МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«МИРЭА – Российский технологический университет»
РТУ МИРЭА

Институт информационных технологий (ИТ)
Кафедра инструментального и прикладного программного обеспечения (ИиППО)

КУРСОВАЯ РАБОТА

по дисциплине: Шаблоны программных платформ языка Джава
по профилю: Разработка программных продуктов и проектирование информационных систем
направления профессиональной подготовки: 09.03.04 «Программная инженерия»

Тема: Приложение «Компания интернет-рекрутмента»

Студент: Быченков Александр Константинович
Группа: ИКБО-32-21
Работа представлена к защите_____(дата)_____/Быченков А.К./
(подпись и ф.и.о. студента)

Руководитель: старший преподаватель кафедры ИиППО Рачков Андрей Владимирович

Работа допущена к защите _____(дата)_____/Рачков А.В./
(подпись и ф.и.о. рук-ля)

Оценка по итогам защиты: _____
_____ / _____ /
_____ / _____ /
(подписи, дата, ф.и.о., должность, звание, уч. степень двух преподавателей, принявших защиту)

М. РТУ МИРЭА. 2023 г

МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«МИРЭА – Российский технологический университет»
РТУ МИРЭА

Институт информационных технологий (ИТ)
Кафедра инструментального и прикладного программного обеспечения (ИиППО)

ЗАДАНИЕ
на выполнение курсовой работы

по дисциплине: Шаблоны программных платформ языка Джава
по профилю: Разработка программных продуктов и проектирование информационных систем
направления профессиональной подготовки: Программная инженерия (09.03.04)

Студент: Быченков Александр Константинович
Группа: ИКБО-32-21
Срок представления к защите: 11.05.2023 г.
Руководитель: старший преподаватель Рачков Андрей Владимирович

Тема: Приложение «Компания интернет-рекрутмента»

Исходные данные: индивидуальное задание на разработку; документация по Spring Framework и JEE, документация по языку Java (версия не ниже 8); инструменты и технологии: JDK (не ниже 8), создание Spring MVC web-приложений, RESTful web-сервисов, Spring ORM и Spring DAO, Gradle, gitHub, IntelliJIDEA. Нормативный документ: инструкция по организации и проведению курсового проектирования СМКО МИРЭА 7.5.1/04.И.05-18.

Перечень вопросов, подлежащих разработке, и обязательного графического материала:
1. Провести анализ предметной области и формирование основных требований к приложению, 2. Обосновать выбор средств ведения разработки. 3. Разработать приложение с использование фреймворка Spring, выбранной технологии и инструментария. 4.Провести тестирование приложения. 5. Оформить пояснительную записку по курсовой работе в соответствии с ГОСТ 7.32-2017. 6. Провести анализ текста на антиплагиат 7. Создать презентацию по выполненной курсовой работе.

Руководителем произведён инструктаж по технике безопасности, противопожарной технике и правилам внутреннего распорядка.

Зав. кафедрой ИиППО: _____/Р. Г. Болбаков/, «20» 02 2023 г.

Задание на КР выдал: _____/ А.В. Рачков/, «20» 02 2023 г.

Задание на КР получил: _____/А.К. Быченков/, «20» 02 2023 г.

АННОТАЦИЯ

Отчет 30 с., 30 рис., 8 источн.

СЕРВЕРНАЯ ЧАСТЬ, JAVA, SPRING, JDK 17, REST API, POSTGRESQL, KEYCLOAK, DOCKER, POSTMAN, ИНТЕРНЕТ-РЕКРУТМЕНТ.

Объект разработки – серверная часть приложения «Компания интернет-рекрутмента».

Цель работы – разработать серверную часть приложения «Компания интернет-рекрутмента».

В ходе работы был проведен краткий анализ предметной области, были выбраны необходимые технологии для разработки приложения, разработано и протестировано приложение «Компания интернет-рекрутмента».

Результатом работы является серверная часть приложения «Компания интернет-рекрутмента», написанная с использованием Spring Framework.

СОДЕРЖАНИЕ

ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И ОБОЗНАЧЕНИЙ

В настоящем отчете применяются следующие сокращения и обозначения:

- OAuth – Open Authorization (открытый протокол авторизации)
- SSO – Single Sign-On (технология единого входа)
- JWT – JSON Web Token (открытый стандарт для создания токенов доступа, основанный на формате JSON)
- REST – Representational State Transfer (передача репрезентативного состояния)
- CRUD – Create, Read, Update, Delete (базовые функции, используемые при работе с базами данных: создание, чтение, модификация, удаление)

ВВЕДЕНИЕ

Современные тенденции в бизнесе и технологическом развитии внесли значительные изменения в процессы подбора персонала для организаций. С ростом конкуренции на рынке труда, компании все чаще обращаются к интернет-рекрутменту как к важному инструменту для обеспечения качественного и эффективного подбора персонала. Этот процесс становится более сложным и требует использования современных информационных технологий для оптимизации, автоматизации и улучшения результатов.

Актуальность выбранной темы исследования заключается в необходимости разработки серверной части приложения "Компания интернет-рекрутмента", которое будет способствовать автоматизации процесса поиска персонала. Это позволит организациям значительно сократить затраты времени и ресурсов, повысить точность и эффективность отбора кандидатов, а также улучшить общее качество процесса рекрутмента.

Целью данной работы является разработка серверной части приложения (RESTful web-сервиса) «Компания интернет-рекрутмента» с использованием языка Java и Spring Framework.

Для достижения этой цели были поставлены следующие задачи:

1. Исследовать современные тенденции в области интернет-рекрутмента;
2. Разработать архитектуру серверной части приложения «Компания интернет-рекрутмента» с учетом требований к безопасности, масштабируемости, надежности;
3. Реализовать функциональность автоматизации процесса поиска персонала, включая создание резюме кандидатов, публикацию требуемых вакансий нанимателями, возможность отправки откликов на вакансии;
4. Провести тестирование и оптимизацию разработанной серверной части приложения для обеспечения надежной и эффективной работы.

Структура данной курсовой работы организована следующим образом. В первом разделе проведен анализ предметной области, представлен обзор существующих решений в области интернет-рекрутмента, сформированы основные требования. Во втором разделе приведены обоснования выбранных технологий разработки. Третий раздел посвящен разработке веб-приложения. В четвертом разделе были описаны результаты проведенного тестирования. Завершающий раздел содержит заключение и выводы по проведенной работе.

Объектом исследования данной курсовой работы является процесс интернет-рекрутмента, а предметом – разработка серверной части приложения «Компания интернет-рекрутмента» на базе Spring Framework.

1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ И ФОРМИРОВАНИЕ ОСНОВНЫХ ТРЕБОВАНИЙ К ПРИЛОЖЕНИЮ

Для анализа предметной области было проведено сравнение трех популярных российских сервисов по поиску работы: HeadHunter (hh.ru), Работа.ру (rabota.ru), SuperJob (superjob.ru). Были выделены общие черты, функционал, который присутствует во всех сервисах.

Клиентами сервиса являются либо наниматели (работодатели), либо соискатели (потенциальные сотрудники). На каждом сервисе им необходимо авторизоваться, используя телефон, электронную почту или сторонние сервисы (VK ID, Yandex ID или другие) для аутентификации по протоколу открытой авторизации OAuth 2, рис 1.1:

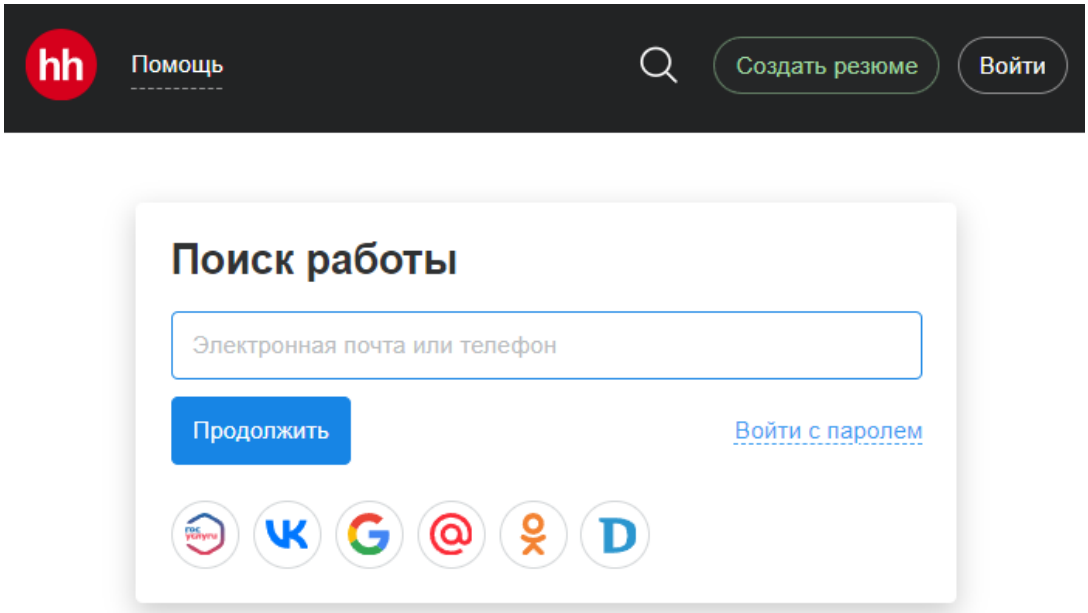


Рисунок 1.1 – Авторизация на сервисе HeadHunter

Затем нанимателям необходимо зарегистрировать свою компанию и разместить вакансии в необходимых регионах. Соискателям же необходимо создать резюме, указав желаемую позицию (должность), заработную плату, имеющееся профессиональные навыки, образование, предыдущие места работы. Для каждого соискателя сервисы предоставляют самые релевантные вакансии, рис. 1.2:

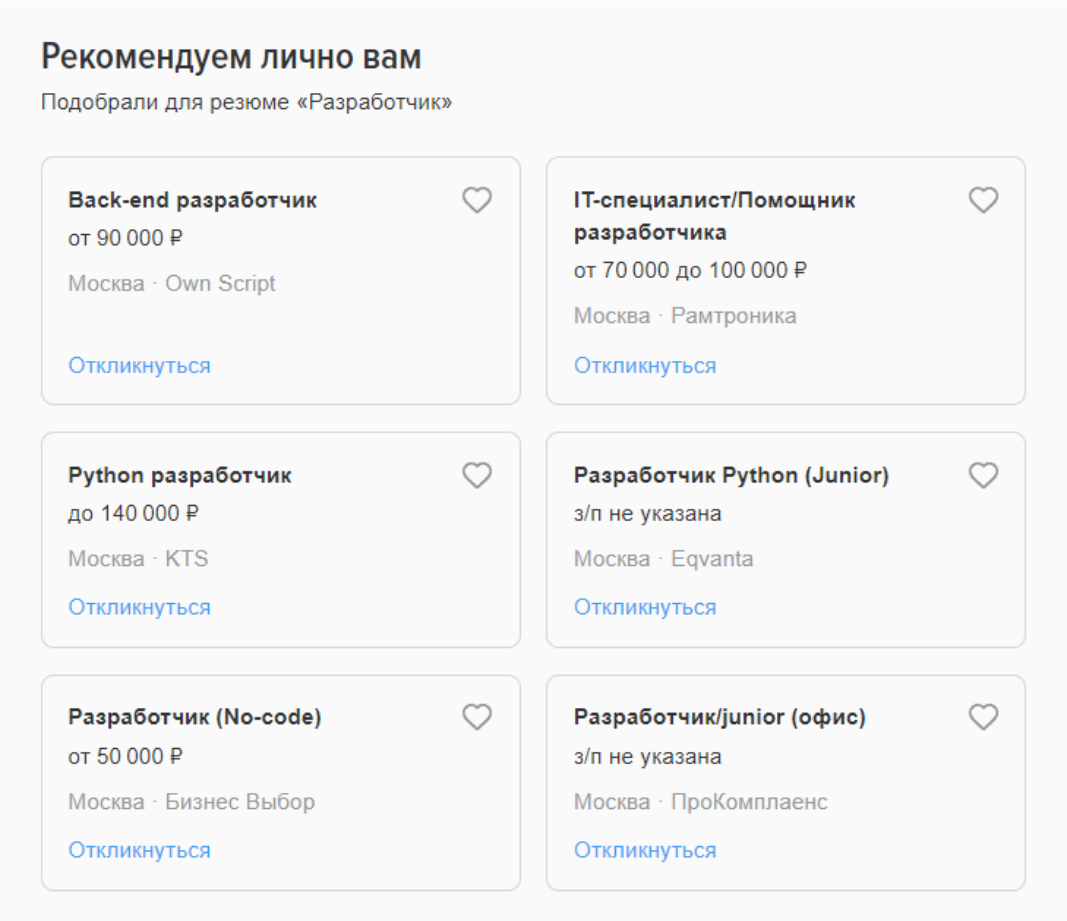


Рисунок 1.2 – Рекомендуемые вакансии

На главной странице сервисов для соискателей отображается список вакансий в их регионах. Вакансии можно отфильтровать по многим параметрам, включая: ключевые слова, уровень дохода, требуемый опыт, график работы, рис. 1.3:

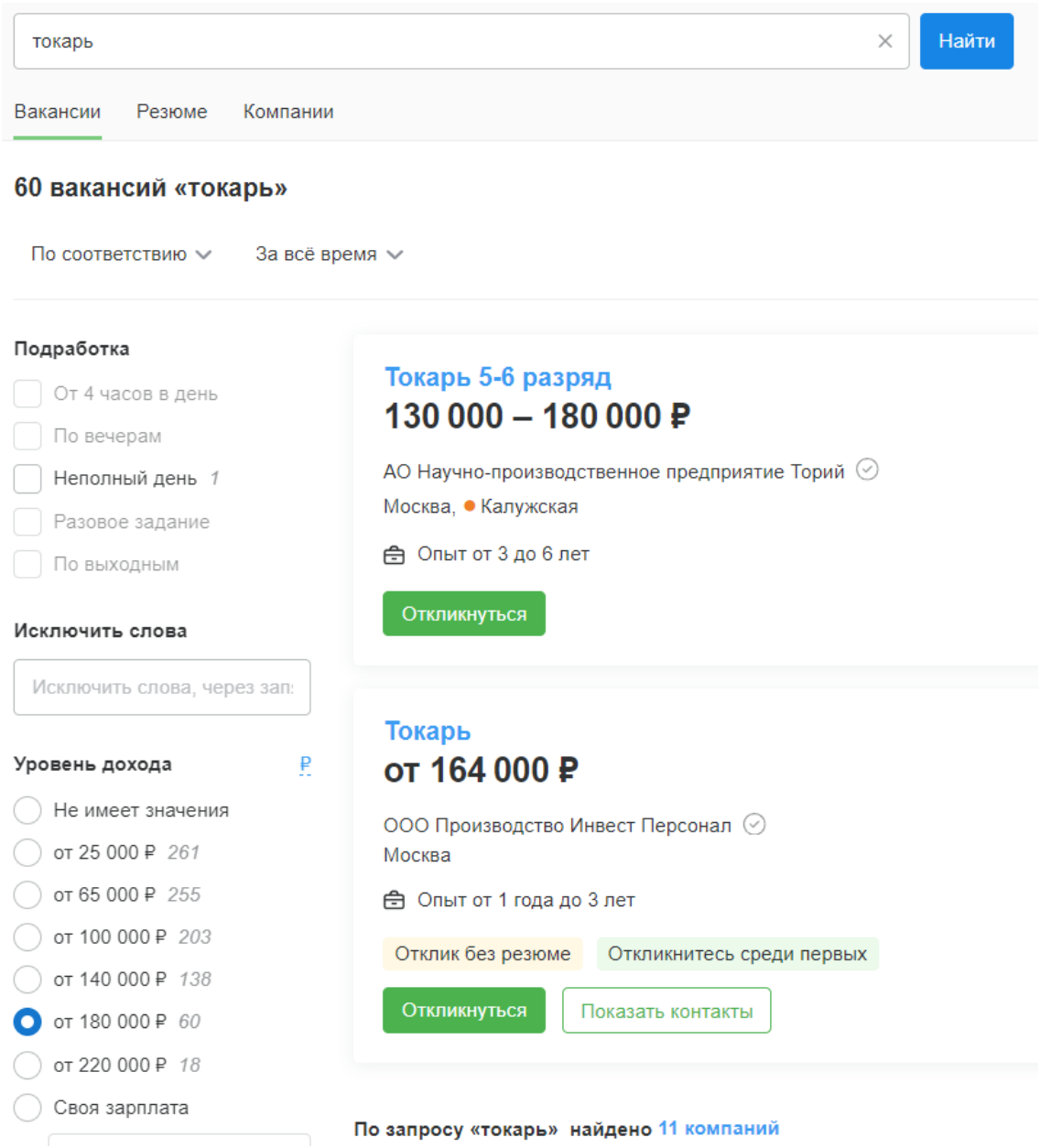


Рисунок 1.3 – Отфильтрованные вакансии

Для работодателей же сервисы предоставляют резюме сотрудников, которые находятся в поиске работы, рис. 1.4:

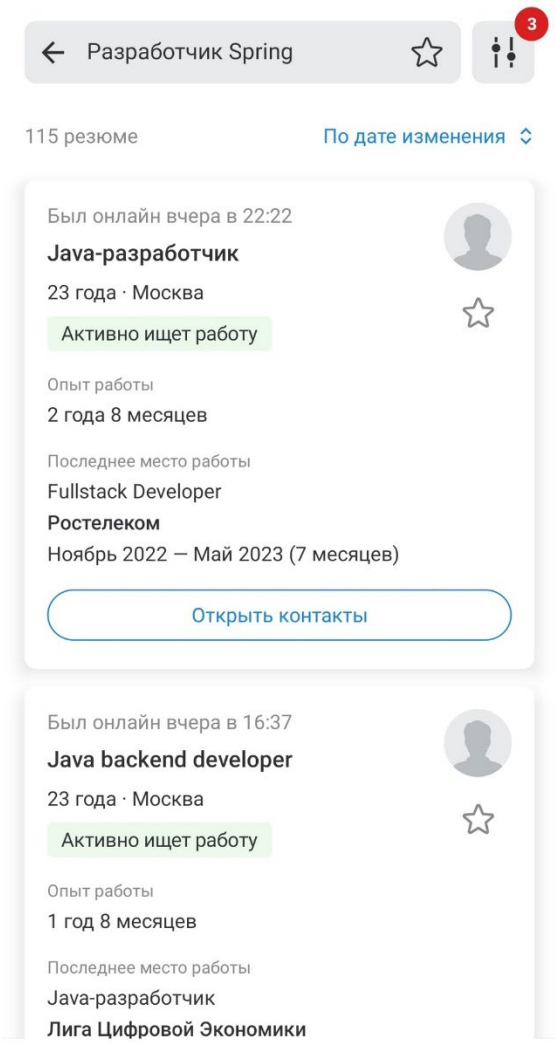


Рисунок 1.4 – Поиск потенциальных сотрудников

Кандидатов на требуемую должность можно найти как самостоятельно написав им, используя поиск по выложенным резюме, так и кандидаты могут откликнуться на вакансии, приложив резюме и написав сопроводительное письмо, рис. 1.5:

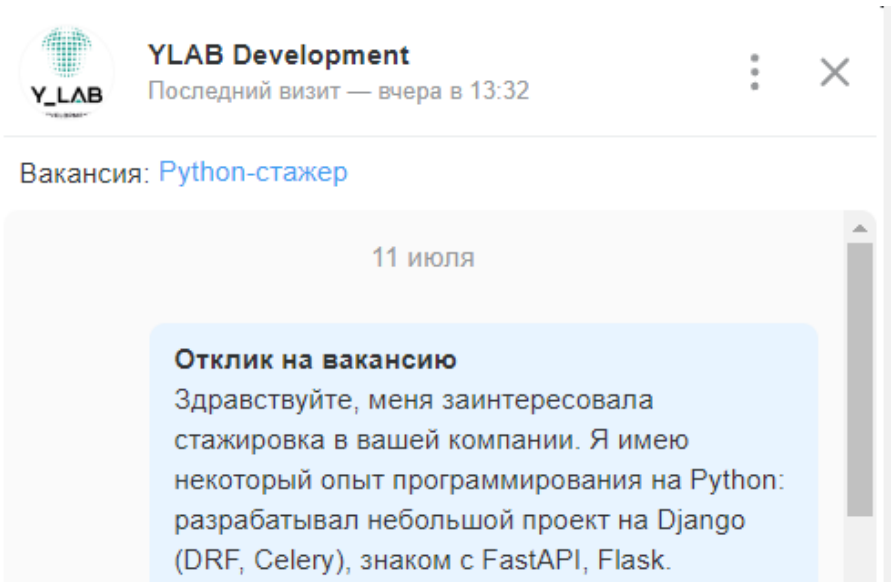


Рисунок 1.5 – Сопроводительное письмо для отклика на вакансию

Изучив вышеперечисленные сервисы, были сформированы основные требования к разрабатываемому приложению:

1. Надежная аутентификация пользователей в системе и их авторизация, используя различные роли: наниматель, соискатель и сервисные роли (модератор, администратор). Пользователи в зависимости от их роли должны иметь доступ только к определенным функциям системы;
2. Реализация сущностей системы: пользователь, регион, компания, вакансия, резюме, отклик;
3. Управление каждой сущностью и их фильтрация по различным параметрам.

2 ВЫБОР СРЕДСТВ ВЕДЕНИЯ РАЗРАБОТКИ

2.1 Прикладное программное обеспечение, необходимое для разработки и функционирования веб-приложения

Для написания программного кода использовалась среда разработки VS Code, для функционирования приложения необходима программная платформа контейнеризации приложений Docker.

2.2 Языки и технологии реализации веб-приложения

Используется язык программирования Java 17 версии в связке с фреймворком Spring. Используется система сборки Gradle.

Данные приложения хранятся внутри СУБД PostgreSQL, запущенной внутри Docker-контейнера. СУБД PostgreSQL широко используется в настоящее время и является промышленным стандартом хранения данных. Взаимодействие с базой данных реализовано, используя модуль Spring ORM.

Для аутентификации пользователей используется KeyCloak – готовое SSO (Single Sign-On) решение, имеющее встроенную реализацию протокола OpenID Connect. Интегрировано с модулем Spring Security.

3 РАЗРАБОТКА ВЕБ-ПРИЛОЖЕНИЯ

3.1 Конфигурация окружения

Для функционирования необходимо подключение к базе данных PostgreSQL, серверу KeyCloak SSO. Приложение и окружение развертывается в Docker-контейнерах. Для этого был написан файл Docker Compose:

Листинг 1 – compose.yaml

```
services:
  postgres:
    build: containers/postgres
    container_name: work-in-postgres
    restart: always
    ports:
      - 5432:5432
    environment:
      POSTGRES_USER: ${POSTGRES_USER}
      POSTGRES_PASSWORD: ${POSTGRES_PASSWORD}
      POSTGRES_DB: workin
    volumes:
      - postgres-data:/var/lib/postgresql/data
  keycloak:
    build: containers/keycloak
    container_name: work-in-keycloak
    restart: always
    command: start-dev --db postgres
    ports:
      - 8443:8443
    environment:
      KC_DB_URL: jdbc:postgresql://postgres:5432/keycloak
      KC_DB_USERNAME: ${POSTGRES_USER}
      KC_DB_PASSWORD: ${POSTGRES_PASSWORD}
      KEYCLOAK_ADMIN: ${KEYCLOAK_ADMIN}
      KEYCLOAK_ADMIN_PASSWORD: ${KEYCLOAK_ADMIN_PASSWORD}
      KC_HTTPS_CERTIFICATE_FILE: /opt/keycloak/conf/certs/tls.crt
      KC_HTTPS_CERTIFICATE_KEY_FILE: /opt/keycloak/conf/certs/tls.key
    healthcheck:
      test:
        ["CMD", "curl", "--head", "fsS",
"https://keycloak:8443/health/ready"]
      interval: 5s
      timeout: 2s
      retries: 15
    depends_on:
      - postgres
  app:
    image: descenty/work_in_spring
```

```
    pull_policy: always
    container_name: work-in-app
    restart: always
    ports:
      - 8001:8001
    environment:
      POSTGRES_HOST: postgres
      POSTGRES_PORT: 5432
      POSTGRES_DB: workin
      KC_PORT: 8443
    env_file:
      - .env
    depends_on:
      - postgres
      - keycloak
volumes:
  postgres-data:
```

3.2 Логическая структура веб-приложения

Все Java-классы разбиты по модулям и слоям, следуя по архитектурному паттерну HMVC (Hierarchical Model–View–Controller), который позволяет решить некоторые проблемы масштабируемости приложений, имеющих классическую MVC-архитектуру. Благодаря разделению по модулям, в проекте легко найти Java-файлы для любого модуля приложения, рисунок 3.1:

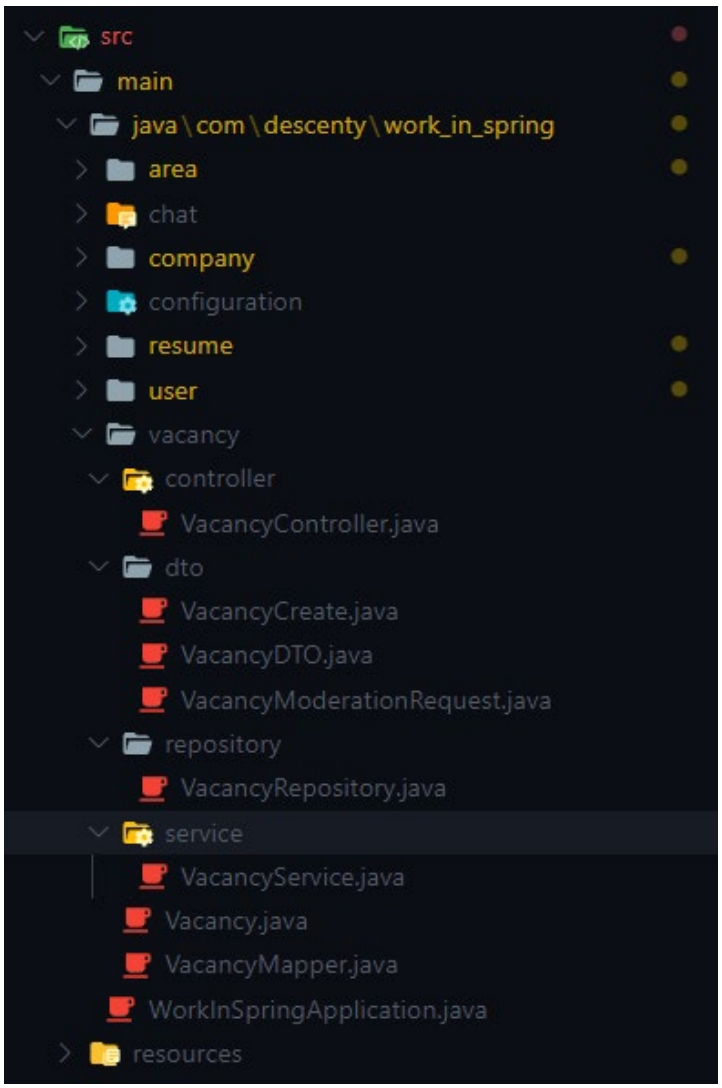


Рисунок 3.1 – Логическая структура веб-приложения

3.3 Реализация авторизации пользователей

Для авторизации пользователей используется готовое SSO (Single Sign-On) решение KeyCloak. Keycloak — это полноценная реализация Identity Provider для OpenID Connect (а значит, Authorization Server для OAuth2). Для работы приложения был создан отдельный Realm с названием workin, в него добавлен клиент под названием backend. Этот клиент будет использоваться приложением для доступа к пользователям, их аутентификации и авторизации, получения ролей:

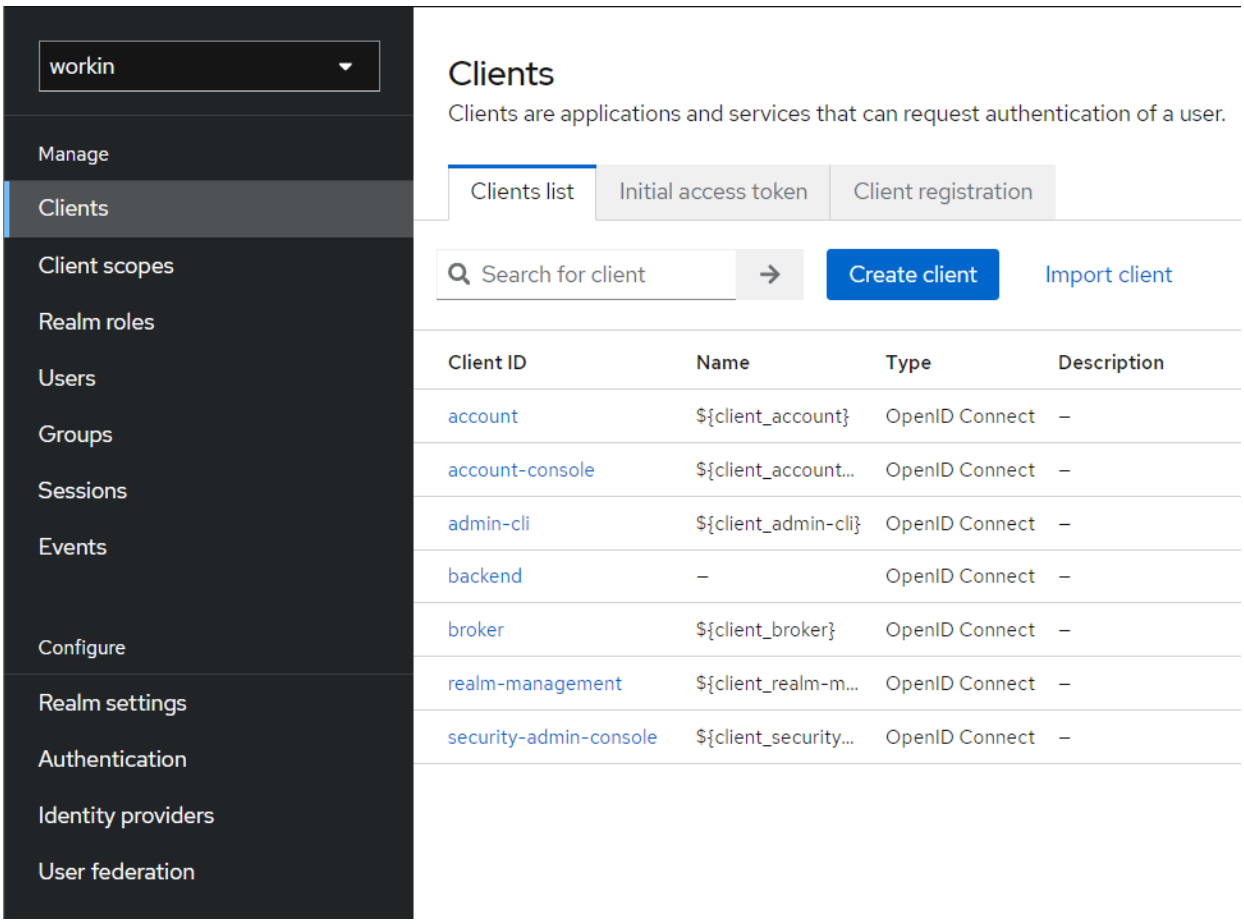


Рисунок 3.2 – Экран клиентов KeyCloak

Для работы с KeyCloak в конфигурации приложения application.yml написаны URL для KeyCloak Resource Server, ID и имя созданного клиента backend, и ключ доступа для него:

```yaml
security:
  oauth2:
    client:
      registration:
        keycloak:
          client-id: ${CLIENT_ID}
          client-name: ${CLIENT_ID}
          authorization-grant-type: authorization_code
          client-secret: ${CLIENT_SECRET}
          scope: openid
          use-resource-role-mappings: false
      provider:
        keycloak:
          issuer-uri: ${KC_URL}/realms/${KC_REALM}
          user-name-attribute: preferred_username
    resourceserver:
      jwt:
        issuer-uri: ${KC_URL}/realms/${KC_REALM}
        jwk-set-uri: ${spring.security.oauth2.resourceserver.jwt.issuer-uri}/protocol/openid-connect/certs
```

Рисунок 3.3 – Конфигурация Spring Security в application.yml

Был также написан Java-класс для конфигурации Spring Security, рис. 3.4:

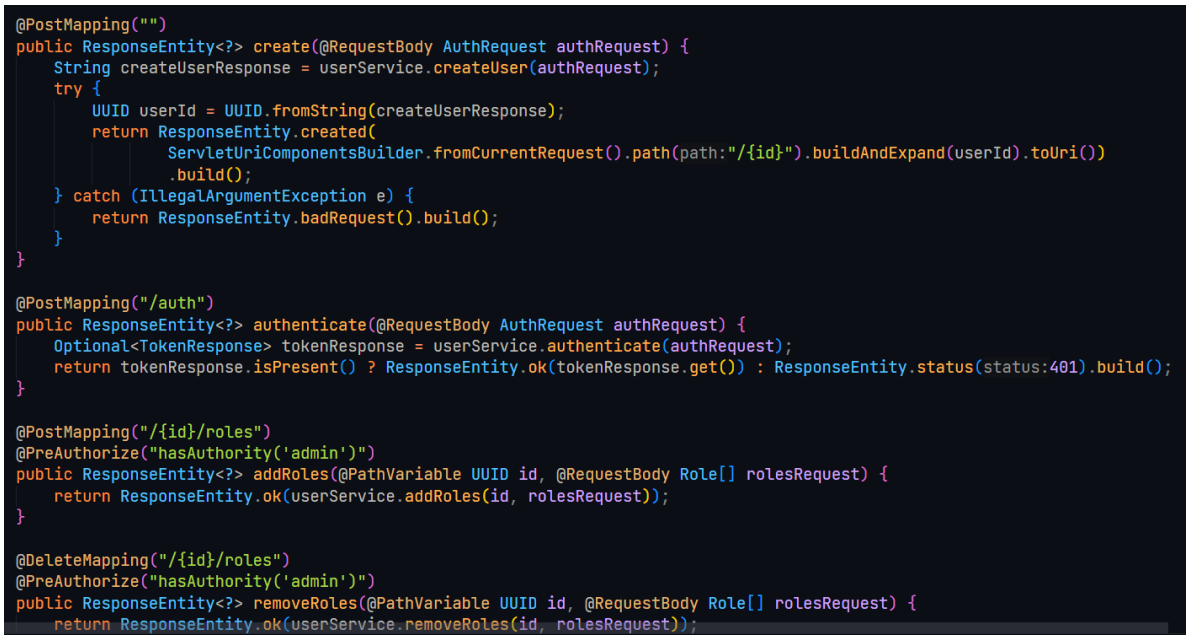
```java
@Configuration
@ConditionalOnProperty(name = "keycloak.enabled", havingValue = "true", matchIfMissing = true)
@EnableMethodSecurity
@EnableWebSecurity
public class SecurityConfiguration {

        @Bean
        SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
                http.cors().configurationSource(request -> {
                        var cors = new org.springframework.web.cors.CorsConfiguration();
                        cors.setAllowedOrigins(List.of("http://localhost:3000"));
                        return cors;
                });
                http.csrf().disable().authorizeHttpRequests().anyRequest().permitAll().and().oauth2ResourceServer()
                                .jwt().jwtAuthenticationConverter(new KeycloakJwtAuthenticationConverter());
                http.oauth2Client().and().oauth2Login().tokenEndpoint().and().userInfoEndpoint();
                http.sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS);
                return http.build();
        }
}
```

Рисунок 3.4 – Java-класс конфигурации Spring Security

Аутентификация и авторизация пользователей производится по JWT-токенам, которые выдаются KeyCloak. В токены также записываются роли пользователя в системы (модератор, администратор).

Для взаимодействия с KeyCloak написан UserController, отвечающий за регистрацию, аутентификацию. Через него администратор также может добавить, удалить роли или удалить пользователя, рисунок 3.5:



```java
@PostMapping("")
public ResponseEntity<?> create(@RequestBody AuthRequest authRequest) {
    String createUserResponse = userService.createUser(authRequest);
    try {
        UUID userId = UUID.fromString(createUserResponse);
        return ResponseEntity.created(
                ServletUriComponentsBuilder.fromCurrentRequest().path("/{id}").buildAndExpand(userId).toUri())
                .build();
    } catch (IllegalArgumentException e) {
        return ResponseEntity.badRequest().build();
    }
}

@PostMapping("/auth")
public ResponseEntity<?> authenticate(@RequestBody AuthRequest authRequest) {
    Optional<TokenResponse> tokenResponse = userService.authenticate(authRequest);
    return tokenResponse.isPresent() ? ResponseEntity.ok(tokenResponse.get()) : ResponseEntity.status(401).build();
}

@PostMapping("/{id}/roles")
@PreAuthorize("hasAuthority('admin')")
public ResponseEntity<?> addRoles(@PathVariable UUID id, @RequestBody Role[] rolesRequest) {
    return ResponseEntity.ok(userService.addRoles(id, rolesRequest));
}

@DeleteMapping("/{id}/roles")
@PreAuthorize("hasAuthority('admin')")
public ResponseEntity<?> removeRoles(@PathVariable UUID id, @RequestBody Role[] rolesRequest) {
    return ResponseEntity.ok(userService.removeRoles(id, rolesRequest));
```

Рисунок 3.5 – Листинг UserController

Этот контроллер является обёрткой над KeyCloak. В сервисном слое лишь отправляются запросы на KeyCloak-сервер, рисунок 3.6:

```
private TokenResponse clientAuthentication() {
    WebClient client = WebClient.create();
    return client.post().uri(serverURL + "/realms/" + realm + "/protocol/openid-connect/token")
            .contentType(MediaType.APPLICATION_FORM_URLENCODED)
            .bodyValue("grant_type=client_credentials&client_id=" + clientID + "&client_secret=" + clientSecret)
            .retrieve().bodyToMono(elementClass:TokenResponse.class).block();

}

public Optional<TokenResponse> authenticate(AuthRequest authRequest) {
    WebClient client = WebClient.create();
    try {
        String bodyValue = authRequest.getRefreshToken() == null
                ? "grant_type=password&username=" + authRequest.getUsername() + "&password=" + authRequest.getPassword()
                        + "&client_id=" + clientID + "&client_secret=" + clientSecret
                : "grant_type=refresh_token&refresh_token=" + authRequest.getRefreshToken() + "&client_id=" + clientID
                        + "&client_secret=" + clientSecret;
        return Optional.of(client.post().uri(serverURL + "/realms/" + realm + "/protocol/openid-connect/token")
                .contentType(MediaType.APPLICATION_FORM_URLENCODED).bodyValue(bodyValue).retrieve()
                .bodyToMono(elementClass:TokenResponse.class).block());
    } catch (WebClientResponseException e) {
        return Optional.empty();
    }
}
```

Рисунок 3.6 – Листинг UserService

Интеграция с KeyCloak настроена, JWT-токены могут передаваться в Authorization заголовке запроса. Контроллеры могут предоставлять доступ к эндпоинтам по различным условиям. К примеру, AreaController ограничивает возможность создания новых регионов для обычных пользователей, только для администраторов, рисунок 3.7:

```
@GetMapping("{id}/main")
public ResponseEntity<List<AreaDTO>> getAllMain(@PathVariable Long id) {
    return areaService.getAllMain(id).map(ResponseEntity::ok).orElse(ResponseEntity.notFound().build());
}

@GetMapping("/{id}")
public ResponseEntity<AreaDTO> getById(@PathVariable Long id) {
    return areaService.getById(id).map(ResponseEntity::ok).orElse(ResponseEntity.notFound().build());
}

@PostMapping("")
@PreAuthorize("hasAuthority('admin')")
public ResponseEntity<?> create(@Valid @RequestBody AreaCreate areaCreate) {
    Optional<Long> areaId = areaService.create(areaCreate);
    return areaId.isPresent()
            ? ResponseEntity.created(ServletUriComponentsBuilder.fromCurrentRequest().path(path:"/{id}")
                    .buildAndExpand(areaId.get()).toUri()).build()
            : ResponseEntity.notFound().build();
}

@PatchMapping("/{id}")
@PreAuthorize("hasAuthority('admin')")
public ResponseEntity<AreaDTO> partialUpdate(@PathVariable Long id, @Valid @RequestBody AreaCreate areaCreate) {
    return areaService.partialUpdate(id, areaCreate).map(ResponseEntity::ok)
            .orElse(ResponseEntity.notFound().build());
}

@PutMapping("/{id}")
@PreAuthorize("hasAuthority('admin')")
```

Рисунок 3.7 – Листинг AreaController

3.4 Реализация бизнес-логики

Сущности базы данных разработаны, используя Spring Data JPA. Все сущности связаны друг с другом внешними ключами и для них настроено каскадное удаление. Пример реализации сущности региона, рисунок 3.8:

```java
@Entity
@Getter
@Setter
public class Area {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    @Column(nullable = false)
    private String name;

    @ManyToOne(fetch = FetchType.LAZY, optional = true)
    @JoinColumn(name = "parent_id", updatable = false, insertable = false)
    private Area parent;

    @Column(name = "parent_id", nullable = true)
    private Long parentId;

    @OneToMany(mappedBy = "parent", cascade = CascadeType.REMOVE, orphanRemoval = true)
    private List<Area> children;

    @OneToMany(mappedBy = "area", cascade = CascadeType.REMOVE, orphanRemoval = true)
    private List<Company> companies;
}
```

Рисунок 3.8 – Листинг Area

Взаимодействие с базой данных реализовано, используя Spring Data JpaRepository. Репозитории автоматически для каждой сущности создают CRUD-методы. Дополнительные методы для запросов к базе данных можно создать, написав соответствующие спецификации методов, используя предопределенные ключевые слова. Например, на рисунке 3.9 показаны пользовательские методы к сущности компании:

```java
@Repository
public interface CompanyRepository extends JpaRepository<Company, Long> {
    List<Company> findAllByAreaId(Long areaId);

    Optional<Company> findByAreaIdAndId(Long areaId, Long id);

    boolean existsByIdAndEmployersIdsContaining(Long id, UUID employerId);

    boolean existsByIdAndCreatorId(Long id, UUID creatorId);

    Long deleteByAreaIdAndId(Long areaId, Long id);
}
```

Рисунок 3.9 – Листинг CompanyRepository

Для работы с репозиториями для сущности региона написан сервисный слой, где вызываются методы репозитория, рисунок 3.9:

```
@Transactional
public Optional<Long> create(AreaCreate areaCreate) {
    if (areaCreate.getParentId() != null && !areaRepository.existsById(areaCreate.getParentId()))
        return Optional.empty();
    return Optional.of(areaMapper.toEntity(areaCreate)).map(areaRepository::save).map(area -> area.getId());
}

@Transactional
public Optional<AreaDTO> partialUpdate(Long id, AreaCreate areaCreate) {
    return areaRepository.findById(id).map(area -> areaMapper.update(area, areaCreate)).map(areaRepository::save)
            .map(areaMapper::toDTO);
}

@Transactional
public AreaDTO update(Long id, AreaCreate areaCreate) {
    return Optional.of(areaCreate).map(area -> {
        area.setId(id);
        return area;
    }).map(areaMapper::toEntity).map(areaRepository::save).map(areaMapper::toDTO).get();
}

@Transactional
public boolean delete(Long id) {
    return areaRepository.findById(id).map(area -> {
        areaRepository.delete(area);
        return true;
    }).orElse(other:false);
}
```

Рисунок 3.10 – Листинг AreaService

В сервисный слой передается одно DTO для создания новых объектов, их изменения, другое DTO возвращается для конвертации объектов в JSON. Для преобразования DTO в сущности базы данных и наоборот используется мапперы из библиотеки Mapstruct. Пример маппера для сущности региона, рисунок 3.10:

```
@Mapper(componentModel = "spring")
public interface AreaMapper {
    AreaDTO toDTO(Area area);

    Area toEntity(AreaCreate areaCreate);

    @BeanMapping(nullValuePropertyMappingStrategy = org.mapstruct.NullValuePropertyMappingStrategy.IGNORE)
    Area update(@MappingTarget Area area, AreaCreate areaCreate);

}
```

Рисунок 3.11 – Листинг AreaMapper

Например, для создания объекта региона необходимо передать объект класса AreaCreate, затем этот объект будет преобразован в сущность Area, сохранен в базе данных, клиенту вернется код 201, а в заголовке ответа Location

будет записан URL для доступа к созданному объекту:

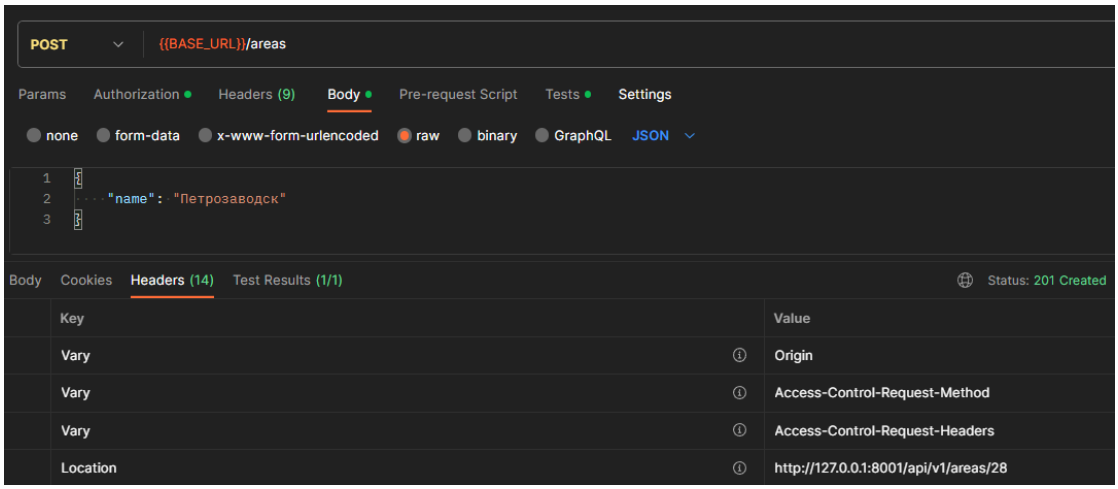


Рисунок 3.12 – Пример запроса в Postman

Для каждой сущности базы данных были реализованы CRUD-операции и фильтрация по различным параметрам. Например, для вывода всех вакансий компании в определенном регионе используется запрос, показанный в Postman на рисунке 3.12:

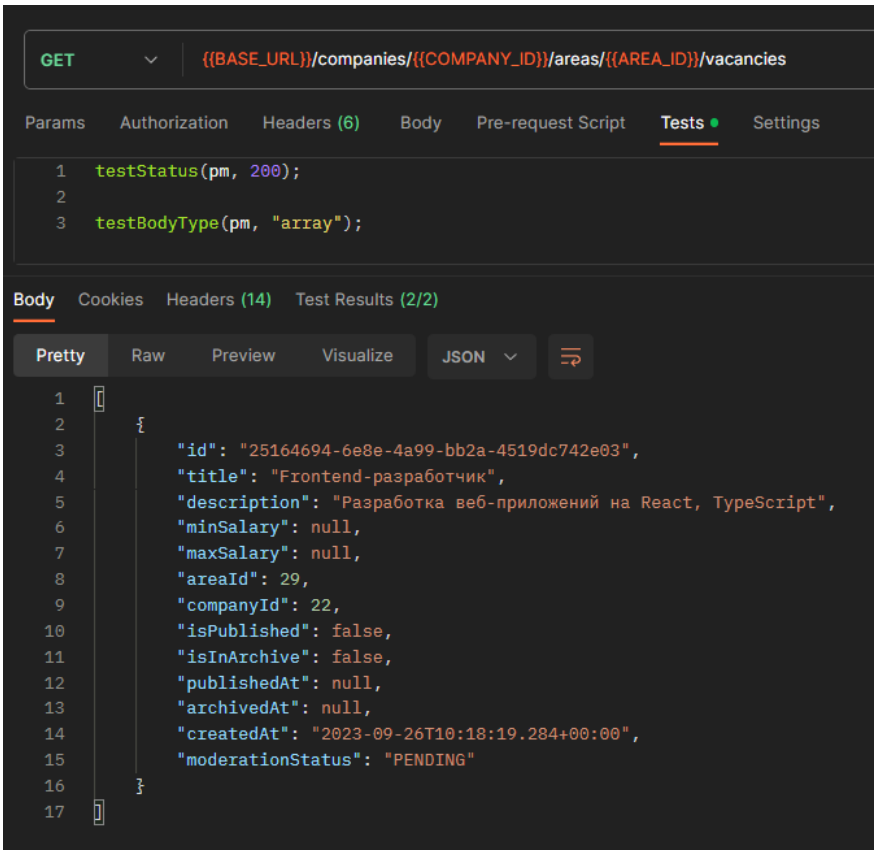


Рисунок 3.13 – Пример вывода всех вакансий компании

4 ТЕСТИРОВАНИЕ

Веб-приложение было протестировано с помощью инструмента тестирования API – Postman. Всего было написано 197 тестов (проверка статусов ответа сервера, типа отправляемого объекта, значений полей, существования объектов в массиве), которые разбиты на 6 тестовых сценариев, рисунок 4.1, 4.2:

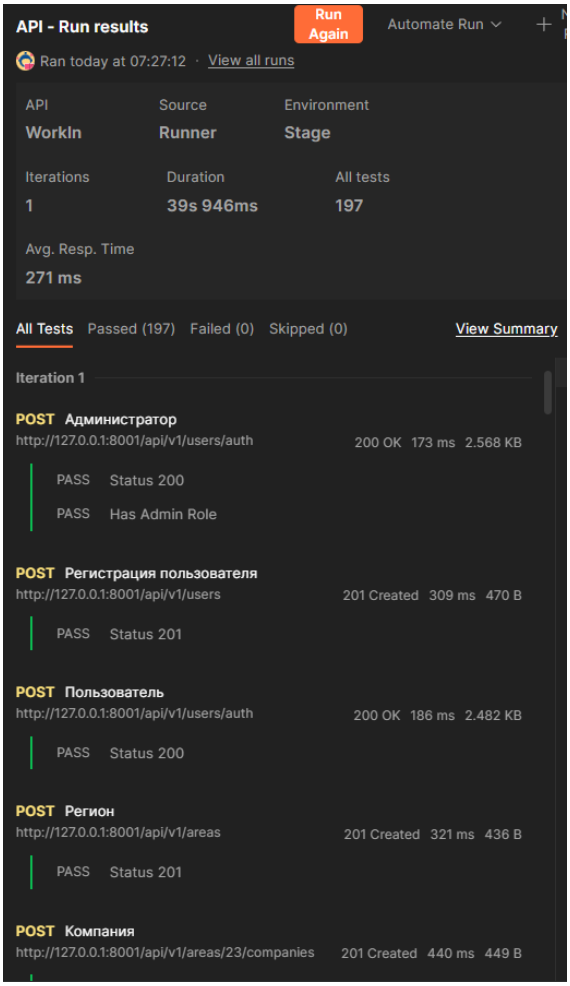


Рисунок 4.1 – Пройденные тесты

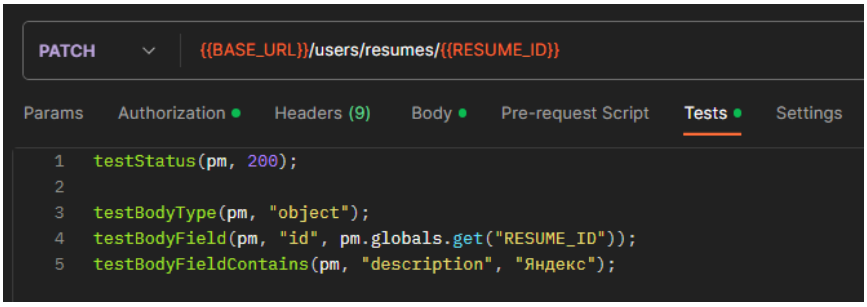


Рисунок 4.2 – Пример тестов для изменения резюме по ID

1. Проверка каскадного удаления всех сущностей

Происходит авторизация сервисного аккаунта администратора и регистрация нового пользователя. От имени администратора создаются Регион, Компания, Вакансия, пользователь создает Резюме и Отклик. Затем удаляется Регион и проверяется, что все связанные сущности (Компания, Вакансия, Отклик) теперь не существуют. Наконец, удаляется пользователь и проверяется на существование Резюме, рисунок 4.3:

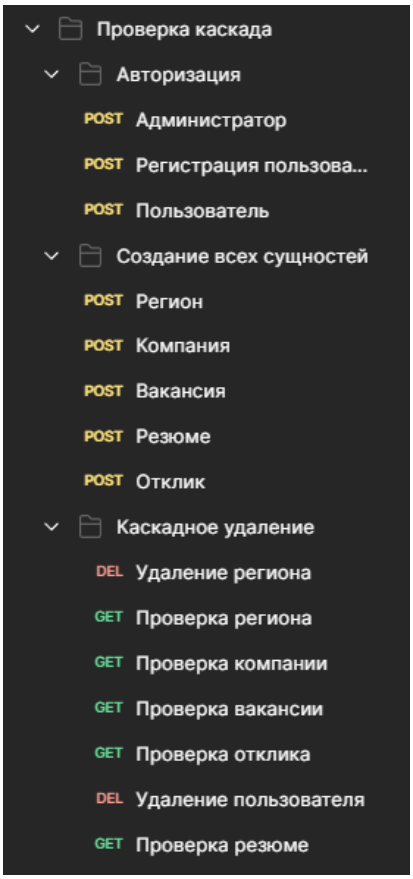


Рисунок 4.3 – Проверка каскадного удаления сущностей

2. CRUD-операции и проверка прав для Региона, рис 4.4:

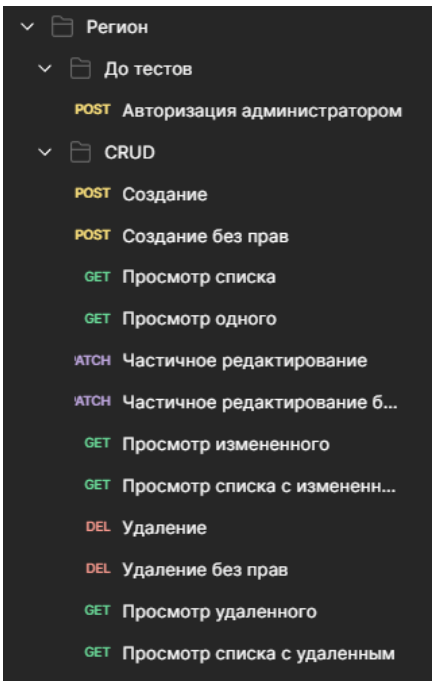


Рисунок 4.4 – Тестирование для Региона

3. CRUD-операции и проверка прав для Компании, рис 4.5:

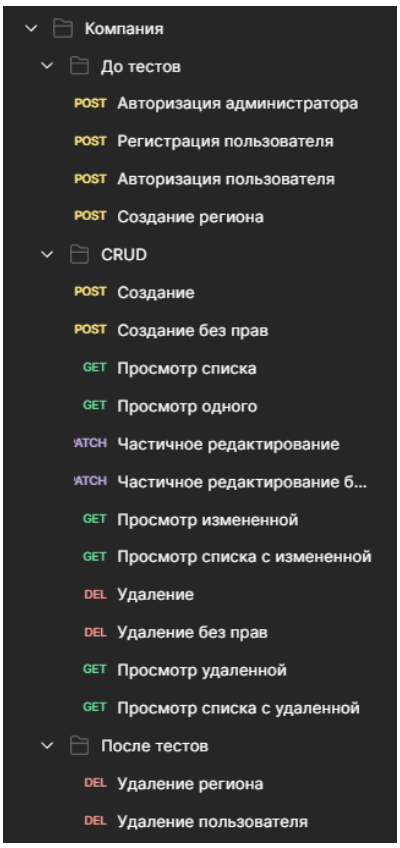


Рисунок 4.5 – Тестирование для Компании

4. CRUD-операции, проверка прав, проверка модератором для Вакансии, рисунок 4.6:

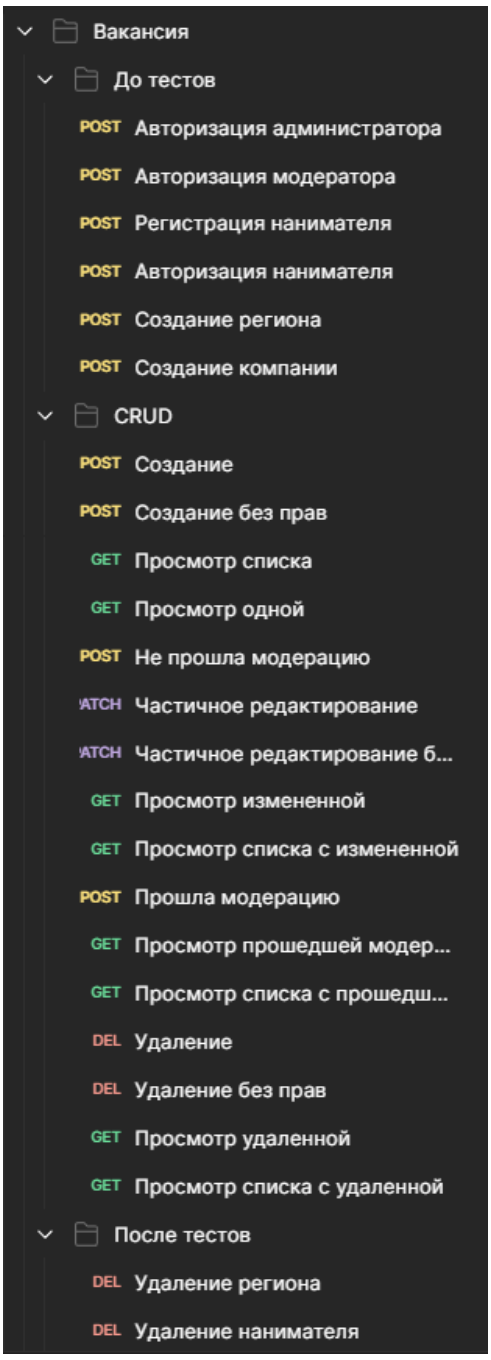


Рисунок 4.6 – Тестирование для Вакансии

5. CRUD-операции, проверка прав и модерация для Резюме, рисунок 4.7:

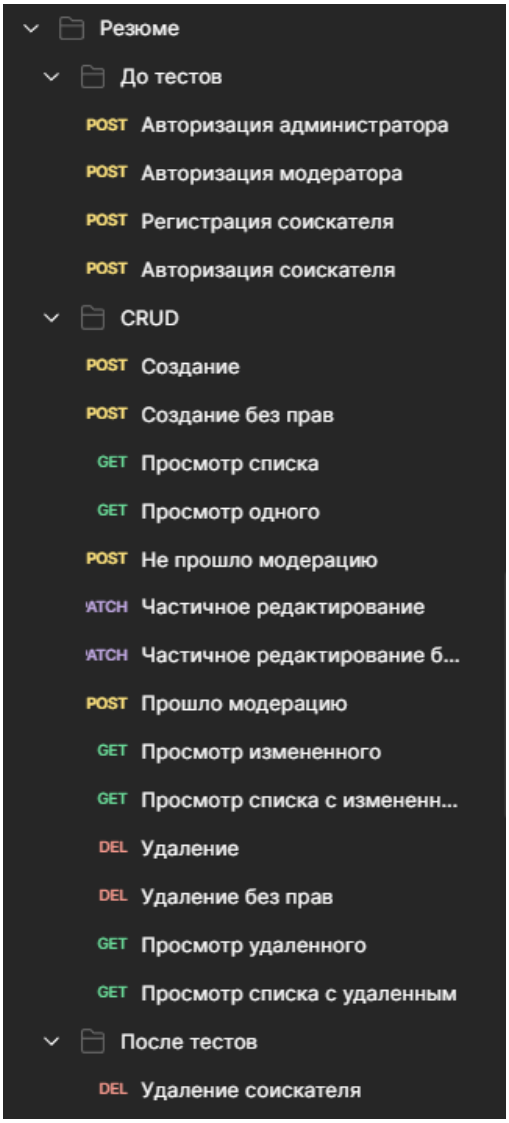


Рисунок 4.7 – Тестирование для Резюме

6. CRUD-операции, проверка прав, модерация для Отклика, рисунок 4.8:

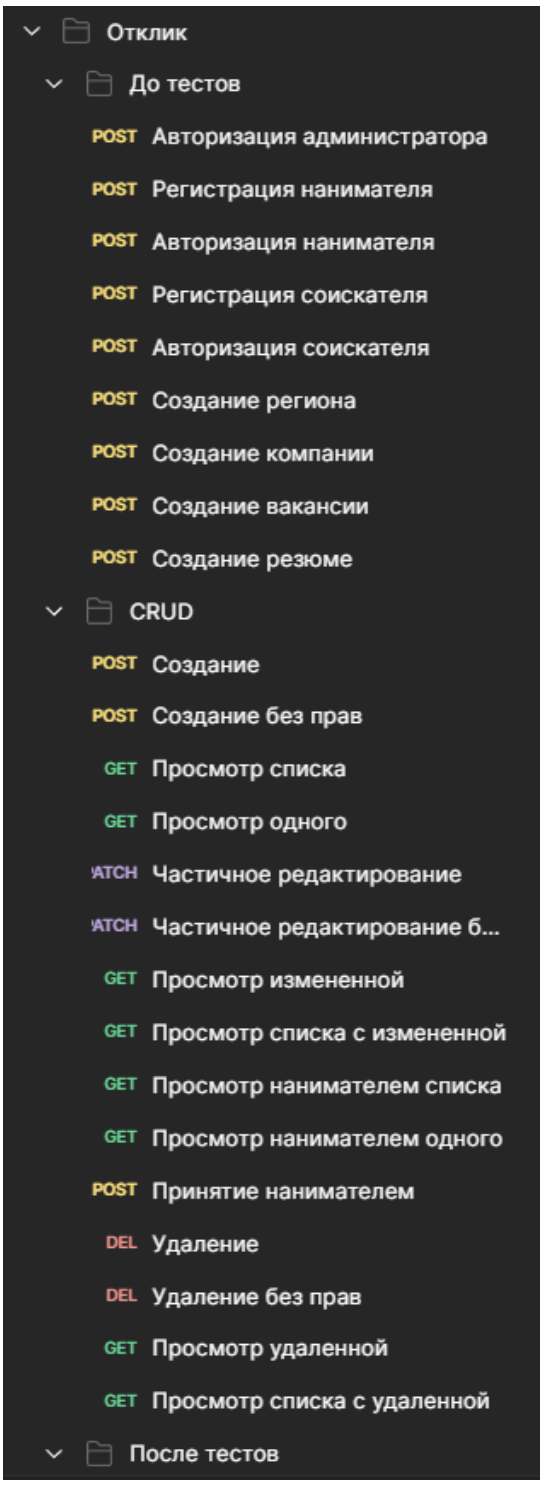


Рисунок 4.8 – Тестирование для Отклика

ЗАКЛЮЧЕНИЕ

В рамках данной курсовой работы, используя возможности языка Java совместно с фреймворком Spring, разработан RESTful веб-сервис для автоматизации поиска потенциальных сотрудников.

Были использованы актуальные подходы к разработке современные веб-сервисов, включая контейнеризацию, OAuth 2 аутентификацию.

Для основного функционала приложения (REST API) написаны тестовые сценарии, используя Postman как инструмент автоматизированного тестирования.

Курсовая работа позволила изучить на практике востребованные веб-технологии и применить в разработке современные подходы. В результате были выполнены поставленные задачи и достигнуты все цели курсовой работы.

С готовым проектом можно ознакомиться в GitHub-репозитории:

https://github.com/descenty/work_in_spring

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Spring Boot Documentation [Электронный ресурс]. – URL: https://docs.spring.io/spring-boot/docs/current/reference/html/ (дата обращения 26.09.2023).

2. JPA @Entity Annotation [Электронный ресурс]. – URL: https://www.baeldung.com/jpa-entity (дата обращения 26.09.2023).

3. PostgreSQL Documentation [Электронный ресурс]. – URL: https://www.postgresql.org/docs/ (дата обращения 26.09.2023).

4. KeyCloak Documentation [Электронный ресурс]. – URL: https://www.keycloak.org/documentation (дата обращения 26.09.2023).

5. Spring Boot Reference Documentation [Электронный ресурс]. – URL: https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/ (дата обращения 26.09.2023).

6. Spring Data JPA [Электронный ресурс]. – URL: https://spring.io/projects/spring-data-jpa (дата обращения 06.05.2023).

7. Spring MVC [Электронный ресурс]. – URL: https://docs.spring.io/spring-framework/docs/current/reference/html/web.html (дата обращения 07.05.2023).

8. Postman Documentation [Электронный ресурс]. – URL: https://learning.postman.com/docs/introduction/overview/ (дата обращения: 26.09.2023)