Министерство образования и науки Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
“НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО”

Факультет программной инженерии и компьютерной техники

ЛАБОРАТОРНАЯ РАБОТА № 4

ПО ДИСЦИПЛИНЕ «ТЕХНОЛОГИИ ВЕБ-СЕРВИСОВ»

Студент: Калинин Даниил Леонидович

Группа: P41141

Преподаватель: Дергачев Андрей Михайлович

\

Санкт-Петербург

2021

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.

Этапы выполнения:

1. Для переноса созданного проекта на REST с использованием JAX-RS, в первую очередь был изменен класс App, осуществляющий запуск приложения

```
public class App {
    private static final URI BASE_URI =
URI.create("http://localhost:8080/rest/");
    public static void main(String[] args) {
        HttpServer server = null;
        try {
            ResourceConfig resourceConfig = new

PackagesResourceConfig(PersonResource.class.getPackage().getName());
            server = GrizzlyServerFactory.createHttpServer(BASE_URI,
resourceConfig);
            server.start();
            System.in.read();
            stopServer(server);
        } catch (IOException e) {
            e.printStackTrace();
            stopServer(server);
        }
    }

    private static void stopServer(HttpServer server) {
        if (server != null)
            server.stop();
    }
}
```

2. Затем был создан класс, осуществляющий обработку входящих HTTP-запросов
   Метод getPersons() возвращает список всех людей, подходящих по параметрам, переданным вместе с GET-запросом

```
@Path("/persons")
@Produces({MediaType.APPLICATION_JSON})
public class PersonResource {

    private final MariaDBDAO mdb = new MariaDBDAO();
    private final SQLQueryBuilder sqlQueryBuilder = new SQLQueryBuilder();

    @GET
    @Produces({MediaType.APPLICATION_JSON})
    public List<Person> getPersons(@QueryParam("name") String name,
                                   @QueryParam("id") int id,
```

```
                                    @QueryParam("surname") String surname,
                                    @QueryParam("country") String country,
                                    @QueryParam("gender") String gender,
                                    @QueryParam("age") int age) throws
PersonDoesNotExistException, ServerException, EmptyRequestException {
        Person p = new Person();
        p.setId(id);
        p.setName(name);
        p.setAge(age);
        p.setSurname(surname);
        p.setCountry(country);
        p.setGender(gender);
            List<Person> result =
mdb.getPersonsBySqlQuery(sqlQueryBuilder.buildSelectQuery(p));
            if (result.size() == 0) {
                throw PersonDoesNotExistException.DEFAULT_INSTANCE;
            } else return result;
    }
}
```

Классы для работы с данными, в целом, остались без изменений, однако было принято решение провести небольшую переработку кода:

1. Объекты теперь преобразовываются в SQL-запросы при помощи аннотаций и механизма рефлексии:
   Для этого были созданы аннотации @QueryClass и @QueryField и соответствующий класс – обработчик.
2. Работа с JDBC теперь производится с использованием функциональных интерфейсов, что позволило существенно повысить чистоту и читаемость кода

```
public class MariaDBDAO {

    private ConnectionUtil connectionUtil = ConnectionUtil.getInstance();


    public List<Person> getPersons() throws ServerException {
        return connectionUtil.statement(stmt -> {
            ResultSet rs = stmt.executeQuery("select * from persons");
            return processQuery(rs);
        });
    }

    public String executeUpdateQuery(String sqlQuery) throws ServerException
{
        return connectionUtil.statement(stmt -> {
            int affected  = stmt.executeUpdate(sqlQuery);
            return  "Query affected " + affected + " rows";
        });
    }

    public List<Person> getPersonsBySqlQuery(String sqlQuery) throws
ServerException {
            return connectionUtil.statement(stmt -> {
                ResultSet rs = stmt.executeQuery(sqlQuery);
                return processQuery(rs);
```

```
            });
    }

    public Boolean checkIfPersonExists(String sqlQuery) throws
ServerException {
            return connectionUtil.statement(stmt -> {
                ResultSet rs = stmt.executeQuery(sqlQuery);
                return processQuery(rs).size() > 0;
            });
    }

    public List<Person> getPersonsByName(String name) throws ServerException
{
        return connectionUtil.preparedStatement("SELECT * FROM persons where
name = ?", stmt -> {
            stmt.setString(1, name);
            ResultSet rs = stmt.executeQuery();
            return processQuery(rs);
        });
    }
}
```

Для работы с REST-сервисом был также изменен клиент:

Класс Person, используемый во взаимодействии с сервером, был оставлен без изменений. Остальные классы, сгенерированные по WSDL-описанию сервиса, были удалены за ненадобностью.

Точка входа в приложение выглядит следующим образом:

```
public class App {
    private static final String URL = "http://localhost:8080/rest/persons";

    public static void main(String[] args) {
        OperationHandler operationHandler = new OperationHandler(new
RestClient(URL));
        operationHandler.operationSelectDialog();
    }
}
```

Класс, осуществляющий отправку запросов на сервер и обрабатывающий ответы:

```
public class RestClient {
    private final Client client;
    private final String URL;
    private final Gson gson = new Gson();
    private final JsonParser p = new JsonParser();

    public RestClient(String URL){
        client = Client.create();
        this.URL = URL;
    }
```

```
    private WebResource personToQueryParams(Person p){
        WebResource webResource = client.resource(URL);
        if (p.getName() != null) webResource = webResource.queryParam("name",
p.getName());
        if (p.getSurname() != null) webResource =
webResource.queryParam("surname", p.getSurname());
        if (p.getAge() != 0) webResource = webResource.queryParam("age",
String.valueOf(p.getAge()));
        else webResource = webResource.queryParam("age", "0");
        if (p.getCountry() != null) webResource =
webResource.queryParam("country", p.getCountry());
        if (p.getGender() != null) webResource =
webResource.queryParam("gender", p.getGender());
        if (p.getId() != 0) webResource = webResource.queryParam("id",
String.valueOf(p.getId()));
        else webResource = webResource.queryParam("id", "0");
        return webResource;
    }

    private void processError(ClientResponse response){
        if(response.getStatus() == 404){
            System.out.println("No person found by given query params");
        }
        else {
            response.bufferEntity();
            String x = response.getEntity(String.class);
            System.out.println(x);
            throw new IllegalStateException("Request failed");
        }
    }

    private List<Person> processJSON(ClientResponse response) throws
ParseException {
        List<Person> list;
        String x = response.getEntity(String.class);
        JsonElement jsonContainer = p.parse(x);
        JsonElement jsonQuery = ((JsonObject) jsonContainer).get("person");
        if (jsonQuery.isJsonArray()) {
            list = gson.fromJson(jsonQuery, new TypeToken<List<Person>>() {
            }.getType());
        } else if (jsonQuery.isJsonObject()) {
            Person result = gson.fromJson(jsonQuery, Person.class);
            list = new ArrayList<>();
            list.add(result);
        } else throw new ParseException("Unable to parse input json", 1);
        return list;
    }

    private List<Person> processResponse(ClientResponse response) {
        List<Person> list = new ArrayList<>();
        if (response.getStatus() != ClientResponse.Status.OK.getStatusCode())
{
            processError(response);
            return list;
        }
        else {
            try{
                return processJSON(response);
            } catch (ParseException e){
                System.out.println(e.getMessage());
            }
        }
        return list;
```

```
    }


    public List<Person> getPersons(Person person) {
        WebResource webResource = personToQueryParams(person);
        ClientResponse response =

webResource.accept(MediaType.APPLICATION_JSON).get(ClientResponse.class);
        return processResponse(response);
    }
```

Выводы:

В ходе выполнения лабораторной работы клиент и сервер были модифицированы в REST-ресурс и приложение для работы с ним. Создан обработчик поиска человека по параметрам на основе параметров GET-запроса

Вопросы:

Ссылка на GitHub:

https://github.com/KalininDL/web_services_spring