Федеральное агентство железнодорожного транспорта

Омский государственный университет путей сообщения

Кафедра «Автоматика и системы управления»

К ЗАЩИТЕ ДОПУСТИТЬ

Доцент кафедры АиСУ

\_\_\_\_\_\_\_\_ Е.А. Альтман

ПРОЕКТИРОВАНИЕ REST-СЕРВЕРА

Пояснительная записка к курсовой работе

по дисциплине «Объектно – ориентированное программировании»

ИНМВ.4040000.000 ПЗ

|  |  |
| --- | --- |
| Оценка  «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2020г.  \_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  Подпись Расшифровка подписи | Студент гр. 28М  \_\_\_\_\_\_\_\_\_\_\_Е.А.Захаров  «\_\_» \_\_\_\_\_\_\_\_ 2020 г.  Руководитель – Доцент кафедры «АиСУ»  \_\_\_\_\_\_\_\_\_\_\_Е.А. Альтман  «\_\_» \_\_\_\_\_\_\_\_2020 г. |

Омск 2020

Задание

Спроектировать REST-сервер магазина аксессуаров.

Сервер позволяет получать и редактировать информацию по товарам, продажам и работникам.

Реферат

УДК 004.42

Пояснительная записка к курсовой работе содержит 56 страниц, 84 рисунка, 4 использованных источника, 2 приложения.

Объектом курсовой работы является проектирование и реализация REST-сервера.

Цель курсовой работы – получение основных навыков проектирования REST-сервера, знакомство с ORM Kotlin Exposed и фреймворком Ktor и реализация REST-сервера сайта магазина аксессуаров для телефонов.

Результатом курсовой работы является REST-сервер сайта магазина аксессуаров для телефонов, написанный на языке Kotlin в программе IntelliJ IDEA.

Пояснительная записка выполнена в текстовом редакторе Microsoft Word 2020.

Содержание

[Введение 5](#_Toc65762389)

[1 ER-диаграмма 6](#_Toc65762390)

[2 Диаграмма классов 7](#_Toc65762391)

[3 Диаграмма прецедентов 8](#_Toc65762392)

[4 Диаграммы последовательностей 9](#_Toc65762393)

[5 Описание моделей 17](#_Toc65762394)

[6 Описание запросов 19](#_Toc65762395)

[Заключение 35](#_Toc65762396)

[Библиографический список 36](#_Toc65762397)

[Приложение А 37](#_Toc65762398)

[Приложение B 56](#_Toc65762399)

# Введение

REST (Representational state transfer) – это стиль архитектуры программного обеспечения для распределенных систем, таких как World Wide Web, который, как правило, используется для построения веб-служб. Термин REST был введен в 2000 году Роем Филдингом, одним из авторов HTTP-протокола. Системы, поддерживающие REST, называются RESTful-системами.

В общем случае REST является очень простым интерфейсом управления информацией без использования каких-то дополнительных внутренних прослоек. Каждая единица информации однозначно определяется глобальным идентификатором, таким как URL. Каждая URL в свою очередь имеет строго заданный формат.

В курсовой работе необходимо разработать REST API.

# ER-диаграмма

ER-диаграммы отображают отношения набора сущностей, хранящиеся в базе данных. ER-диаграммы могут объяснить логическую структуру баз данных.

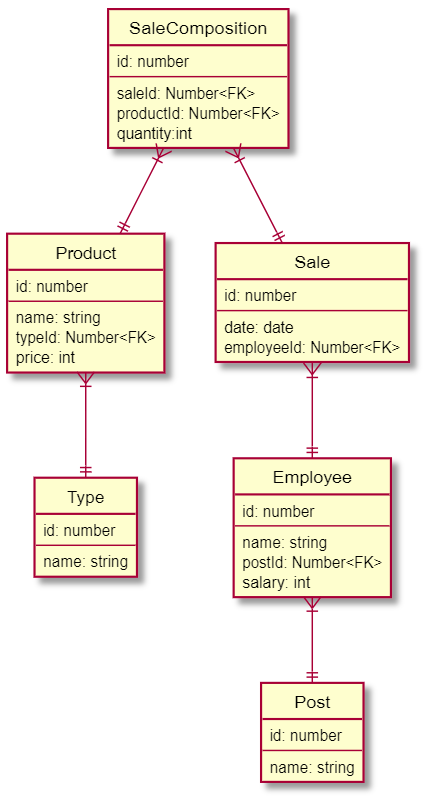


Рисунок 1 – ER-диаграмма

По данной ER-диаграмме была спроектирована база данных для сайта магазина аксессуаров.

# Диаграмма классов

Всего в курсовой работе реализовано 6 классов: Post, Employee, Sale, SaleComposition, Product, Type(рисунок 2).

Класс Post представляет собой должность, которую занимает сотрудник(Employee).

Класс Employee представляет собой сотрудника и содержит ссылку на id Post.

Sale представляет собой продажу и содержит ссылку на id Employee.

SaleComposition представляет собой промежуточную таблицу для объединения Sale и Product.

Product представляет товар и содержит ссылку на id Type.

Type представляет собой тип продукта.

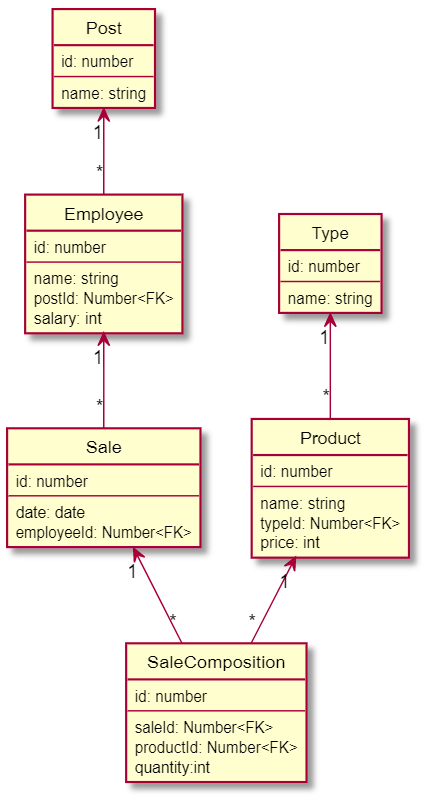


Рисунок 2 – Диаграмма классов

## Диаграмма прецедентов

Для реализации функций получения/ изменения/ создания объектов. Например, получать списки продаж по различным критериям были реализованы следующие прецеденты(рисунок 3).

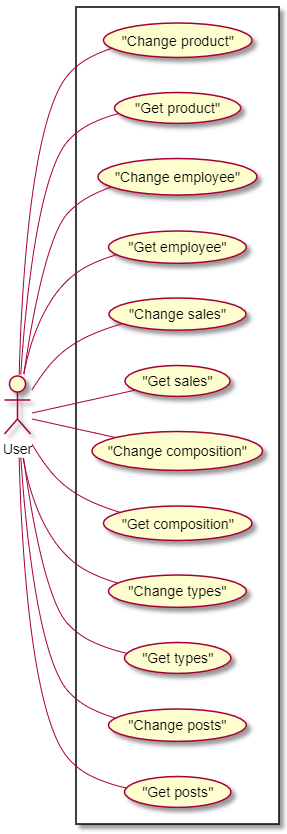


Рисунок 3 – Диаграмма прецедентов

## 4 Диаграммы последовательностей

При выполнении прецедента get product, пользователь отправляет запрос на соответствующий адрес, обрабатывая который, сервер возвращает пользователю JSON объект, соответствующий экземпляру класса Product. В случае, если обработка запроса не была завершена успешна, пользователю будет возвращено уведомление об ошибке.

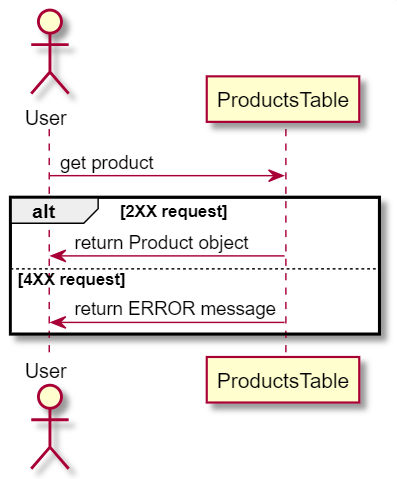


Рисунок 4 – Диаграмма последовательности get product

При выполнении прецедента change product (создании/изменении/удалении товара), пользователь отправляет соответствующий запрос, в тело которого передает объект в формате JSON, соответствующий экземпляру класса Product, который, при методе POST, будет помещен в новую строку таблицы ProductTable. При методе PUT данные в строке данной таблицы будут изменены, при методе DELETE удалены.

Если пользователь передал верные параметры в запросе, то операция будет выполнена, а пользователю будет возвращено соответствующее подтверждающее сообщение, в ином случае пользователь получит уведомление об ошибке.

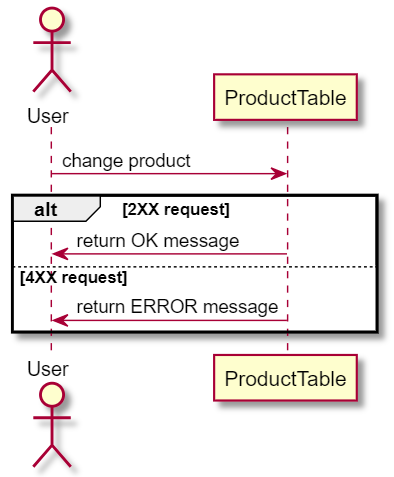


Рисунок 5 – Диаграмма последовательности change product

При выполнении прецедента get employee, пользователь отправляет запрос на соответствующий адрес, обрабатывая который, сервер возвращает пользователю JSON объект, соответствующий экземпляру класса Employee. В случае, если обработка запроса не была завершена успешна, пользователю будет возвращено уведомление об ошибке.

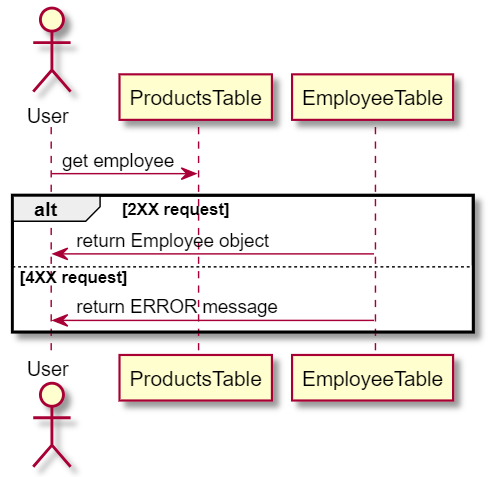


Рисунок 6 – Диаграмма последовательности get product

При выполнении прецедента change employee (создании/изменении/удалении сотрудника), пользователь отправляет соответствующий запрос, в тело которого передает объект в формате JSON, соответствующий экземпляру класса Employee, который, при методе POST, будет помещен в новую строку таблицы EmployeeTable. При методе PUT данные в строке данной таблицы будут изменены, при методе DELETE удалены.

Если пользователь передал верные параметры в запросе, то операция будет выполнена, а пользователю будет возвращено соответствующее подтверждающее сообщение, в ином случае пользователь получит уведомление об ошибке.

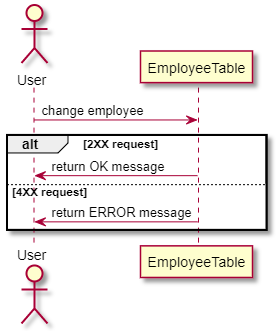


Рисунок 7 – Диаграмма последовательности change product

При выполнении прецедента get sales, пользователь отправляет запрос на соответствующий адрес, обрабатывая который, сервер возвращает пользователю JSON объект, соответствующий экземпляру класса Sale. В случае, если обработка запроса не была завершена успешна, пользователю будет возвращено уведомление об ошибке.

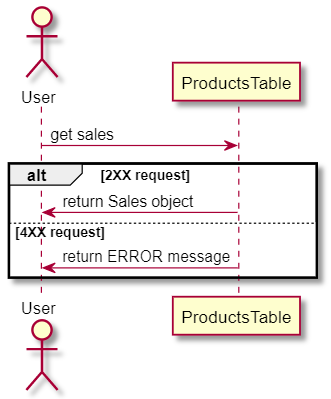


Рисунок 8 – Диаграмма последовательности get sales

При выполнении прецедента change sales (создании/изменении/удалении продажи), пользователь отправляет соответствующий запрос, в тело которого передает объект в формате JSON, соответствующий экземпляру класса Sale, который, при методе POST, будет помещен в новую строку таблицы SaleTable. При методе PUT данные в строке данной таблицы будут изменены, при методе DELETE удалены.

Если пользователь передал верные параметры в запросе, то операция будет выполнена, а пользователю будет возвращено соответствующее подтверждающее сообщение, в ином случае пользователь получит уведомление об ошибке.

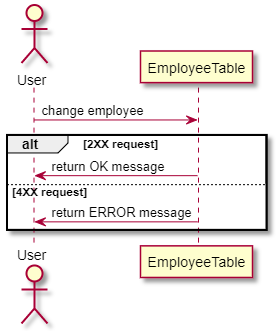


Рисунок 9 – Диаграмма последовательности change sales

При выполнении прецедента get compositions, пользователь отправляет запрос на соответствующий адрес, обрабатывая который, сервер возвращает пользователю JSON объект, соответствующий экземпляру класса Composition. В случае, если обработка запроса не была завершена успешна, пользователю будет возвращено уведомление об ошибке.

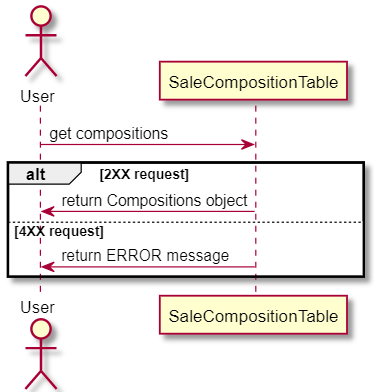


Рисунок 10 – Диаграмма последовательности get compositions

При выполнении прецедента change compositions (создании/изменении/удалении состава продажи), пользователь отправляет соответствующий запрос, в тело которого передает объект в формате JSON, соответствующий экземпляру класса Composition, который, при методе POST, будет помещен в новую строку таблицы SaleCompositionTable. При методе PUT данные в строке данной таблицы будут изменены, при методе DELETE удалены.

Если пользователь передал верные параметры в запросе, то операция будет выполнена, а пользователю будет возвращено соответствующее подтверждающее сообщение, в ином случае пользователь получит уведомление об ошибке.

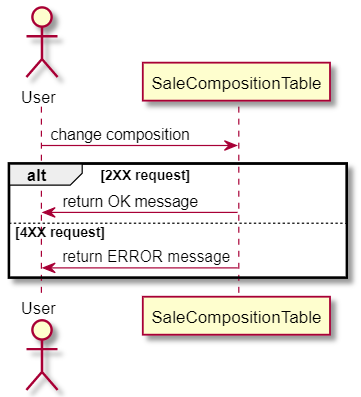


Рисунок 11 – Диаграмма последовательности change compositions

При выполнении прецедента get types, пользователь отправляет запрос на соответствующий адрес, обрабатывая который, сервер возвращает пользователю JSON объект, соответствующий экземпляру класса Type. В случае, если обработка запроса не была завершена успешна, пользователю будет возвращено уведомление об ошибке.

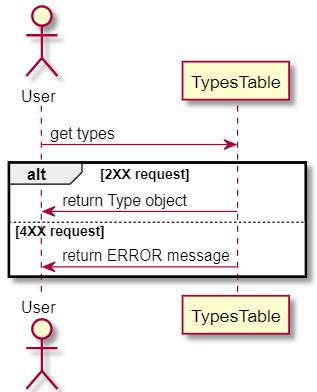


Рисунок 12 – Диаграмма последовательности get types

При выполнении прецедента change types (создании/изменении/удалении типа товара), пользователь отправляет соответствующий запрос, в тело которого передает объект в формате JSON, соответствующий экземпляру класса Type, который, при методе POST, будет помещен в новую строку таблицы TypesTable. При методе PUT данные в строке данной таблицы будут изменены, при методе DELETE удалены.

Если пользователь передал верные параметры в запросе, то операция будет выполнена, а пользователю будет возвращено соответствующее подтверждающее сообщение, в ином случае пользователь получит уведомление об ошибке.

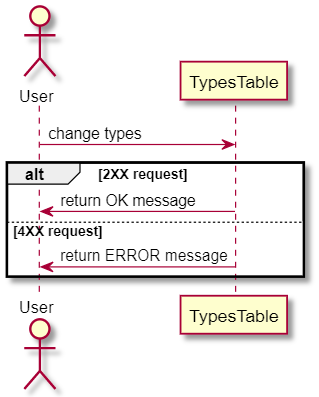


Рисунок 13 – Диаграмма последовательности change types

При выполнении прецедента get posts, пользователь отправляет запрос на соответствующий адрес, обрабатывая который, сервер возвращает пользователю JSON объект, соответствующий экземпляру класса Post. В случае, если обработка запроса не была завершена успешна, пользователю будет возвращено уведомление об ошибке.

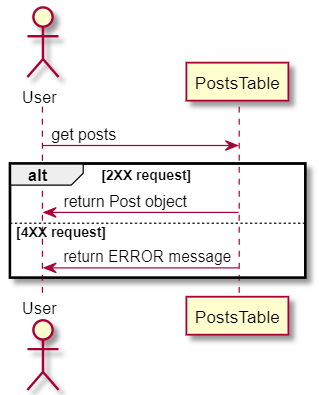


Рисунок 14 – Диаграмма последовательности get posts

При выполнении прецедента change posts (создании/изменении/удалении должности сотрудника), пользователь отправляет соответствующий запрос, в тело которого передает объект в формате JSON, соответствующий экземпляру класса Post, который, при методе POST, будет помещен в новую строку таблицы PostsTable. При методе PUT данные в строке данной таблицы будут изменены, при методе DELETE удалены.

Если пользователь передал верные параметры в запросе, то операция будет выполнена, а пользователю будет возвращено соответствующее подтверждающее сообщение, в ином случае пользователь получит уведомление об ошибке.

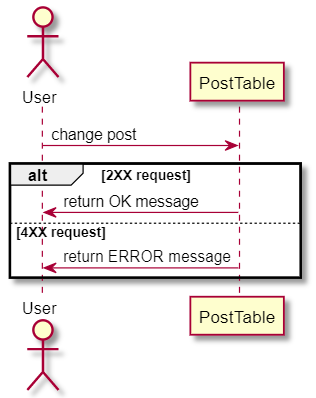


Рисунок 15 – Диаграмма последовательности change post

## 5 Описание моделей

Модель Employee представляет собой сотрудника, содержит в себе имя, id должности и зарплату.

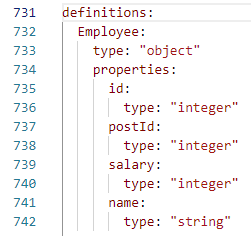


Рисунок 16 – Модель Employee

Модель Product представляет собой продукт, содержит в себе название, id типа продукта и цену.

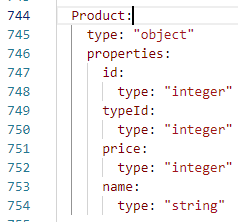


Рисунок 17 – Модель Product

Модель Sale представляет собой продажу. Содержит в себе id сотрудника, который совершил продажу и дату продажи.

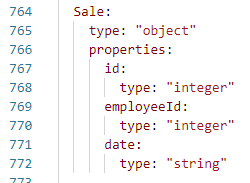


Рисунок 18 – Модель Sale

Модель SaleComposition представляет собой промежуточную таблицу между Sale и Product, служит для хранения состава продажи. Содержит в себе id продажи, id продукта и количество продукта.

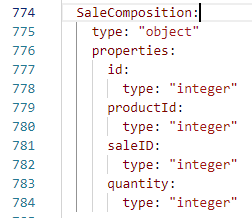


Рисунок 19 – Модель SaleComposition

Модель Type представляет собой тип продукта. Содержит в себе имя.

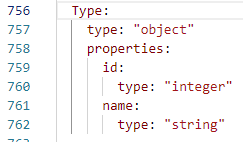


Рисунок 20 – Модель Type

Модель Post представляет собой должность, которую занимает сотрудник. Содержит в себе название должности.

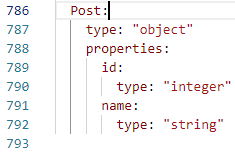


Рисунок 21 – Модель Post

## 6 Описание запросов

При запросе типа GET по пути /products, сервер ответит пользователю массивом объектов, соответствующих модели Product в формате JSON.

При запросе типа POST по пути /products, с указанием в теле запроса данных нового продукта в формате JSON, будет создана новая запись в базе данных, то есть будет создана новая строка в таблице с данными, переданными пользователем.



Рисунок 22 – / products

При запросе типа GET по пути /products/id, сервер ответит пользователю объектом, соответствующим модели Product в формате JSON с соответствующим id.

При запросе пользователя DELETE/ products /id, будет удалена запись с соответствующим id.

При запросе типа PUT по пути /products/id, с указанием в теле запроса новых данных для товара в формате JSON, будет изменена существующая запись с id, соответствующим этому объекту в базе данных.



Рисунок 23 – /products/id

При запросе типа GET по пути /products/ type={typeid}, сервер ответит пользователю массивом объектов, соответствующих модели Product с типом, соответствующим переданному параметру typeid в формате JSON.



Рисунок 24 – / products/type={typeid}

При запросе типа GET по пути /employee, сервер ответит пользователю массивом объектов, соответствующих модели Employee в формате JSON.

При запросе типа POST по пути / employee, с указанием в теле запроса данных нового сотрудника в формате JSON, будет создана новая запись в базе данных, то есть будет создана новая строка в таблице с данными, переданными пользователем.

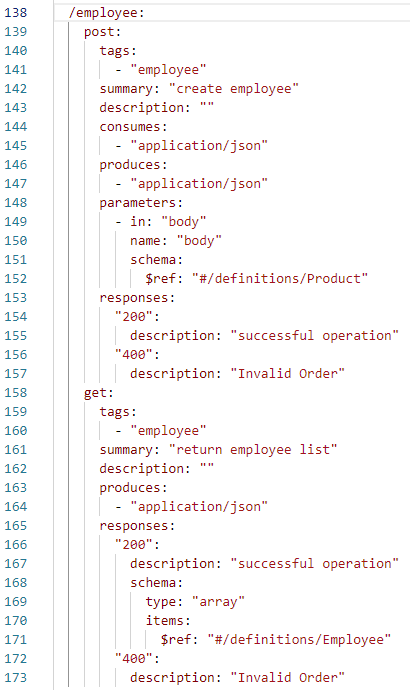


Рисунок 25 – / employee

При запросе типа GET по пути /employee/id, сервер ответит пользователю объектом, соответствующим модели Employee в формате JSON с соответствующим id.

При запросе пользователя DELETE/ employee /id, будет удалена запись с соответствующим id.

При запросе типа PUT по пути / employee /id, с указанием в теле запроса новых данных сотрудника в формате JSON, будет изменена существующая запись с id, соответствующим этому объекту в базе данных.



Рисунок 26 – / employee /id

При запросе типа GET по пути /sales, сервер ответит пользователю массивом объектов, соответствующих модели Sale в формате JSON.

При запросе типа POST по пути /sales, с указанием в теле запроса данных новой продажи в формате JSON, будет создана новая запись в базе данных, то есть будет создана новая строка в таблице с данными, переданными пользователем.

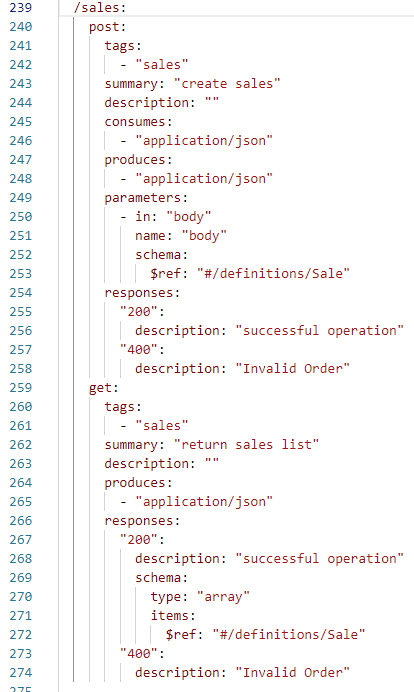


Рисунок 27 – / sales

При запросе типа GET по пути / sales /id, сервер ответит пользователю объектом, соответствующим модели Sale в формате JSON с соответствующим id.

При запросе пользователя DELETE/ sales /id, будет удалена запись с соответствующим id.

При запросе типа PUT по пути / sales /id, с указанием в теле запроса новых данных для товара в формате JSON, будет изменена существующая запись с id, соответствующим этому объекту в базе данных.

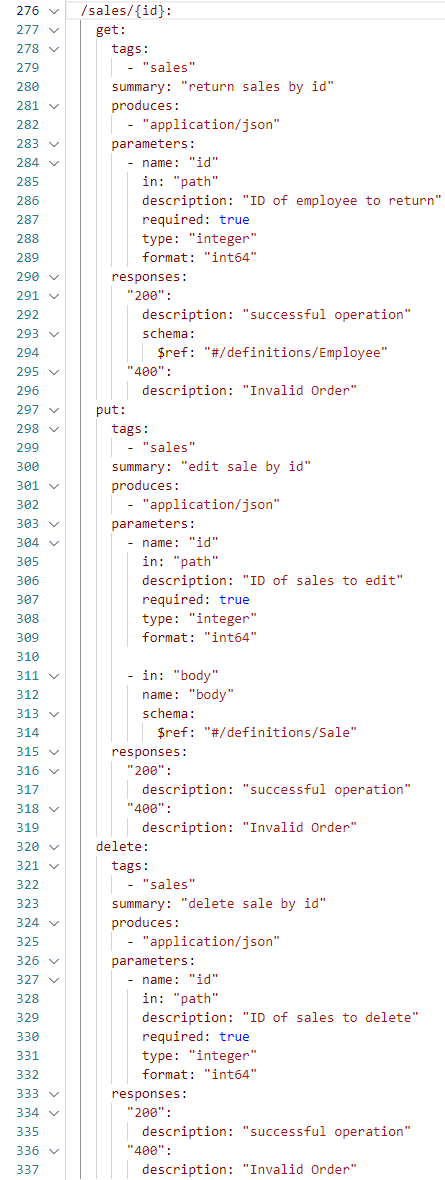


Рисунок 28 – / sales /id

При запросе типа GET по пути / sales/id/composition, сервер ответит пользователю массивом объектов, соответствующих модели SaleComposition с id продажи, соответствующим переданному параметру id, в формате JSON.



Рисунок 29 – / sales /id/composition

При запросе типа GET по пути /composition, сервер ответит пользователю массивом объектов, соответствующих модели SaleComposition в формате JSON.

При запросе типа POST по пути / composition, с указанием в теле запроса данных нового состава продажи в формате JSON, будет создана новая запись в базе данных, то есть будет создана новая строка в таблице с данными, переданными пользователем.



Рисунок 30 – / composition

При запросе типа GET по пути / composition /id, сервер ответит пользователю объектом, соответствующим модели SaleComposition в формате JSON с соответствующим id.

При запросе пользователя DELETE/ composition /id, будет удалена запись с соответствующим id.

При запросе типа PUT по пути / composition /id, с указанием в теле запроса новых данных для товара в формате JSON, будет изменена существующая запись с id, соответствующим этому объекту в базе данных.

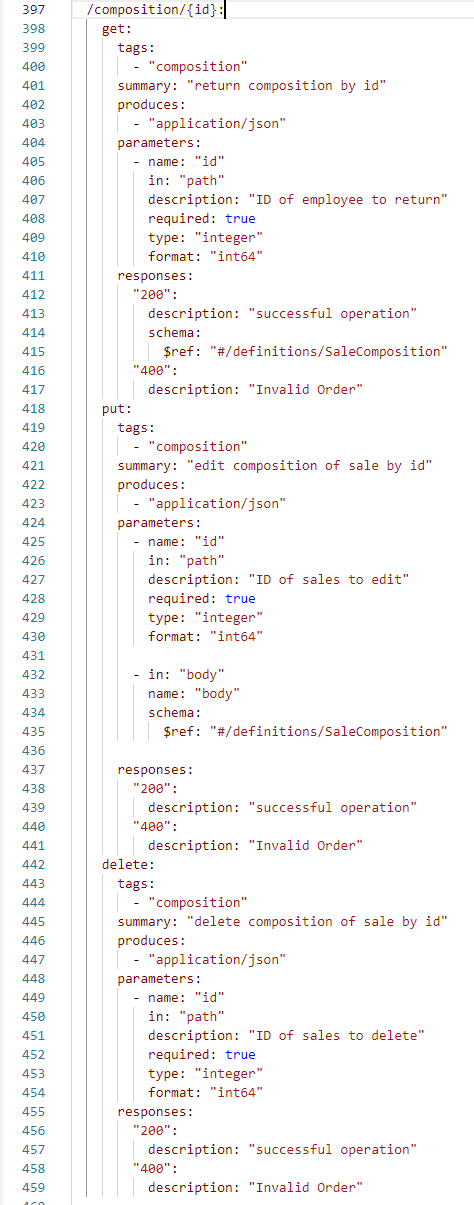


Рисунок 31 – / composition /id

При запросе типа GET по пути / composition / price<{price}, сервер ответит пользователю массивом объектов, соответствующих модели SaleComposition с суммой продажи меньше чем переданный параметр price, в формате JSON.



Рисунок 32 – / composition /price<{price}

При запросе типа GET по пути / composition / price>{price}, сервер ответит пользователю массивом объектов, соответствующих модели SaleComposition с суммой продажи больше чем переданный параметр price, в формате JSON.



Рисунок 33 – / composition /price>{price}

При запросе типа GET по пути / composition / price={price}, сервер ответит пользователю массивом объектов, соответствующих модели SaleComposition с суммой продажи равной переданному параметру price, в формате JSON.

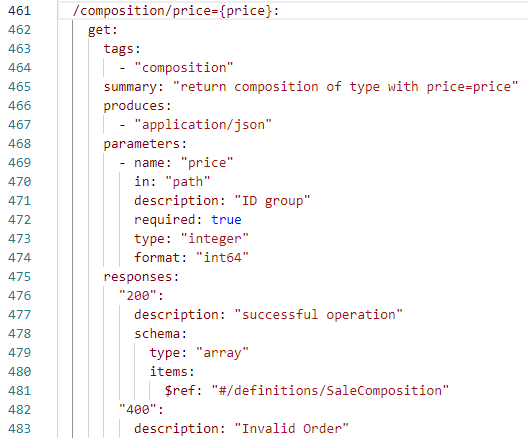


Рисунок 34 – / composition /price={price}

При запросе типа GET по пути /types, сервер ответит пользователю массивом объектов, соответствующих модели Type в формате JSON.

При запросе типа POST по пути / types, с указанием в теле запроса данных нового типа товара в формате JSON, будет создана новая запись в базе данных, то есть будет создана новая строка в таблице с данными, переданными пользователем.



Рисунок 35– / types

При запросе типа GET по пути / types /id, сервер ответит пользователю объектом, соответствующим модели Type в формате JSON с соответствующим id.

При запросе пользователя DELETE/ types /id, будет удалена запись с соответствующим id.

При запросе типа PUT по пути / types /id, с указанием в теле запроса новых данных для товара в формате JSON, будет изменена существующая запись с id, соответствующим этому объекту в базе данных.

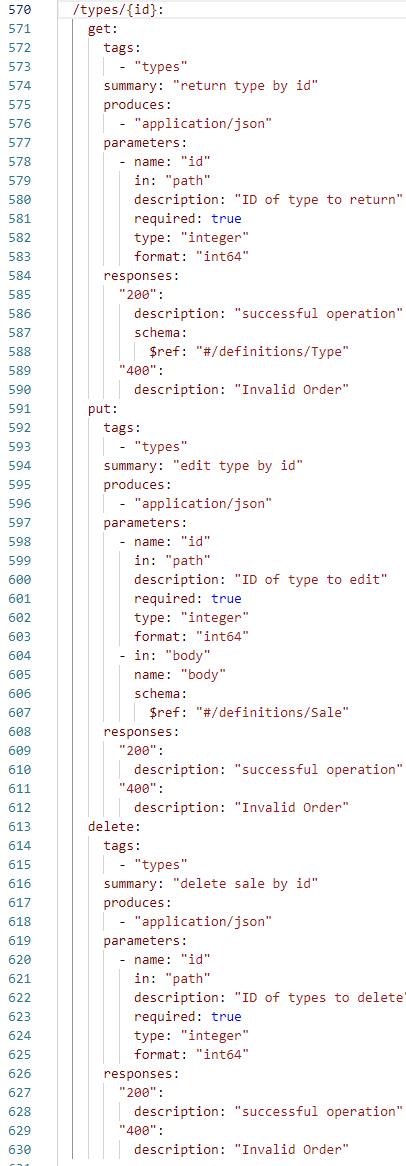


Рисунок 36 – / types /id

При запросе типа GET по пути / posts, сервер ответит пользователю массивом объектов, соответствующих модели Post в формате JSON.

При запросе типа POST по пути / posts, с указанием в теле запроса данных новой должности в формате JSON, будет создана новая запись в базе данных, то есть будет создана новая строка в таблице с данными, переданными пользователем.



Рисунок 37 – / posts

При запросе типа GET по пути / posts /id, сервер ответит пользователю объектом, соответствующим модели Post в формате JSON с соответствующим id.

При запросе пользователя DELETE/ posts /id, будет удалена запись с соответствующим id.

При запросе типа PUT по пути / posts /id, с указанием в теле запроса новых данных должности в формате JSON, будет изменена существующая запись с id, соответствующим этому объекту в базе данных.

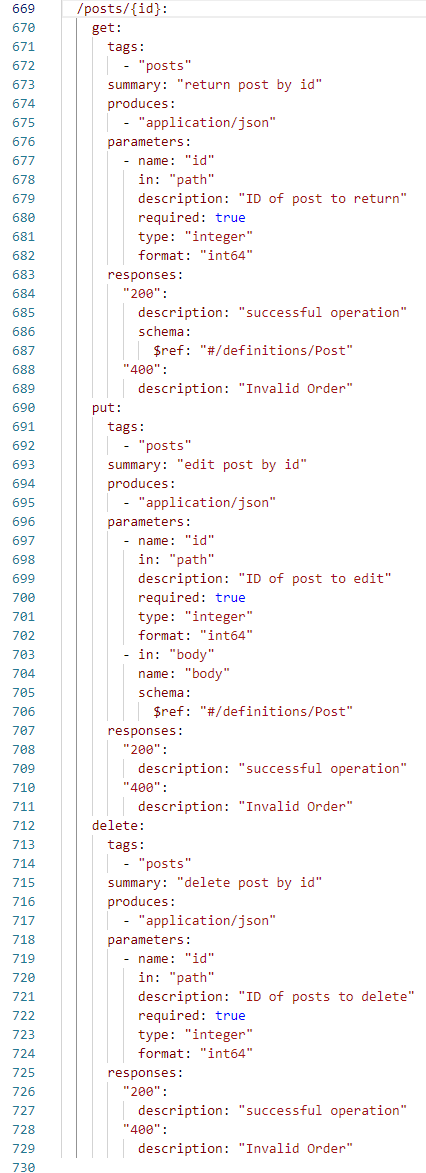


Рисунок 38 – / posts /id

# Заключение

В ходе выполнения работы были получены навыки разработки REST API , а так же навыки взаимодействия с инструментом swagger. Так же были получены навыки построения uml диаграмм.

В курсовой работе был REST API магазина аксессуаров для мобильных телефонов.

# Библиографический список

1. Kotlin exposed [Электронный ресурс] / Режим доступа: [https://github.com/JetBrains/Exposed](https://github.com/JetBrains/Exposed%20)  (Exposed)
2. plantuml [Электронный ресурс] / Режим доступа: [https://plantuml.com/ru/use-case-diagram](https://plantuml.com/ru/use-case-diagram%20)  (UML)
3. Википедия. Свободная энциклопедия [Электронный ресурс] / Режим доступа: https://ru.wikipedia.org/wiki/Диаграмма\_(UML)
4. ktor [Электронный ресурс] / Режим доступа:

[https://ktor.io/](https://ktor.io/%20)  (ktor)

# Приложение А

(обязательное)

REST API

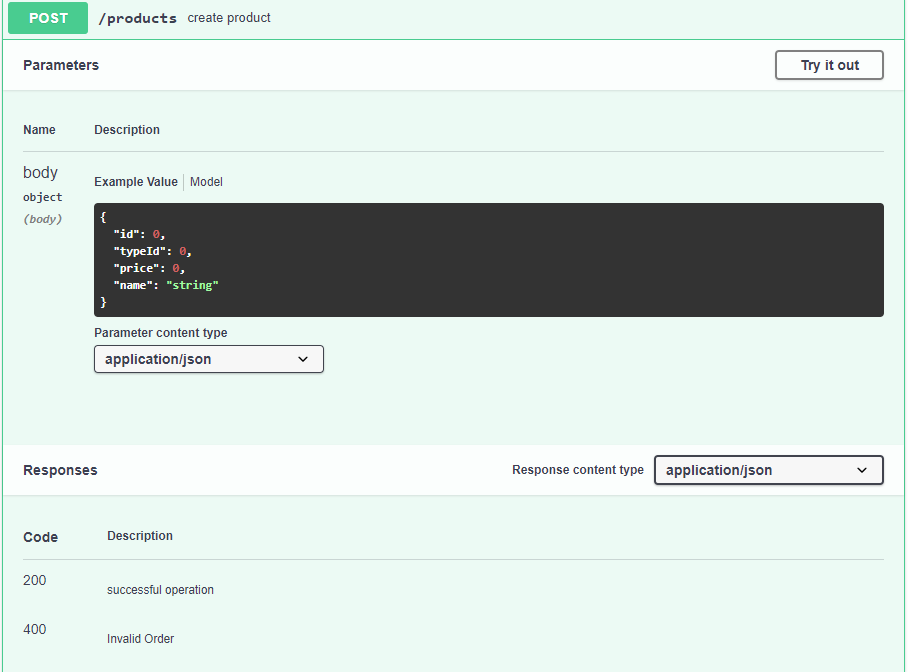


Рисунок А.1 – REST API

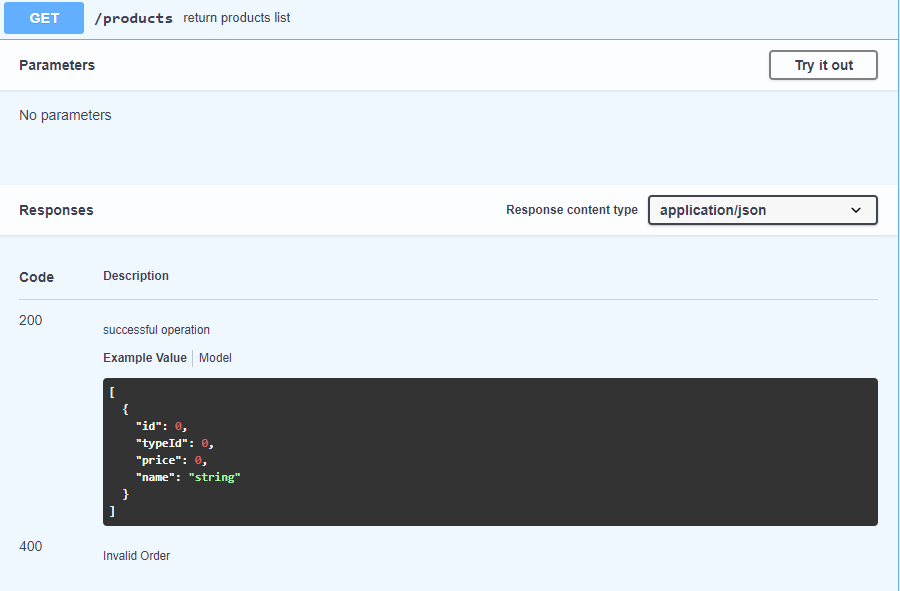


Рисунок А.2 – REST API

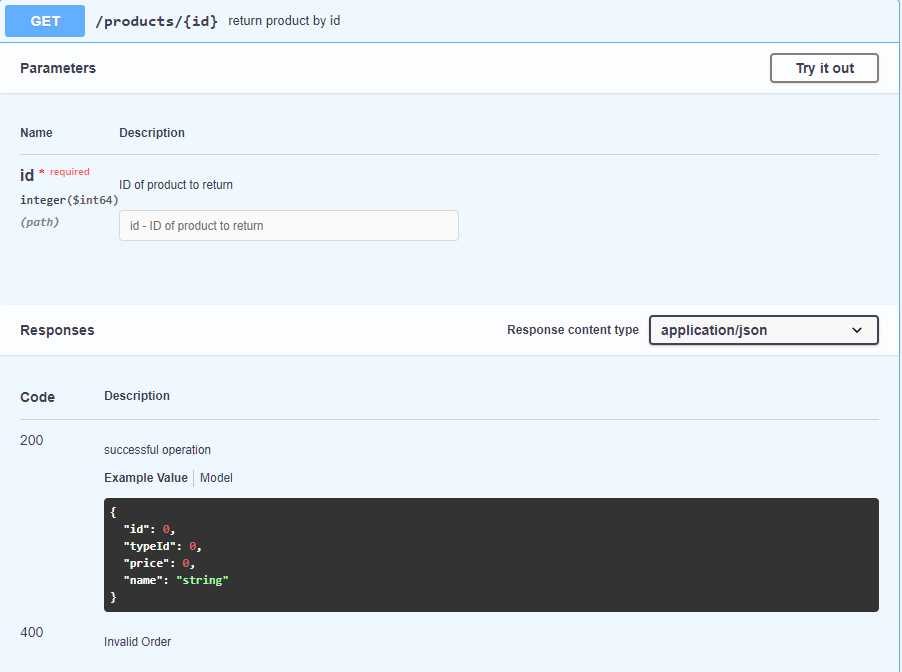


Рисунок А.3 – REST API

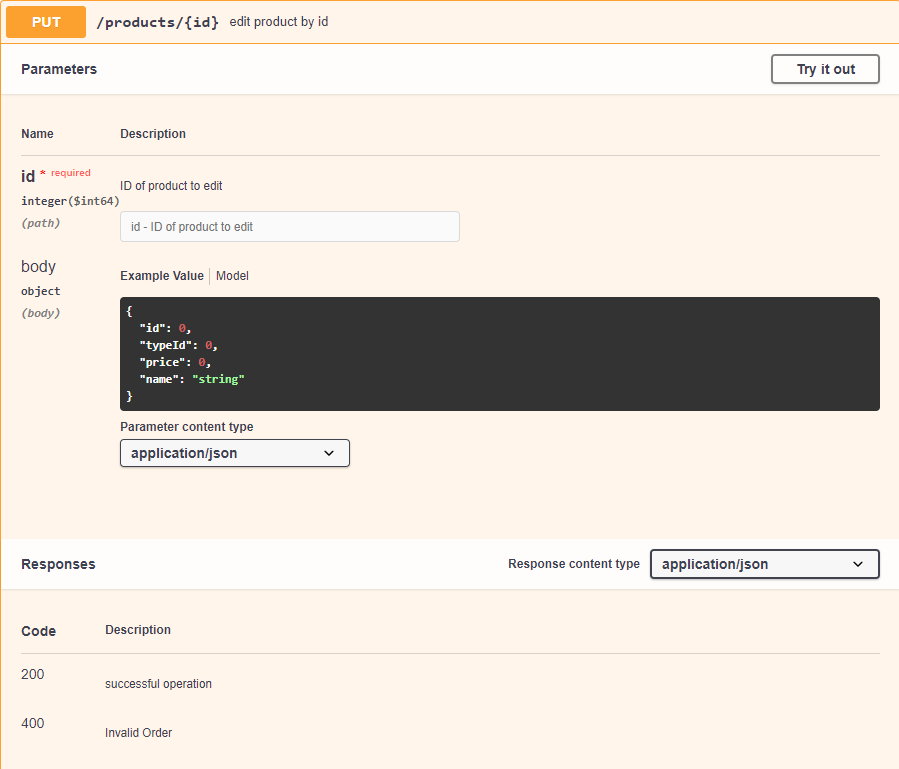


Рисунок А.4 – REST API

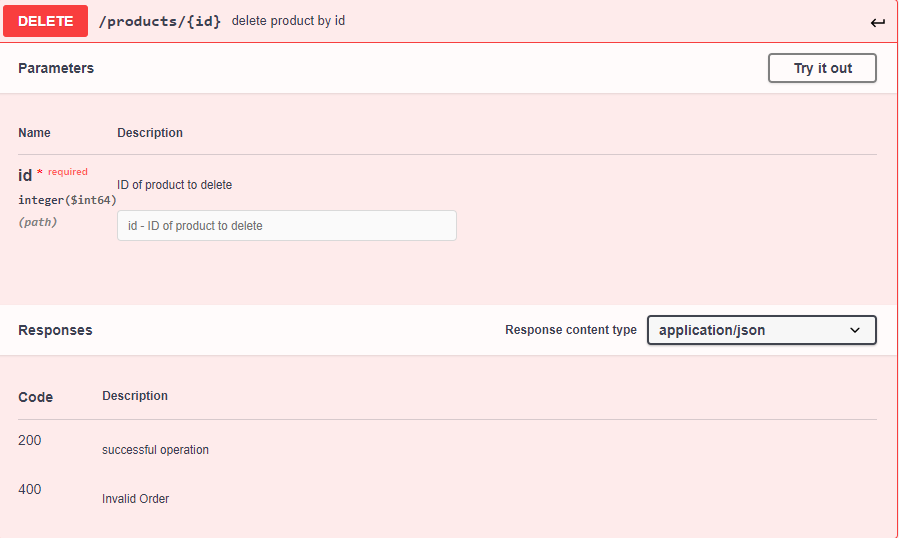


Рисунок А.5 – REST API

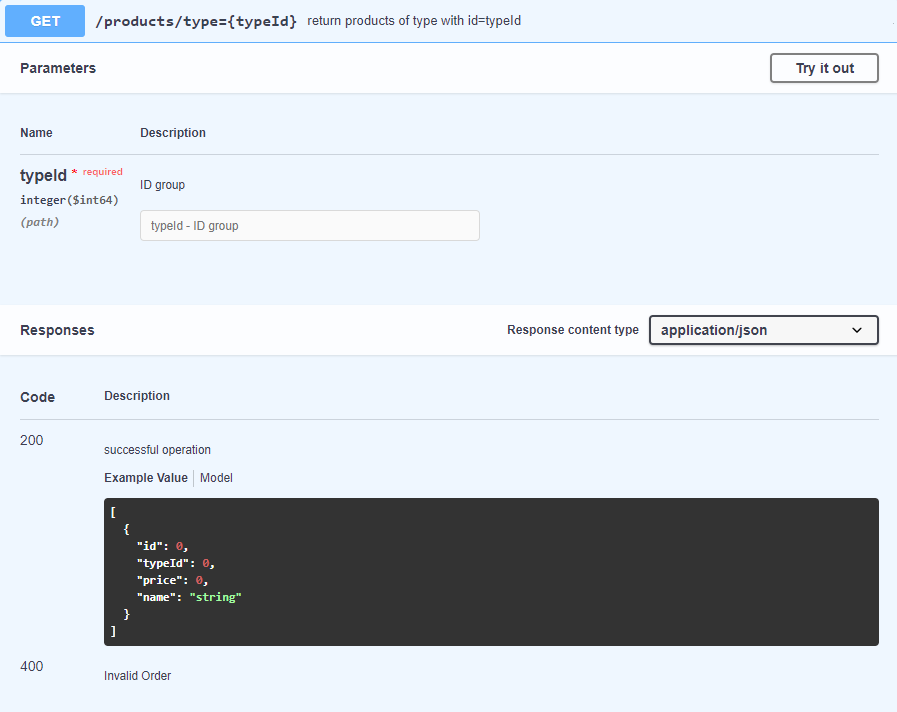


Рисунок А.6 – REST API

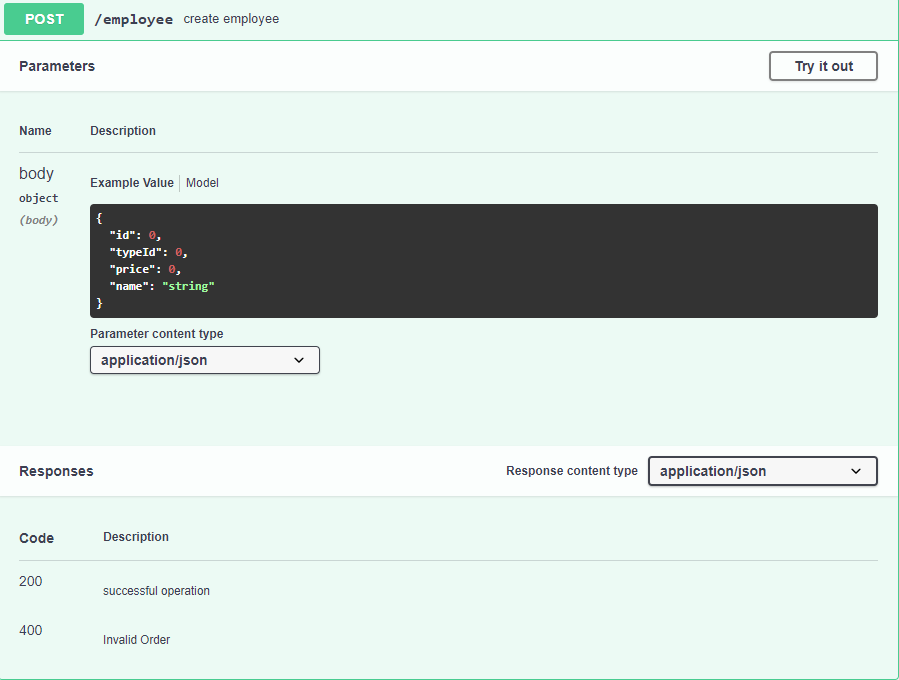


Рисунок А.7 – REST API

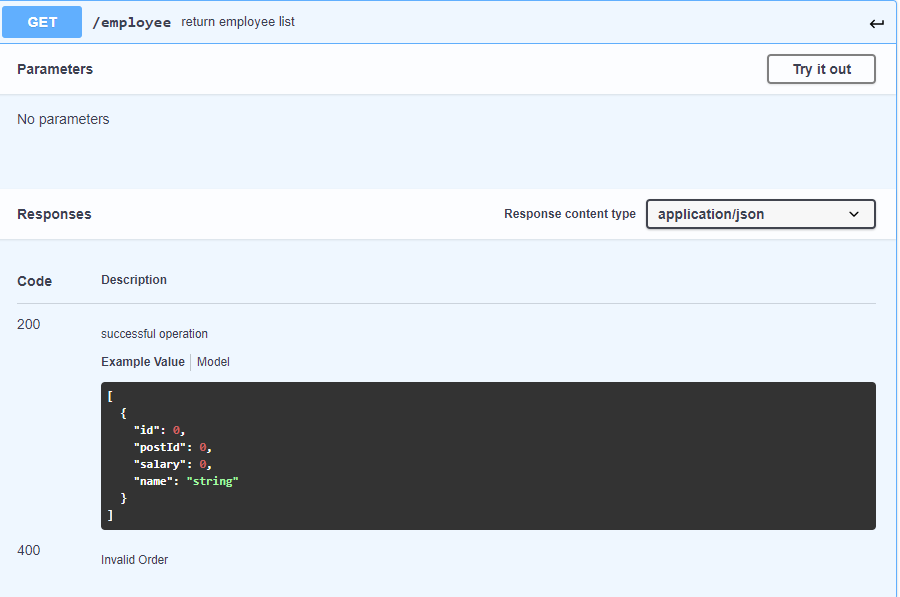


Рисунок А.8 – REST API

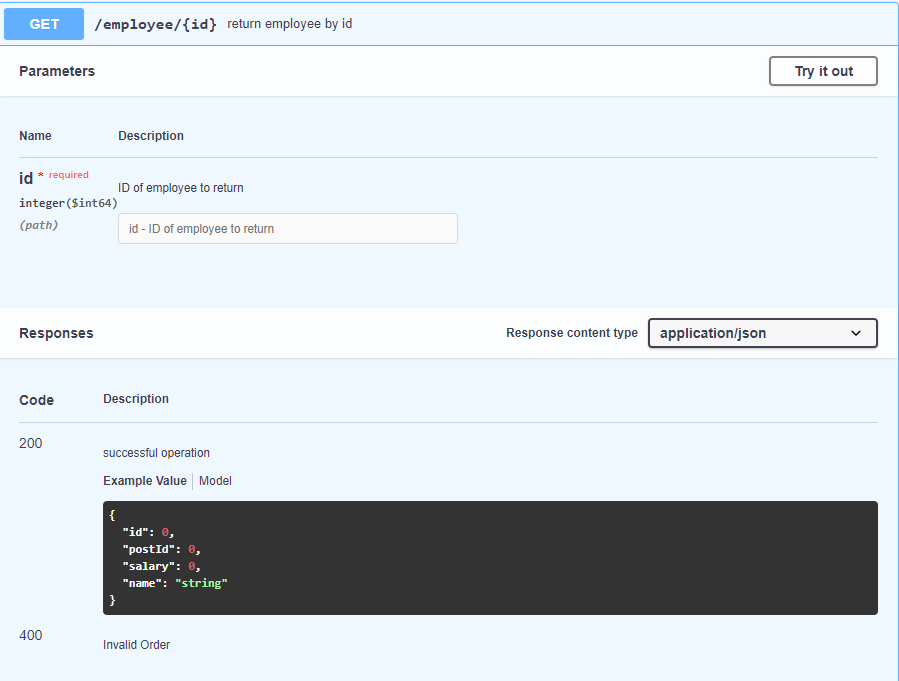


Рисунок А.9 – REST API

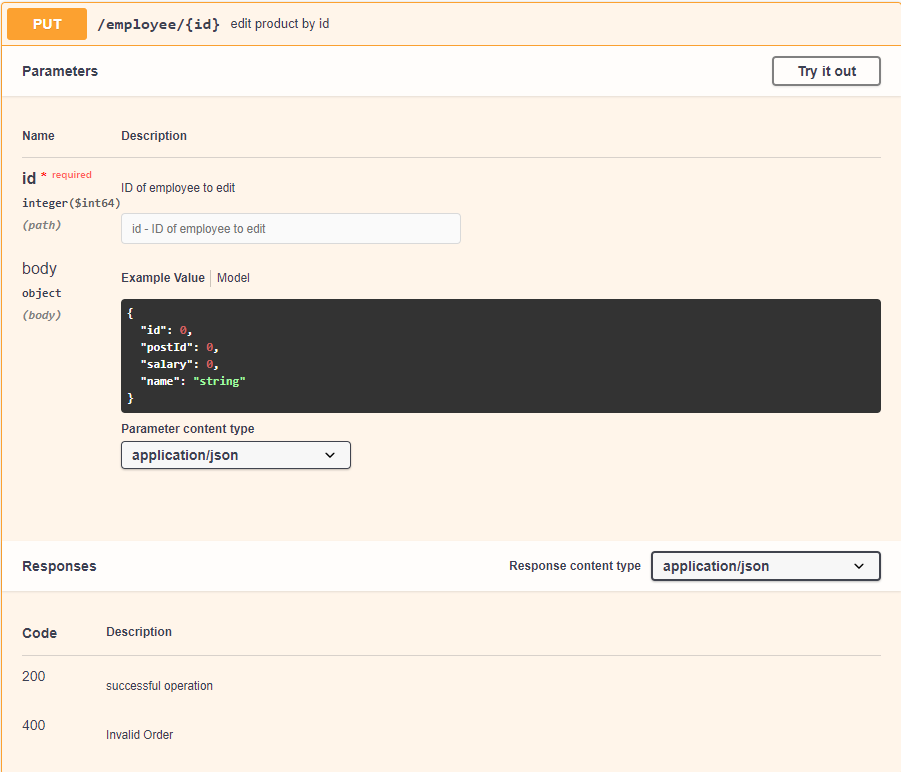


Рисунок А.10 – REST API

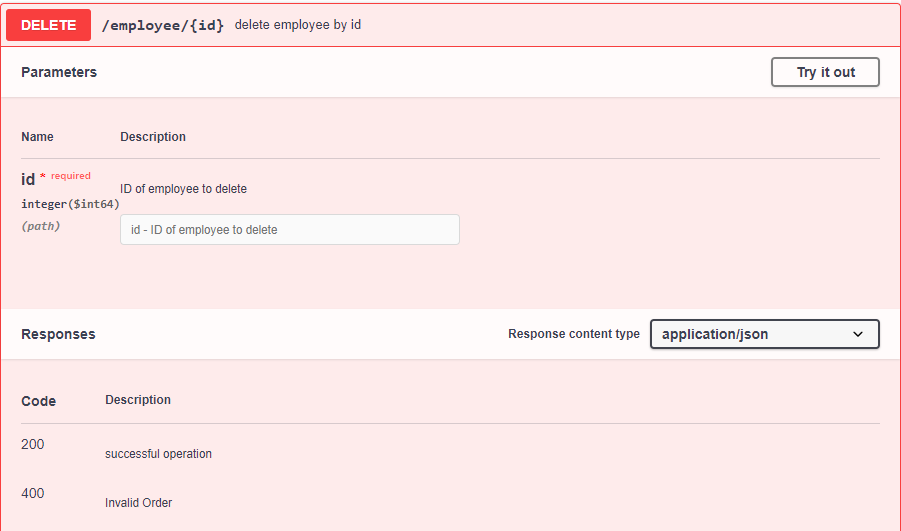


Рисунок А.11 – REST API

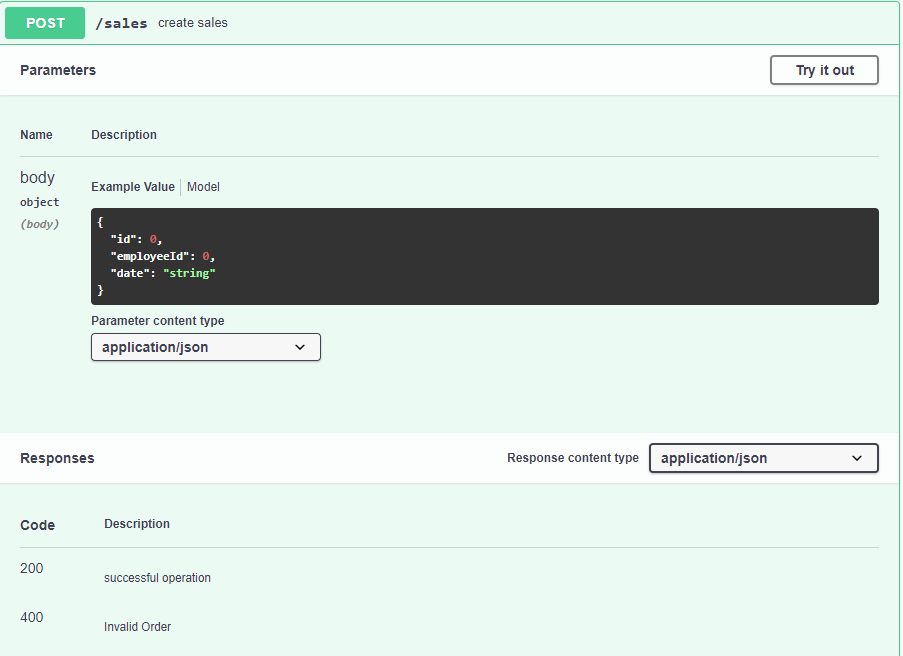


Рисунок А.12 – REST API

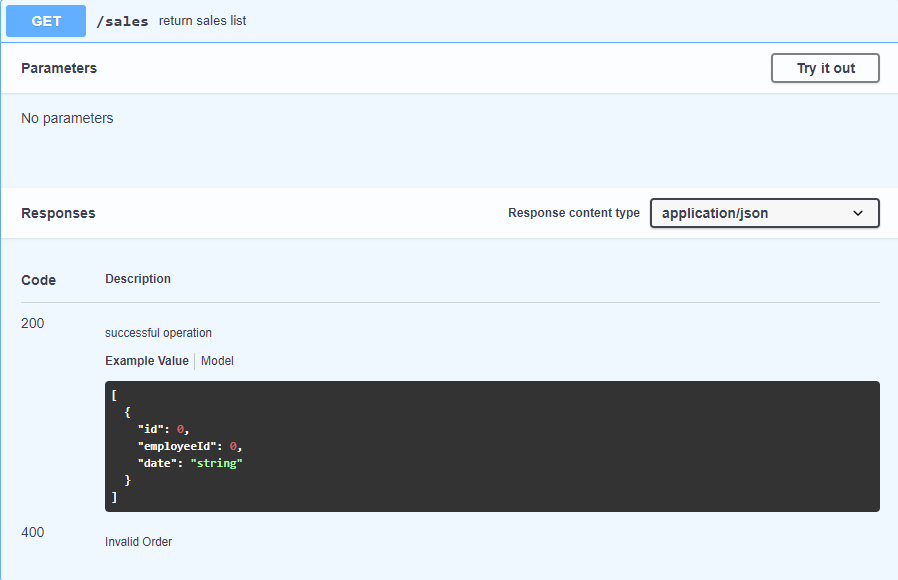


Рисунок А.13 – REST API

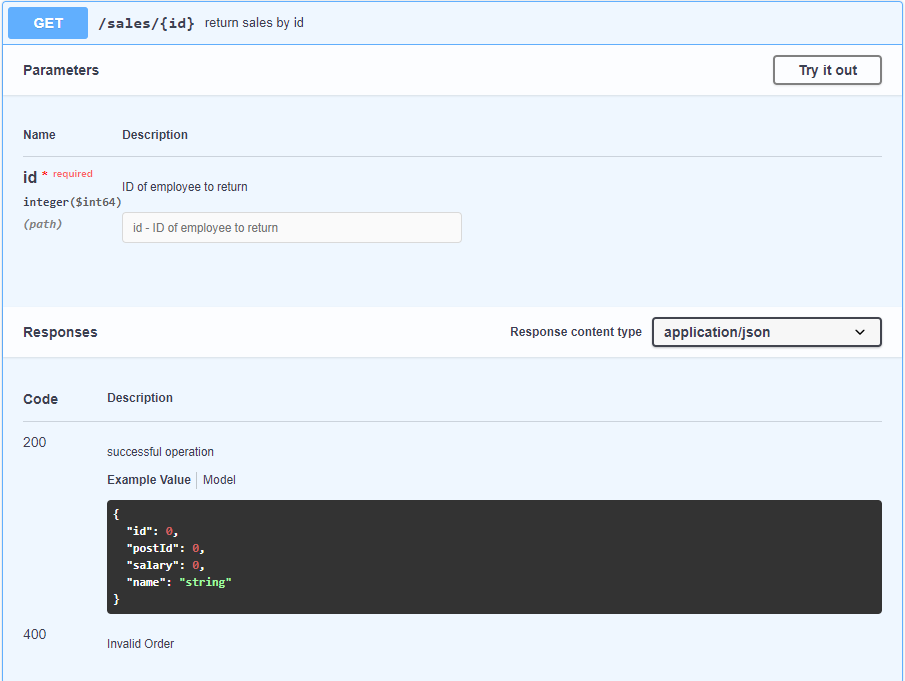


Рисунок А.14 – REST API

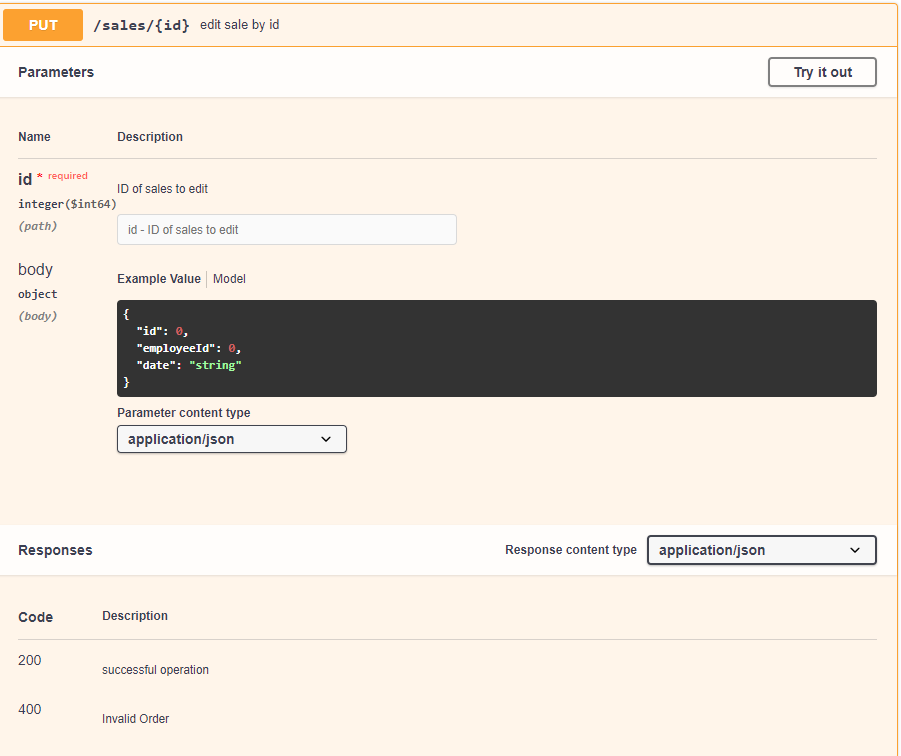


Рисунок А.15 – REST API

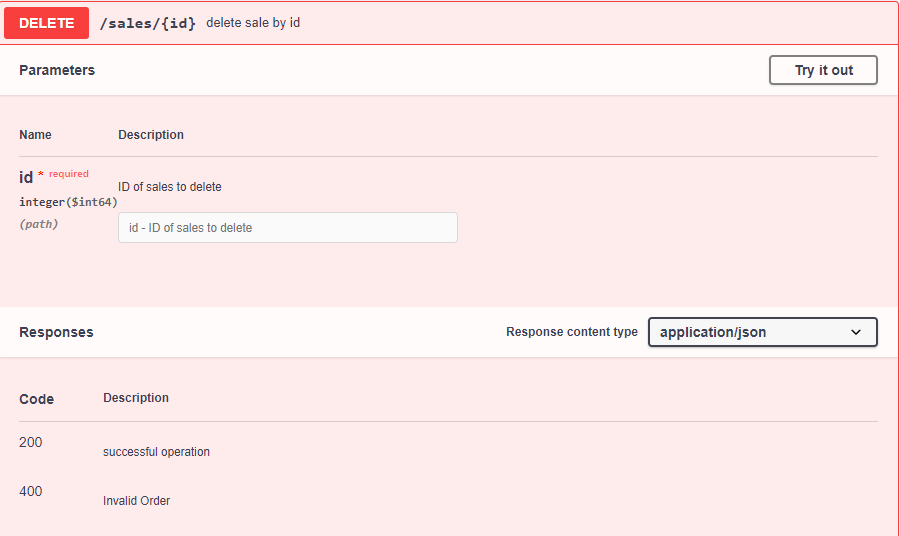


Рисунок А.16 – REST API

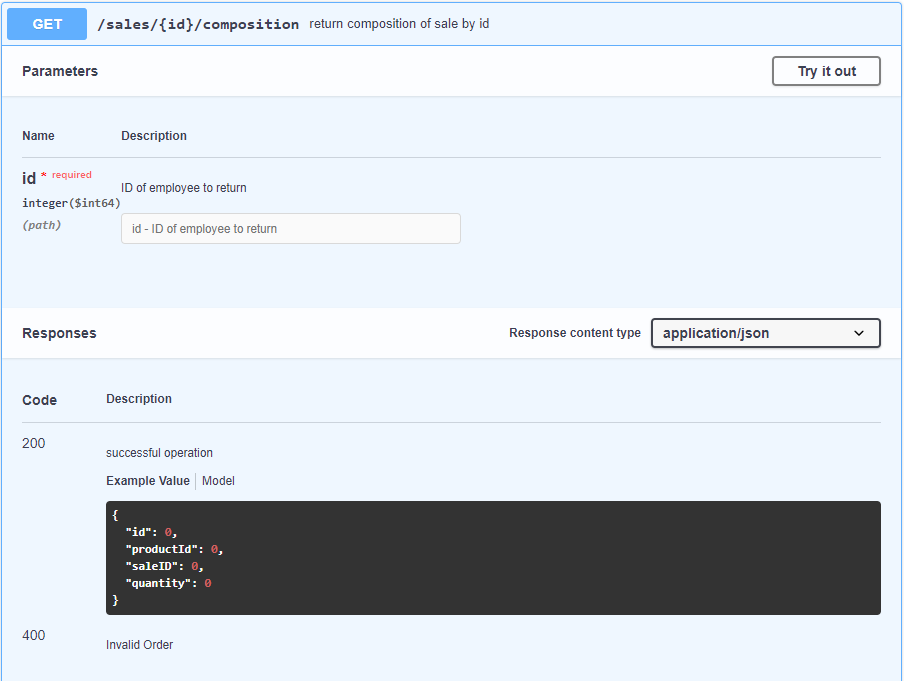


Рисунок А.17 – REST API

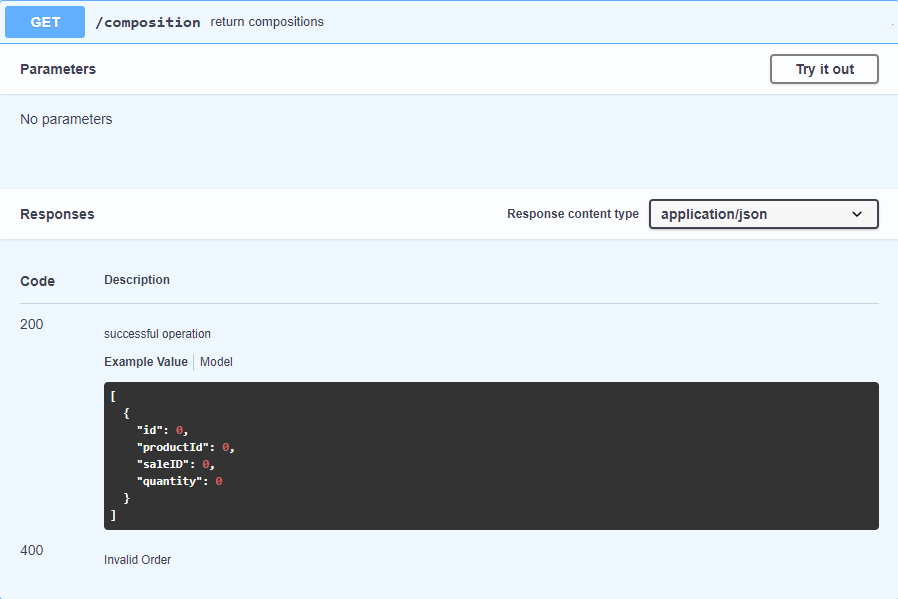


Рисунок А.18 – REST API

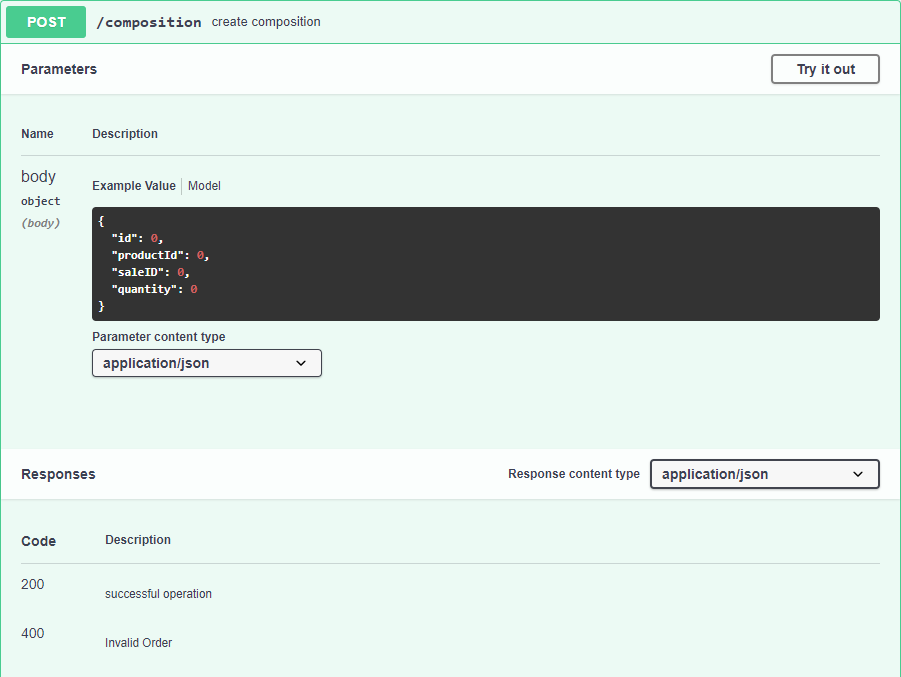


Рисунок А.19 – REST API

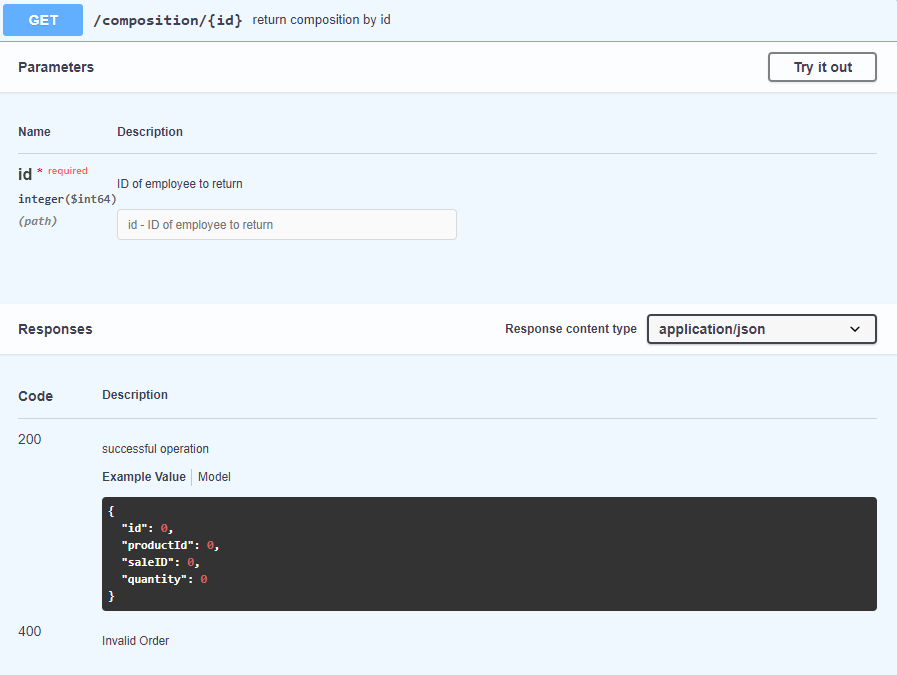


Рисунок А.20 – REST API

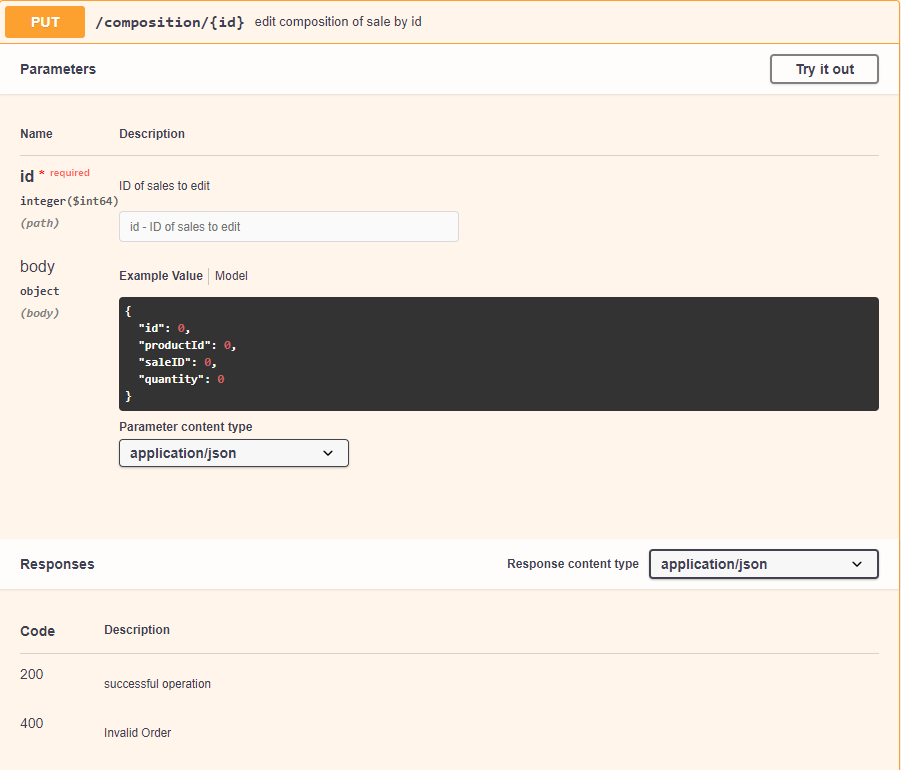


Рисунок А.21 – REST API

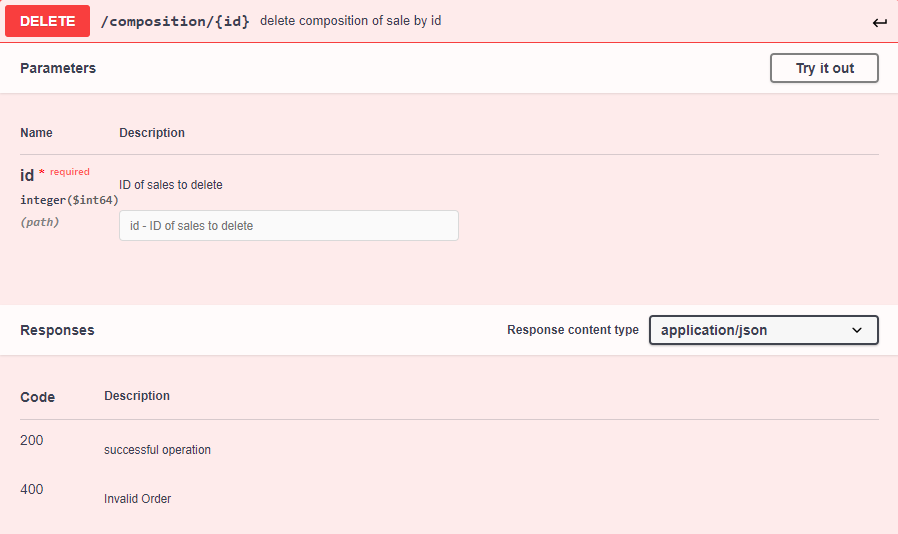


Рисунок А.22 – REST API

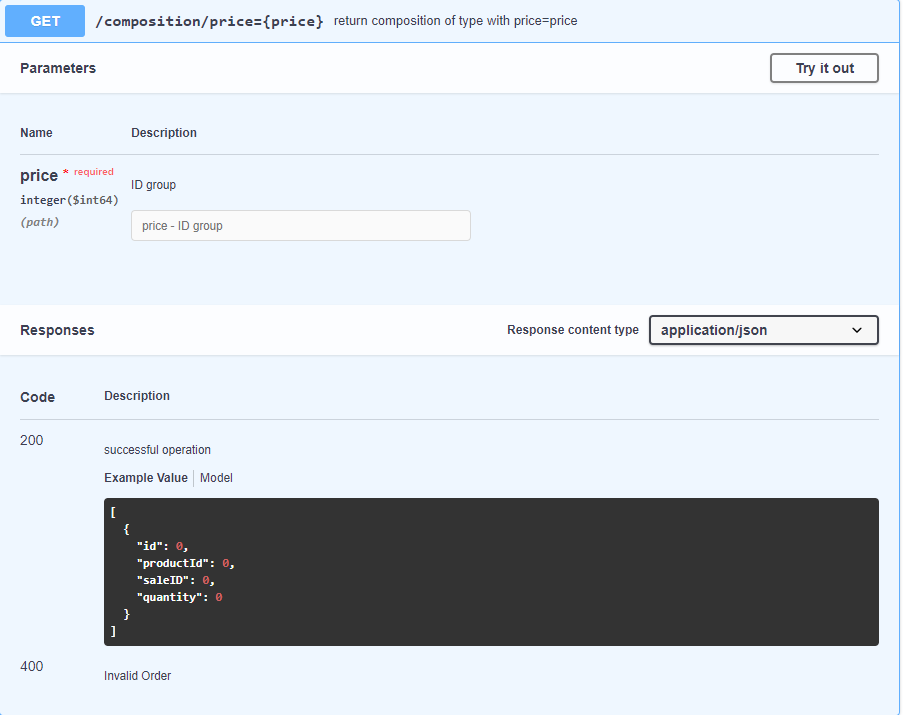


Рисунок А.23 – REST API

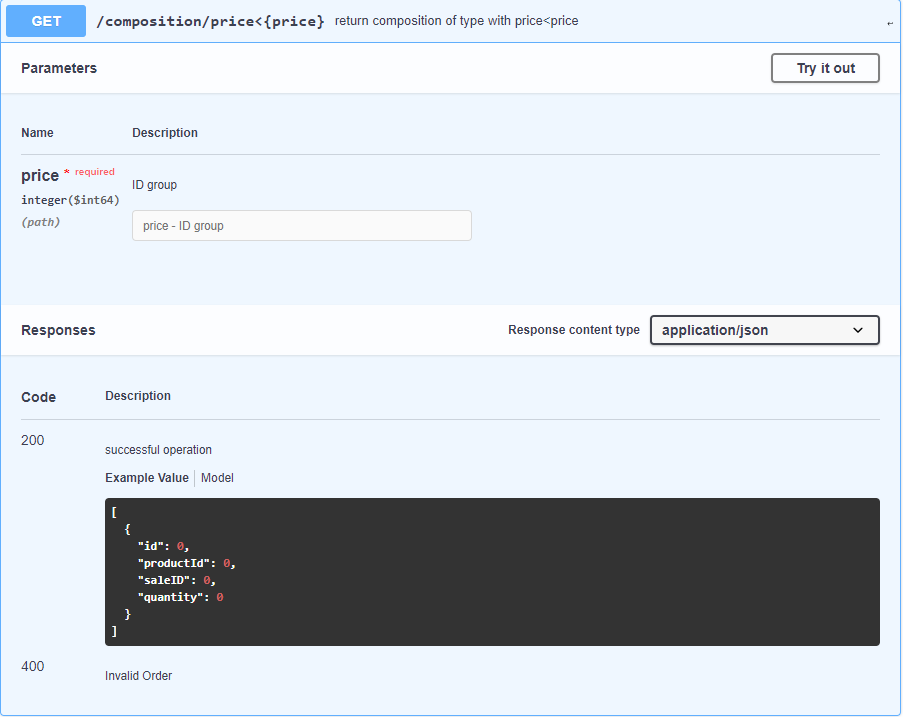


Рисунок А.24 – REST API

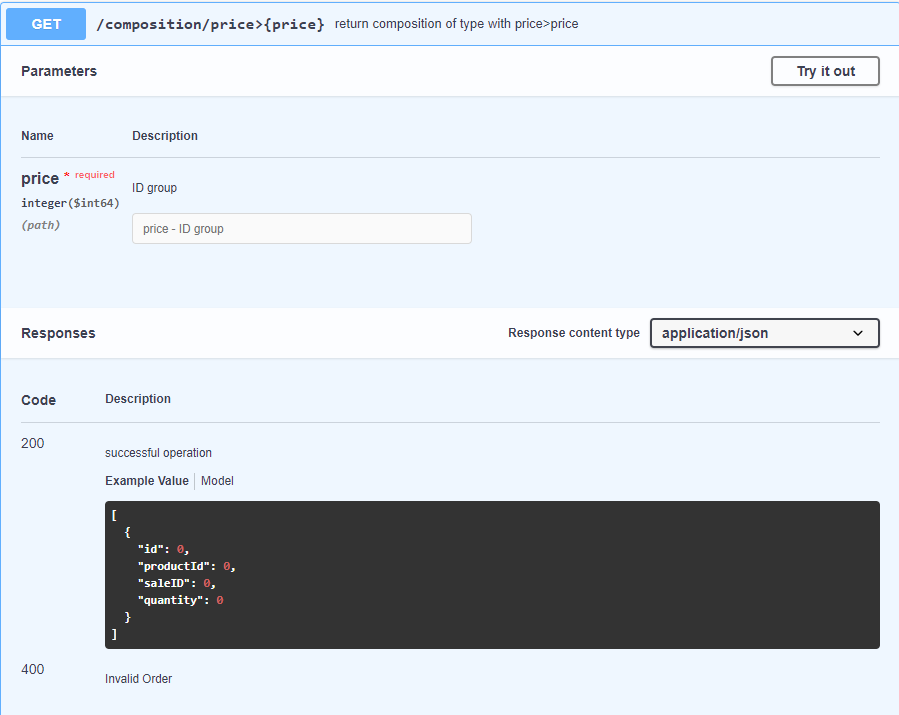


Рисунок А.25 – REST API

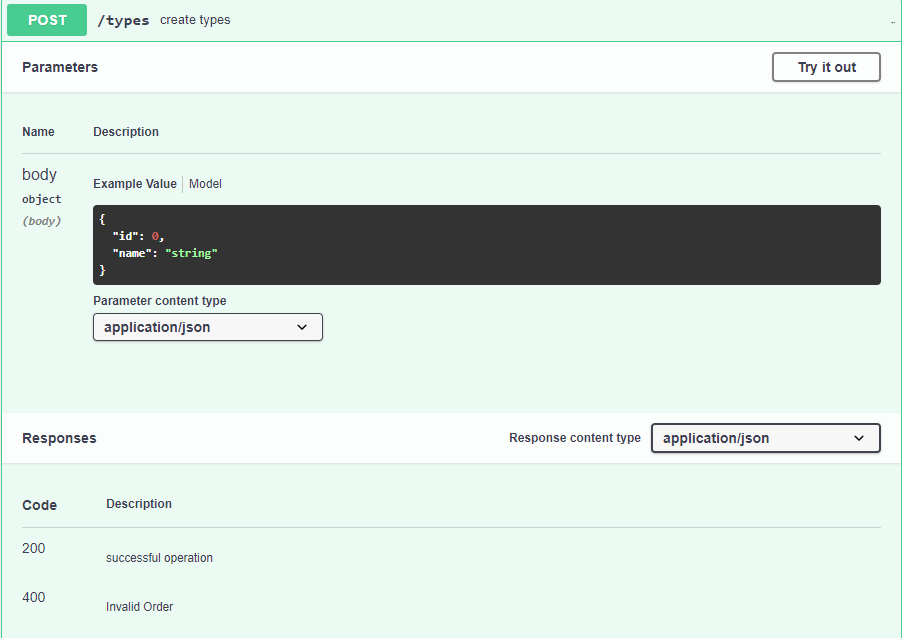


Рисунок А.26 – REST API

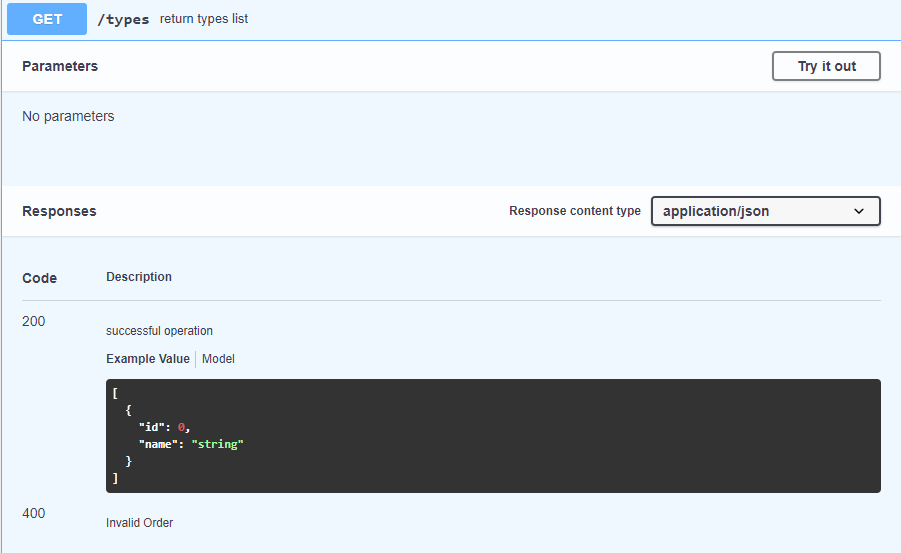


Рисунок А.27 – REST API

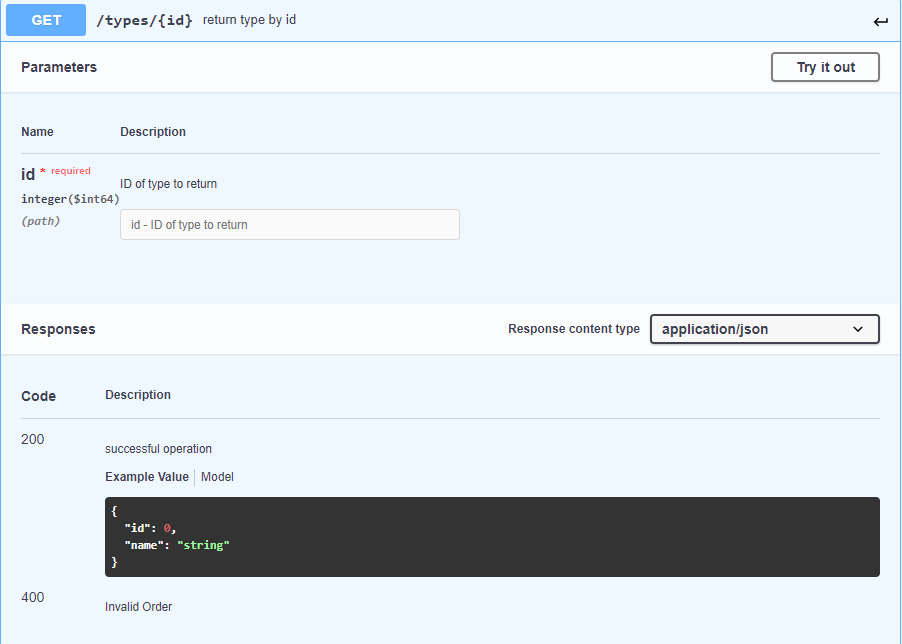


Рисунок А.28 – REST API

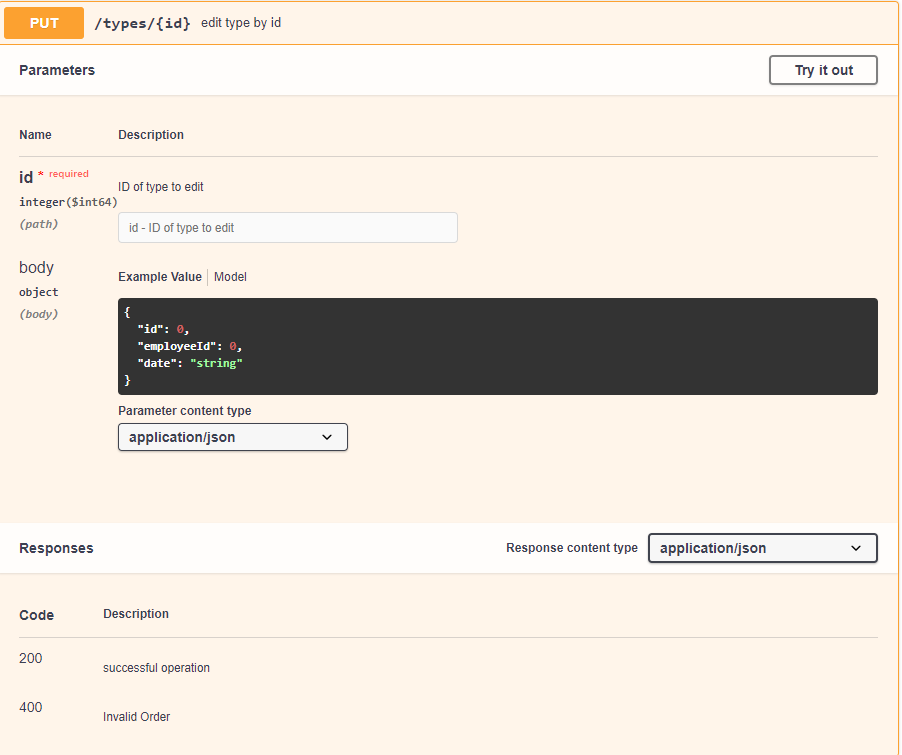


Рисунок А.29 – REST API

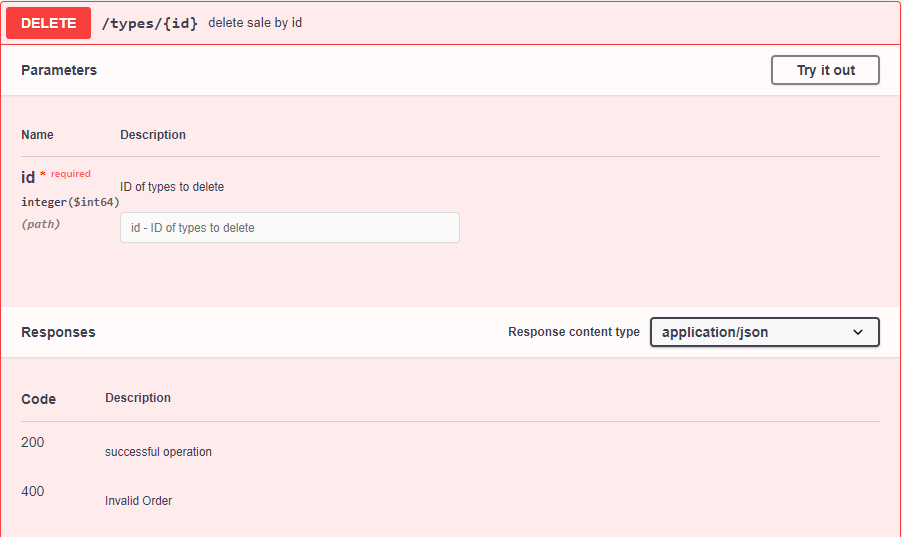


Рисунок А.30 – REST API

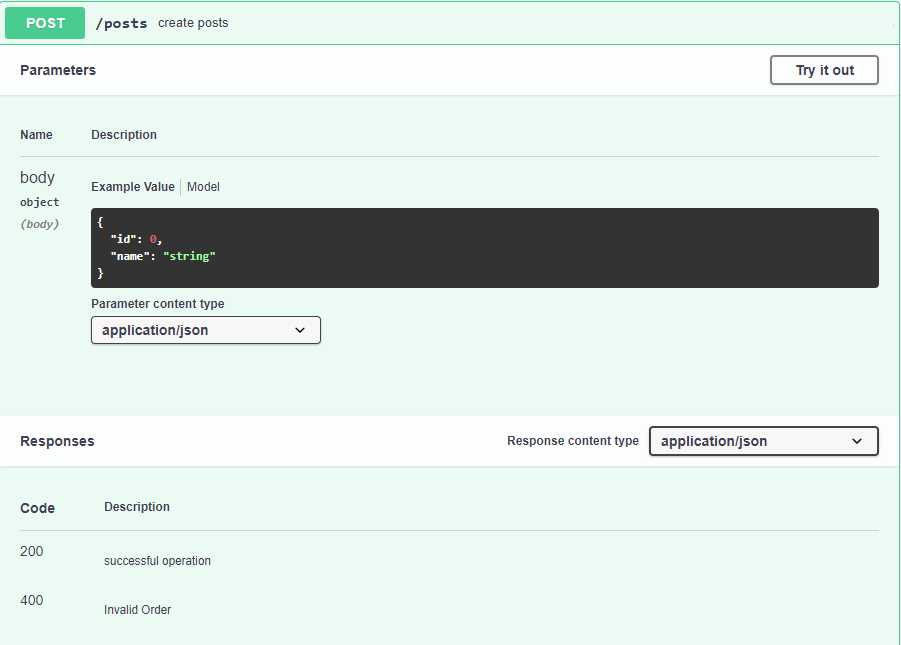


Рисунок А.31 – REST API

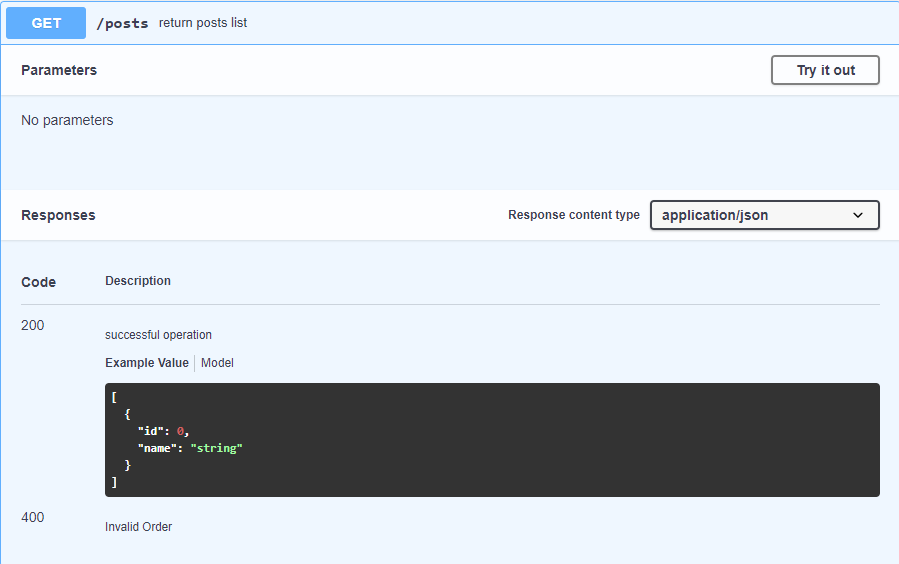


Рисунок А.32 – REST API

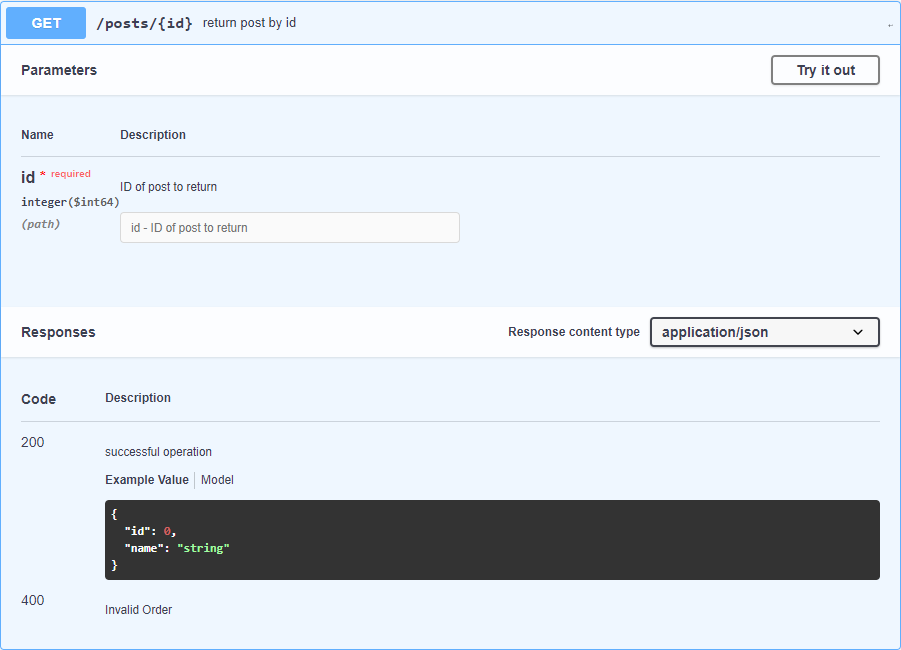


Рисунок А.33 – REST API

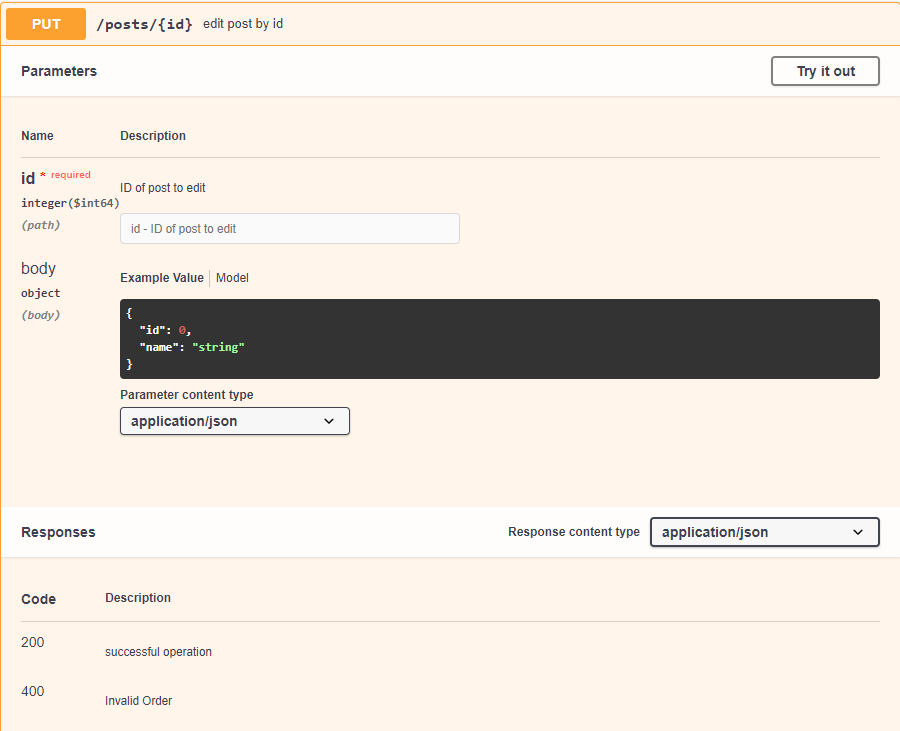


Рисунок А.34 – REST API

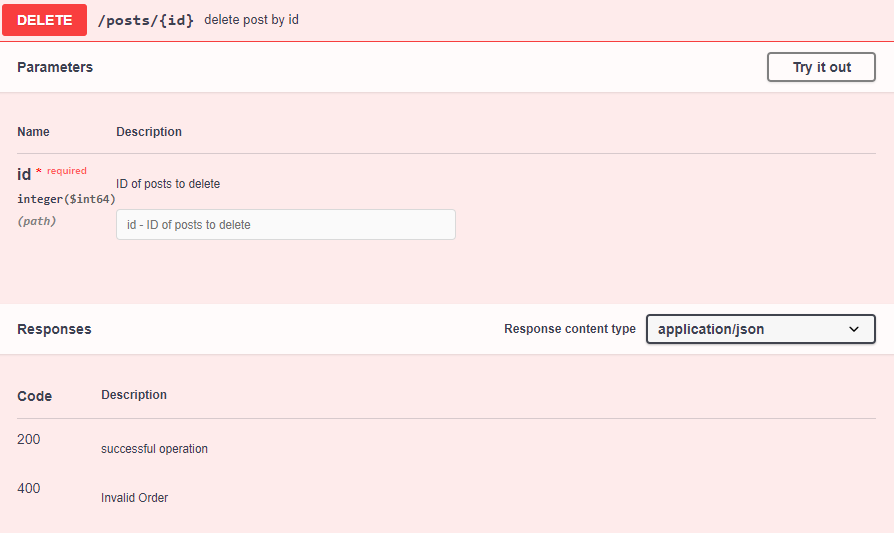


Рисунок А.35 – REST API



Рисунок А.36 – REST API

Приложение B

(обязательное)

Код проекта

Код проекта доступен по ссылке:

https://github.com/egorzakharovv/kurs