Федеральное государственное автономное образовательное учреждение высшего образования

Университет ИТМО

Дисциплина: Технологии веб-сервисов

Лабораторная работа 4

Выполнили:
Кривоносов Егор Дмитриевич

Группа: P4214

Преподаватель:
Сафронов Андрей Геннадьевич

2025 г.

Санкт-Петербург

Оглавление

Техническое задание

Необходимо выполнить задание из первой работы, но с использованием REST-сервиса. Таблицу базы данных, а также код для работы с ней можно оставить без изменений.

Постановка задачи

Создать REST-сервис для работы с сущностью `Person`, который предоставляет следующие функции:

- Поиск записей по параметру `query` с поддержкой параметров `limit` и `offset`.
- Получение записи по идентификатору `id`.
- Предоставление данных в формате JSON.

Этапы выполнения

1. Реализован REST-контроллер `PersonRestController` с двумя методами:
    a. `searchPersons`: выполняет поиск записей по параметру `query`, возвращая результаты с пагинацией.
    b. `findPersonById`: находит запись по идентификатору.
2. Настроен сервер с использованием `Jersey`:
    a. `searchPersons`: выполняет поиск записей по параметру `query`, возвращая результаты с пагинацией.
    b. `findPersonById`: находит запись по идентификатору.
3. Подключен слой сервиса `PersonService` для работы с данными из базы.
4. Проведено тестирование запросов через инструменты (например, Postman или cURL).

Комментарии к архитектурным и функциональным аспектам реализации

Контроллер:

1. Использование аннотации `@Path` задает базовый путь для API.
2. Аннотации `@Produces` и `@Consumes` задают формат данных (JSON).
3. Методы обрабатывают запросы `GET` и возвращают данные с помощью `Response`.

```
@Path("/persons")
@Produces(MediaType.APPLICATION_JSON)
@Consumes(MediaType.APPLICATION_JSON)
public class PersonRestController {
    private final PersonService personService;

    @Inject
    public PersonRestController(PersonService personService) {
        this.personService = personService;
    }

    @GET
    public Response searchPersons(@QueryParam("query") String query,
                                  @QueryParam("limit") @DefaultValue("10") int limit,
                                  @QueryParam("offset") @DefaultValue("0") int offset) {
        List<Person> persons = personService.searchPersons(query, limit, offset);
        return Response.ok(persons).build();
    }


    @GET
    @Path("/{id}")
    public Response findPersonById(@PathParam("id") int id) {
        Person person = personService.readPerson(id);
        return person != null ? Response.ok(person).build() :
Response.status(Response.Status.NOT_FOUND).build();
    }
}
```

Настройка сервера:

1. Используется библиотека `Jersey` для настройки REST-сервиса.
2. Провайдер `JacksonJsonProvider` обеспечивает автоматическую конвертацию объектов Java в JSON.
3. Сервер запускается на порту `8080`.

```
ResourceConfig config = new ResourceConfig();
config.register(JacksonJsonProvider.class); // Подключение провайдера JSON
config.register(new PersonRestController(personService)); // Регистрация контроллера

// Настройка и запуск сервера
URI url = UriBuilder.fromUri("http://localhost/").port(8080).build();
HttpServer server = JdkHttpServerFactory.createHttpServer(url, config);
```

Архитектура сервиса разделена на три слоя:

1. **Контроллер**: Обрабатывает HTTP-запросы и возвращает ответы в формате JSON.
2. **Сервис**: Реализует бизнес-логику для поиска и получения данных.
3. **Данные**: Работает с базой данных для извлечения информации.

Rest-клиент:

1. В качестве клиента использовался: `jakarta.ws.rs.client.Client`
2. Посылается запрос, если происходит ошибка, тогда выбрасываеся throw new … и причина ошибки отображается клиенту в консоле.

```
public class PersonRestClient {
    private final String baseUrl;
    private final Client client;
    private final ObjectMapper objectMapper = new ObjectMapper();

    public PersonRestClient(String baseUrl, Client client) {
        this.baseUrl = baseUrl;
        this.client = client;
    }

    public List<PersonDto> searchPersons(String query, int limit, int offset) throws Exception {
        URI uri = UriBuilder.fromUri(baseUrl)
                .path("/persons")
                .queryParam("query", query.replace("%", "%25"))
                .queryParam("limit", limit)
                .queryParam("offset", offset)
                .build();

        Response response =
client.target(uri.toURL().toString()).request(MediaType.APPLICATION_JSON).get();

        if (response.getStatus() == Response.Status.OK.getStatusCode()) {
            return response.readEntity(new GenericType<List<PersonDto>>() {});
        } else {
            throw new RuntimeException("Error fetching persons: " +
response.readEntity(String.class));
        }
    }

    public PersonDto findPersonById(int id) throws Exception {
        URI uri = UriBuilder.fromUri(baseUrl)
                .path("/persons/{id}")
                .resolveTemplate("id", id)
                .build();
        Response response =
client.target(uri.toURL().toString()).request(MediaType.APPLICATION_JSON).get();
        if (response.getStatus() == Response.Status.OK.getStatusCode()) {
            return response.readEntity(new GenericType<>(PersonDto.class));
        } else {
            throw new RuntimeException("Error fetching person: " +
response.readEntity(String.class));
        }
    }
}
```

Ссылка на код

https://github.com/RedGry/TVS-LABS/tree/lab456

Вывод

В ходе выполнения работы был разработан REST-сервис для работы с сущностью Person. Сервис предоставляет функционал поиска записей и получения данных по идентификатору, возвращая их в формате JSON. Реализованы принципы модульности. Настроенный сервер успешно принимает и обрабатывает запросы.