МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ

ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ

ВЫСШЕГО ОБРАЗОВАНИЯ

«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Факультет автоматики и вычислительной техники

Кафедра CCОД



**Отчет**

по проекту

**RESTful микросервис**

по дисциплине “Мобильные приложения”

|  |  |
| --- | --- |
| Факультет: АВТФ  Группа: АТ-04  Студенты:  Буханцев Максим Олегович  Белая Светлана Владимировна | Преподаватель:  Кухто Андрей Вадимович |

Новосибирск

2022

Оглавление

[**Введение** 3](#_Toc120476926)

[**Постановка задачи** 3](#_Toc120476927)

[**Краткое описание работы приложения.** 4](#_Toc120476928)

[**Работа с нашим приложением** 13](#_Toc120476929)

[**Заключение** 19](#_Toc120476930)

# **Введение**

Стек используемых технологий: Java, Spring Boot, LiquiBase, MySQL, Swagger UI.

**Spring Framework**, или просто **Spring** — один из самых популярных фреймворков для создания веб-приложений на Java. **Фреймворк** — программная платформа, определяющая структуру программной системы; программное обеспечение, облегчающее разработку и объединение разных компонентов большого программного проекта. Основная задача Spring - это упростить и сократить работу программиста при создании Java-приложения. Spring представляет контейнер внедрения зависимостей с несколькими модулями. В случае необходимости модули можно легко подключить к проекту.

**LiquiBase** – система управления миграциями баз данных.

**Swagger UI** - веб-фреймворк, который анализирует документ в спецификации OpenAPI и создает веб-страницу интерактивной документации.

# **Постановка задачи**

Целью данного проекта является:

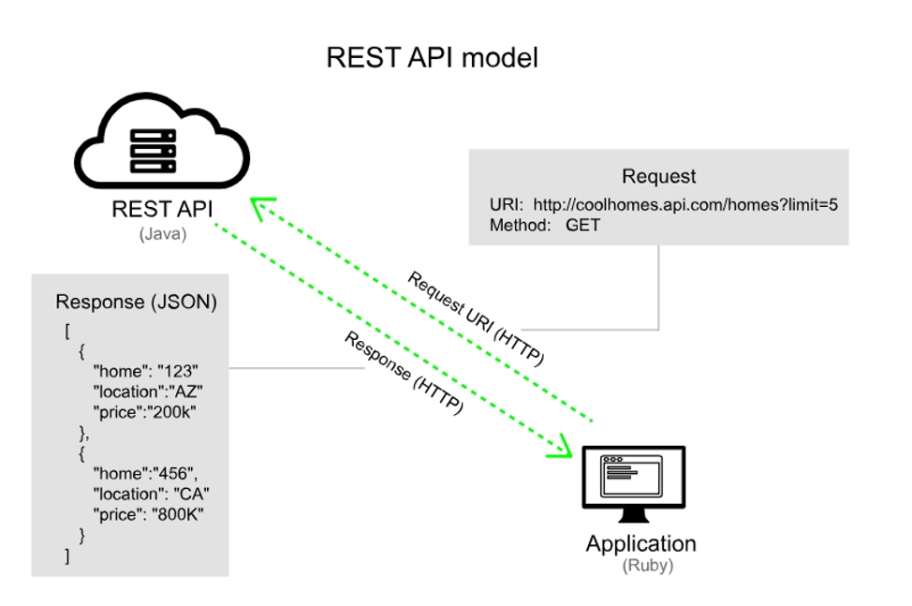
* Разработка собственного REST API backend приложения
* Получение знаний работы с Spring Framework, базой данных MySQL
* Развертывание и публикация Spring Boot приложения с помощью сервиса <railway.app>

Наше приложение является REST API микросервисом для ведения заметок. Оно имеет свою базу данных, в которой хранит сведения о пользователях и их заметках. Микросервис имеет JWT-аутентификацию, с помощью, которой можно осуществлять авторизацию и регистрацию пользователей. В зависимости от ролей, которыми наделен пользователь, он имеет право на выполнение определенных действий. Также микросервис был опубликован в сети Интернет под доменом <https://your-note.ru/>. Приложение имеет документацию с помощью Swagger UI, она находится по адресу <https://your-note.ru/swagger-ui.html#/>.

# **Краткое описание работы приложения.**

Для начала стоит понять что вообще такое REST API.

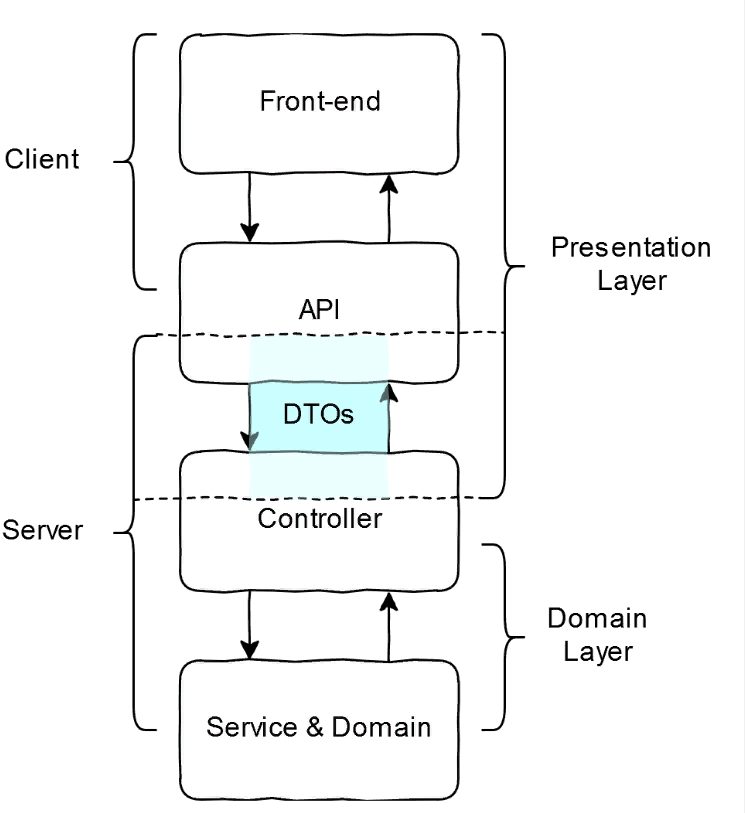
REST – это очень удобный способ коммуникации между приложениями. Он описывает стандарты, используя которые Клиент взаимодействует с Сервером посредством HTTP протокола. REST API не принуждает использовать какой-то конкретный язык программирования. Клиентская и серверная части нашего приложения могут быть написаны на разных языках программирования. Для передачи информации можно использовать не только JSON, но и любой другой формат данных.



Таким образом, наше приложение обрабатывает HTTP запросы с помощью контроллеров, которые мы создали.

Существует около пятнадцати видов HTTP запросов от Клиента к Серверу, но в нашем приложении используются всего 4: GET, POST, PUT и DELETE.

Стоит отметить, что клиент и сервер обмениваются запросами с помощью DTO (Data Transfer Object)



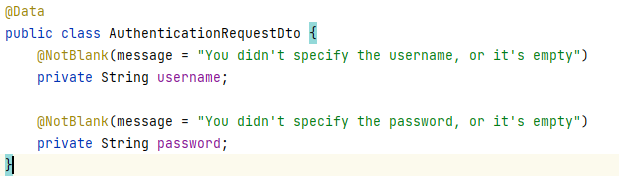
Рассмотрим работу нескольких контроллеров

Наши пользователи для начала должны либо войти в свой аккаунт, либо зарегистрироваться, за это отвечает AuthenticationRestController

В нем есть метод login, отвечающий за авторизацию пользователя.



На вход он получает AuthenticationRequestDto который содержит логин и пароль, вводимые пользователем

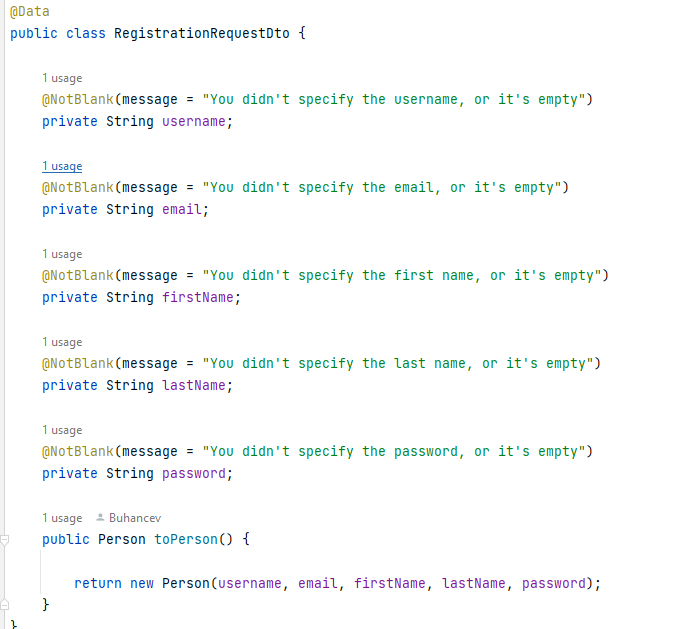


Далее идет процесс аутентификации, если в базе данных есть такой пользователь, то мы генерируем JWT-токен. В конце мы отправляет Клиенту его JWT-токен, который будет в дальнейшем служить для процесса аутентификации запросов между Клиентом и Сервером.

Также есть метод registration, который отвечает за регистрацию пользователей

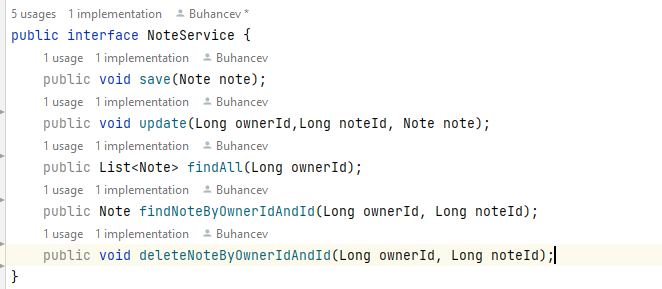


На вход он получает RegistrationRequestDto



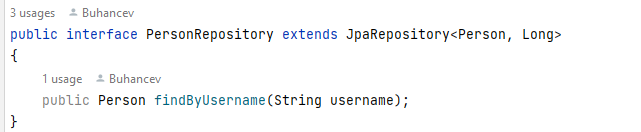
Все поля должны быть NotBlank – не пустые и не null.

Далее идет процесс регистрации, мы обращаемся к сервисному слою personService вызываем функцию registration , в которую передаем нового пользователя.



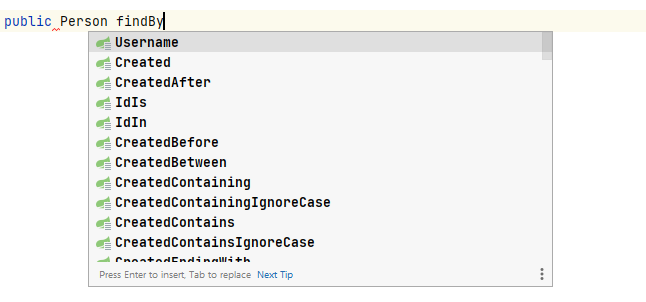
Сервисный слой – слой, который инкапсулирует бизнес логику. Он вызывает слой репозитория, который является отображением работы с базой данных.

Слой репозитория содержит огромное количество методов, для работы с БД. Но если нам недостаточно функционала по умолчания, то можно описать сигнатуру метода, который нам нужен



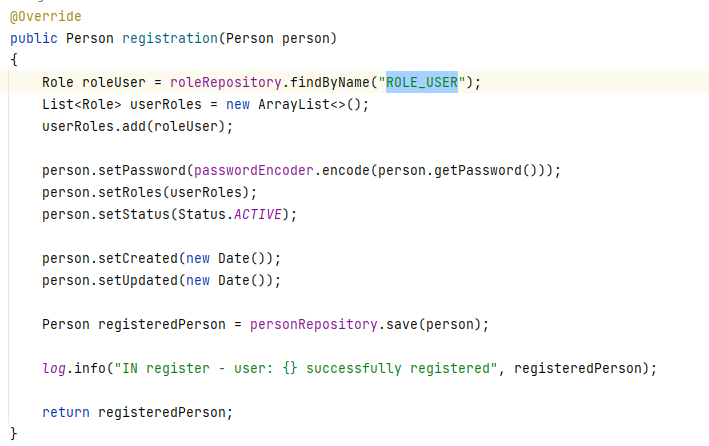
Данную сигнатуру даже не нужно реализовывать, ведь Spring сам (если правильно составить сигнатуру) самостоятельно занимается этим.

Вот пример того, какие есть возможности поиска пользователя из БД



Там огромное количество вариантов, но если всё же нам будет недостаточно, то всегда можно реализовать его вручную.

Вернемся к сервисному слою. Рассмотрим процесс регистрации.



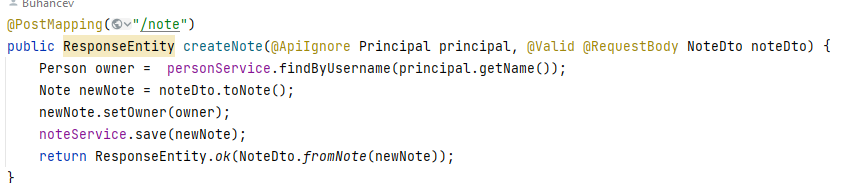
Мы получаем данные нового пользователя, заполняем Create и Update, выдаем роли (USER по умолчанию). Далее мы обращаемся к ранее рассмотренному слою репозитория – personRepository который вызывает save –

встроенная реализация INSERT в базу данных

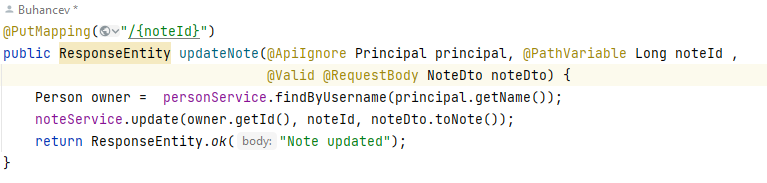
После успешной авторизации пользователь, в зависимости от своих ролей, может пользоваться функционалом нашего сервисом.

Рассмотрим сценарий обычного пользователя, который не имеет прав администратора.

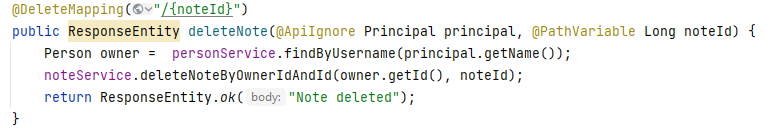
Он может создать заметку



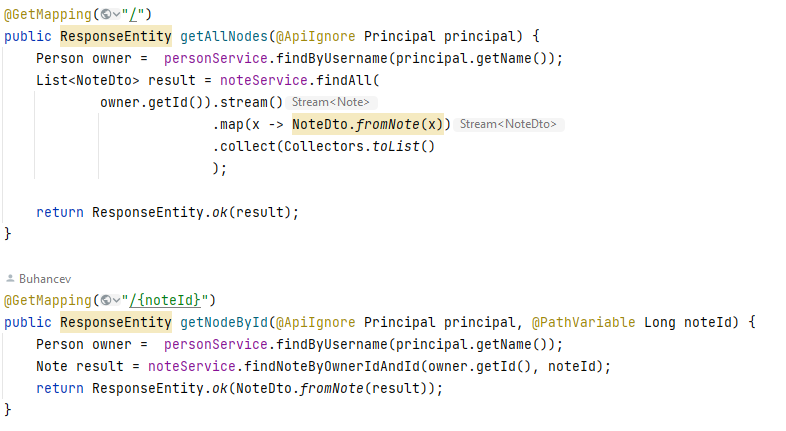
Может обновить уже существующую заметку



Может удалить заметку, если она у него есть



И может вывести на экран

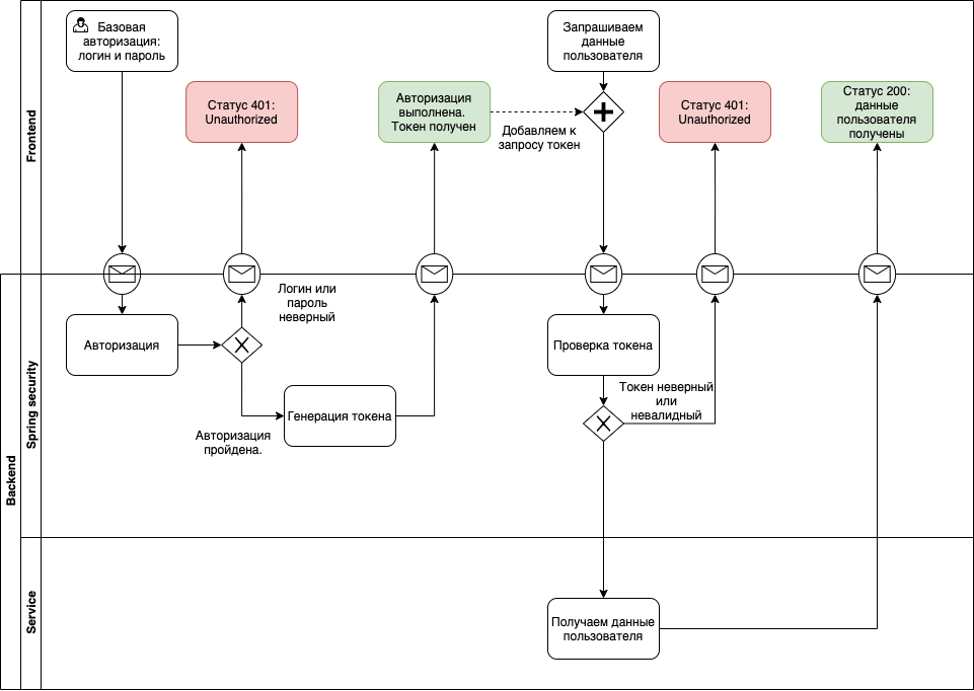


Здесь каждый метод получает Principal. Principal - это пользователь, вошедший в систему в данный момент. Однако мы извлекаем его через контекст безопасности, который привязан к текущему потоку.

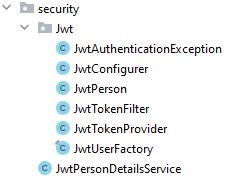
Нам нужно знать какой пользователь обращается на Сервер, чтобы вывести данные, соответствующие именно ему.

Остальные функции снова обращаются к сервисному слою, который содержит бизнес логику и работает с репозиторные слоем, который, в свою очередь, содержит логику работы с БД.

Теперь стоит кратко рассмотреть процесс аутентификации с помощью Spring Security и JWT-token



В данной пакете находится полная реализация JsonWebToken



Примерная работа:

* С помощью конфигурации Spring Security мы создаем уровни доступа

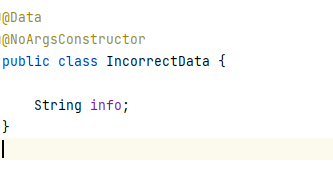
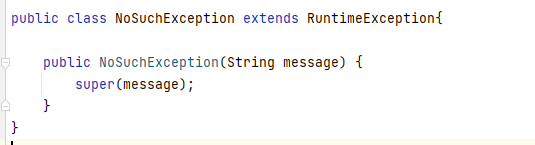


Здесь указано, что все запросы, кроме авторизации и регистрации, должны проходить аутентификацию с помощью JWT-token.

* Теперь Spring будет искать JWT токен и пытаться валидировать его при каждом HTTP запросе

Так же мы сделали обработку ошибок, которые могут появится при работе микросервиса.

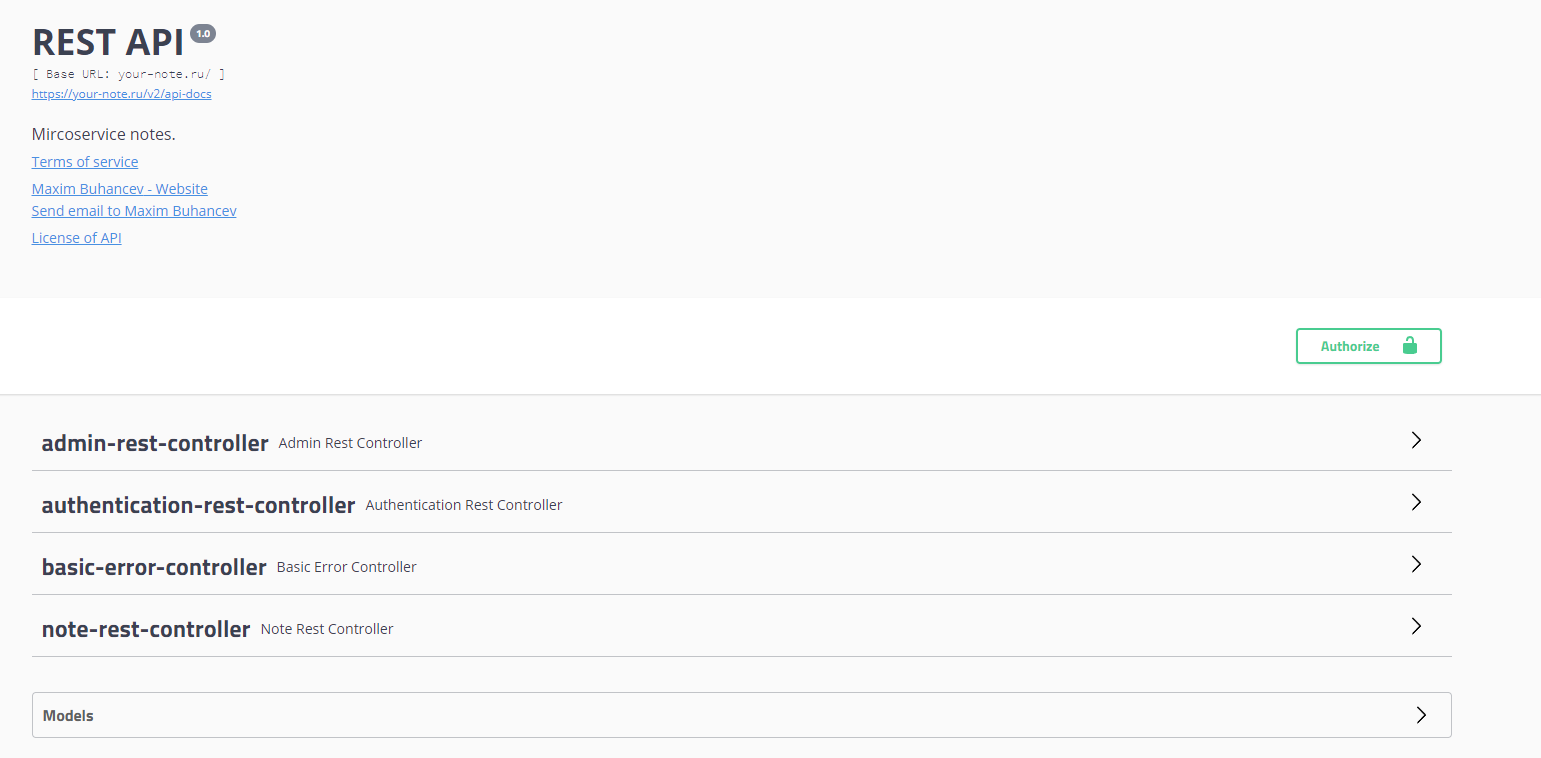
Условно, ошибки делится на 2 типа – ошибки ввода данных и все остальные.

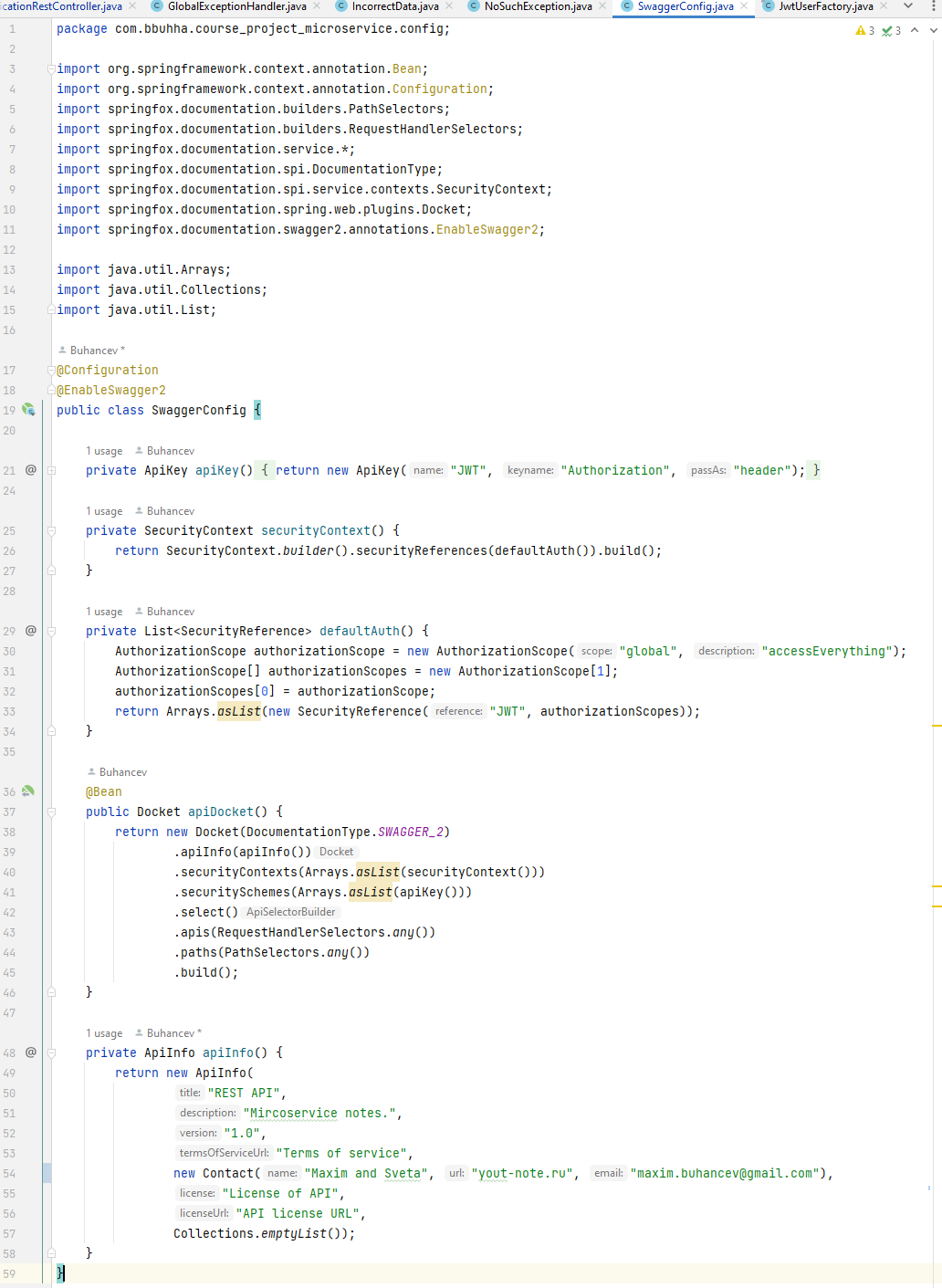
 



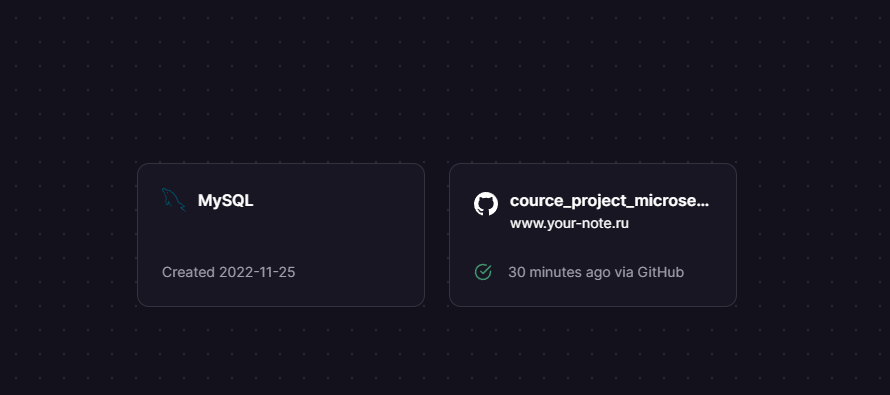
Наш Advice перехватывает ошибки, и возвращает их Клиенту.

Для работы с REST API был добавлен Swagger UI, который анализирует документ в спецификации OpenAPI и создает веб-страницу интерактивной документации по адресу <https://your-note.ru/swagger-ui.html#/>





Стоит отметить, что мы опубликовали наше веб-приложение с помощью сервиса <https://railway.app/>

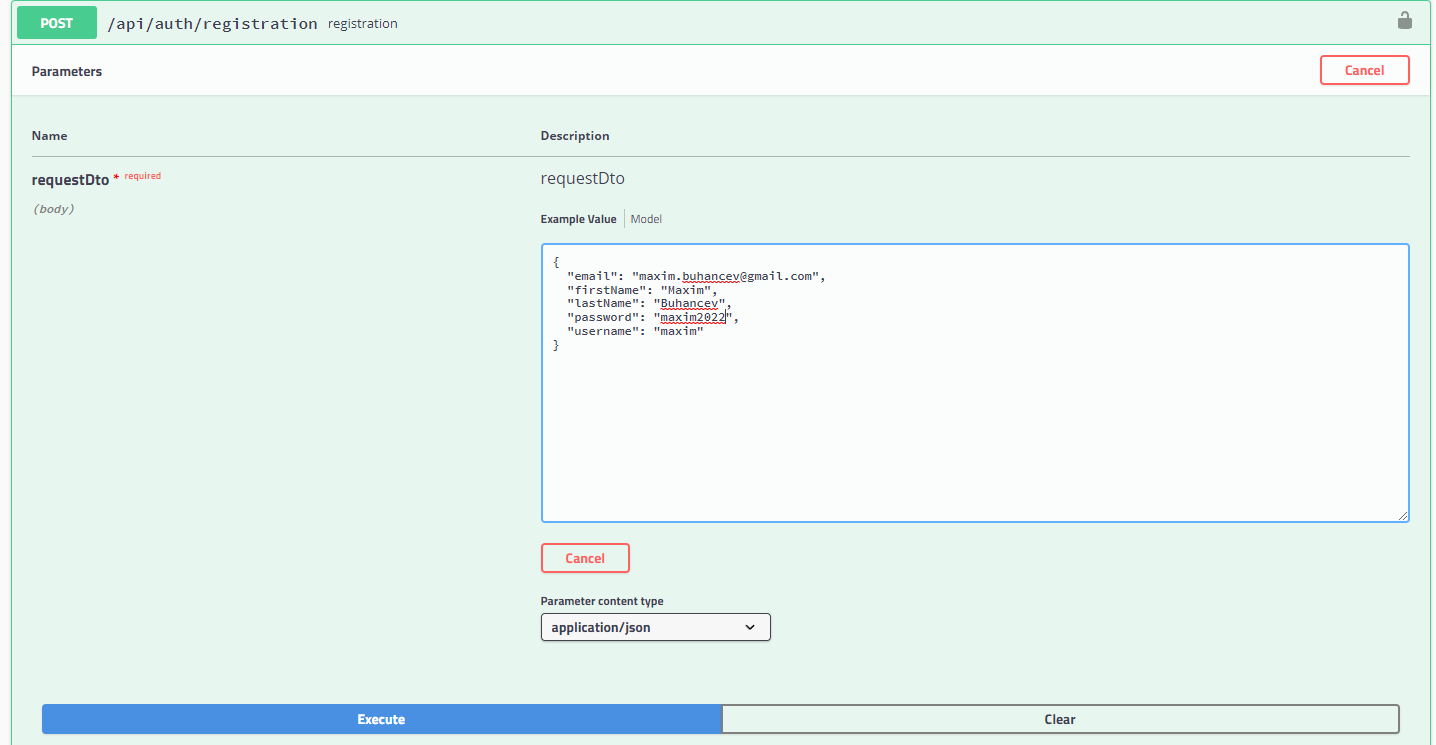


Это очень просто, нужно всего лишь создать базу данных и связать GitHub репозиторий, в котором находится исходный код проекта. После этого сервис сам деплоит и публикует приложение.

# **Работа с нашим приложением**

Теперь попробуем на практике наше приложение! Отправим несколько HTTP запросов. Для работы с REST API можно использовать как Swagger UI, так и другие приложения, например, Postman. Мы используем Swagger UI.

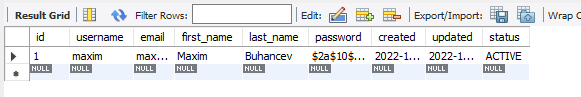
Зарегистрируем нового пользователя



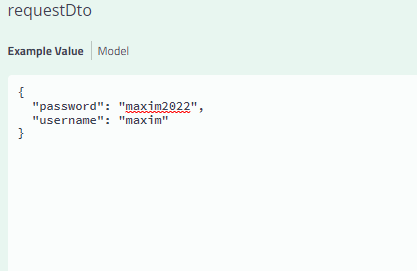
Получаем ответ – пользователь зарегистрирован



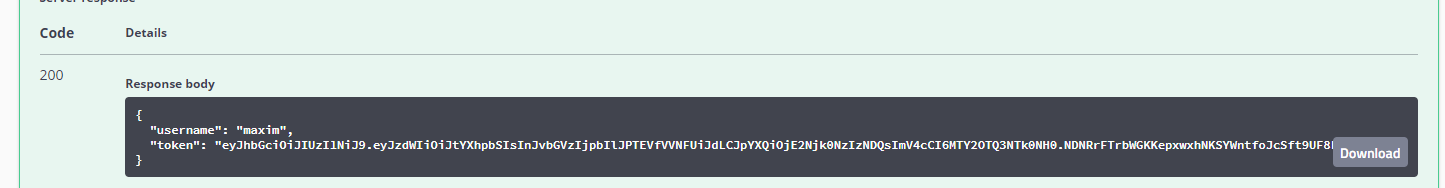
В БД тоже появился пользователь



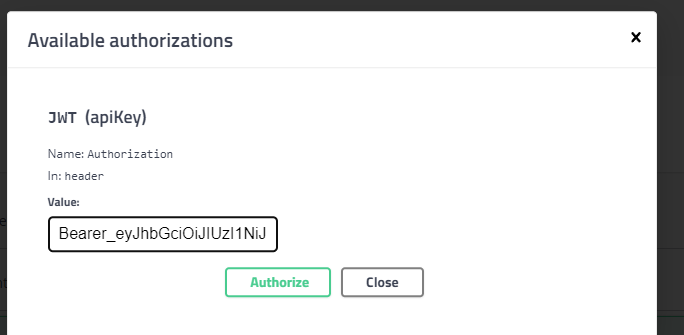
Теперь авторизуемся



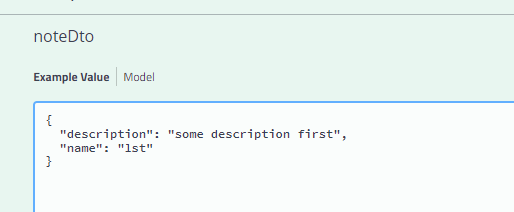
Успешно, получаем токен

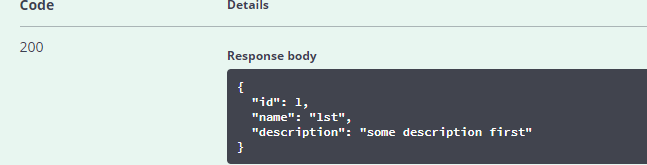


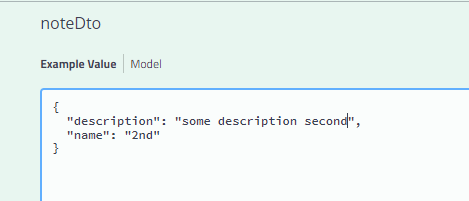
Вводим его в авторизации Swagger

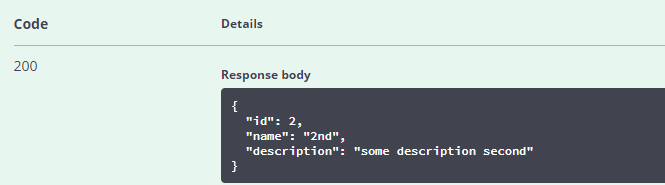


Cоздадим несколько заметок

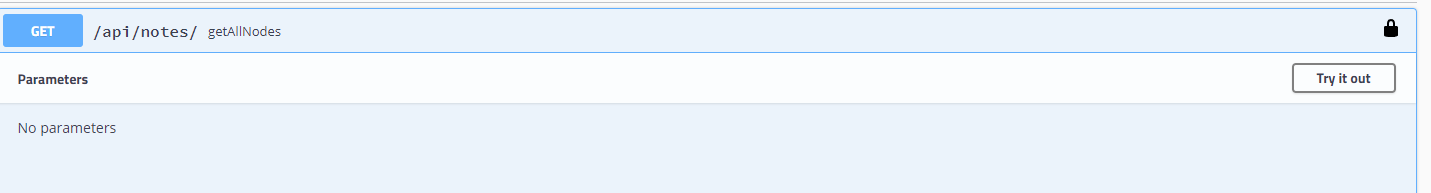






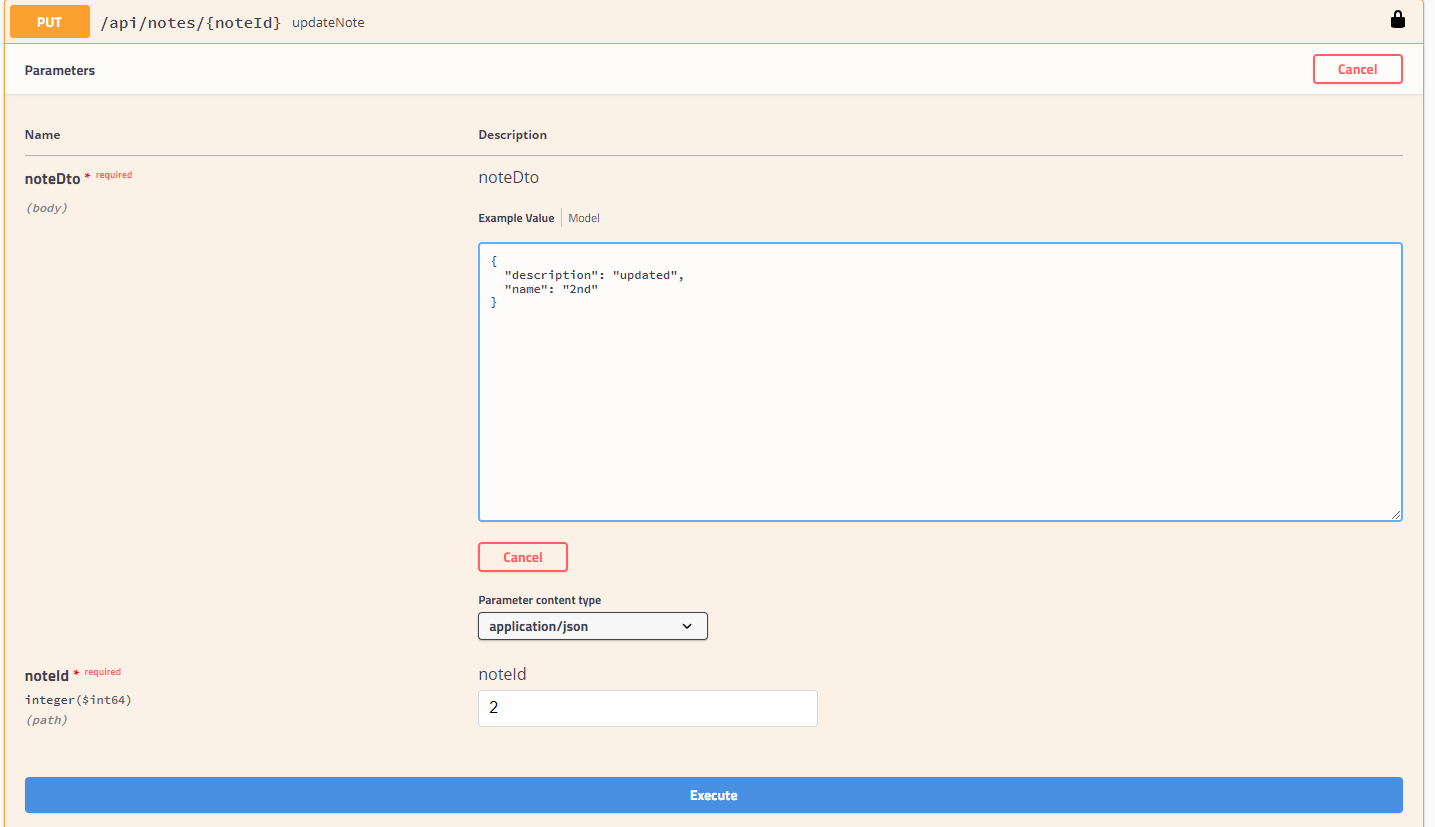


Попробуем вывести все



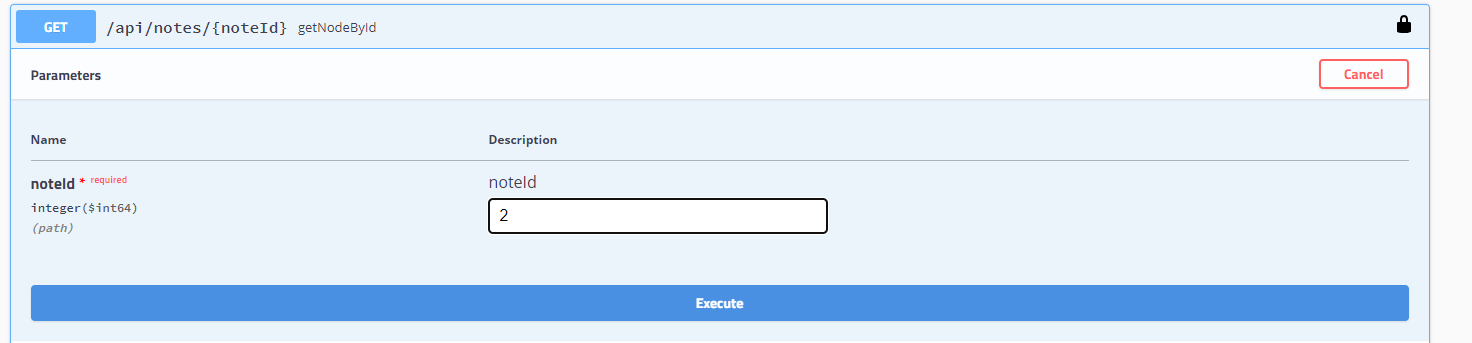


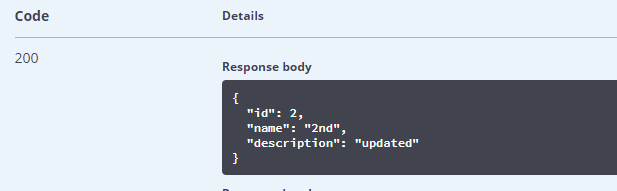
Обновим вторую заметку



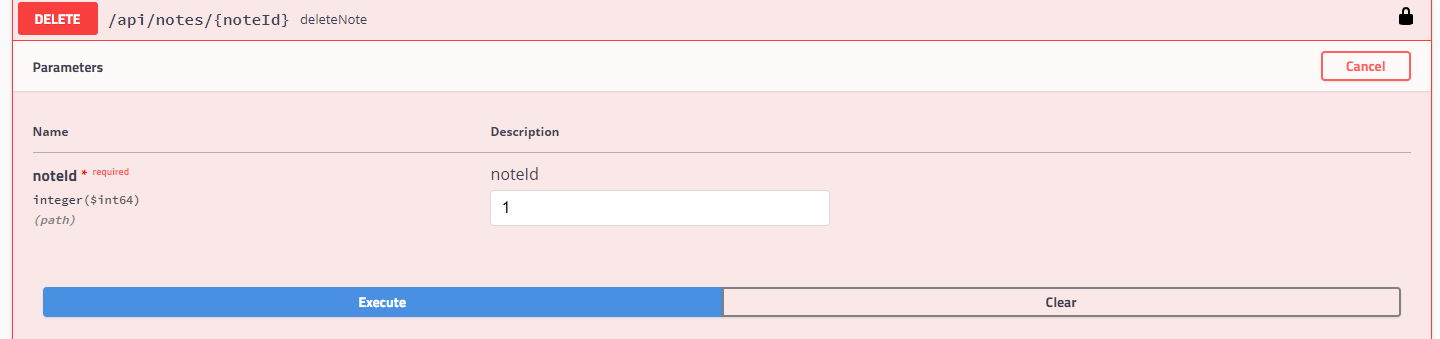


Проверим вторую заметку



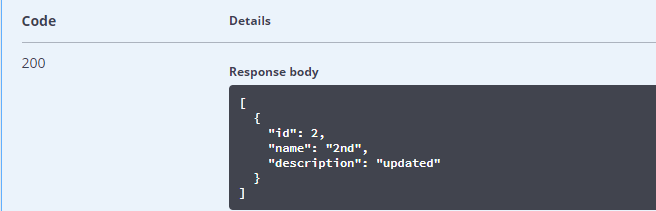


Удалим первую заметку

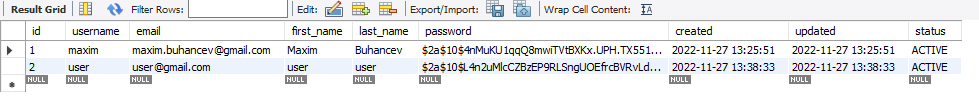
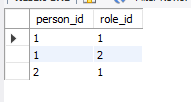




Cнова выведем все наши заметки, чтобы проверить удалилась ли первая заметка



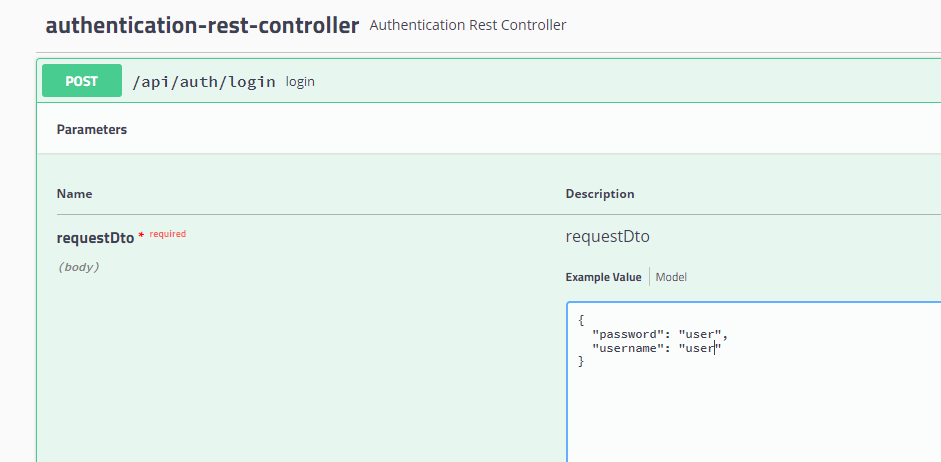
Теперь выдадим права администратора нашему пользователю, создадим еще одного пользователя и проверим функционал администратора.

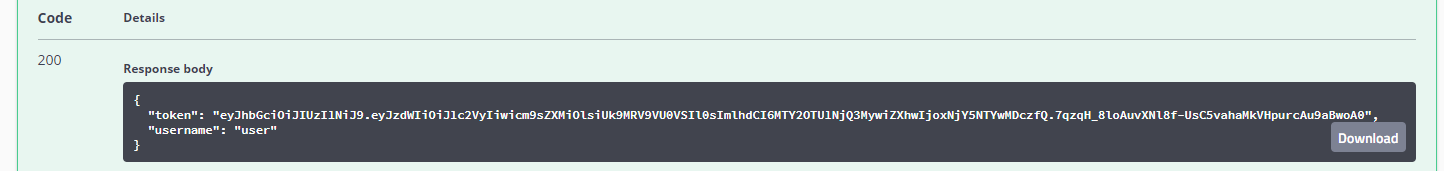


Теперь у нас есть Администратор (роль номер 2) и обычный пользователь.

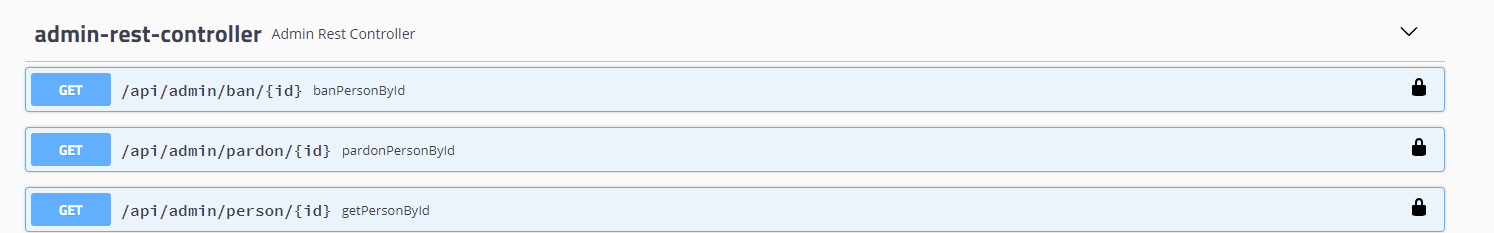
Сначала попробуем получить с помощью обычного пользователя функционал, который ему не доступен

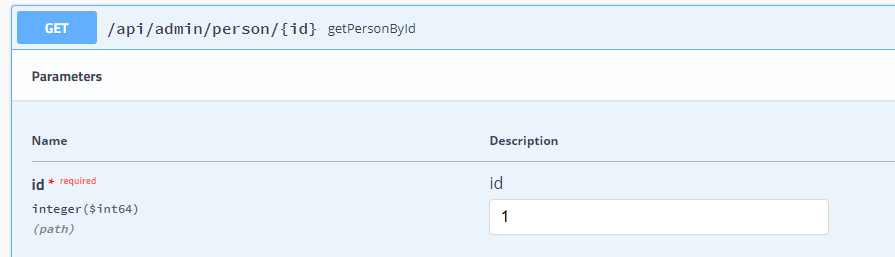
Авторизуемся в системе



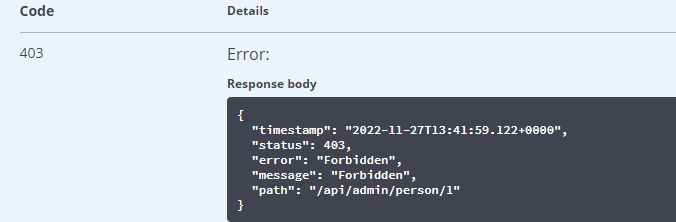


Попробуем получить данные о пользователе

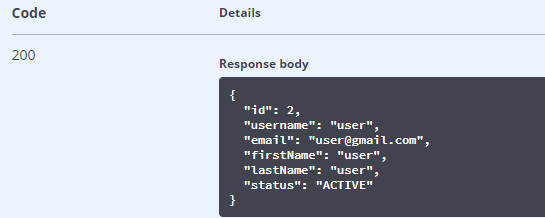




И… получаем 403 ошибку, которая говорит нам, что мы не имеем достаточных прав для работы с данным функционалом! Все верно! Spring Security и JWT работают корректно.



Теперь зайдем с аккаунта администратора и попробуем тоже самое



Всё работает верно!

# **Заключение**

Во время проектирования и разработки данного приложения:

* Мы изучили основы REST API разработки backend приложения
* Научились работать с базой данных MySQL
* Освоили Spring Framework, систему миграций LiquiBase
* Опубликовали готовый проект в сети Интернет по адресу <https://your-note.ru/>
* Научились работать с интерактивной документацией Swagger UI

Весь исходный код находится в нашем GitHub-репозитории <https://github.com/Buhancev/cource_project_microservice>