[1 Теоретическое введение 3](#_Toc121359560)

[2 Выполнение поставленных задач в практической №5 4](#_Toc121359561)

[2.1 Аналитическая часть 4](#_Toc121359562)

[2.1.1 Распределение задач между серверной и клиентской частями 4](#_Toc121359563)

[2.1.2 Построение интерфейса взаимодействия клиентской и серверной часте 5](#_Toc121359564)

[2.2 Разработка серверной части информационной системы 14](#_Toc121359565)

[2.2.1 Реализация моделей данных в серверном приложении 14](#_Toc121359566)

[2.2.2 Реализация хранения данных в серверном приложении 16](#_Toc121359567)

[2.2.3 Реализация контроллеров для взаимодействия с клиентской частью 19](#_Toc121359568)

[2.2.4 Реализация логики ролевой модели на стороне серверного приложения 27](#_Toc121359569)

[2.2.5 Сборка и дистрибьюция разработанного серверного приложения 28](#_Toc121359570)

[2.3 Разработка клиентской части информационной системы 31](#_Toc121359571)

[2.3.1 Доработка моделей данных в мобильном приложении 31](#_Toc121359572)

[2.3.2Реализация интерфейса взаимодействия клиентской части информационной системы с сервером 33](#_Toc121359573)

[2.3.3 Реализация класса корневой логики сетевого взаимодействия с серверной частью информационной системы 34](#_Toc121359574)

[2.3.4 Доработка классов предоставления данных для реализации смешанного взаимодействия 38](#_Toc121359575)

[3 Демонстрация работы приложения 40](#_Toc121359576)

[4 Приложение 43](#_Toc121359577)

# Теоретическое введение

В наше время очень распространены различные маркетплейсы, в них продают множество товаров различного рода происхождения и порой покупать тот или иной продукт довольно страшно, поэтому мы заходим читать отзывы различных людей, которые часто состоят из пары слов и не могут дать представление о предмете вожделения, происходит из-за шаблонного представления отзывов, состоящих из 3 пунктов: достоинства, недостатки и комментарии, при таких инструментах сложно сделать идейный отзыв о товаре после которого тот или иной читатель сможет понять стоит ли тот или иной товар покупки.

Программа «Отзовик» позволяет синхронизирует данные с помощью удалённого сервера, обрабатывающего различные запросы к нему.

# Выполнение поставленных задач в практической №5

## Аналитическая часть

### Распределение задач между серверной и клиентской частями

В виду требования разработки сервера для информационной системы, необходимо произвести распределение задач между клиентской частью приложения и серверной частью. Для разрабатываемого приложения «Отзовик» важно соблюдать актуальность данных и их централизованность, следовательно сервер должен служить ресурсом данных для клиентской стороны, чтобы она всегда хранила актуальную информацию о пользователях и отзывах созданными первыми. В результате анализа схема распределения функциональных задач стала выглядеть следующим образом, показанной в таблице 1.

Таблица 1. Распределение задач

|  |  |
| --- | --- |
| **Клиент** | **Сервер** |
| Отображение отзывов | Хранение отзывов |
| Отображение пользовательских данных | Хранение персональной пользовательской информации |
| Выполнения ролевых функций | Взаимодействие с клиентом через API |
| Создание отзывов | Контроль уровней доступа пользователей |
| Хранение состояния после завершения работы |  |

Таким образом был произведён анализ разрабатываемой информационной системы, в ходе которого были распределены функциональные задачи между клиентской и серверной сторон.

### Построение интерфейса взаимодействия клиентской и серверной частей

Для выполнения функциональных задач разрабатываемой информационной системы необходимо обеспечить интерфейс позволяющий создавать новую сущность отзывов и пользователей, а также выполнять ролевые задачи в зависимости от роли исполнителя определять доступ к действиям. Требования были разработаны с помощью Swagger UI и представлены на рисунках 1 - 12 с описанием требований к данным и вид возвращаемых данных.

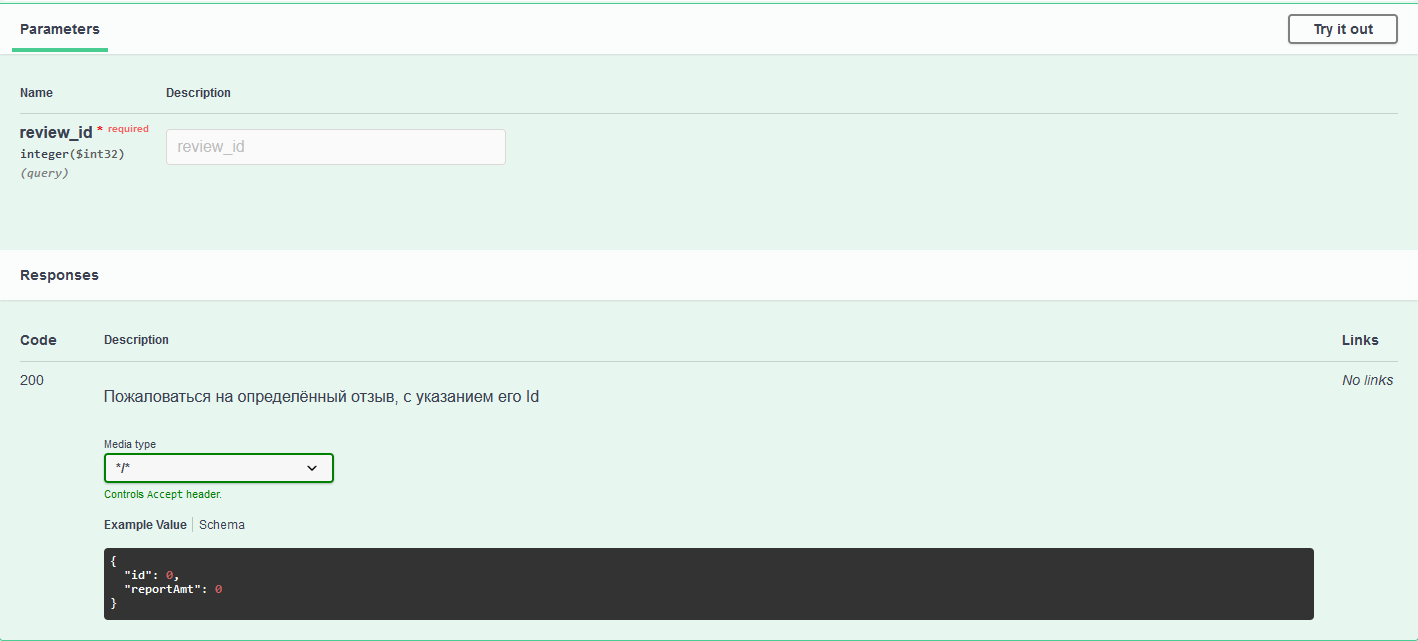


Рисунок 1 – жалоба на отзыв

На основе review\_id – который является id отзыва будет сформирован новый список жалоб

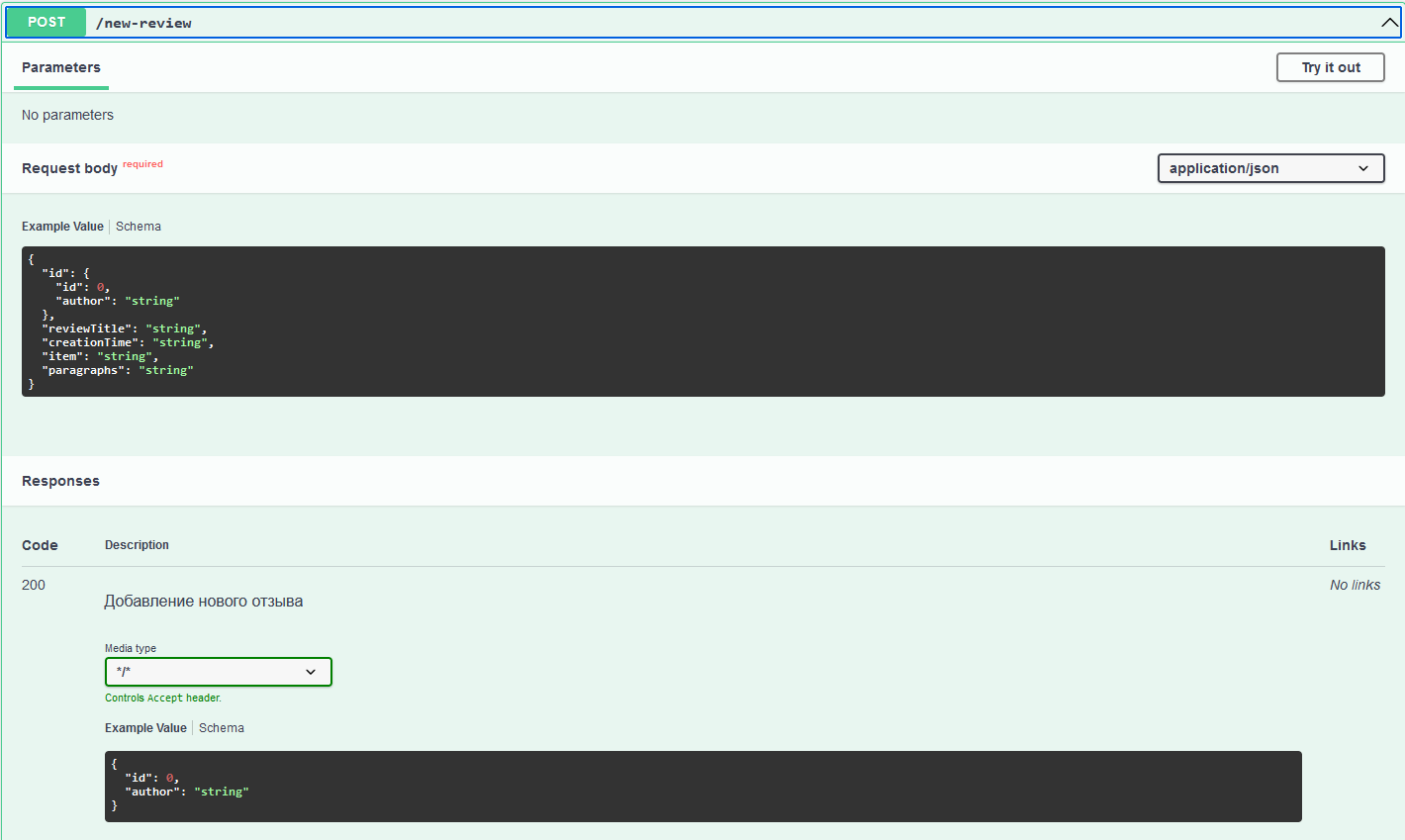


Рисунок 2 – создание нового отзыва

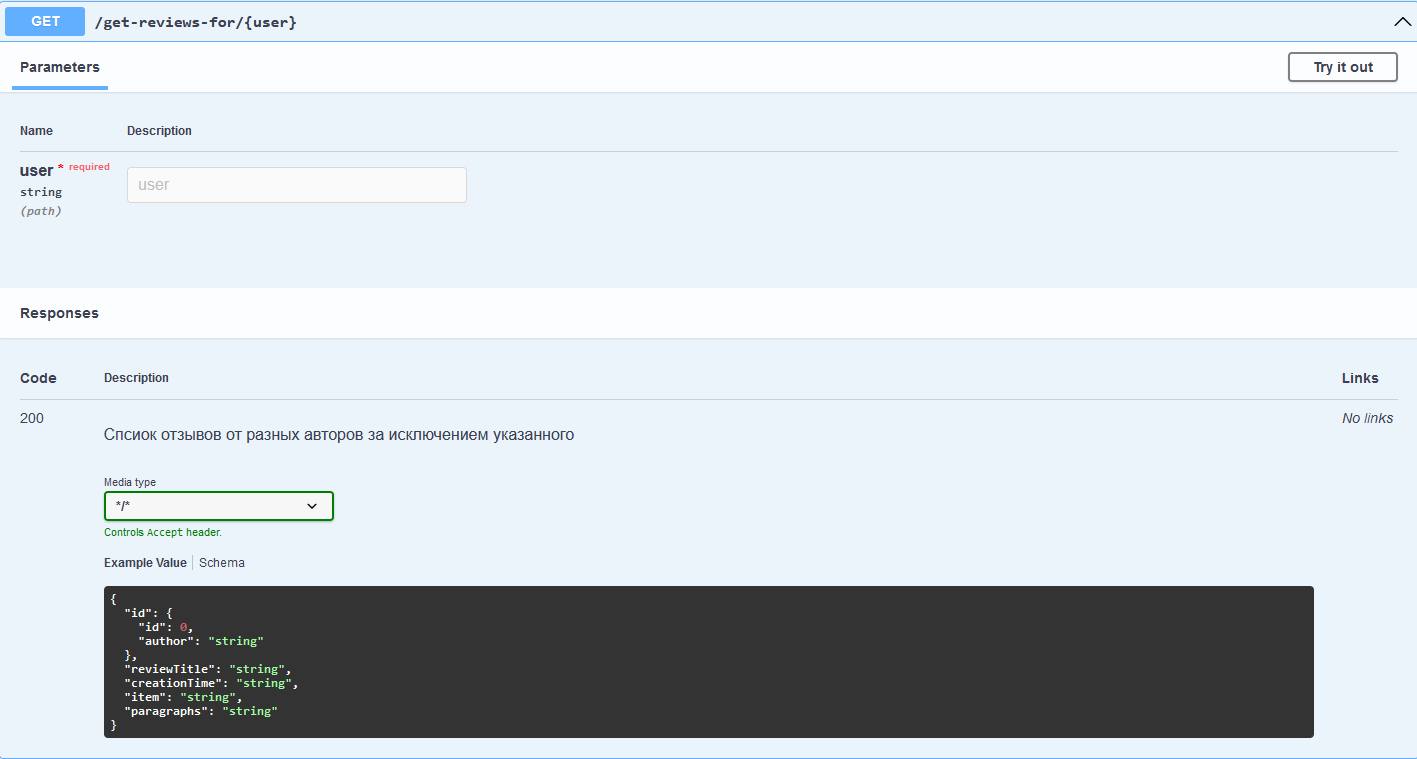
****

Рисунок 3 – список отзывов кроме указанного пользователя

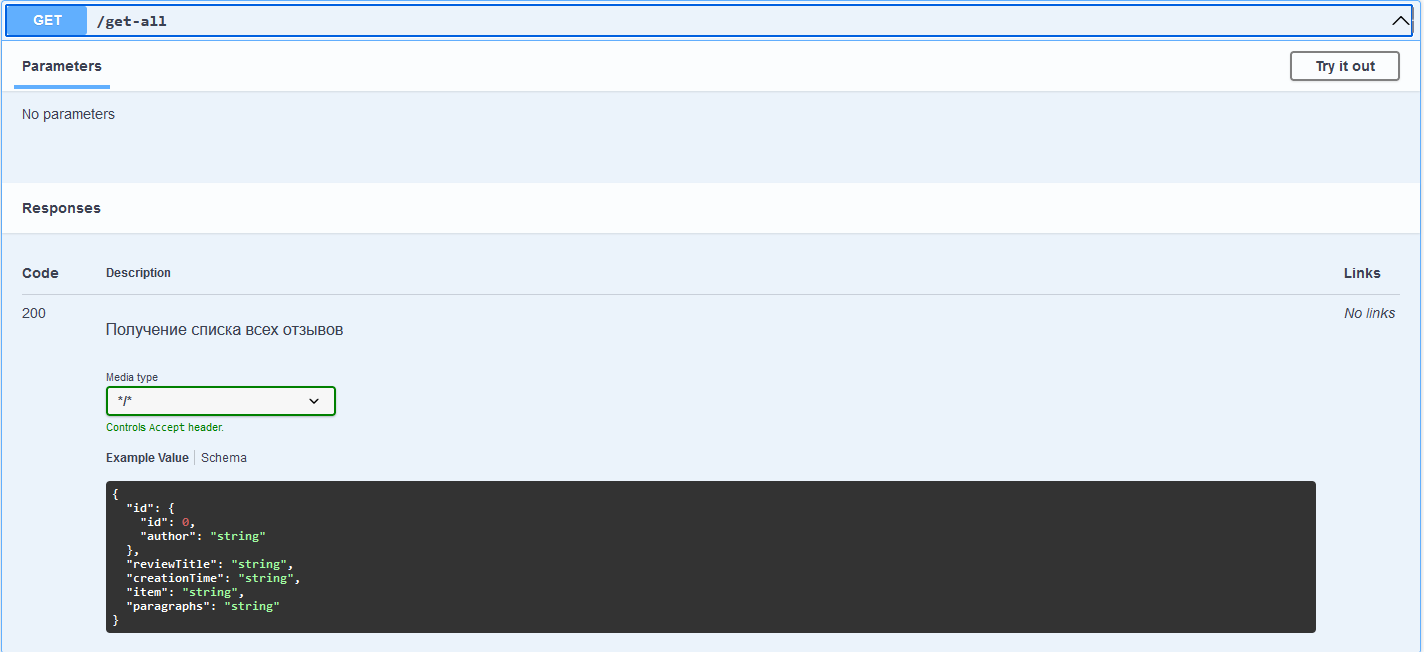


Рисунок 4 – список всех отзывов

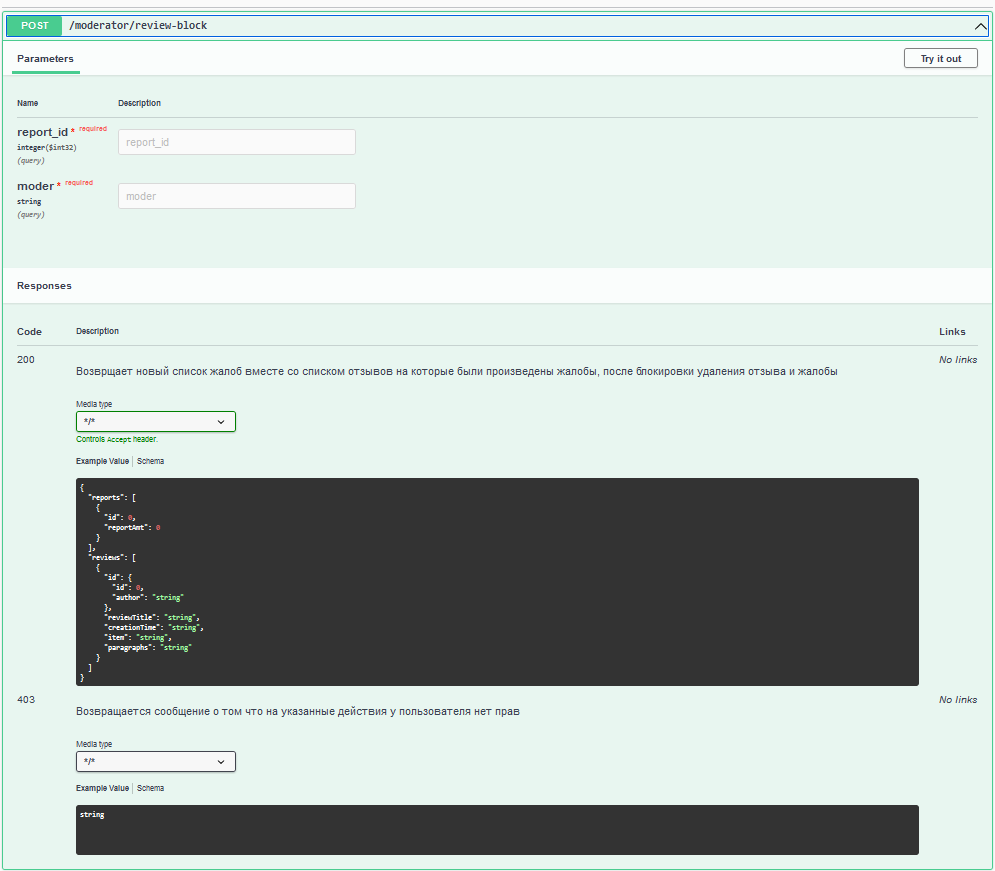


Рисунок 5 – блокировка отзыва

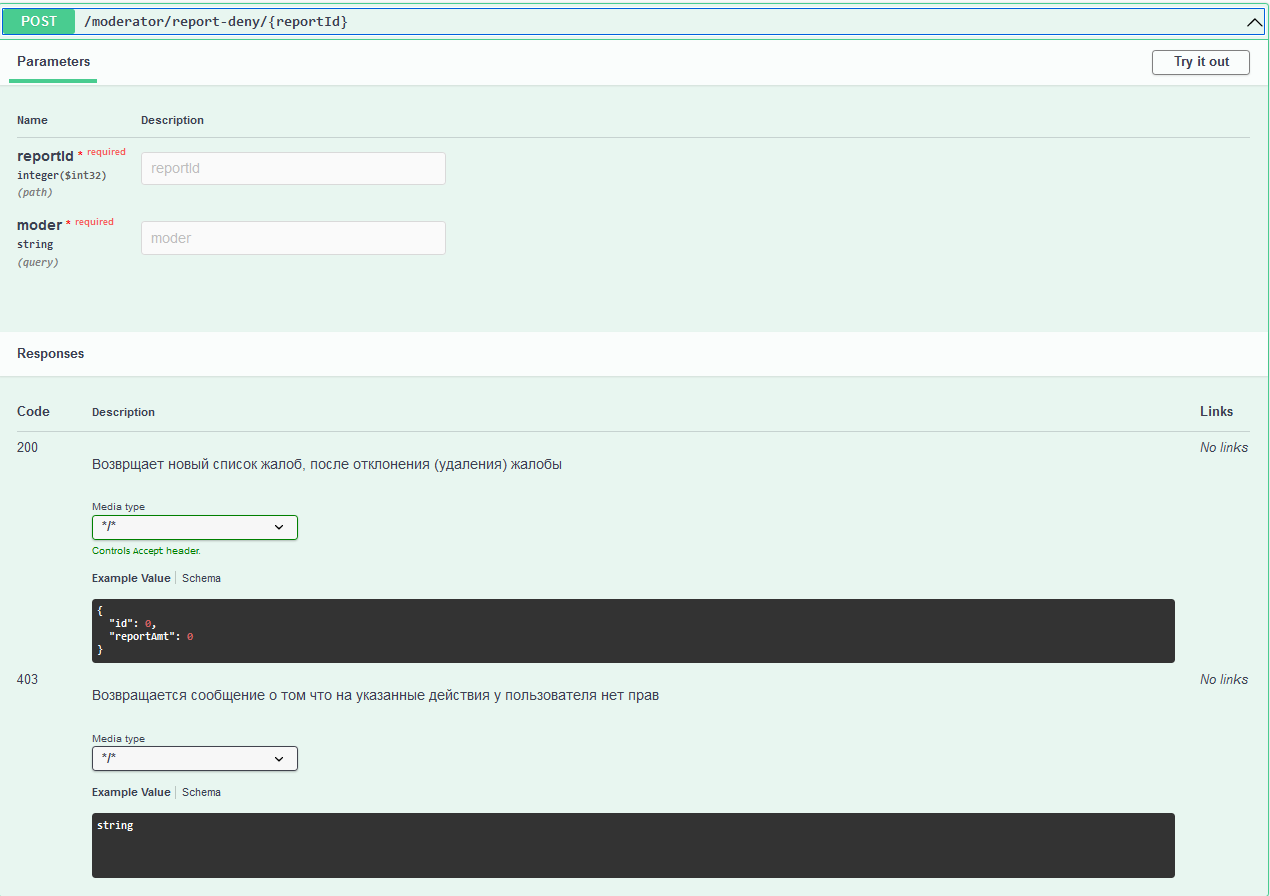


Рисунок 6 – отклонение жалобы

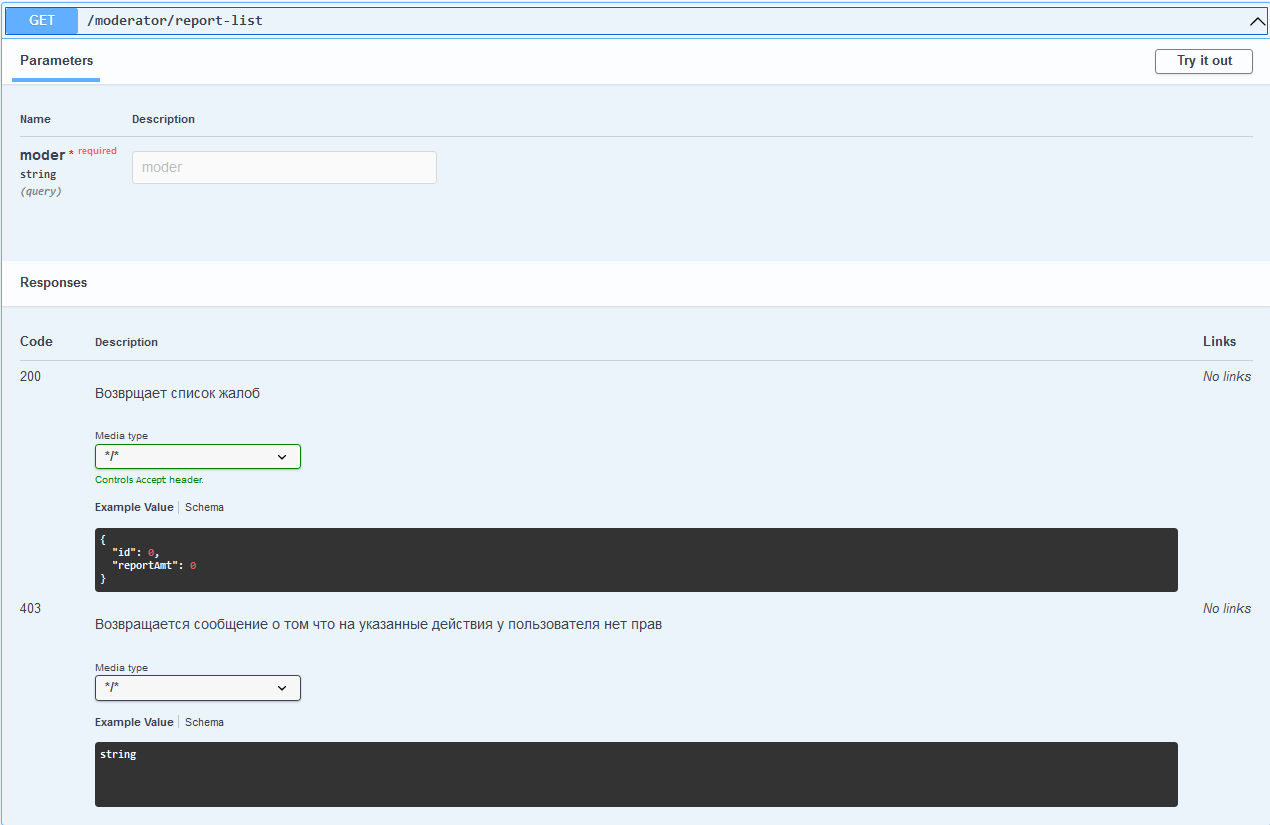


Рисунок 7 – список жалоб

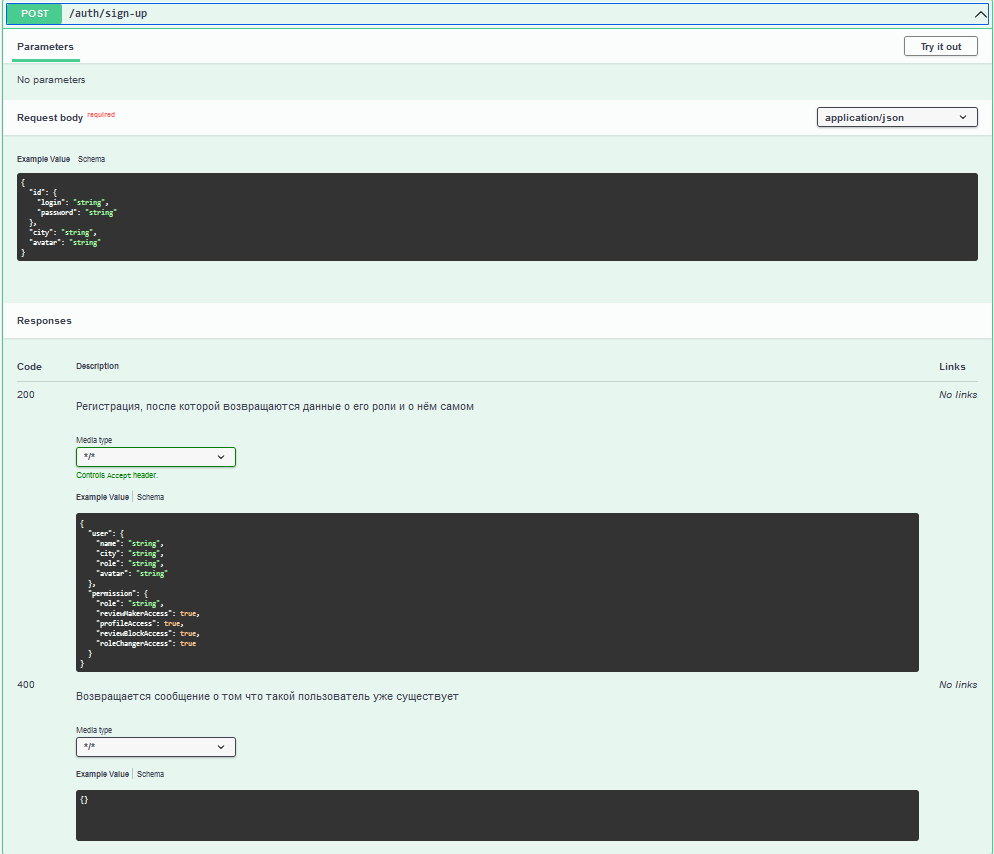


Рисунок 8 – регистрация

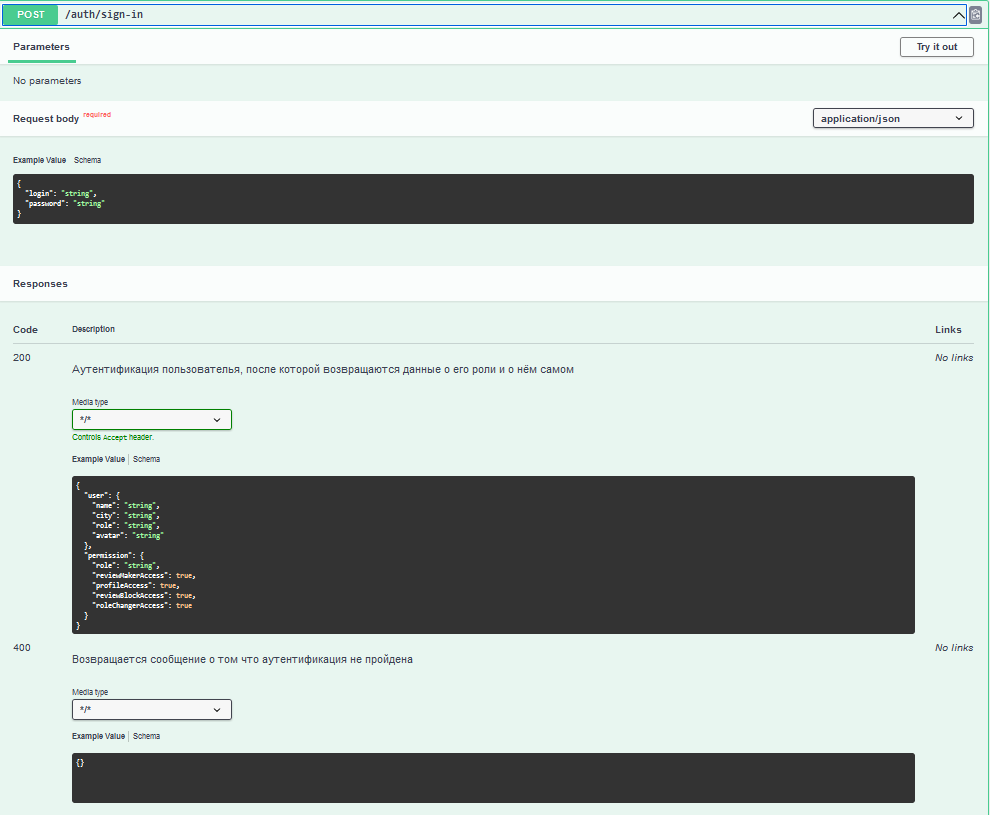


Рисунок 9 – авторизация

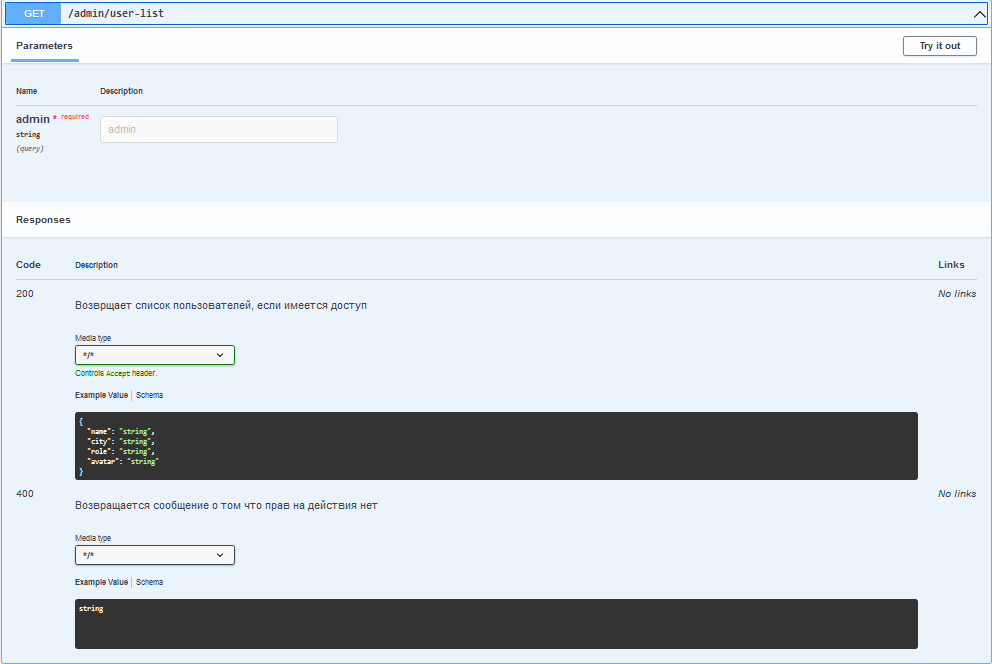


Рисунок 10 – получение списка пользователей для администратора

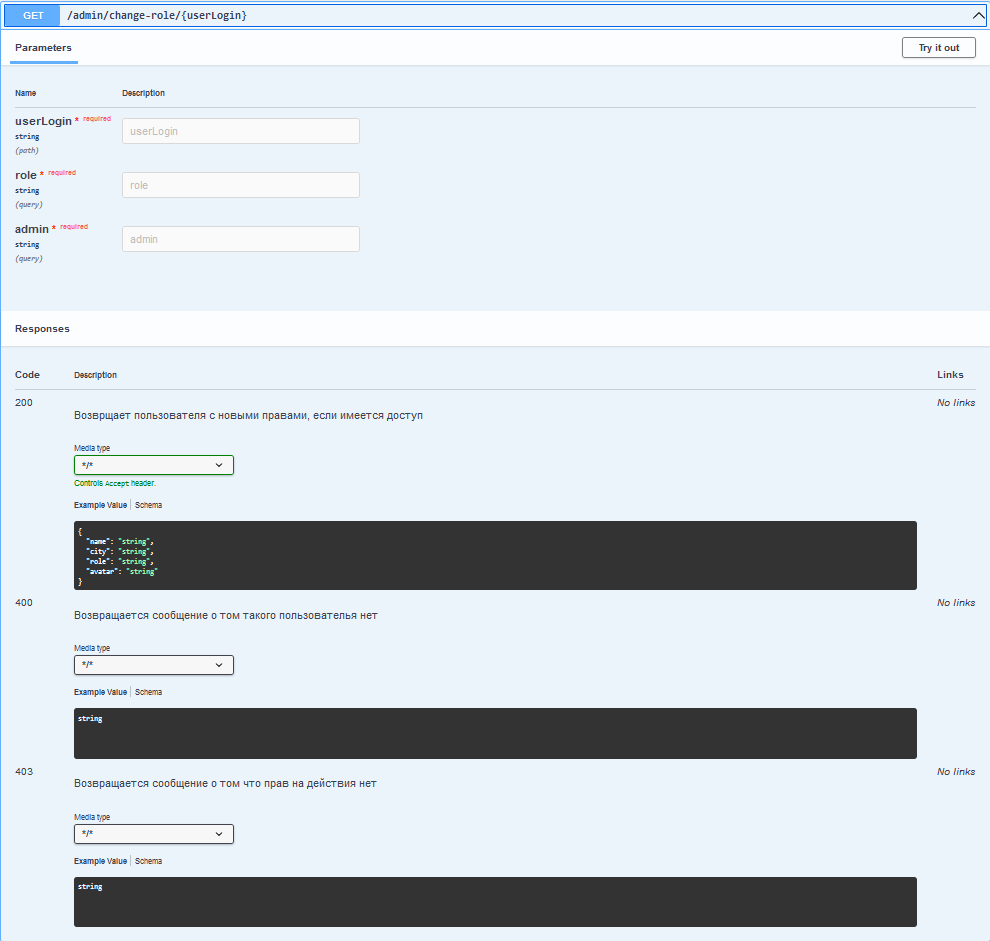


Рисунок 11 – изменение роли указанному пользователю

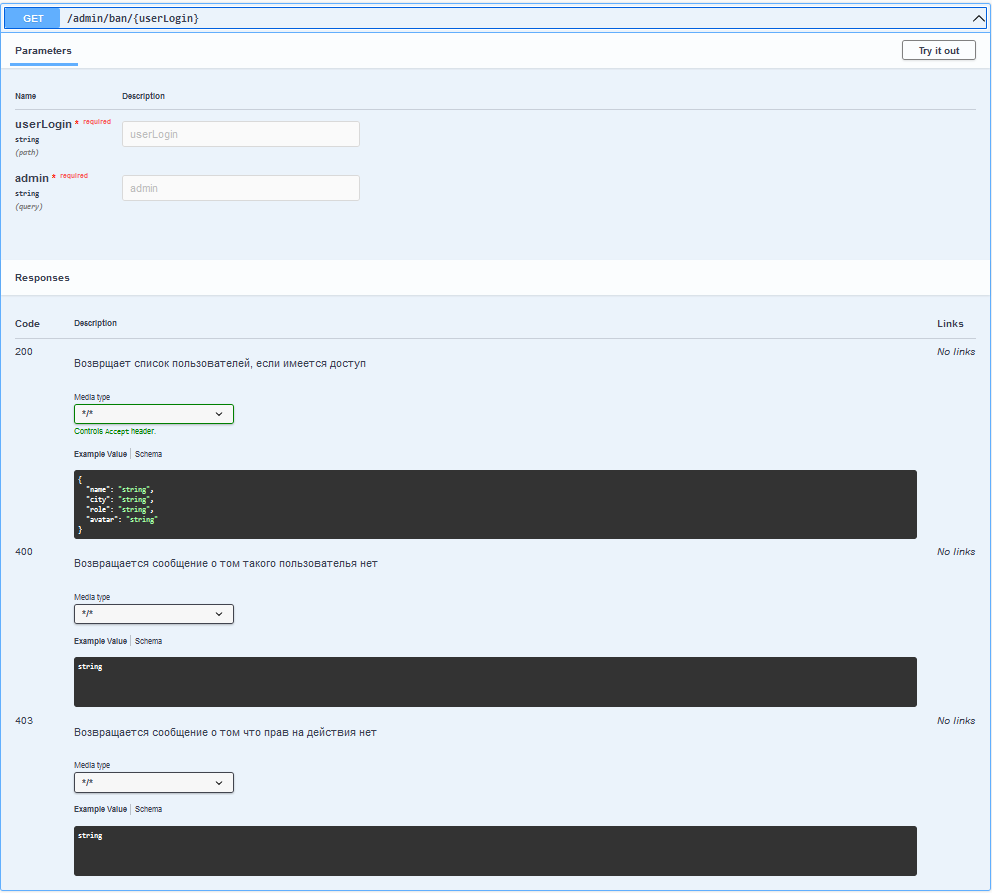


Рисунок 12 – блокировка указанного пользователя

Обращаться к последним двум методам можно используя любой метод запроса.

## Разработка серверной части информационной системы

### Реализация моделей данных в серверном приложении

Для разработки серверной части был выбран framework spring-boot, поскольку он реализован на языке Java, на котором также пишется и разрабатываемое приложение.

Для инициализации начального проекта был выбран ресурс spring.io, где можно собрать начальный проект бесплатно и сразу указать необходимые зависимости из предложенного списка, параметры разрабатываемого приложения показаны на рисунке 13.

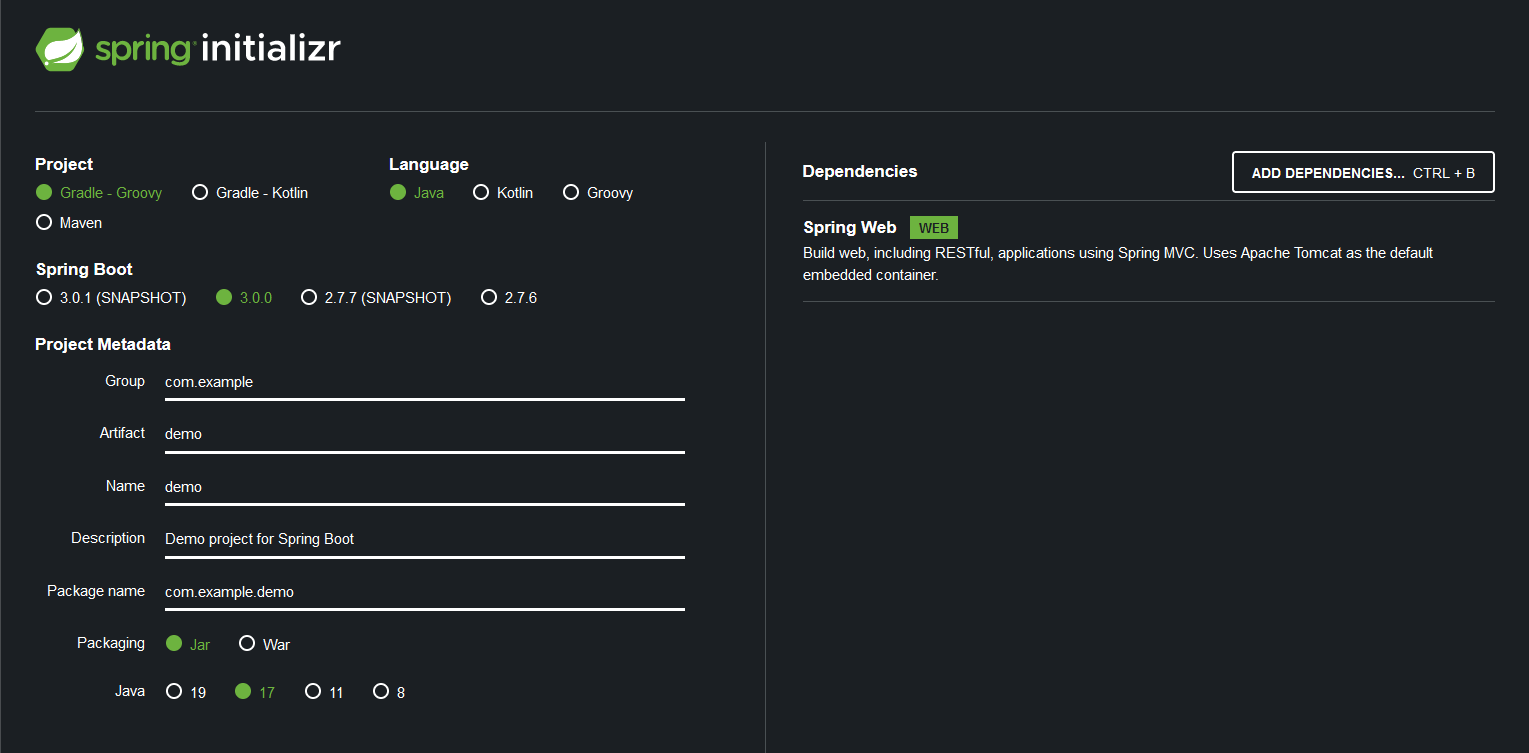


Рисунок 13 – конфигурация начального проекта сервера

Для поставленных задач из зависимостей необходимо только Spring Web для построения RESTful API.

Spring Boot удобен тем что его запуск производится как и все java приложения через функцию main() что облегчает процесс отладки и запуска. Поскольку для работы оригинальной Spring Framework необходимо собирать конфигурационные файлы, которые составлены автоматически в выбранном варианте.

Проект был поделён по представленным на рисунке 14 пакетам.

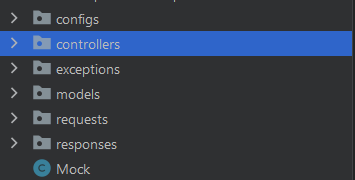


Рисунок 14 – пакеты проекта

Где во пакете configs будут хранится различные конфигурационные классы, например для Spring Docs, позволяющий автоматически генерировать страницу с документацией API, которая будет описана над методами в виде аннотаций над ними.

В controllers будут храниться классы, реализующие начальную обработку поступающих запросов и производить ответы в итоге обработки.

В exceptions будут храниться созданные исключения вызываемые при различных нарушениях

В models будут храниться прототипы моделей для будущих сущностей для БД.

В requests и responses храниться модели для маппинга запросов и отправление ответов на них.

Mock же является имитатором работы БД, аналогично первой части практической работы №1.

### Реализация хранения данных в серверном приложении

Для создания моделей были использованы модели с клиентской части, например, модели-сущности «Report», «Permission», «UserResponse» аналогичны моделям-сущностям из клиентской части приложения. Модель Review также аналогичны, как и User, но поля, являющиеся их идентификаторами, были объединены в классы, как показано на рисунках 15, 16. User храниться в Mock так как хранит пароль пользователя, но в ответ на запрос всегда отправляется UserResponse, поскольку не содержит приватной информации

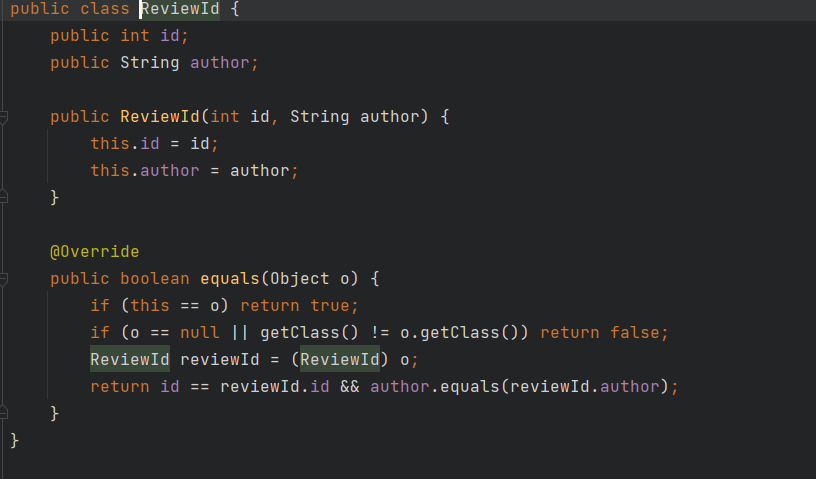


Рисунок 15 – класс ReviewId



Рисунок 16 – класс UserId

Также были созданы модели для отправки составных классов, например, списка с жалобами и отзывов, на которые были поданы жалобы, она представлены на рисунках 17, 18.

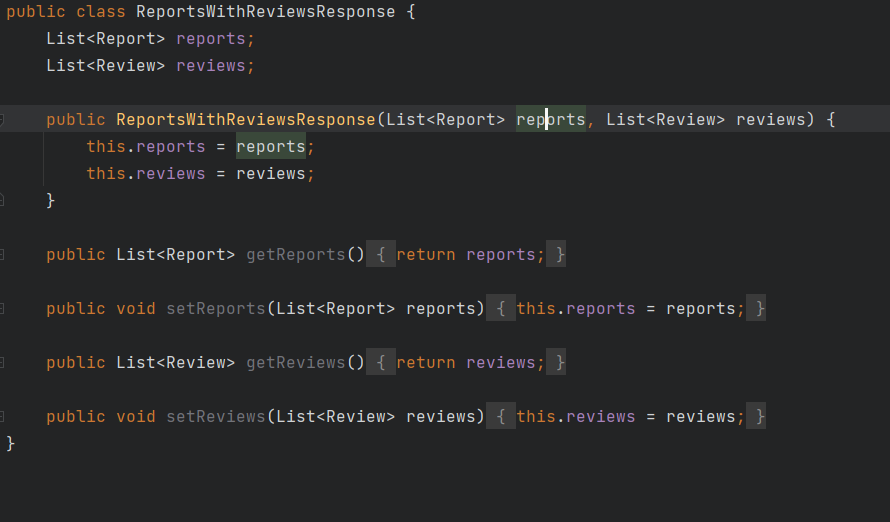


Рисунок 17 – класс ReportsWithReviewsResponse

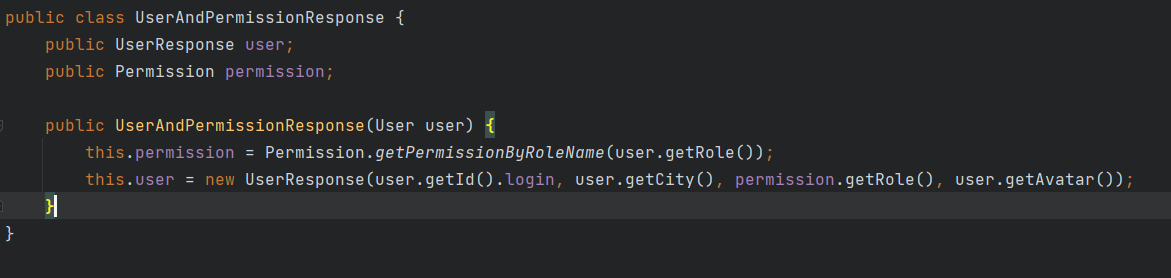


Рисунок 18 – класс UserAndPermission

Показанный класс является аналогичным тому, что присутствует в клиентской части. В большей степени модели были перенесены без изменений или были подстроены под модели API.

### Реализация контроллеров для взаимодействия с клиентской частью

Для того чтобы класс считался контроллером необходимо указать аннотацию @RestController, далее над методом, который будет производить обработку запросов необходимо указать либо аннотацию @{Method}Mapping либо @RequestMapping, где Method является название производимого методы с большой буквы, если указать второй вариант, то обработка запроса будет производиться вне зависимости от метода запроса, если не указывать поле method с методом запроса.

На рисунке 19 показаны методы, которые работают в котроллере ReviewController

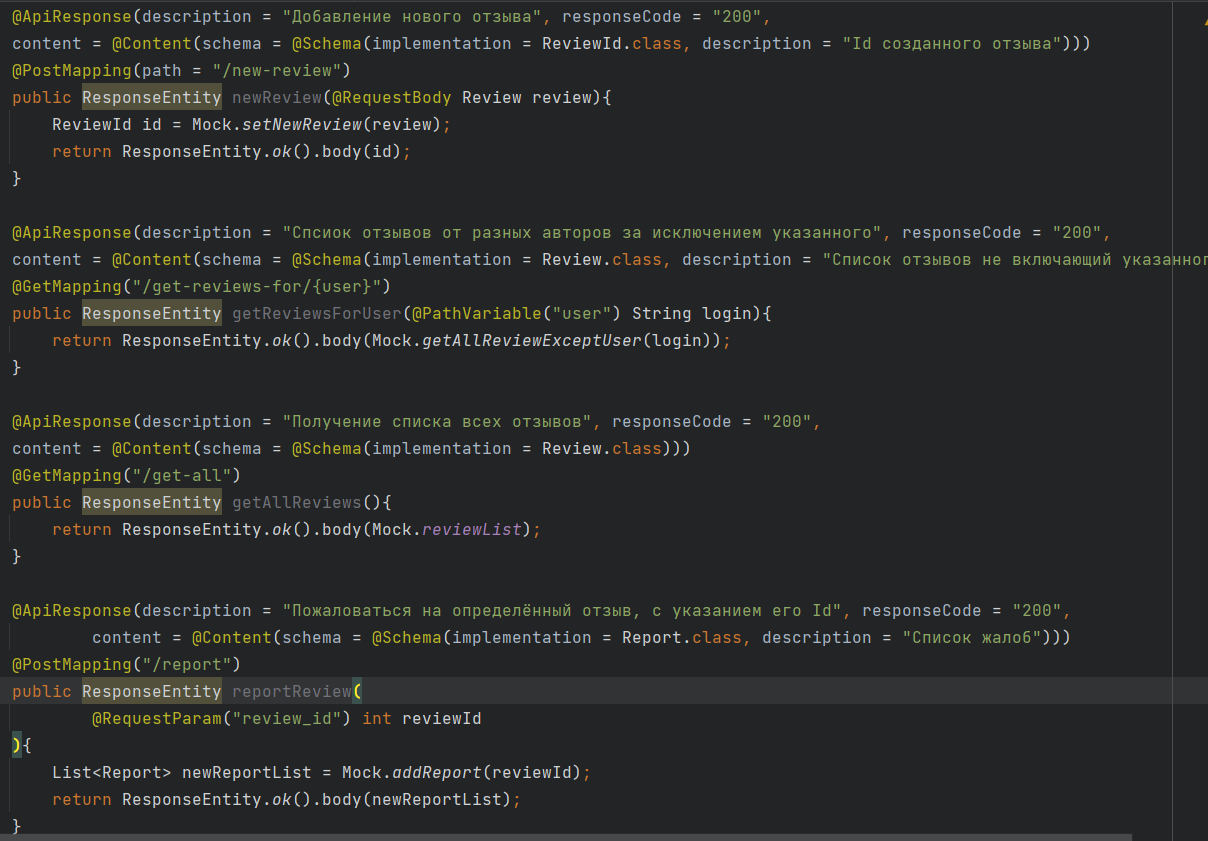


Рисунок 19 – методы контроллера ReviewController

Значениями в аннотациях методов запроса – путь, по которым можно обращаться в API, для реализации вложенности путей необходимо использовать RequestMapping на уровне классе и указать название корневого названия пути для доступа к подразделам. Для того чтобы Spring понимал какие данные необходимо с какими моделями необходимо указывать соответсвующие аннотации, например, аннотация @RequestBody означает, что следующий за ней параметр будет использован для маппинга тела запроса, или @RequestParam с указанием названия параметра, требуемого для запроса и следующий за ней параметр также будет использован для маппинга.

Аннотации @ApiResponse являются аннотациями библиотеки swagger UI, описывающая то, что было представлено выше, загружается вместе с spring-doc, которую необходимо указывать в отличие от мобильного приложения в файле pom.xml, сборщиком является maven.

На рисунках 20-22 изображены остальные контроллеры выполняющие методы, описанные выше.

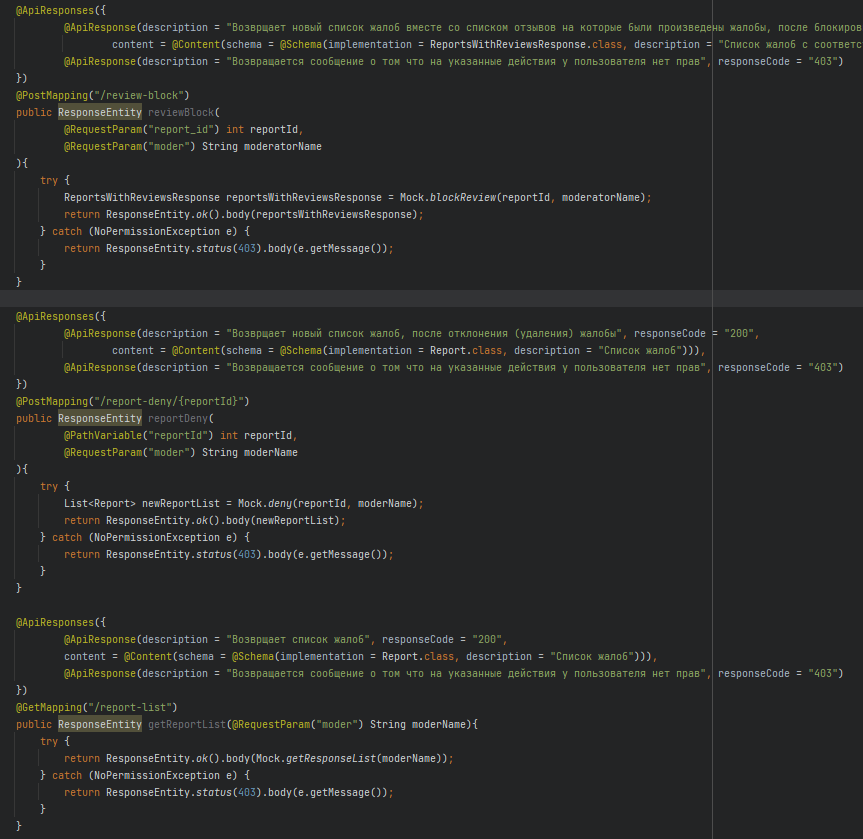


Рисунок 20 – ModeratorController

Данный контроллер выполняет запросы от модераторов.



Рисунок 21 – AdministratorController

Выполняет аналогичную ModeratorController работу.

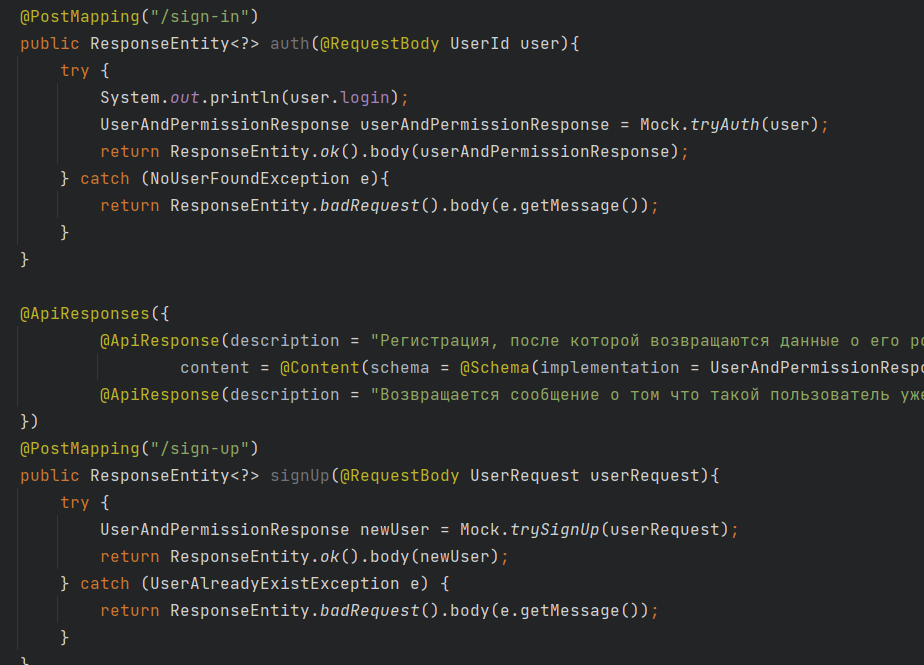


Рисунок 22 – AuthController

Выполняет обработку запросов, связанную с авторизацией и регистрацией.

Mock имитирует работу с базой данных и сервисом, выполняя сортировку данных и её обработку до необходимого состояния, также на выбрасывает исключения при каких-либо нарушениях, например, если пользователь не имеет доступ к действию которое он запрашивает произвести, например блокировка отзыва. Mock изображён на рисунках 23 – 26 и все методы выполняют работы по обработке данных запроса способом описанным выше в описании методов API.

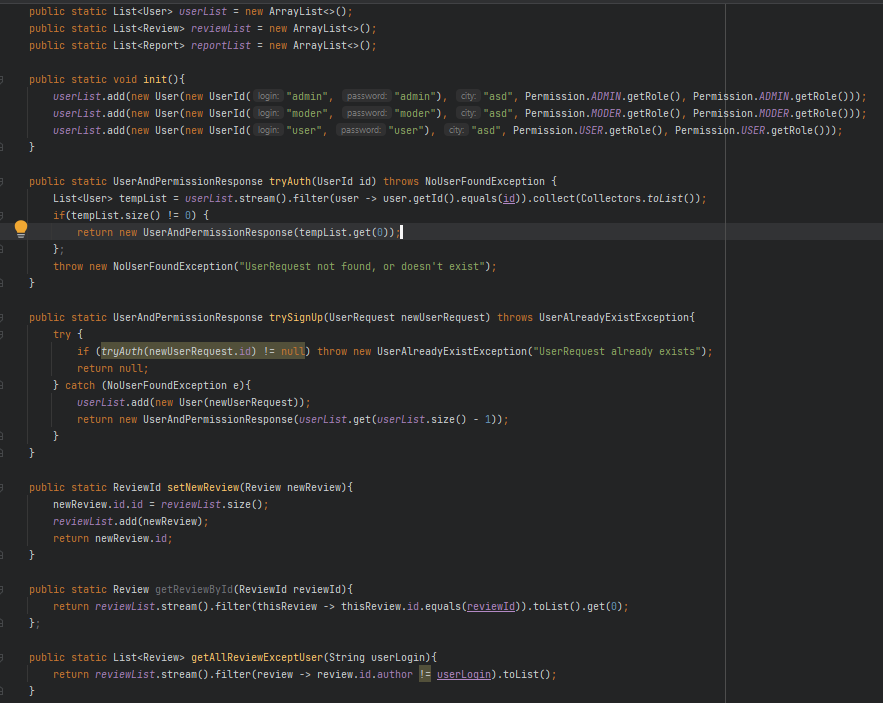


Рисунок 23 – Mock-класс ч.1

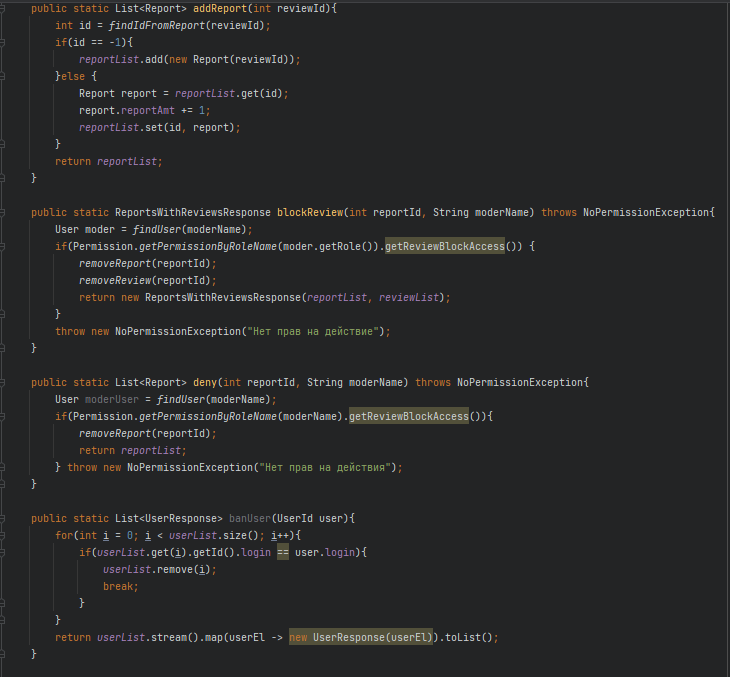


Рисунок 24 – Mock-класс ч.2



Рисунок 25 – Mock-класс ч.3



Рисунок 26 – Mock-класс ч.4

Методы в Mock являются простыми методами поиска со сравнением входного id с требуемым, или удалением аналогичным способом.

### Реализация логики ролевой модели на стороне серверного приложения

Для реализации доступа к данным была перенесена модель Permission, из клиентской части. Проверка доступа к определённым действием производится с помощью поиска наличия указанного пользователя, а затем поиска его разрешений по его роли, как показано на рисунке 27.

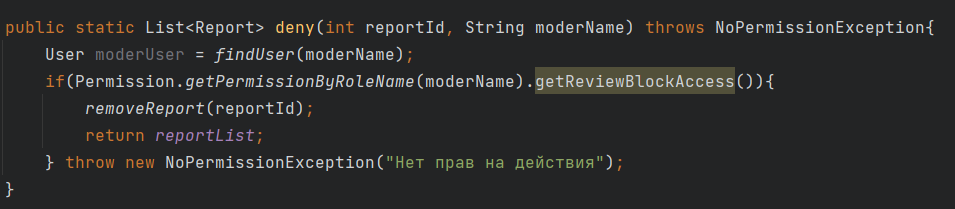


Рисунок 27 – способ проверки доступности действия пользователю

Таким образом если пользователь не имеет доступ, то будет брошено исключение сообщающий об отсутствии прав на действие.

### Сборка и дистрибьюция разработанного серверного приложения

Для сборки сервера, как указано выше необходимо лишь запустить main функцию – точку входа приложения, и выбранный сборщик, в данной работе это maven, соберёт программу и запустит сервер и далее можно будет им пользоваться. Для дистрибьюции сервера необходимо создать исполняемый файл, для этого нужно ввести в терминале в среде программирования команду, показанную на рисунке 28.



Рисунок 28 – команда для генерации исполняемого файла

Для дальнейшей контейнеризации необходимо написать сборочный файл – Docerfile (рисунок 29) описывающий последовательность действия для контейнеризации проекта в образ. Так ранее был создан исполняемый файл в папке target с раширением jar его и нужно будет контейнеризировать.

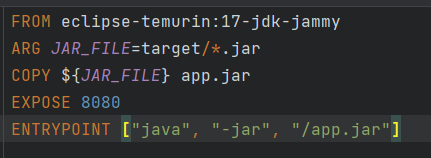


Рисунок 29 – Dockerfile проекта

В файле в качестве родительского слоя был указан eclipse-temurin поскольку он является быстрым и лёгким дистрибутивом, и также указан используемый в проекте jdk, чтобы получить необходимый образ.

ARG создаёт переменную используемую в скрипте хранящую название созданного исполняемого файла. Далее этот файл копируется в образ в файл под названием app.jar. И в качестве входной точки – команды исполняемой при запуске образа, будет указан запуск с помощью java исполняемого класса. Также в качестве порта для работы сервера указываем 8080 с помощью EXPOSE

Для генерации образа docker необходимо после создания Dockerfile в терминале обратившись к инструменту docker вызвать команду указанную на рисунке 30.



Рисунок 30 – команда создания образа

Далее необходимо воспользоваться программой Docker Desktop позволяющий взаимодействовать с созданными образами через графический интерфейс, там же и появляется созданный образ, который можно сразу запустить как показано на рисунке 31.

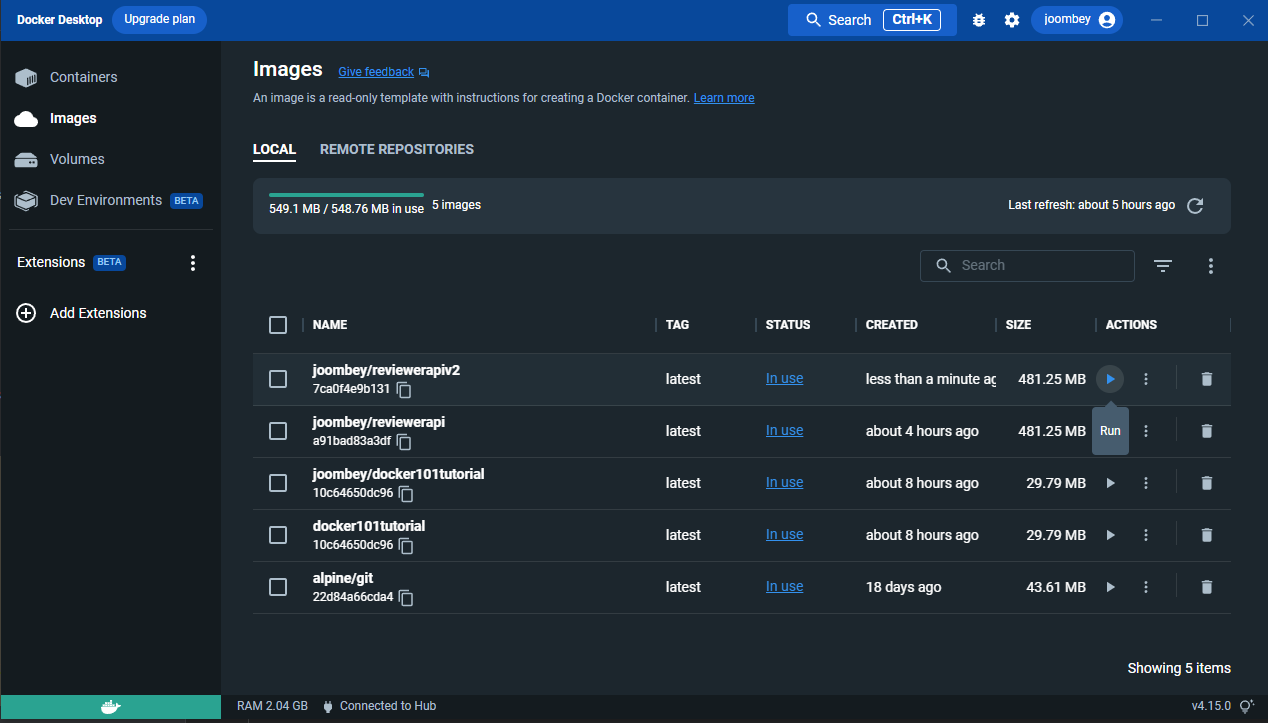


Рисунок 31 – интерфейс Docker Desktop

После необходимо будет выбрать начальные настройки из представленного необходимо лишь указать порт через который мы будем обращаться к образу а тот в свою очередь будет перенаправлять по порту указанному в Dockerfile

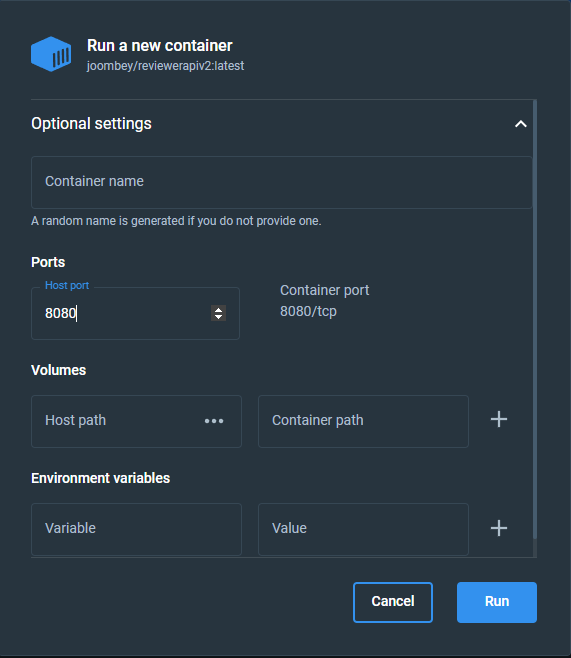


Рисунок 32 – окно предзагрузачной конфигурации

Затем будет произведён запуск контейнера как показано на рисунке 33.



Рисунок 33 – запуск контейнера

Как видно запуск был произведён успешно и сервер работает через контейнер и занимает порт 8080.

Созданный образ можно сохранить в DockerHub аналог GitHub куда можно отправлять и хранить образы. Это можно произвести в Docker Desktop.

## Разработка клиентской части информационной системы

### Доработка моделей данных в мобильном приложении

Производить доработку моделей данных не пришлось поскольку, ответу приходящие от сервера приходят в формате имеющихся моделей, но для случаем, где структура отличается были созданы соответсвующие классы-модели аналогичные моделям на сервере, как показано на рисунказ 34 – 38

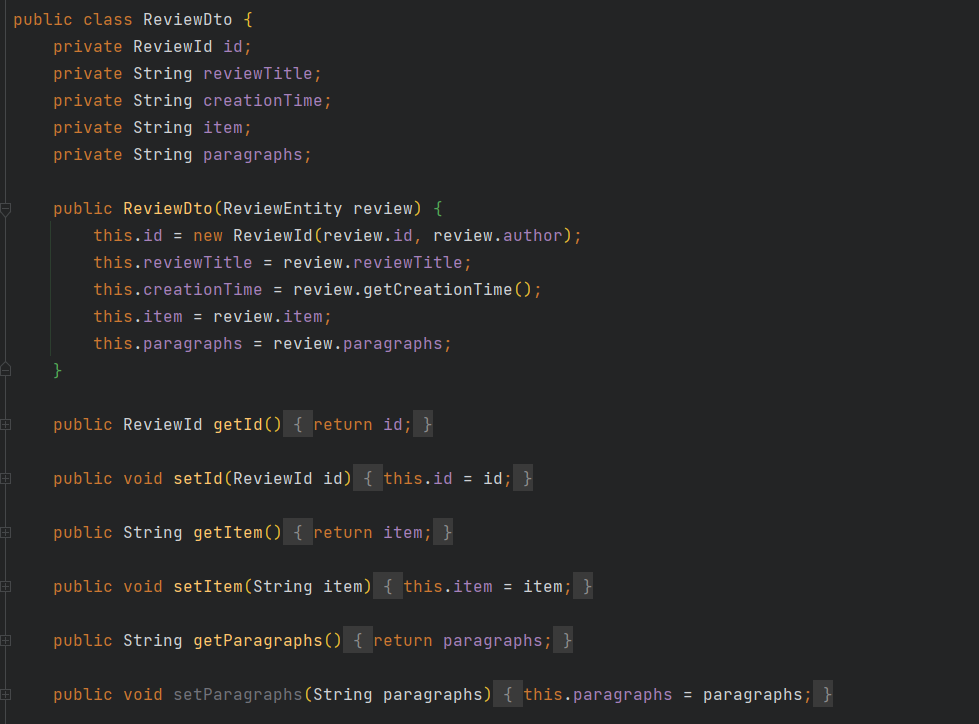


Рисунок 34 – ReviewDto – аналог Review из сервера

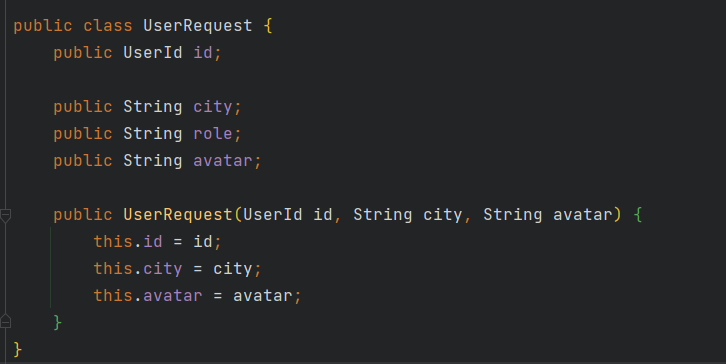


Рисунок 35 – UserRequest – аналог User, используется при регистрации.

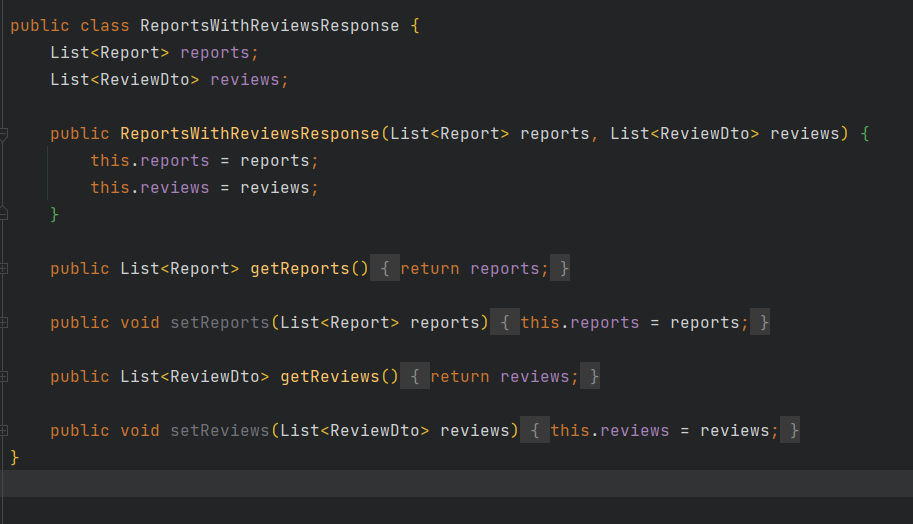


Рисунок 35 – ReportsWithReviews – аналог ReportsWithReviewsResponse

Используется при регистрации.

Таким образом доработка моделей данных была минимальна.

### Реализация интерфейса взаимодействия клиентской части информационной системы с сервером

Для того чтобы клиентская часть приложения могла взаимодействовать с сервером необходимо в первую очередь создать соответствующий интерфейс-сервис с использованием «Retrofit 2» (рисунок 36).

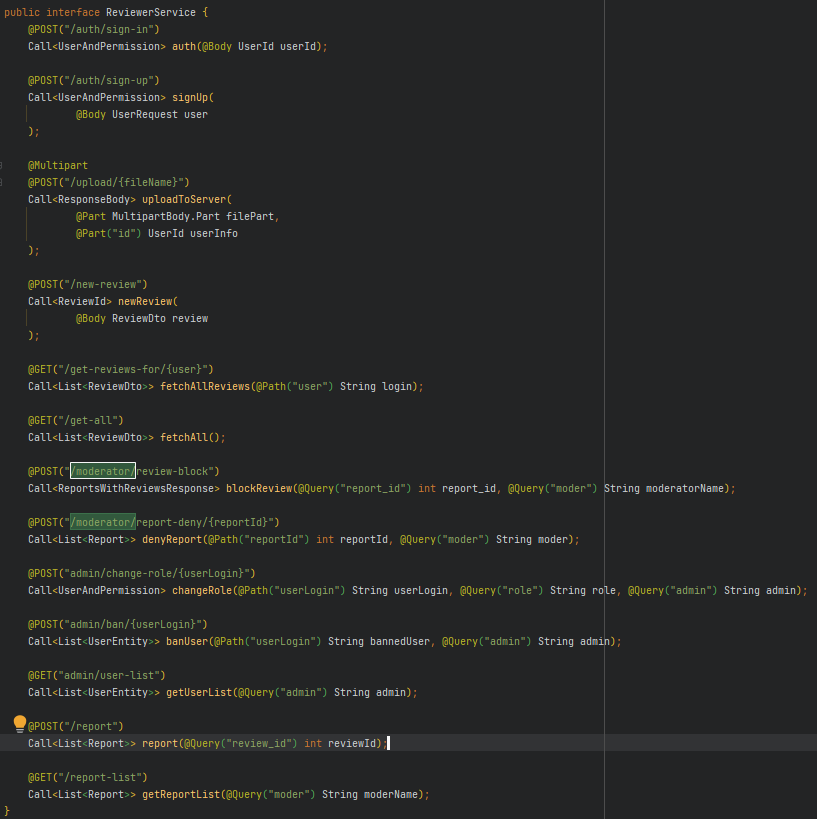
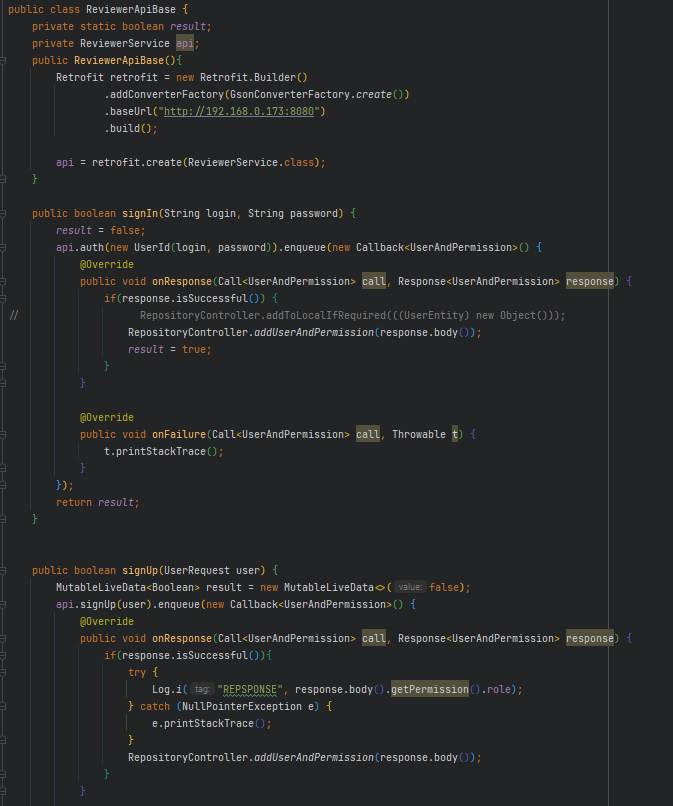


Рисунок 36 – сервис для сетевых запросов

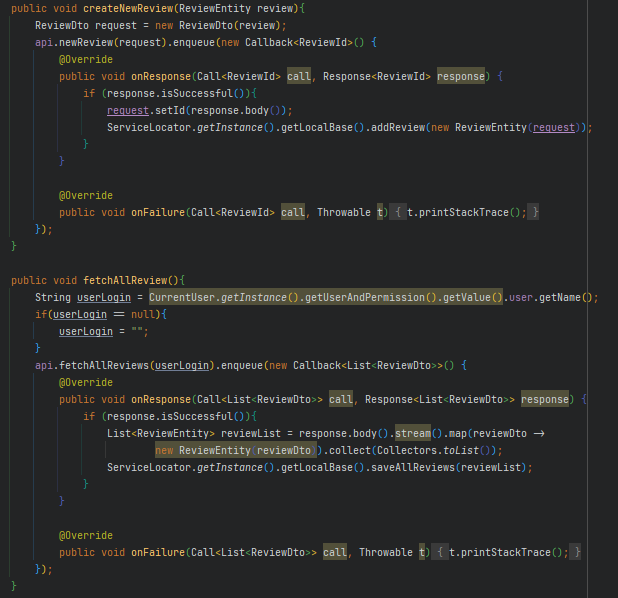
Запросы являются адаптацией созданного API и следуют его требованиям.

### Реализация класса корневой логики сетевого взаимодействия с серверной частью информационной системы

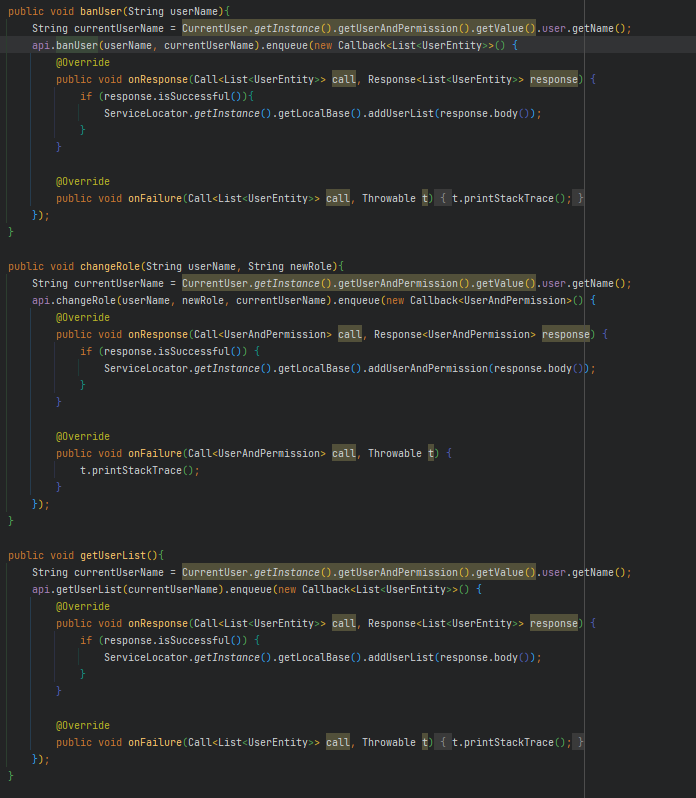
Базовый класс строился на подобии тех, чтобы были созданы в предыдущих практических работах и представлен на рисунках 37 – 41.



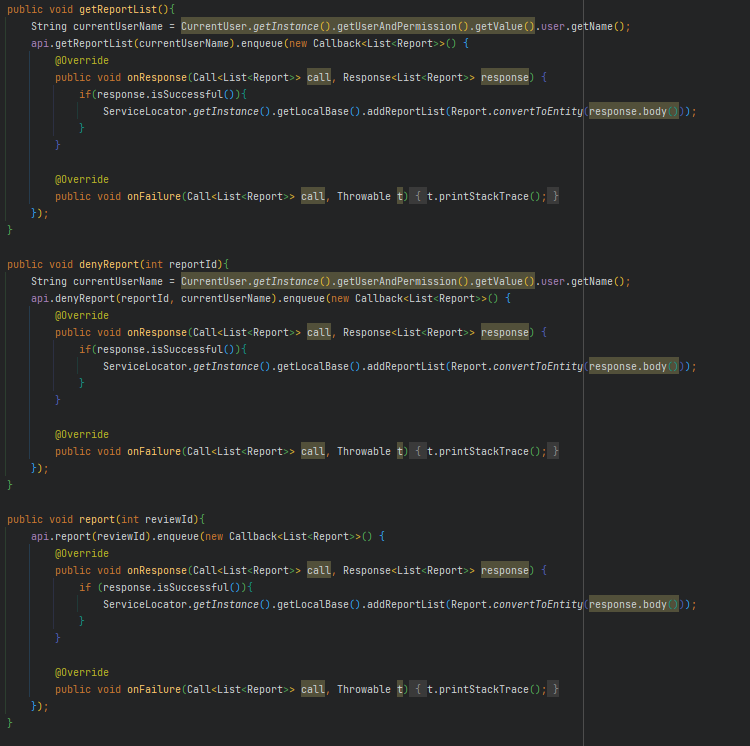
Рисунке 37 –класс ReviewerApiBase для реализации сетевого взаимодействия ч. 1



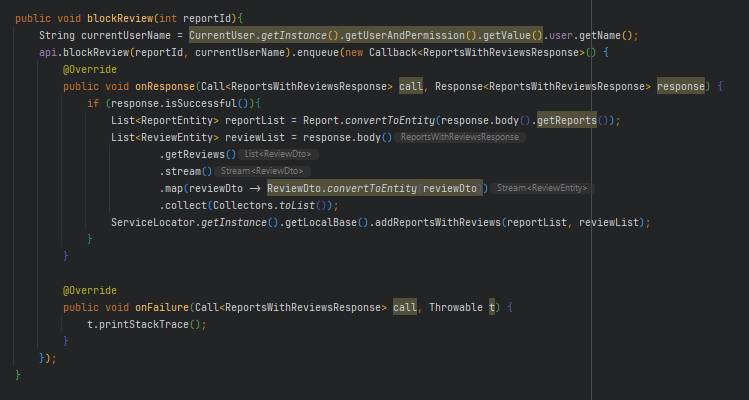
Рисунке 38 –класс ReviewerApiBase для реализации сетевого взаимодействия ч. 2



Рисунке 39 – класс ReviewerApiBase для реализации сетевого взаимодействия ч. 3



Рисунке 40 –класс ReviewerApiBase для реализации сетевого взаимодействия ч. 4



Рисунке 41 –класс ReviewerApiBase для реализации сетевого взаимодействия ч. 5

Методы производят обработку данных схожим образом – полученные данные конвертируются в модели сущности, если это необходимо, далее данные записываются в локальную базу данных.

### Доработка классов предоставления данных для реализации смешанного взаимодействия

Поскольку по ходу обработки ответов данные лишь записываются в локальное хранилище, то, так как она обновилась, подписанные на неё View, реагируют прежним способом поэтому изменения в работу реакции на изменения данных не производились.

При удалении определённых данных, например, жалобы, сначала производиться удаление данных из локального хранилища, а затем вызов сетевого метода, как показано на рисунке 42



Рисунок 42 – методы при удалении сущности

Также при вызове onViewCreated жизненного цикла Fragment, как показано на рисунке 43 клиент запрашивает новые данные с сервера и тем самым клиент всегда хранит актуальную информацию.

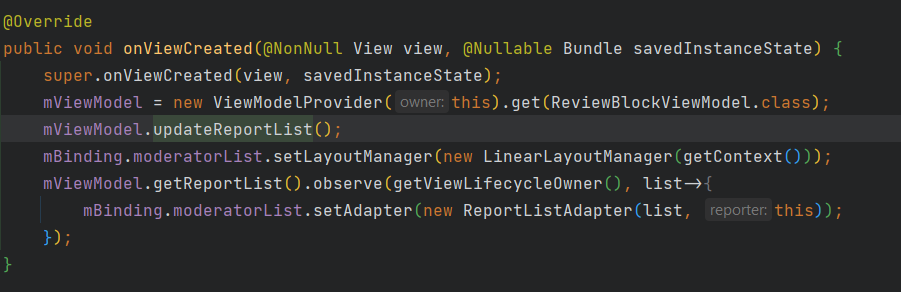


Рисунок 43 – обновление данных о жалобах

Таким образом в ходе проделанной работы клиентская часть подверглась малым доработкам.

# Демонстрация работы приложения

Для демонстрации работы приложения создадим отзыв на одном устройстве и запустим его на другом рисунки 44 - 45.



Рисунок 44 – отзыв с личного устройства

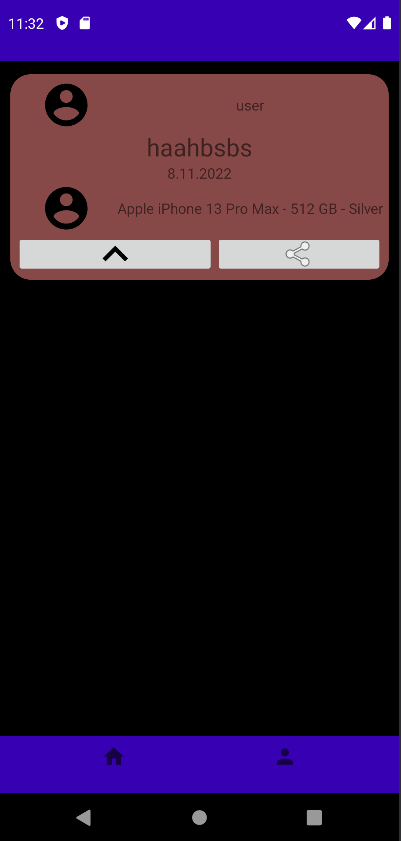


Рисунок 45 – отзыв на эмуляторе

Взаимодействие происходило на запущенном контейнере, изображённом на рисунке 46

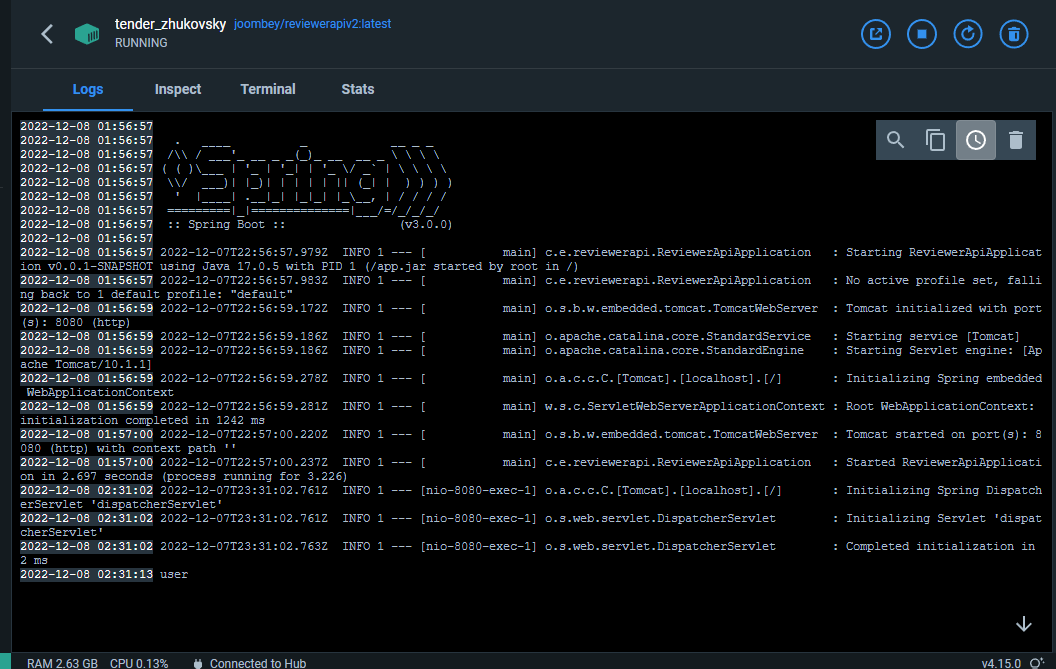


Рисунок 46 – запущенный образ сервера

По результатам демонстрации приложение работает с сервером, соответственно поставленные задачи выполнены.

# Приложение

Git Hub сервера - <https://github.com/Joombey/ReviewerAPI>

Git Hub клиента - <https://github.com/Joombey/ReviewerJAVA>

Docker Hub образа - <https://hub.docker.com/r/joombey/reviewerapiv2>