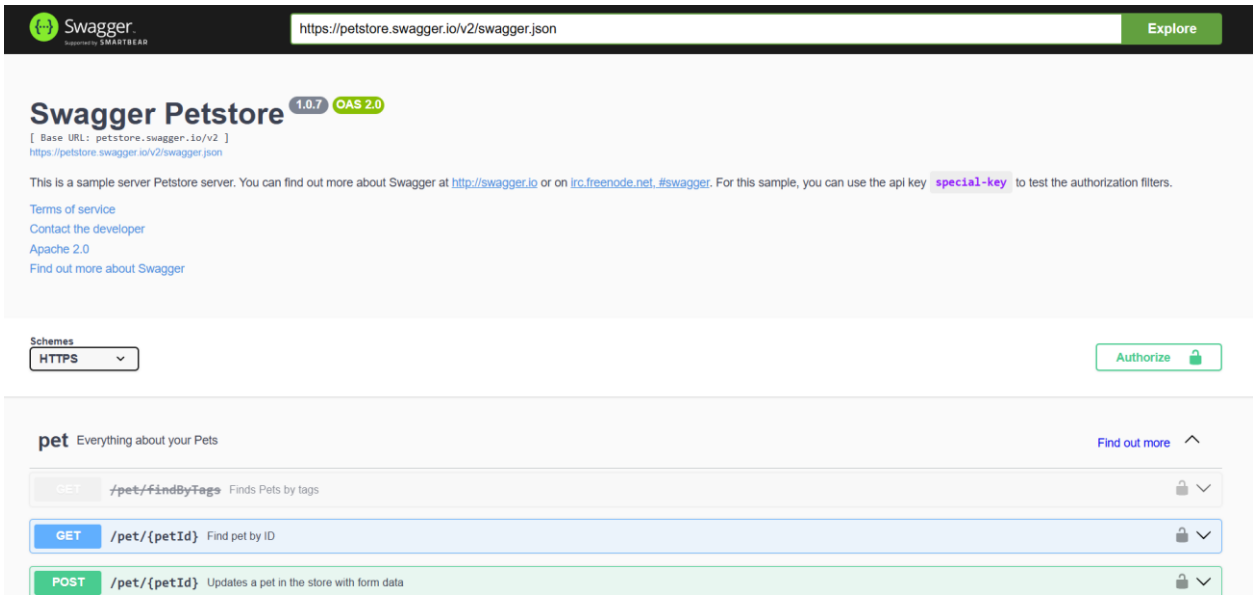
Теоретическая часть (изучить перед выполнением домашнего задания)

OpenAPI Specification — это стандарт для описания RESTful API в машиночитаемом формате (обычно в YAML или JSON). Он позволяет автоматизировать документацию, тестирование, генерацию клиента и сервера.

Swagger — это набор инструментов, основанный на OpenAPI, который помогает создавать, документировать и тестировать API. В 2016 году проект Swagger был передан Linux Foundation и стал частью OpenAPI Initiative.
Swagger-документ — это файл, в котором подробно описаны основные характеристики API. В нем указываются такие элементы, как конечные точки (эндпоинты), параметры запросов и ответов, используемые типы данных и другие важные детали. Такой документ состоит из нескольких разделов, каждый из которых отвечает за определённую часть API:

- Общая информация о версии и названии API.
- Описание доступных конечных точек и методов HTTP, которые можно использовать для обращения к ним.
- Детальное описание параметров запросов и формата ответов.
- Описание типов данных, применяемых в запросах и ответах.
- Структурированные схемы для запросов и ответов.
- Настройки безопасности, включая авторизацию.
- Дополнительные параметры работы API, такие как лимиты на количество запросов или механизмы кэширования.

Для удобства взаимодействия с этим описанием используется компонент Swagger UI — интерактивный интерфейс, который визуализирует документацию в удобной форме.
Благодаря такой интерактивной документации можно легко просматривать все доступные методы сервиса и сразу же тестировать их — отправлять запросы прямо из браузера и получать ответы без необходимости писать дополнительный код.



Пример сваггер – документа (доступен по ссылке: https://petstore.swagger.io/)

Способы создания Swagger-документации
Существует несколько подходов к формированию документации API с использованием Swagger:

1. **Ручное написание YAML или JSON файла**

Можно самостоятельно создать файл в формате YAML или JSON, в котором подробно описываются все конечные точки, параметры, схемы данных и другие элементы API. Этот способ требует знания синтаксиса Swagger и хорошего понимания структуры API.

Допустимо применять какой-то из синтаксисов, а для «перевода» в другой использовать конверторы, **пример конвертора**: https://codebeautify.org/json-to-yaml

2. **Использование генераторов на основе аннотаций в коде**
   Многие фреймворки и библиотеки позволяют автоматически генерировать Swagger-документацию из аннотированных исходных кодов. Например, в языках Java, Python или C# можно добавлять специальные аннотации к методам и классам, после чего инструмент автоматически создает актуальную документацию.
3. **Инструменты визуального моделирования и редакторы**

   Существуют графические редакторы и онлайн-сервисы (например, Swagger Editor), которые позволяют создавать и редактировать документацию через удобный интерфейс. Такие инструменты часто поддерживают автоматическую проверку синтаксиса и предварительный просмотр результата.

4. **Конвертация из других форматов документации**

   Можно преобразовать существующие спецификации или документацию (например, RAML или API Blueprint) в формат Swagger с помощью специальных конвертеров.

5. **Автоматическая генерация из спецификаций OpenAPI**

   В случае использования стандарта OpenAPI (который является развитием Swagger) есть возможность автоматически генерировать документацию на основе описаний API, размещённых в репозиториях или системах управления проектами.

Примеры инструментов

Существует множество инструментов, которые помогают автоматизировать процесс разработки и поддержки спецификаций OpenAPI. Вот некоторые из наиболее популярных:

1. **Swagger Editor**

   Онлайн-редактор и локальная версия, позволяющие создавать, редактировать и проверять спецификации OpenAPI в формате YAML или JSON с мгновенной визуализацией и подсветкой синтаксиса.
2. **Swagger UI**

   Интерактивная документация, которая автоматически генерируется из спецификации OpenAPI и позволяет просматривать API, тестировать методы прямо в браузере.
3. **OpenAPI Generator**

   Инструмент для автоматической генерации клиентских SDK, серверных шаблонов и документации на основе спецификации OpenAPI. Поддерживает множество языков программирования.
4. **Redoc**
   Еще один популярный генератор статической документации по спецификациям OpenAPI с современным дизайном и удобным интерфейсом.

5. **Insomnia Designer**

   Инструмент для проектирования API, который поддерживает создание и редактирование спецификаций OpenAPI с возможностью тестирования и совместной работы.
6. **Postman**
   Помимо тестирования API, Postman позволяет импортировать или экспортировать спецификации OpenAPI, а также генерировать документацию.

7. **VS Code Extensions (например, "OpenAPI (Swagger) Editor")**

   Расширения для редактора Visual Studio Code, которые обеспечивают подсветку синтаксиса, проверку ошибок и автодополнение при работе с файлами OpenAPI.

8. **APIMatic**
   Платформа для автоматического преобразования, генерации и поддержки API-спецификаций в различных форматах, включая OpenAPI.

Основные понятия

1**. OpenAPI Specification**

- Формат описания API
- Включает описание путей, методов, параметров, схем данных и т.д.
- Версии: OAS 2.0 (Swagger 2.0), OAS 3.0.x и более новые

2. Swagger Tools

- Swagger Editor — онлайн или локальный редактор для написания спецификаций
- Swagger UI— автоматическая генерация интерактивной документации
- Swagger Codegen — генерация клиентских SDK и серверных скелетов
- SwaggerHub — платформа для совместной работы над API

Структура файла OpenAPI (на примере YAML)

```yaml
openapi: 3.0.3
info:
  title: Пример API
  version: 1.0.0
servers:
  - url: https://api.example.com/v1
paths:
  /users:
    get:
      summary: Получить список пользователей
      responses:
        '200':
```

```
        description: Успешный ответ
        content:
          application/json:
            schema:
              type: array
              items:
                $ref: '#/components/schemas/User'
components:
  schemas:
    User:
      type: object
      properties:
        id:
          type: integer
        name:
          type: string
```

Основные разделы спецификации OpenAPI

1. `info`
Описание API (название, версия, описание).

2. `servers`
Массив серверов, на которых доступен API.

3. `paths`
Определение маршрутов (эндпоинтов), методов HTTP (GET, POST и т.д.), параметров и ответов.

4. `components`
Общие схемы данных, параметры, ответы и другие компоненты для повторного использования.

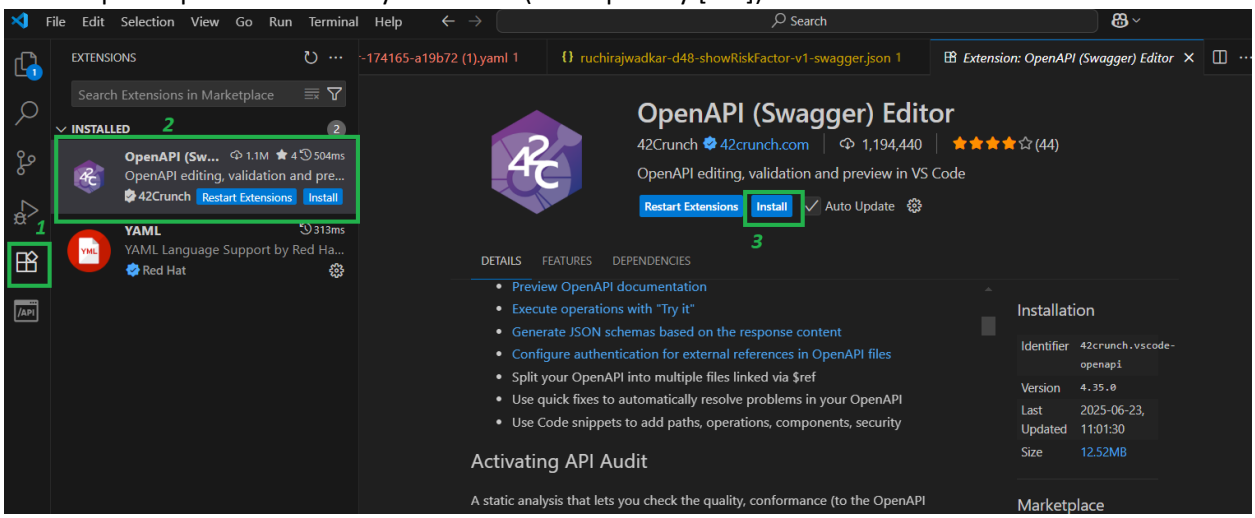
5. `security`
Механизмы аутентификации и авторизации.

Практические рекомендации

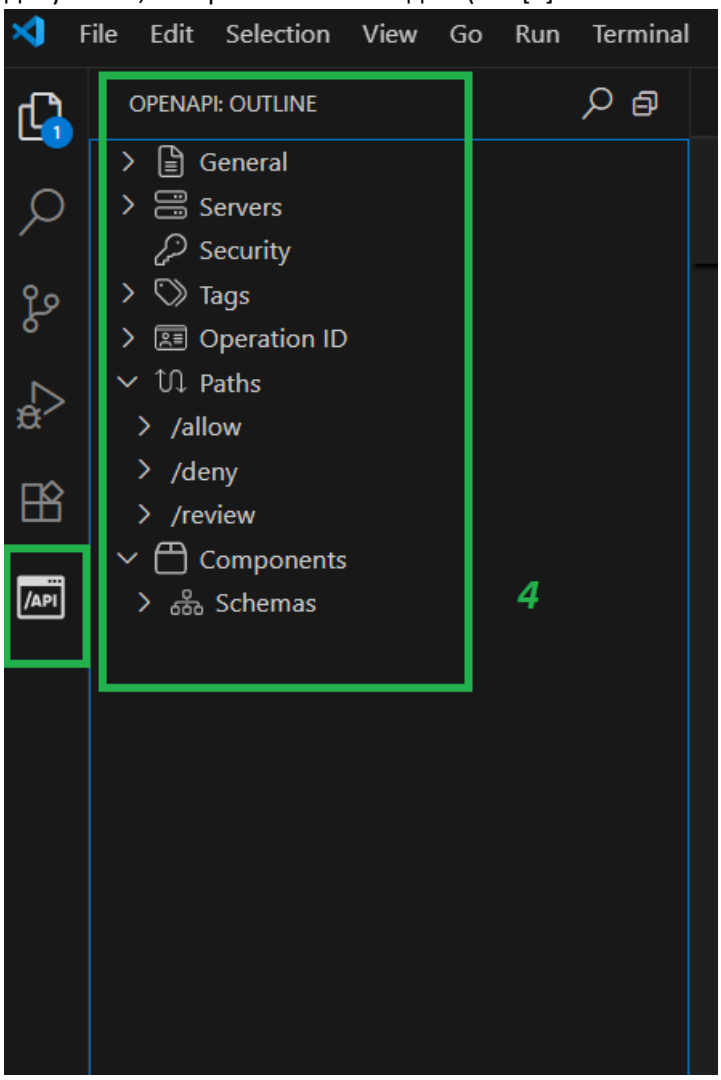
1. Стандартизация описаний: Используйте компоненты (`components`) для повторно используемых схем и параметров.
2. Версионирование: Обновляйте спецификацию при изменениях API.
3. Автоматизация: Интегрируйте генерацию документации и тестирование в CI/CD процессы.
4. Безопасность: Описывайте механизмы аутентификации (`securitySchemes`).

Помимо общего теоретического материала, рассмотрим на примере (в текущем примере и в домашнем задании предложено использование расширения Swagger Editor для VS Code):

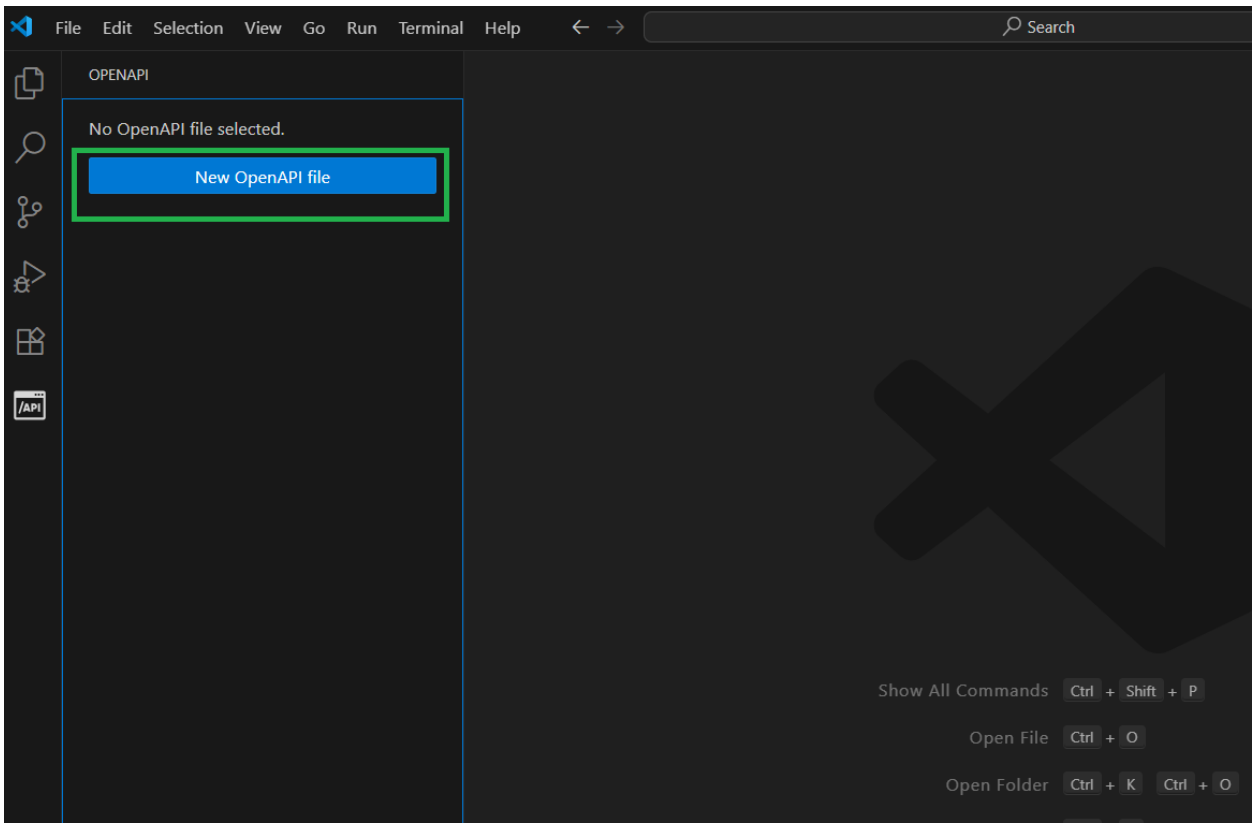
Для того чтобы установить расширение OpenApi Swagger необходимо открыть VC Code-> расширения и нажать установить (см. картинку [1-3]):



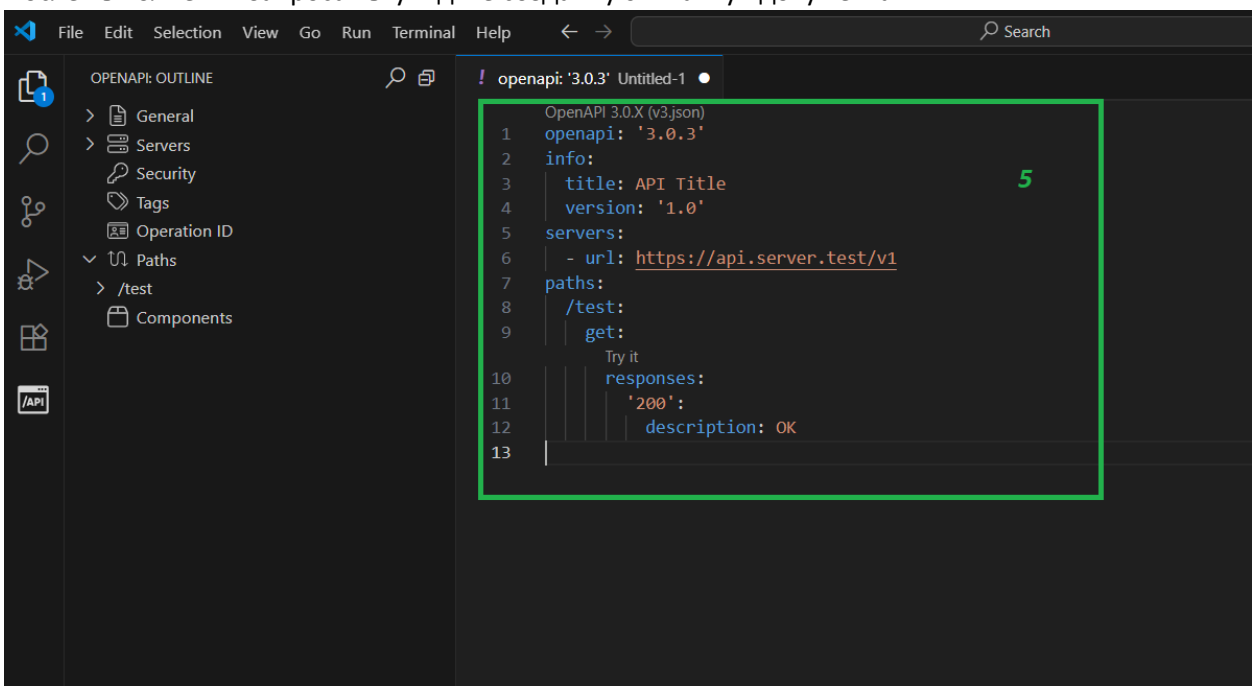
В левом боковом меню будет раздел OpenAPI, в нем вы очень быстро сможете найти тот фрагмент документа, который вам необходим (см. [4]:



После установки выбираем «New OpenApi file»



После выполнения запроса вы увидите созданную «шапку» документа:



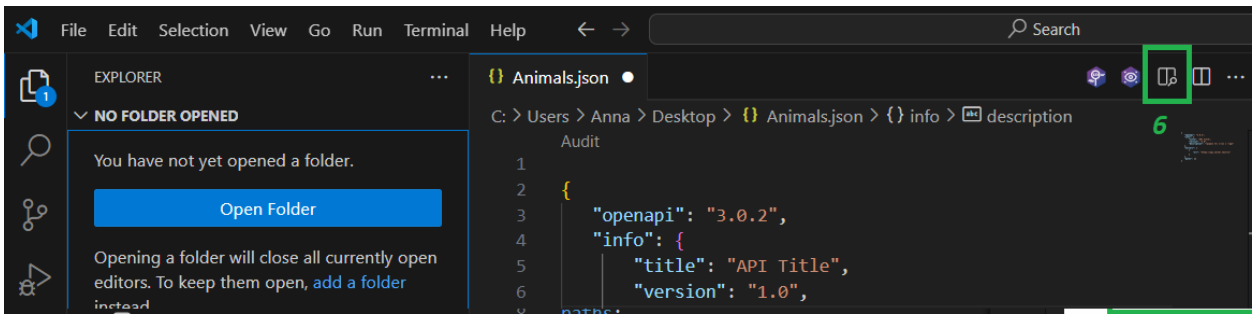
После этого сохраните файл (“Cmd+S” на Mac или “Файл => Сохранить”) в формате json в удобном месте.

(ВНИМАНИЕ! Обратите внимание, что после сохранения в формате JSON синтаксис изменится)

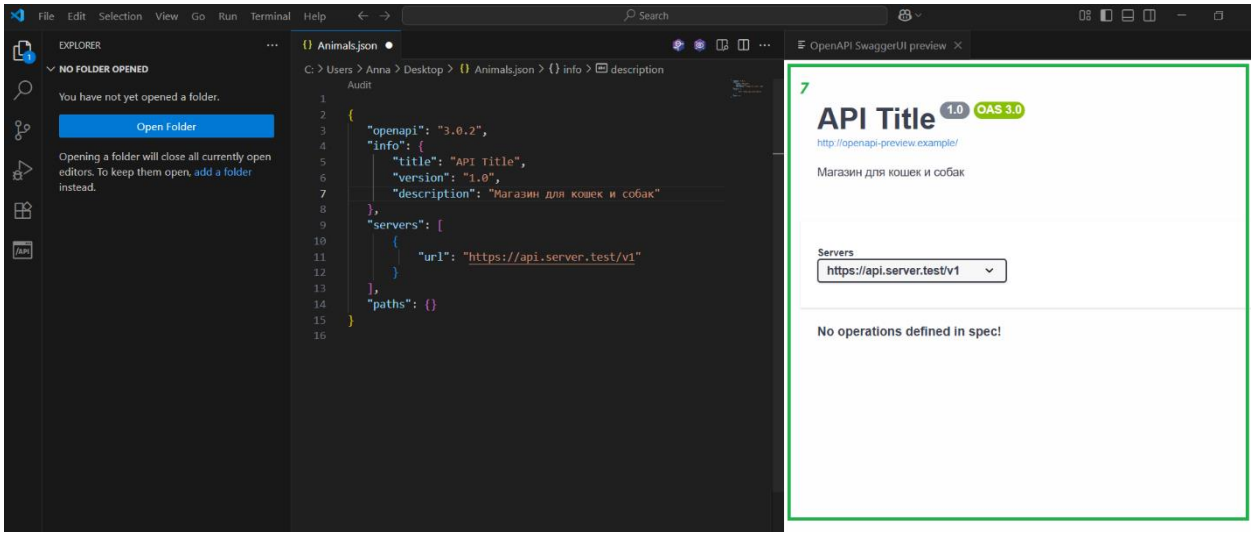
Запись «openapi»: «3.0.3» говорит о том, что документ описан в спецификации OpenAPI “3.0”. От версии OpenAPI зависит структура документа и доступность некоторых функций.

Объект «info» содержит параметры по умолчанию: «title» — заголовок и «version» — версия именно нашего API.

Если для удобства вам необходимо видеть UI версию Swagger нажмите кнопку [6]



После выполнения запроса вы увидите превью в Swagger-UI [7]



Рассмотрим пример написания спецификации для сайта бронирования отелей.

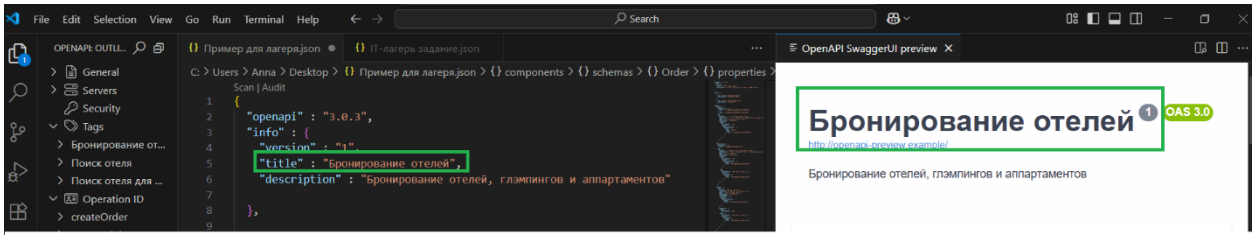
В **рамках примера** мы рассмотрим как реализовать спецификацию на методе получения списка отелей:

-получение списка отелей

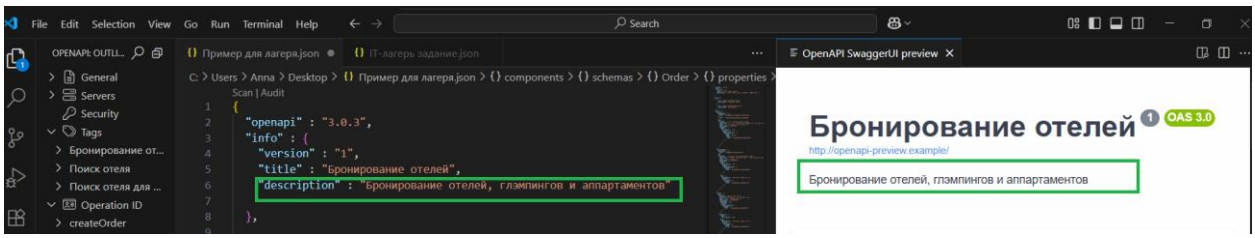
Два метода вам будет предложено доделать в рамках домашнего задания:

-получение выбранного отеля
-бронь выбранного номера

Начнем с изменения шапки спецификации,в частности, ее названия:



Добавим описание:



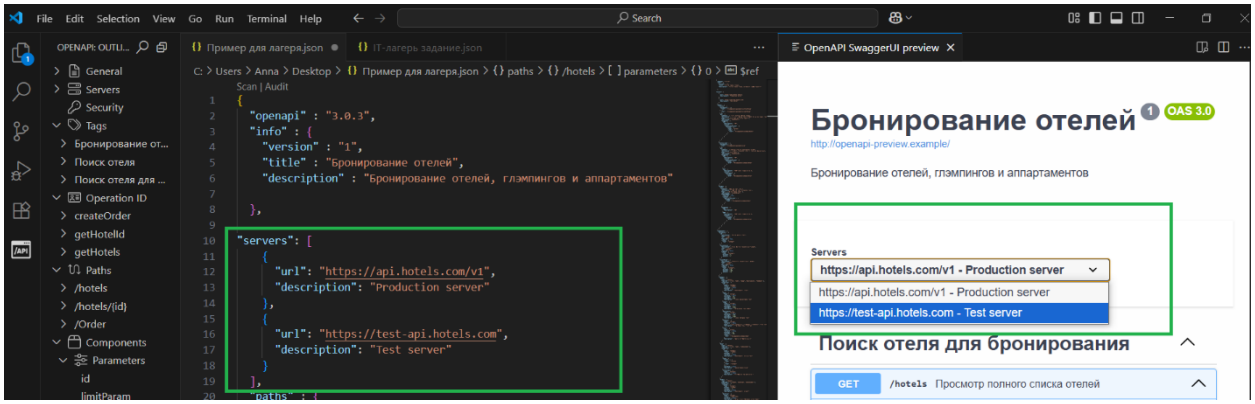
Объект «servers» содержит информацию о том, на каких серверах доступна реализуемая API. Эти данные обычно предоставляют разработчики после завершения работы над сервисом, и их необходимо включить в документацию. На этапе проектирования обычно указываются временные «заглушки» — тестовые серверы.

Перечисленные в этом разделе серверы будут отображаться в виде выпадающего списка в интерактивной документации, что позволит пользователю выбрать нужный вариант. Если на выбранных серверах действительно запущена API-реализация, пользователь сможет отправлять запросы и получать ответы — как тестовые, так и реальные.

Для каждого сервера задаем параметры:

«url» — адрес конечной точки сервера; «description» — краткое описание назначения этого сервера.
Добавим тестовый и продовый сервер:

```
"servers": [
  {
    "url": "https://api.hotels.com/v1",
    "description": "Production server"
  },
  {
    "url": "https://test-api.hotels.com",
    "description": "Test server"
  }
]
```



В объекте «paths» хранится вся информация о доступных в API методах и их реализации.
Endpoints (эндпойнты далее) и HTTP-заголовки указываются внутри объекта «paths».
В рамках нашего примера необходимо будет реализовать следующие методы:

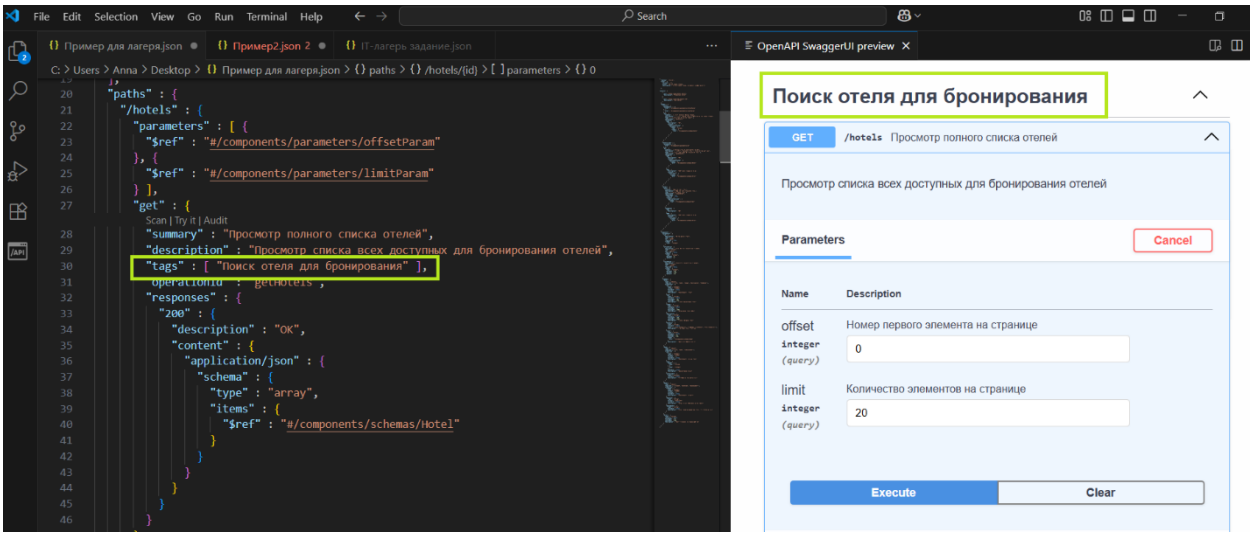
- Получение списка отелей (GET /hotels)
- Создание бронирования/заказа (POST /order)
- Получение выбранного отеля и его параметров GET/hotels/{id})

HTTP-заголовки для методов get, post и других указываются внутри соответствующих разделов эндпойнтов

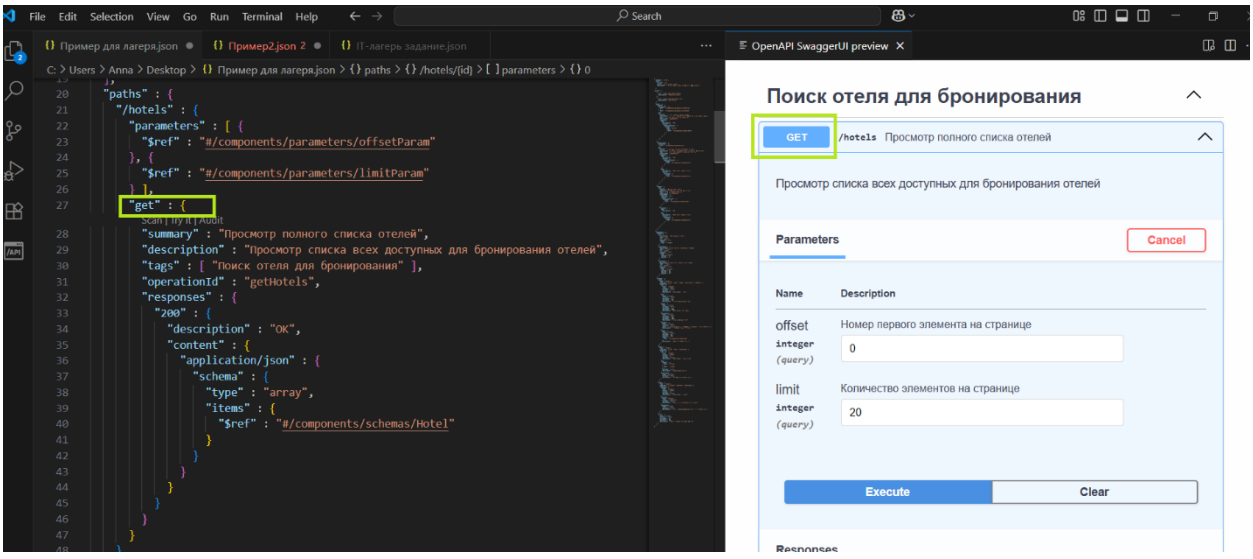
В каждом методе заполним следующую информацию:

- «tags» – используется для группировки методов по типу объекта. Обычно используется один тег на несколько методов.
- «summary» – краткое описание метода, которое отображается рядом с названием HTTP глагола
- «operationId» — уникальный идентификатор операции. Используется для точной идентификации метода при разборе ошибок. Идентификатор должен быть уникальными среди всех операций, описанных в API.

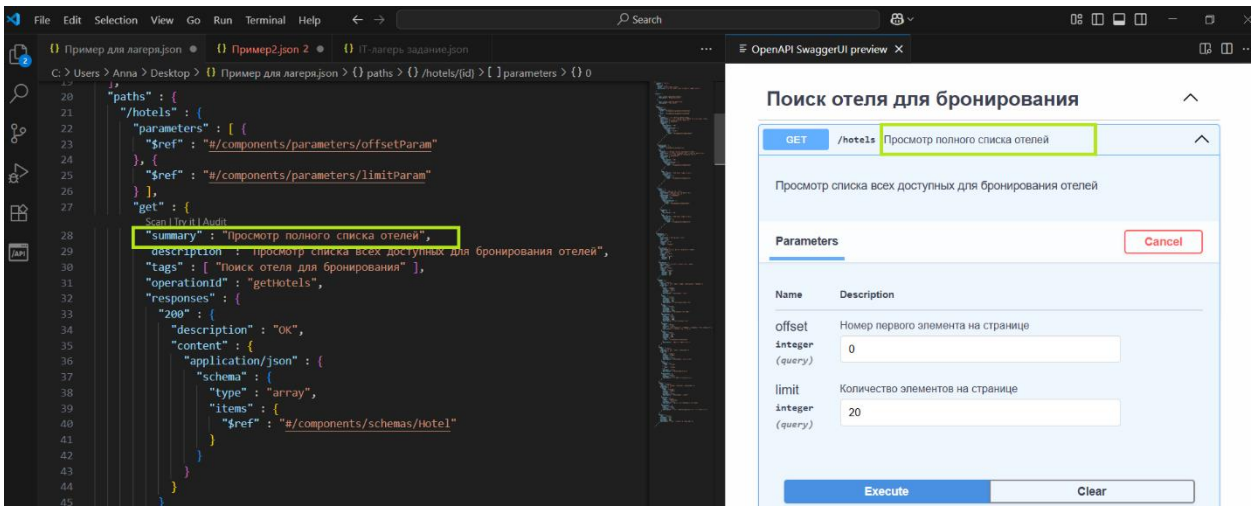
Как выглядит тег в методе:



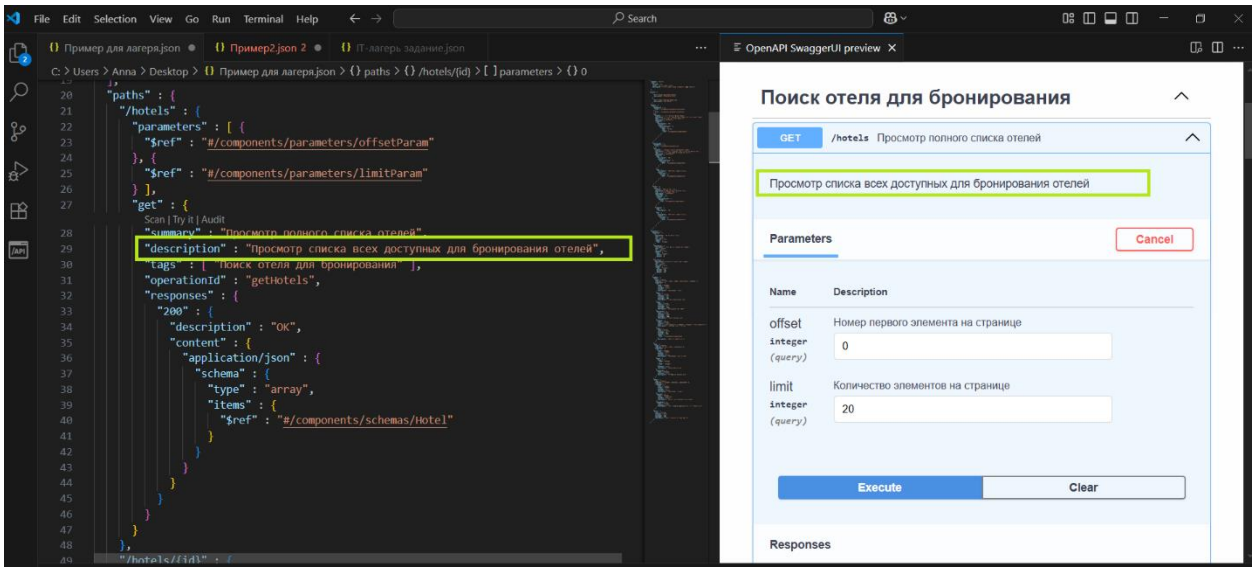
Сам метод:



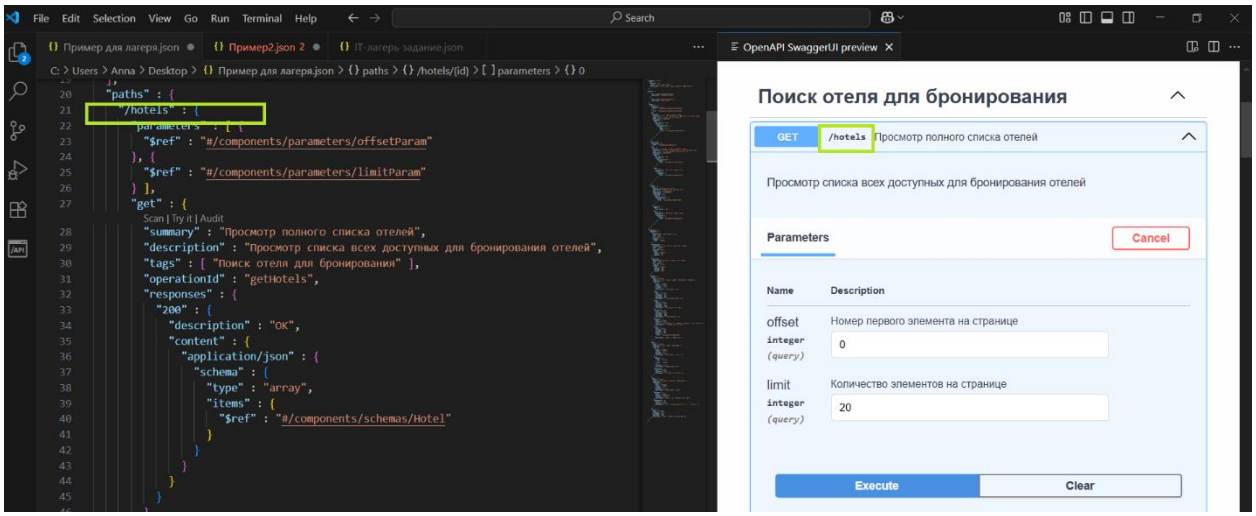
Заголовок:



Описание:



Эндпойнт:



Тело запроса и содержимое ответа для методов описываются в следующих объектах:

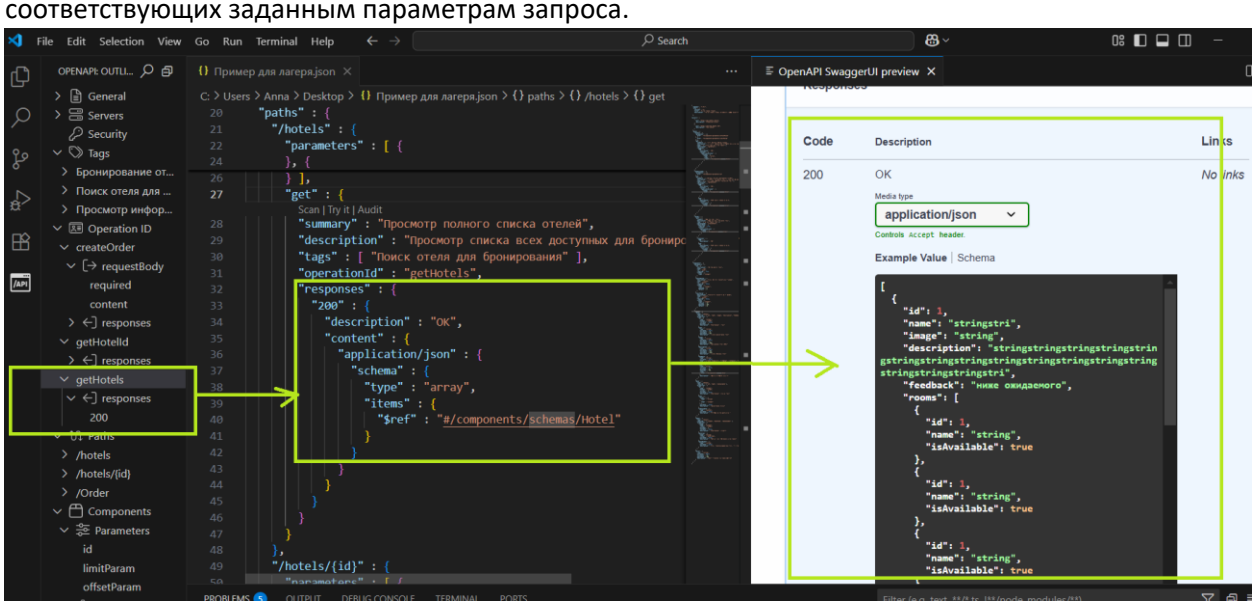
- «requestBody» — содержит тело запроса
- «responses» — содержит ответ на запрос

Для таких методов, как метод создания (POST) и обновления (PATCH/PUT) в «requestBody» необходимо передать все параметры, за исключением ID. **(пример см. далее по тексту после описания schemas)**

При создании объекта ID генерируется автоматически на уровне базы данных. При обновлении карточки ID передается в URL запроса и служит для точной идентификации объекта, который нужно изменить (параметр Path).

При выполнении запроса параметры, указанные в «requestBody», отправляются на сервер. Как при создании нового объекта, так и при обновлении существующего, успешное выполнение запроса приведет к изменению данных в базе сервиса.

Для метода получения списка объектов (GET) «requestBody» не заполняется, поскольку GET-запрос предназначен только для получения данных с сервера и не создает или обновляет записи, как POST и PATCH/PUT.
Опишем информации в блоке «reponses».
В объект **“responses”** передаются технические метрики запроса, а также значимые для конечного пользователя данные.
Для методов POST и PATCH, при успешном выполнении запроса возвращается информация о созданном объекте.
Для метода GET, при успешном выполнении запроса, возвращается список объектов, соответствующих заданным параметрам запроса.

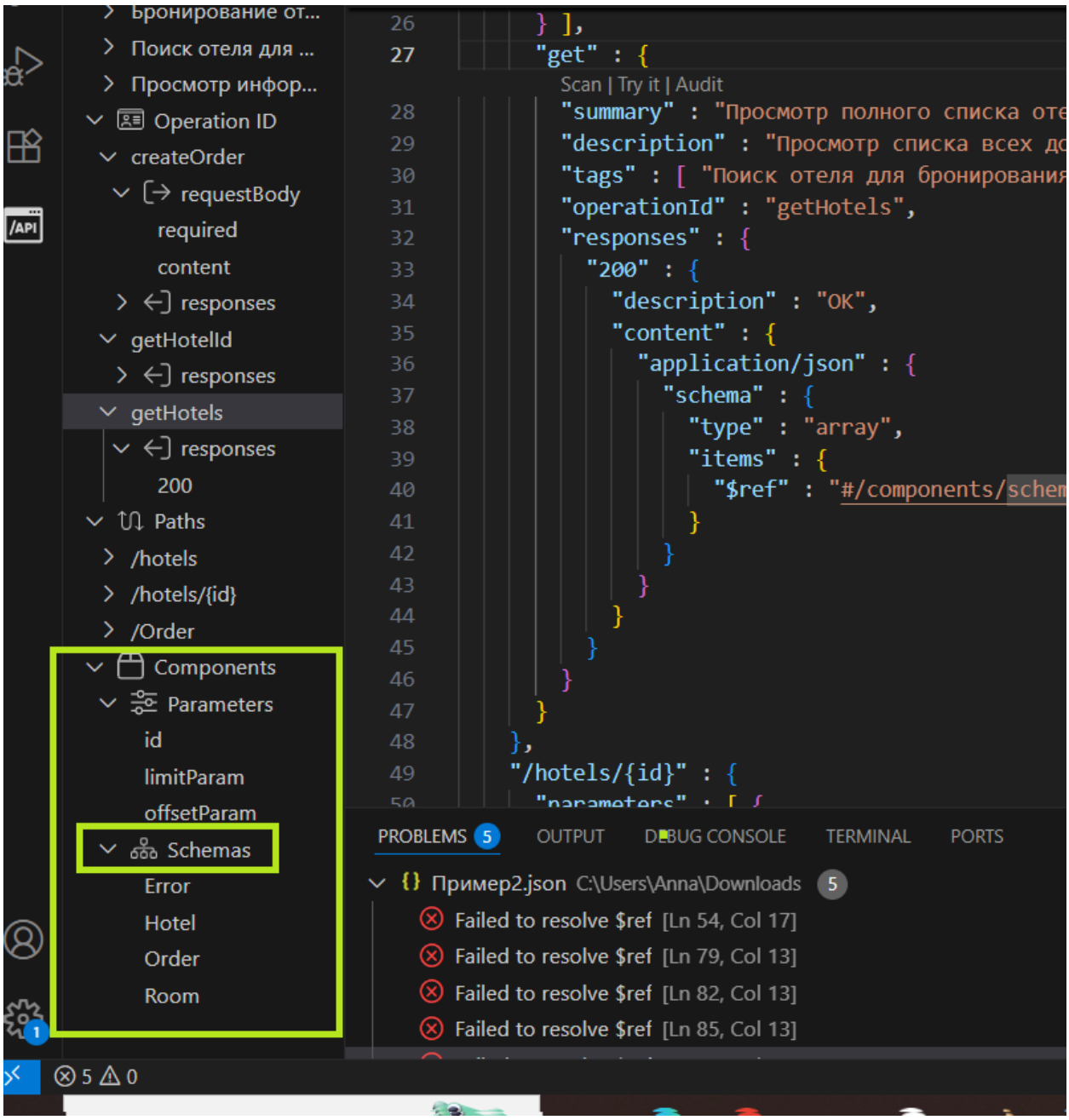


Все ответы блока **“responses”** разделены по кодам ответа. Примеры:

- 200 — запрос успешно выполнен
- 500 – внутренняя ошибка сервера

Исключение дублей в документе и оптимизация описания:

Зачастую в документе будет много повторов, поскольку есть типовые ошибки, есть одинаковые параметры, а также может быть однотипный результат
Дублирующиеся блоки информации описываются в объект «schemas», который вложен в объект «components».



Везде, где необходимо использовать повторяющиеся блоки, указываются ссылки на нужные компоненты.

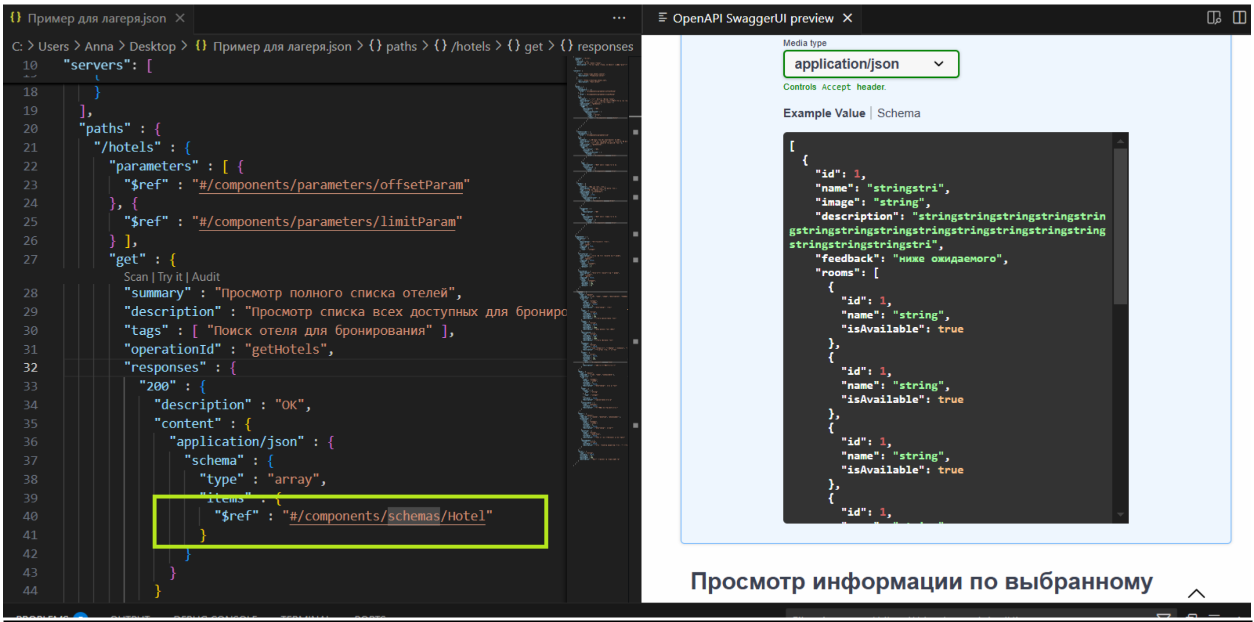
Ссылка на схему выглядит так:

«$ref»: «#/components/schemas/{Название компонента}»

Объявление схемы:

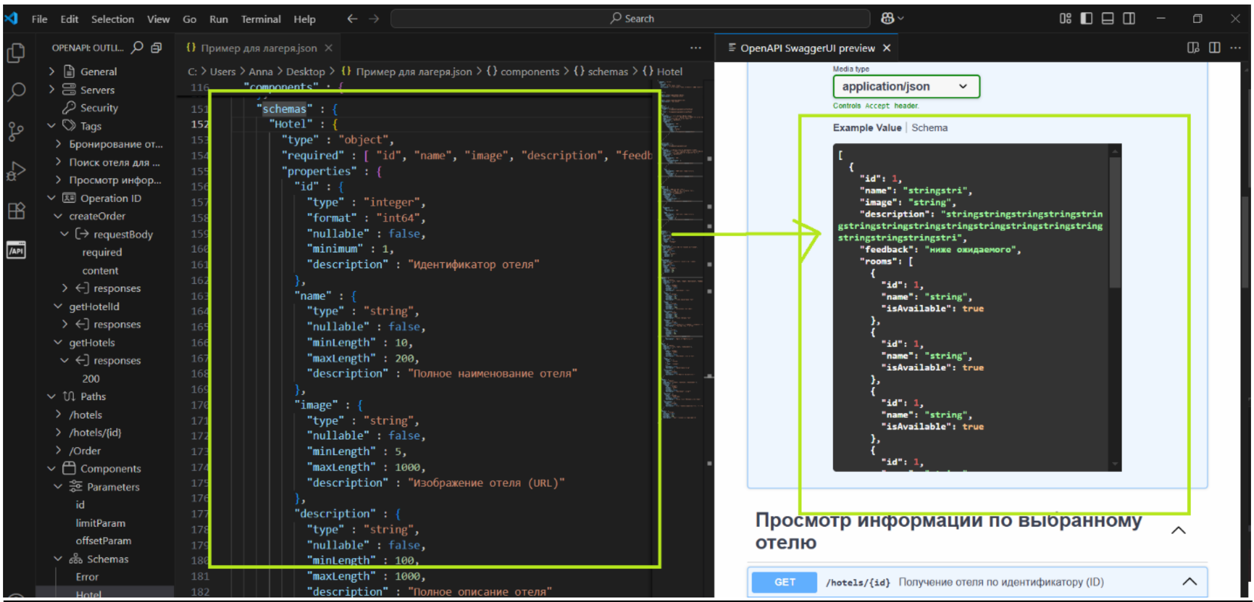
"components": {
"schemas": {
"Название компонента": {
<описание параметров>
}
}
}

На примере метода GET /hotels, указываем ссылку на схему:



Описываем в схеме параметры объекта, который мы хотим получить (на данном примере «Отель», он включает в себя наименование, изображение, рейтинг/обратную связь, наименование, комнаты, а также уникальный Id самого отеля и комнат).

Описанный нами объект, и указанный в виде ссылки в результате мы видим на превью в блоке «Responses»:



Преимущества, которые появляются, если корректно формировать объект «components»:

1. **Скорость, экономия времени.**

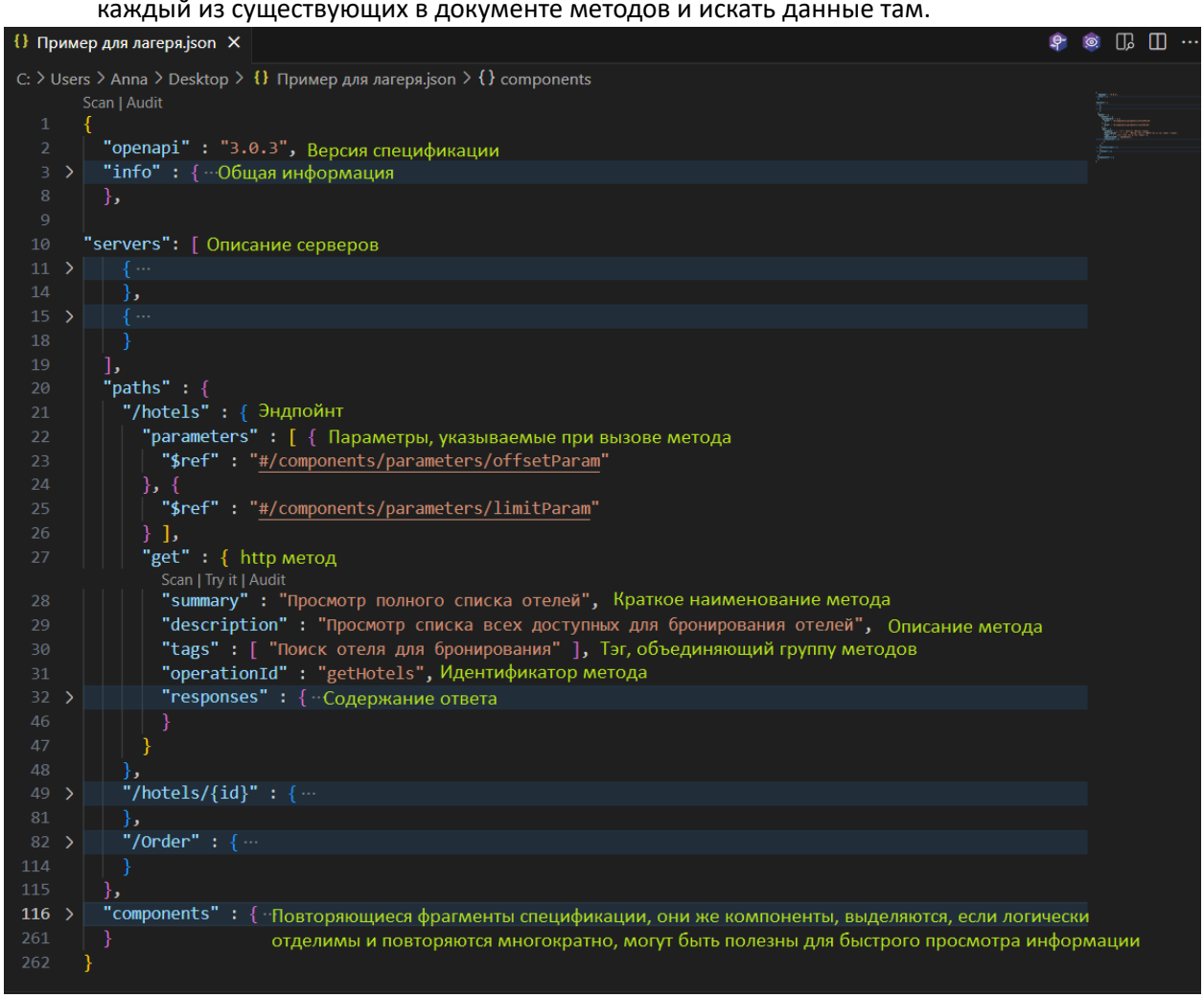
   Один раз описываешь нужный фрагмент кода в «components» и ссылаешься на него, где необходимо.

2. **“Чистота” документации.**

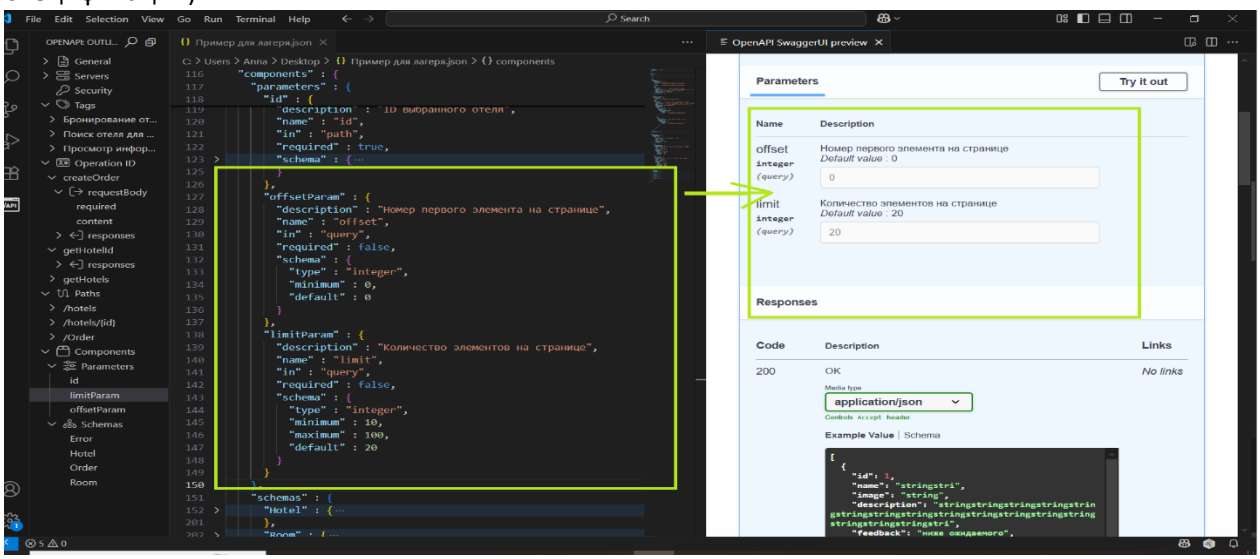
   При использовании такого подхода, документ легче читается и лучше выглядит. Значительно уменьшается количество строк.

3. **Удобство изучения документации.**

Если нужно понять каков атрибутивный состав объектов или структуру запросов/ответов, удобней и быстрее – посмотреть нужную информацию в компонентах, а не разворачивать каждый из существующих в документе методов и искать данные там.



В блоке «Parameters» (на примере для метода GET/hotels) по аналогии с группировкой в Schemas в виде ссылки указаны параметры пагинации (limit и offset), параметры пагинации описанные в спецификации отображены на превью (таким образом можно сгруппировать любые повторяющиеся параметры, которые будут подаваться на вход в запросе методов, текущей спецификации)



Полное описание правил описания : https://docs.swagger.io/spec.html