Федеральное государственное автономное образовательное учреждение высшего образования

«**Университет ИТМО»**

Факультет ПИиКТ

Дисциплина: Технологии веб-сервисов

Лабораторная работа 4

Выполнили: Гурин Евгений Иванович, Камышанская Ксения Васильевна

Преподаватель: Дергачев Андрей Михайлович

Группа: P4216

Санкт-Петербург 2024г.

Задача

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.

Выполнение

В данной лабораторной не пришлось делать кардинальных изменений для реализации REST вместо SOAP. Для этого используется спецификация JAX-RS

И JSON провайдер Resteasy. В данной лабораторной возникли трудности при реализации standalone сервиса, так как встроенного сервера эндпоинтов в данной спецификации нет в отличие от JAX-WS. Поэтому пришлось настраивать встроенный сервер Undertow и Weld servlet container. Это заняло довольно много времени и было сложно подобрать рабочий набор версий. При деплое на wildfly таких проблем не возникло. Кроме того был добавлен swagger UI интерфейс для удобства просмотра апи.

1. Основные изменения в коде затронули контроллер. Изменился способ передачи аргументов, а также набор аннотаций.

```
/**
 * REST API resource for City entity.
 */
@RequestScoped
@Path("/city")
@Produces(MediaType.APPLICATION_JSON)
public class CityResource {
    /**
     * Logger.
     */
    private static final Logger LOGGER =
LogManager.getLogger(CityResource.class);

    /**
     * The city service.
     */
    @Inject
    private CityService cityService;

    /**
     * Find city paginated list by filter.
     *
     * @param name city name. Optional.
     * @param area city area. Optional.
```

```
     * @param population city population. Optional.
     * @param metersAboveSeaLevel city meters above sea level. Optional.
     * @param populationDensity city population density. Optional.
     * @param carCode city car code. Optional.
     * @param limit limit number of entities. Optional.
     * @param offset offset number of entities. Used for pagination.
Optional
     * @return paginated entities list
     */
    @GET
    @Path("/")
    @SneakyThrows
    @Operation(
            summary = "Find cities by filters",
            tags = {"city"},
            description = "Returns a list of cities",
            responses = {
                @ApiResponse(
                        description = "The city list",
                        content = @Content(schema = @Schema(implementation
= PaginationDTO.class))),
                @ApiResponse(responseCode = "400", description = "Invalid
request data"),
            })
    public PaginationDTO<CityDTO> findByFilter(
            @Parameter(description = "City name") @QueryParam("name") final
String name,
            @Parameter(description = "City area") @QueryParam("area") final
Integer area,
            @Parameter(description = "City population")
@QueryParam("population") final Integer population,
            @Parameter(description = "City meters_above_sea_level")
@QueryParam("meters_above_sea_level")
                    final Integer metersAboveSeaLevel,
            @Parameter(description = "City population_density")
@QueryParam("population_density")
                    final Integer populationDensity,
            @Parameter(description = "City car_code")
@QueryParam("car_code") final Integer carCode,
            @Parameter(description = "City limit") @QueryParam("limit")
final Integer limit,
            @Parameter(description = "City offset") @QueryParam("offset")
final Integer offset) {
        LOGGER.info("Run findByFilter");
        var pagination = PaginationRequestDTO.builder()

.limit(Optional.ofNullable(limit).orElse(PaginationRequestDTO.DEFAULT_LIMIT
))

.offset(Optional.ofNullable(offset).orElse(PaginationRequestDTO.DEFAULT_OFF
SET))
                .build();
        Validator.validate(pagination);
        var filterMap = new HashMap<String, Object>() {
            {
                put("name", name);
```

```
                put("area", area);
                put("population", population);
                put("metersAboveSeaLevel", metersAboveSeaLevel);
                put("populationDensity", populationDensity);
                put("carCode", carCode);
            }
        };
        var filters = filterMap.entrySet().stream()
                .filter(it -> Objects.nonNull(it.getValue()))
                .map(it -> FilterArgumentDTO.builder()
                        .field(it.getKey())
                        .value(it.getValue().toString())
                        .build())
                .toList();

        return this.cityService.findByFilter(filters, pagination);
    }
}
```

2. Также был создан класс RestApplication, в котором выполняется регистрация сервиса

```
/**
 * Rest application configuration.
 */
@ApplicationPath("/api")
@OpenAPIDefinition(servers = @Server(url = "/lab4-server"))
public class RestApplication extends Application {
    /**
     * The logger.
     */
    private static final Logger LOGGER =
LogManager.getLogger(RestApplication.class);

    /**
     * Provide services and mappers classes for app.
     * @return classes set.
     */
    @Override
    public Set<Class<?>> getClasses() {
        return Stream.of(CityResource.class, OpenApiResource.class,
ThrowableMapper.class)
                .collect(Collectors.toSet());
    }

    /**
     * Post init.
     */
    @PostConstruct
    public void init() {
        LOGGER.info("init RestApplication");
    }
}
```

3. Обновление Standalone запуска с использованием Undertow сервера
4. Далее необходимо было обновить proxy api для доступа к апи. Для этого использовался JAX-RS Client и реализация от resteasy. Использовалась возможность создания клиента на основе интерфейса.

```
**
 * API interface for city entity management.
 */
@Path("/api/city")
@Produces(MediaType.APPLICATION_JSON)
public interface CityApi {
    /**
     * Find city paginated list by filter.
     *
     * @param name city name. Optional.
     * @param area city area. Optional.
     * @param population city population. Optional.
     * @param metersAboveSeaLevel city meters above sea level. Optional.
     * @param populationDensity city population density. Optional.
     * @param carCode city car code. Optional.
     * @param limit limit number of entities. Optional.
     * @param offset offset number of entities. Used for pagination.
Optional
     * @return paginated entities list
     */
    @GET
    @Path("/")
    PaginationDTO<CityDTO> findByFilter(
            @QueryParam("name") String name,
            @QueryParam("area") Integer area,
            @QueryParam("population") Integer population,
            @QueryParam("meters_above_sea_level") Integer
metersAboveSeaLevel,
            @QueryParam("population_density") Integer populationDensity,
            @QueryParam("car_code") Integer carCode,
            @QueryParam("limit") Integer limit,
            @QueryParam("offset") Integer offset);
}
```

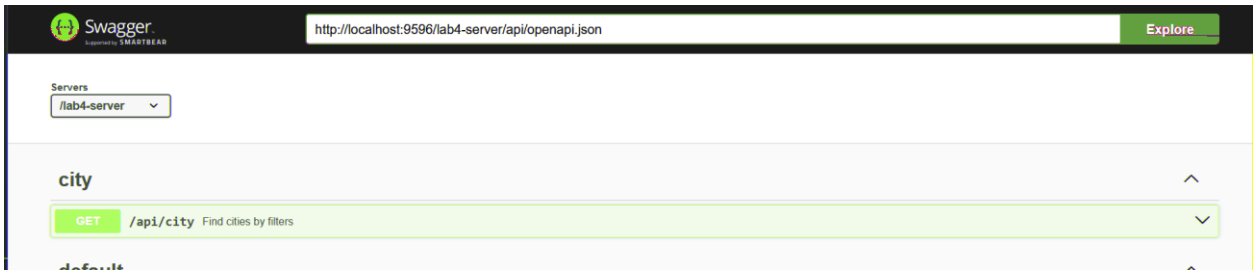
```
/**
 * Builder class for city API rest client.
 */
public class ClientApiBuilder {
    /**
     * Base URI of application.
     */
    private final URI baseURI;

    /**
     * Constructor.
     * @param baseURI Base URI of application.
```

```
     */
    public ClientApiBuilder(final URI baseURI) {
        this.baseURI = baseURI;
    }

    /**
     * Build API client wrapper.
     * @return CityApi service implementation
     */
    public CityApi buildCityApiClient() {
        Client client = ClientBuilder.newBuilder()
                .register(ResteasyJackson2Provider.class)
                .build();
        ResteasyWebTarget target = (ResteasyWebTarget)
client.target(this.baseURI);
        return target.proxy(CityApi.class);
    }
}
```

5. В standalone клиенте обноления потребовало только использование proxy api



Вывод

В данной лабораторной работе был реализован REST сервис на основе спецификации JAX-RS. Основные трудности возникли при реализации standalone версии из-за необходимости настройки embedded сервера с использование servlet контейнера для java se и последующим запуском REST приложения.