

Допустим, есть некое приложение, которое среди прочего функционала позволяет работать со статьями (создавать/редактировать/удалять). Любое действие, которое при этом выполняется – отправляет АПИ запрос на сервер. Каждый эндпойнт – определенное действие. При чем у эндпойнта есть определенные свойства. В частности, REST принципы предполагают, что структура идентификатора должна быть простой, предсказуемой и понятной. URI также должны быть статичными, чтобы при изменениях ресурсов или реализации сервиса ссылка оставалась той же. Один из способов достичь такого уровня удобства пользования - построение URI по аналогии со структурой каталогов (почти как путь до файла на локальном компьютере). На примере ниже можем интуитивно, что, например, запрос GET /api/articles вернет все созданные статьи.

Запрос GET /api/article/{id} вернет данные по конкретной статье (потому что мы введем ее id).

Запрос Delete /api/article/{id} удалит определенную статью по ее id и тд.

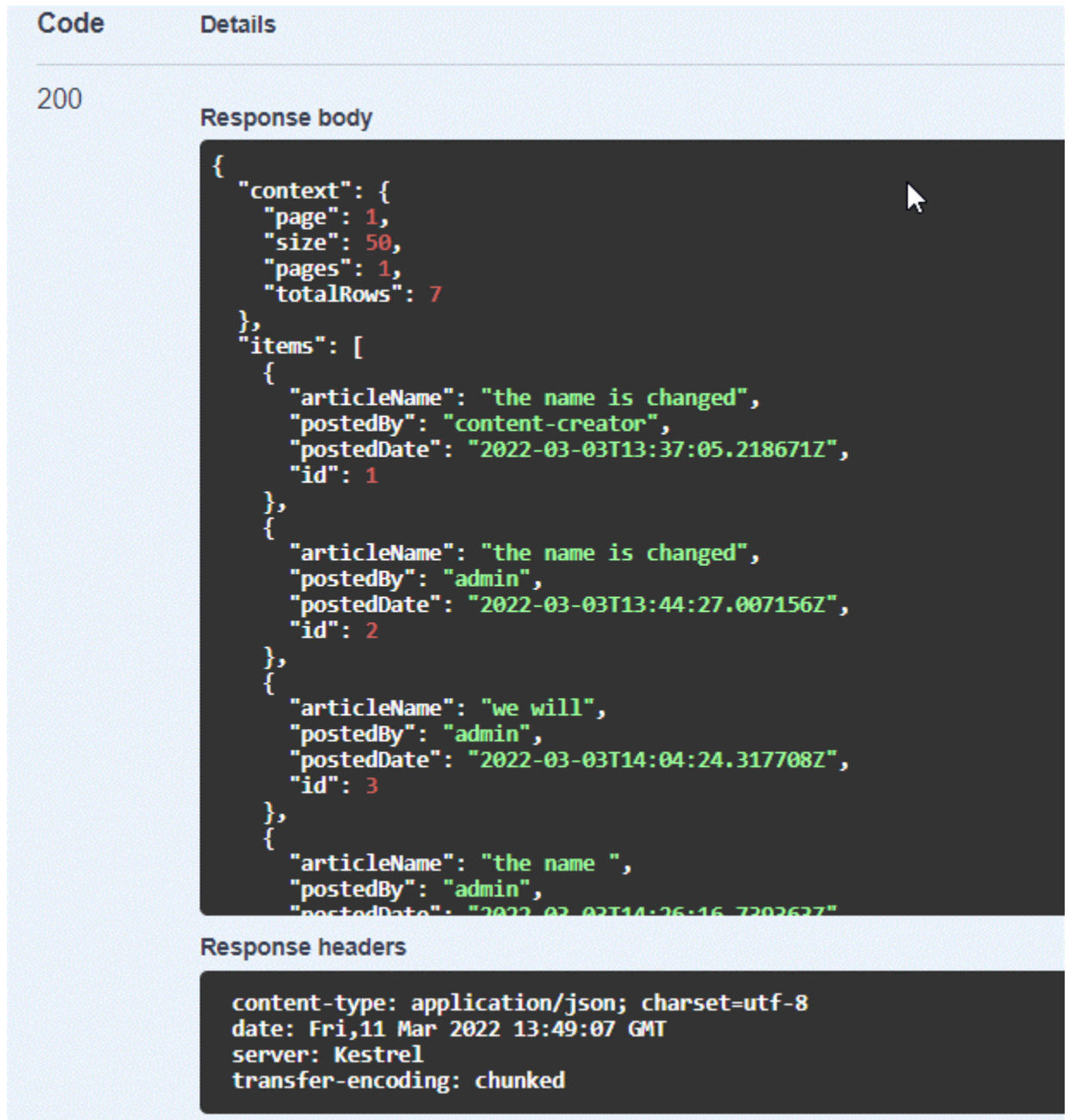
articles	
GET	/api/articles
GET	/api/article/filter/author
GET	/api/article/{id}
DELETE	/api/article/{id}
PUT	/api/article/{id}
POST	/api/article
PUT	/api/article/{id}/upload

Дальше – например, нам нужно протестировать первый апи запрос:

GET /api/articles.

Метод GET значит, что мы просто запрашиваем определенную информацию – и сервер нам ее предоставляет.

Отправляем запрос (через постман или, как в моем случае, с помощью сваггера). Сервер нам возвращает ответ:



The screenshot shows a Swagger UI interface with two tabs: 'Code' and 'Details'. The 'Code' tab is active, displaying a 200 status code. The 'Details' tab is also visible, showing the 'Response body' and 'Response headers'. The response body is a JSON object containing a 'context' object and an 'items' array. The 'context' object has 'page': 1, 'size': 50, 'pages': 1, and 'totalRows': 7. The 'items' array contains three article objects. The first article has 'articleName': 'the name is changed', 'postedBy': 'content-creator', 'postedDate': '2022-03-03T13:37:05.218671Z', and 'id': 1. The second article has 'articleName': 'the name is changed', 'postedBy': 'admin', 'postedDate': '2022-03-03T13:44:27.007156Z', and 'id': 2. The third article has 'articleName': 'we will', 'postedBy': 'admin', 'postedDate': '2022-03-03T14:04:24.317708Z', and 'id': 3. The response headers are 'content-type: application/json; charset=utf-8', 'date: Fri, 11 Mar 2022 13:49:07 GMT', 'server: Kestrel', and 'transfer-encoding: chunked'.

```
200

Response body

{
  "context": {
    "page": 1,
    "size": 50,
    "pages": 1,
    "totalRows": 7
  },
  "items": [
    {
      "articleName": "the name is changed",
      "postedBy": "content-creator",
      "postedDate": "2022-03-03T13:37:05.218671Z",
      "id": 1
    },
    {
      "articleName": "the name is changed",
      "postedBy": "admin",
      "postedDate": "2022-03-03T13:44:27.007156Z",
      "id": 2
    },
    {
      "articleName": "we will",
      "postedBy": "admin",
      "postedDate": "2022-03-03T14:04:24.317708Z",
      "id": 3
    },
    {
      "articleName": "the name ",
      "postedBy": "admin",
      "postedDate": "2022-03-03T14:26:16.720262Z",
      "id": 4
    }
  ]
}

Response headers

content-type: application/json; charset=utf-8
date: Fri, 11 Mar 2022 13:49:07 GMT
server: Kestrel
transfer-encoding: chunked
```

На нем мы видим, что запрос прошел успешно: статус код 200 OK и в Body находится JSON-файл, в котором содержится запрашиваемая информация, а именно – данные по статьям (имя, кем опубликована, дата публикации и id).

Это – успешный вариант тестирования. Какие могут быть ошибки посмотрим дальше.

Возьмем запрос POST /api/articles., который, как мы понимаем из самого эндпойнта, позволит опубликовать статью.

Чтобы отправить его, нам нужно добавить в запрос JSON файл собственно с информацией в статье, которую мы хотим опубликовать.

Request body

file  
string(\$binary)  No file chosen

☒ Send empty value

data  
object

```
{  
  "name": "string",  
  "text": "string"  
}
```

☐ Send empty value

Json характеризуется тем, что внутри него есть набор неупорядоченных (могут идти в любом порядке, значения не имеет) пар «значение-ключ». Что мы видим в этом конкретном json-е: две пары обязательных данных – название статьи и сам текст. Дефолтное значение String мы заменяем своим, и в таком виде отправляем на сервер.

data  
object

```
{  
  "name": "Here's the name",  
  "text": "Here's the text"  
}
```

Сервер нам присылает ответ:

Code	Details
400	<p>Error: Bad Request</p> <p>Response body</p> <pre>{   "success": false,   "code": "validation",   "message": "'File' must not be empty." }</pre> <p>Response headers</p> <pre>content-type: application/json; charset=utf-8 date: Fri, 11 Mar 2022 13:57:11 GMT server: Kestrel transfer-encoding: chunked</pre>

400 ошибка, а следом сообщение – ‘File’ must not be empty.

В данном случае нашими программистами стоит ограничение на создаваемые статьи – они должны идти только с картинками присоединенными. Поэтому мы снова идем в запрос и присоединяем картинку в поле File, после чего запрос отправляем.

Code	Details
200	<p>Response body</p> <pre>{   "name": "Here's the name",   "text": "Here's the text",   "images": [     {       "imageType": "ScaleHeight800",       "imageUrl": "https://d34ci584ct75fg.cloudfront.net/78acb69b-bdbb-4787-8c9d-07ac01717f7e.jpg"     },     {       "imageType": "Original",       "imageUrl": "https://d34ci584ct75fg.cloudfront.net/1cd55e58-7d48-49fb-86ba-8fbfbf9af.jpg"     }   ],   "interestTags": [],   "id": 8 }</pre> <p>Response headers</p> <pre>content-type: application/json; charset=utf-8 date: Fri, 11 Mar 2022 14:00:24 GMT server: Kestrel transfer-encoding: chunked</pre>

Видим, что нам прилетел замечательный ответ от сервера – 200 OK, а в response body информация по созданной статье – имя, текст, данные по приложенной картинке, id. Так же есть параметр

“interestTags”, но его мы не указали в request body, поэтому значение в скобках не указано. Соответствующая запись также была добавлена в базу данных.

Но это только позитивный сценарий тестирования. Нам нужно проверить, что негативные кейсы тоже сервером обрабатываются адекватно. Поэтому берем и убираем вообще всю информацию из request body:

file  
string(\$binary) Choose File No file chosen

☒ Send empty value

data  
object

☒ Send empty value

Сервер присылает в ответ

**Code** Details

400

Error: Bad Request

Response body

```
{
  "success": false,
  "code": "validation",
  "message": "data is empty."
}
```

Значит все ок. Сигнализирует о том, что данные пусты.

Пробуем отправить запрос не с полностью пустым телом, а убрав один из параметров:

file  
string(\$binary)  No file chosen

☒ Send empty value

data  
object

```
{  
  "name": "Here's the name"  
}
```

Что присылает сервер в ответ:

Code	Details
400	<p>Error: Bad Request</p> <p>Response body</p> <pre>{   "success": false,   "code": "validation",   "message": "'Text' must not be empty." }</pre>

Тоже все ок, наш запрос он никуда не добавил, отправил нам обратно с сообщением о том, что «Text» – параметр обязательный и он должен быть заполнен.

Какие еще ошибки часто встречаются:

Если я разлогинюсь и попробую выполнить любой из запросов, от сервера придет

Code	Details
401	<p>Error: Unauthorized</p> <p>Response headers</p> <pre>content-length: 0 date: Fri, 11 Mar 2022 14:04:09 GMT server: Kestrel www-authenticate: Bearer</pre>

401 – не авторизован.

Если я залогинюсь пользователем, которому доступ в этот раздел запрещен и отправлю запрос, то увижу:

Code	Details
403	<div><div>Error: Forbidden</div><div>Response headers</div><div><div>content-length: 0</div><div>date: Fri, 11 Mar 2022 14:06:27 GMT</div><div>server: Kestrel</div></div></div>
Responses	

Соответственно наша задача подобным образом протестировать доступные апишки, чтобы нам возвращался 200 ответ и необходимые данные в нем. Пробовать неверные Id, невалидные данные, проверять граничные значения (возвращает ли сервер ошибку о том, что превышено допустимое количество символов) и т.д.

Интереса ради можно для себя посмотреть пример вот таких запросов, относящихся к функционалу чата, чтобы понять, какой запрос что выполнит:

## mobile-instant-messaging

**GET** /api/mobile/instant-messaging/chat/configuration

**GET** /api/mobile/instant-messaging/chats

**GET** /api/mobile/instant-messaging/chat/{id}

**DELETE** /api/mobile/instant-messaging/chat/{id}

**POST** /api/mobile/instant-messaging/chat

**POST** /api/mobile/instant-messaging/chat/{id}/invite

**POST** /api/mobile/instant-messaging/chat/{id}/leave

**POST** /api/mobile/instant-messaging/chat/{id}/mute

**POST** /api/mobile/instant-messaging/chat/{id}/unmute

**DELETE** /api/mobile/instant-messaging/chat/{chatId}/user/{userId}

**GET** /api/mobile/instant-messaging/chat/{id}/messages

**POST** /api/mobile/instant-messaging/chat/{id}/message