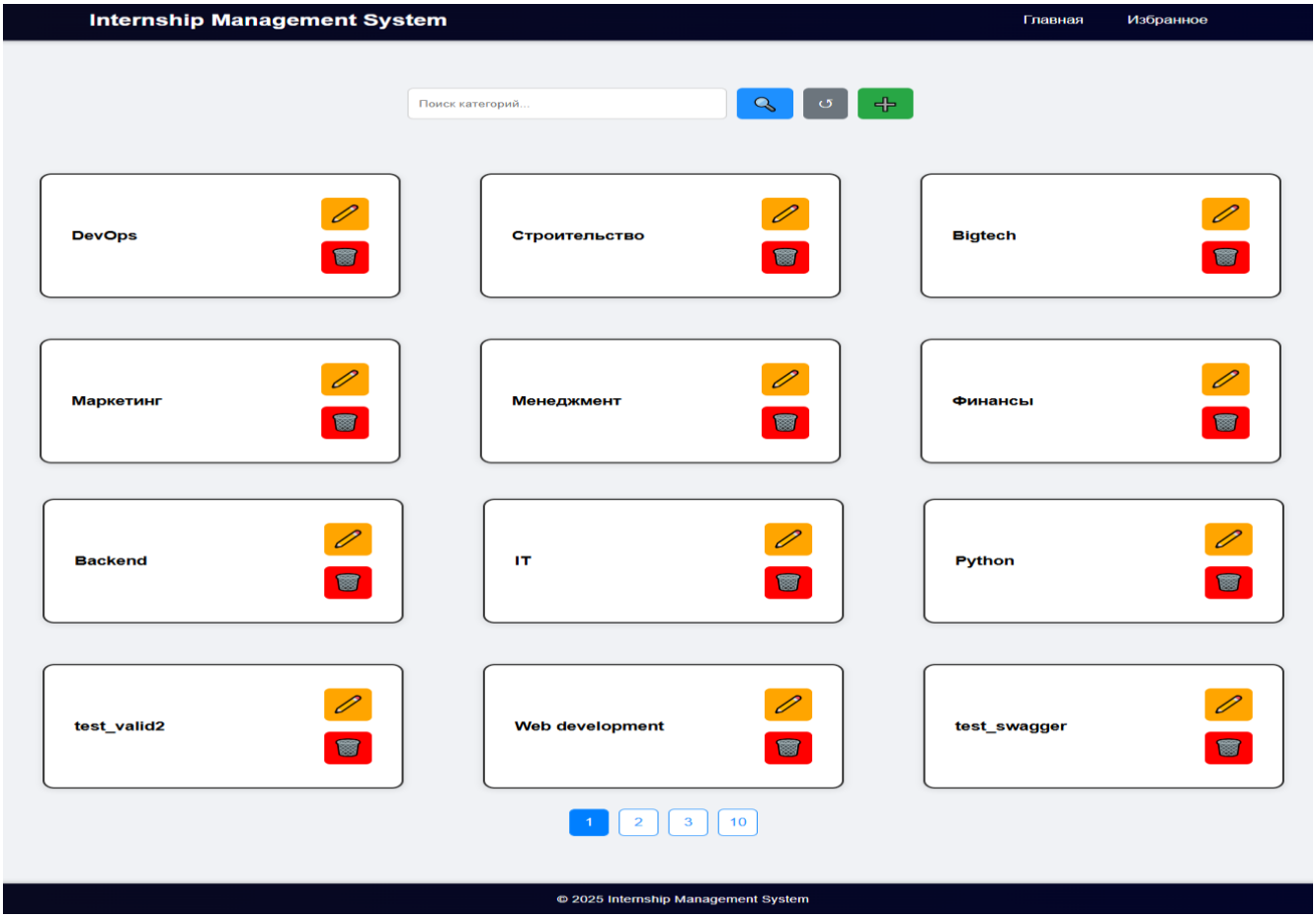


Рис.5.11. Страница для управления категориями (только для администратора).

Рис.5.12. Формы управления категориями (только для администратора).

ИТОГИ

В ходе выполнения курсовой работы было спроектировано и реализовано полноценное веб-приложение, обеспечивающее удобный интерфейс для взаимодействия пользователей со стажировками. Проект охватывает все ключевые этапы разработки: от проектирования базы данных и построения архитектуры серверной части до реализации клиентского интерфейса и настройки безопасности.

На начальном этапе была разработана структура базы данных, учитывающая требования к хранилищу информации о стажировках, пользователях, категориях и избранных записях. Были реализованы связи между таблицами с соблюдением принципов нормализации, что обеспечило целостность и расширяемость данных.

Серверная часть проекта построена на многослойной архитектуре, разделяющей ответственность между слоями: контроллеры, сервисы, репозитории и модели. Это сделало код удобным для сопровождения и расширения. Особое внимание было уделено вопросам безопасности: реализована регистрация, авторизация с использованием JWT, хеширование паролей, а также разграничение доступа к ресурсам. Также предусмотрена работа с файлами: загружаемые изображения сохраняются в файловой системе, а пользователи получают к ним доступ через статические URL.

Клиентская часть веб-приложения обеспечивает интуитивно понятный интерфейс и взаимодействует с сервером через API. Интерфейс реализован с учётом отзывчивости и пользовательского удобства. Все страницы были протестированы на корректность отображения и функционирования.

В результате была получена стабильная и функциональная система, готовая к использованию и дальнейшему развитию. Работа над проектом позволила закрепить практические навыки разработки полноценных веб-приложений.