Министерство науки и высшего образования РФ
Федеральное государственное автономное
образовательное учреждение высшего образования
«**СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ**»

Институт космических и информационных технологий
институт

Программная инженерия
кафедра

ОТЧЕТ О ПРАКТИЧЕСКОЙ РАБОТЕ

Документирование API при помощи Swagger
Тема

Преподаватель _____ К. В. Богданов
подпись, дата | инициалы, фамилия

Студент КИ21-16/2Б, 032161381 _____ Л. М. Соколов
номер группы, зачётной книжки | подпись, дата | инициалы, фамилия

Красноярск 2024

СОДЕРЖАНИЕ

1 Цель

Изучить принципы построения и документирования RESTful API.

2 Задание

Основываясь на данных, собранных в предыдущей работе, построить API для получения и обновления набора данных.

API должно обеспечивать:

1. Авторизацию доступа (через Bearer token или ключ в параметре).
2. Получение информации о количестве содержащихся в наборе данных сущностей.
3. Получение информации о дате последнего обновления набора данных.
4. Получение набора сущностей по ключу (ключ выбирается по усмотрению студента; например - все данные о нобелевских лауреатах из заданной страны).
5. Добавление нового элемента данных.

API должно соответствовать спецификации OpenAPI (Swagger).

API должно быть документировано с помощью Swagger UI

3 Ход выполнения

3.1 Загрузка датасета

Для работы был выбран датасет, содержащий данные о нобелевских лауреатах. Так как он содержит помимо записей о лауреатах еще и данные о их призах и аффилиациях, было создано три сущности для базы данных. Для них были подготовлены DTO (Data Transfer Object) и реализованы способы конвертации одного в другое.

Загрузка данных с внешнего API осуществлялась при помощи Spring Web Client, а для десереализации данных в DTO использовался Jackson.

В листинге 1 представлен класс, отвечающий за обработку запроса, а в листинге 2 – класс – десериализатор Jackson.

Листинг 1 – Обработчик ответа с полученными данными

```
@Component
class Handler(
    private val objectMapper: ObjectMapper,
    private val laureateService: LaureateService
) {

    fun handleResponse(response: String): UpdateFromApiResponse {
        val jsonTree = objectMapper.readTree(response)

        println("Response: ${jsonTree.shortPSting()}")
        val laureates = jsonTree["laureates"].map {
objectMapper.treeToValue<LaureateDto>(it) }.map { it.mapToEntity() }
        val nullables: HashMap<String, Boolean> = hashMapOf()
        laureates.forEach { item ->
            item::class.declaredMemberProperties.forEach { prop ->
                if (prop.getter.call(item) == null) {
                    nullables[prop.name] = true
                }
            }
        }
        if (nullables.isNotEmpty()) {
            log.debug { "Nullables: $nullables" }
        }
        return laureateService.saveAllIfNotExists(laureates)
    }
}
```

Листинг 2 – Десериализатор для представления Json в виде списка DTO

```
class LaureateDeserializer : JsonDeserializer<LaureateDto>() {

    val dateFormatter: DateTimeFormatter = DateTimeFormatter.ofPattern("yyyy-MM-
dd")

    override fun deserialize(p: JsonParser, ctxt: DeserializationContext):
LaureateDto {
        val node: JsonNode = p.codec.readTree(p)

        val laureate = LaureateDto(
            originId = if (node["id"] == null) node["originId"]?.asInt() else
node["id"].asInt(),
            firstname = wrapNullStr(node["firstname"]?.asText()),
            surname = wrapNullStr(node["surname"]?.asText()),
            born = parseBornDate(node["born"]?.asText()).toString(),
            died = parseDiedDate(node["died"]?.asText()).toString(),
            bornCountry = wrapNullStr(node["bornCountry"]?.asText()),
            bornCountryCode = wrapNullStr(node["bornCountryCode"]?.asText()),
            bornCity = wrapNullStr(node["bornCity"]?.asText()),
            diedCountry = wrapNullStr(node["diedCountry"]?.asText()),
            diedCountryCode = wrapNullStr(node["diedCountryCode"]?.asText()),
            diedCity = wrapNullStr(node["diedCity"]?.asText()),
```

```
            gender = wrapNullStr(node["gender"]?.asText()),
            prizes = parsePrizes(node["prizes"])
        )

        return laureate
    }

    private fun wrapNullStr(str: String?): String? {
        return if (str == null || str == "null") null else str
    }

    private fun parseBornDate(born: String?): LocalDate? {
        return when {
            born == null -> null
            born == "null" -> null
            born.endsWith("-00-00") -> LocalDate.parse("${born.substring(0, 4)}-
01-01", dateFormatter)
            else -> born.let { LocalDate.parse(it, dateFormatter) }
        }
    }


    private fun parseDiedDate(died: String?): LocalDate? {
        return when (died) {
            null -> null
            "null" -> null
            "0000-00-00" -> null
            else -> died.let { LocalDate.parse(it, dateFormatter) }
        }
    }

    private fun parsePrizes(prizesNode: JsonNode?): List<PrizeDto> {
        val prizes = mutableListOf<PrizeDto>()
        prizesNode?.forEach { prizeNode ->
            prizes.add(parsePrize(prizeNode))
        }
        return prizes
    }

    private fun parsePrize(prizeNode: JsonNode): PrizeDto {
        val affiliationsNode = prizeNode["affiliations"]

        val affiliations = if (affiliationsNode.isArray && affiliationsNode.all
{ it.isArray && it.isEmpty }) {
            emptyList()
        } else {
            prizeNode["affiliations"]?.map { parseAffiliation(it) } ?: listOf()
        }

        return PrizeDto(
            id = null,
            year = prizeNode["year"]?.asInt(),
            category = prizeNode["category"]?.asText(),
            share = prizeNode["share"]?.asInt(),
            motivation = prizeNode["motivation"]?.asText(),
            affiliations = affiliations
        )
    }
```

```
    private fun parseAffiliation(affiliationNode: JsonNode): AffiliationDto {
        if (affiliationNode.isObject) {
            return AffiliationDto(
                id = null,
                name = affiliationNode["name"]?.asText(),
                city = affiliationNode["city"]?.asText(),
                country = affiliationNode["country"]?.asText()
            )
        }
        return AffiliationDto()
    }
}
```

Далее полученный список сохранялся в таблице при помощи Spring Data JPA и реализованного сервиса для работы с данными о лауреатах.

3.2 Реализация авторизации

Авторизация была настроена с использованием JWT ключа, выдаваемым Spring Security сроком на 1 день. Была добавлена сущность пользователя и все сопутствующие классы для корректной конфигурации Spring Security.

Для прохождения аутентификации пользователь должен отправлять Bearer token в заголовке Authorizaton.

3.3 Реализация метаданных

Для получения информации о дате последнего обновления и кол-ве данных была реализована триггер-функция, обновляющая кол-во данных и дату последнего обновления при каждом обновлении таблицы laureates. Она представлена на листинге 3. А данные, получаемые функцией, сохраняются в единственную запись таблицы laureates_metadata.

Листинг 3 – Триггер функция обновления метаданных

```
CREATE TABLE laureates_metadata
(
    id              SERIAL PRIMARY KEY,
    last_updated    TIMESTAMP NOT NULL,
    laureates_count INT       NOT NULL
);

CREATE OR REPLACE FUNCTION update_laureates_metadata()
    RETURNS TRIGGER AS $$
BEGIN
    UPDATE laureates_metadata
    SET last_updated = CURRENT_TIMESTAMP,
```

```
        laureates_count = (SELECT COUNT(*) FROM laureates)
    WHERE id = 1;

    IF NOT FOUND THEN
        INSERT INTO laureates_metadata (last_updated, laureates_count)
        VALUES (CURRENT_TIMESTAMP, (SELECT COUNT(*) FROM laureates));
    END IF;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trigger_update_laureates_metadata
    AFTER INSERT OR UPDATE OR DELETE ON laureates
    FOR EACH STATEMENT
EXECUTE FUNCTION update_laureates_metadata();
```

Для получения этих данных пользователю необходимо указать Query параметр «meta» без аргументов в адресной строке, на рисунке 1 продемонстрировано данное поведение.

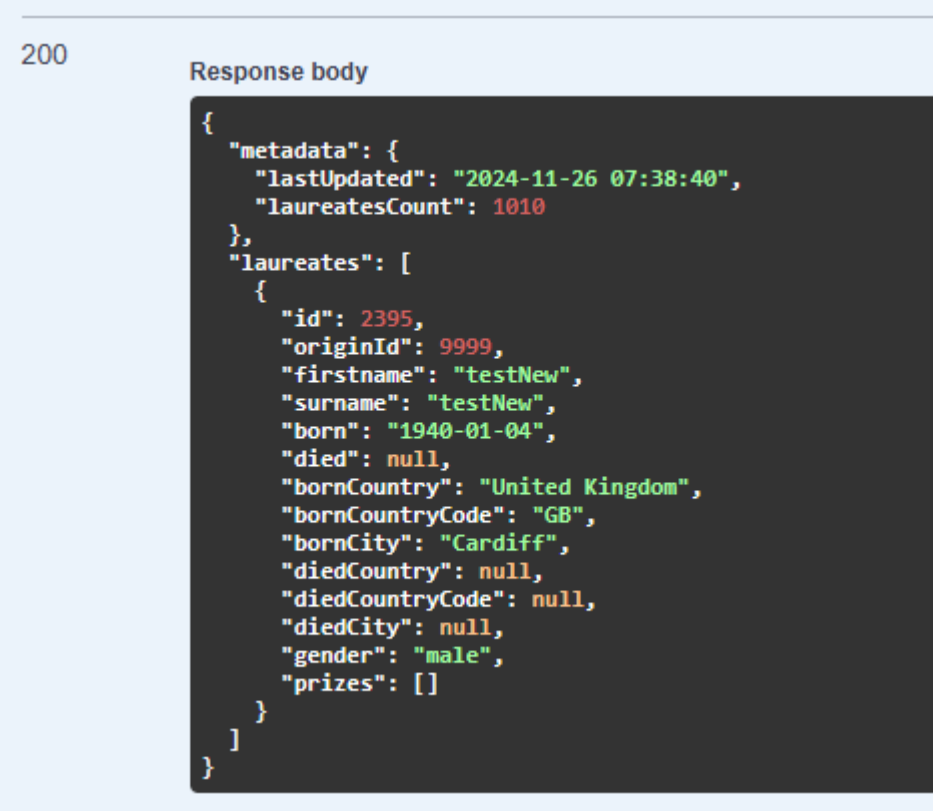

Рисунок 1 – Ответ, содержащий метаданные

3.4 Получение набора сущностей по ключу

Для получения набора сущностей по ключу необходимо также указать ключ и его значение в качестве Query параметра для запроса на получение лауреатов. Например, ответ на рисунке 3 был получен таким запросом:

/api/laureates?firstname=testNew&meta

Таким образом можно фильтровать по любому текстовому полю, однако только лишь по одному.

3.5 Добавление нового элемента данных

Для добавления нового элемента данных были реализованы соответствующие методы контроллера и сервиса. Чтобы добавить запись необходимо отправить запрос, содержащий Json вида, как на рисунке 2.

Рисунок 2 – Вид запроса на добавление записи о лауреате

3.6 Добавление OpenAPI и Swagger UI

Для документации API и его графического представления использовались аннотации над методами, а также класс конфигурации, указывающий данные об API и способ аутентификации. Внешний вид получившегося интерфейса представлен на рисунке 3.

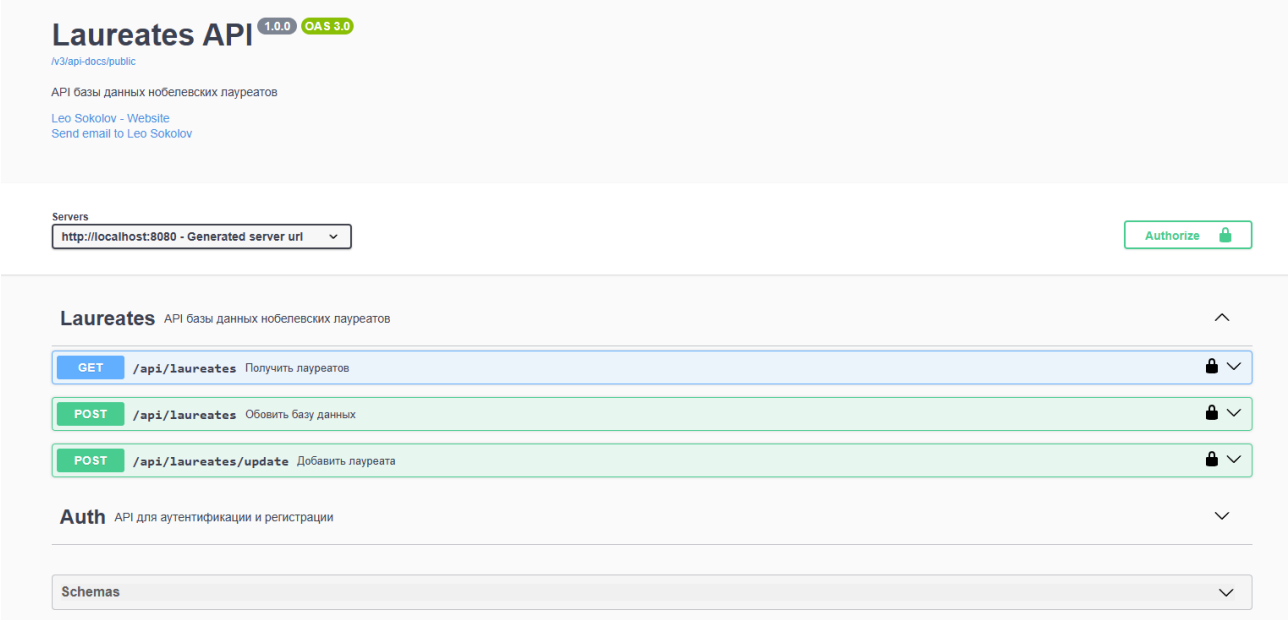

Рисунок 3 – Внешний вид страницы /swagger-ui/index.html

Были задокументированы все эндпоинты, а также схемы данных (DTO).

4 Выводы

В ходе выполнения практической работы была реализовано API, задокументированное при помощи аннотаций OpenAPI, реализующее получение данных, их фильтрацию. Ручное добавление записей, получение их при помощи запроса к стороннему API, а также аутентификация при помощи Bearer Token. Был создан Swagger UI – графический интерфейс, представляющий собой интерактивную документацию созданного API.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. СТУ 7.5-07-2021 Система менеджмента качества. Общие требования к построению, изложении и оформлению документов учебной деятельности. Дата введения – 20.12.2021.