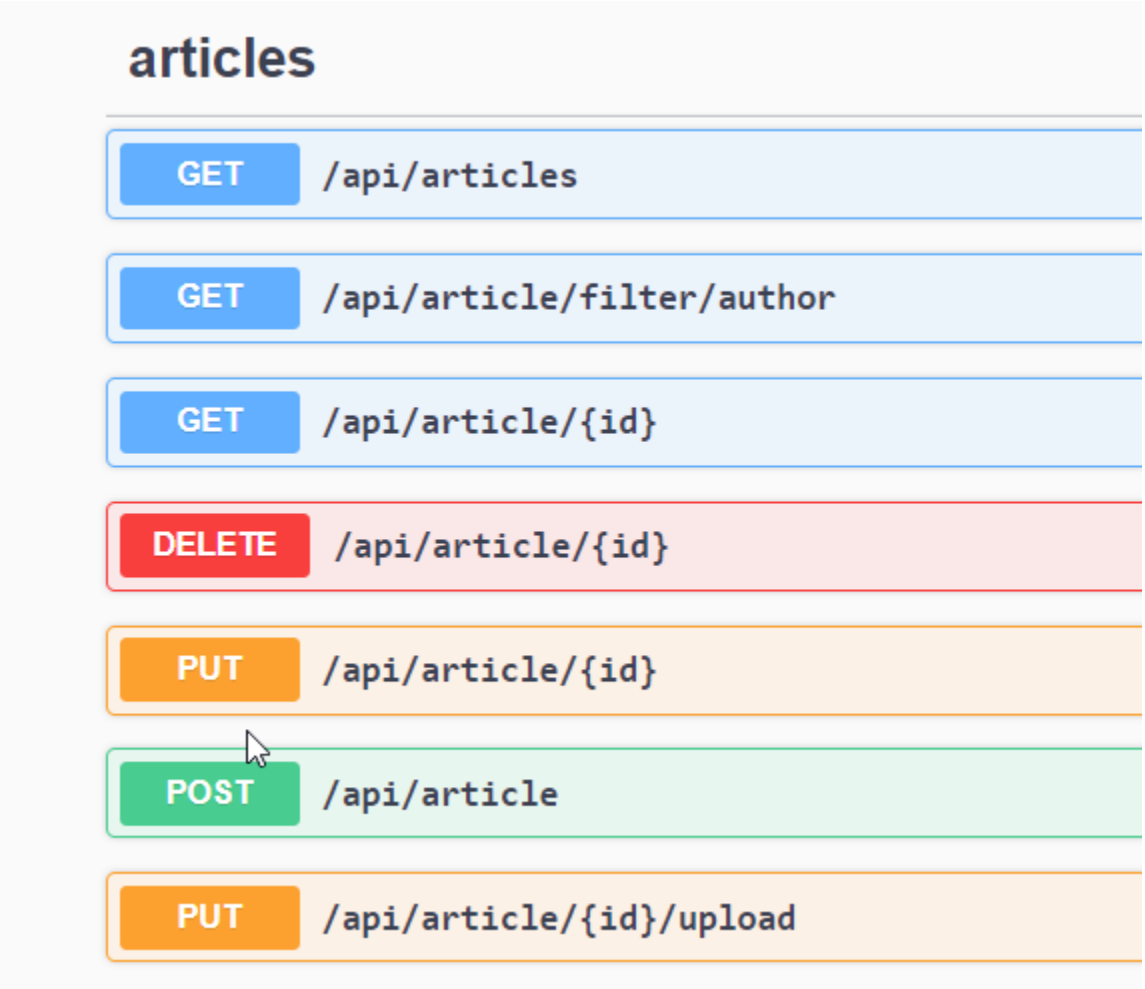
Допустим, есть некое приложение, которое среди прочего функционала позволяет работать со статьями (создавать/редактировать/удалять). Любое действие, которое при этом выполняется – отправляет АПИ запрос на сервер. Каждый эндпойнт – определенное действие. При чем у эндпойнта есть определенные свойства. 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.

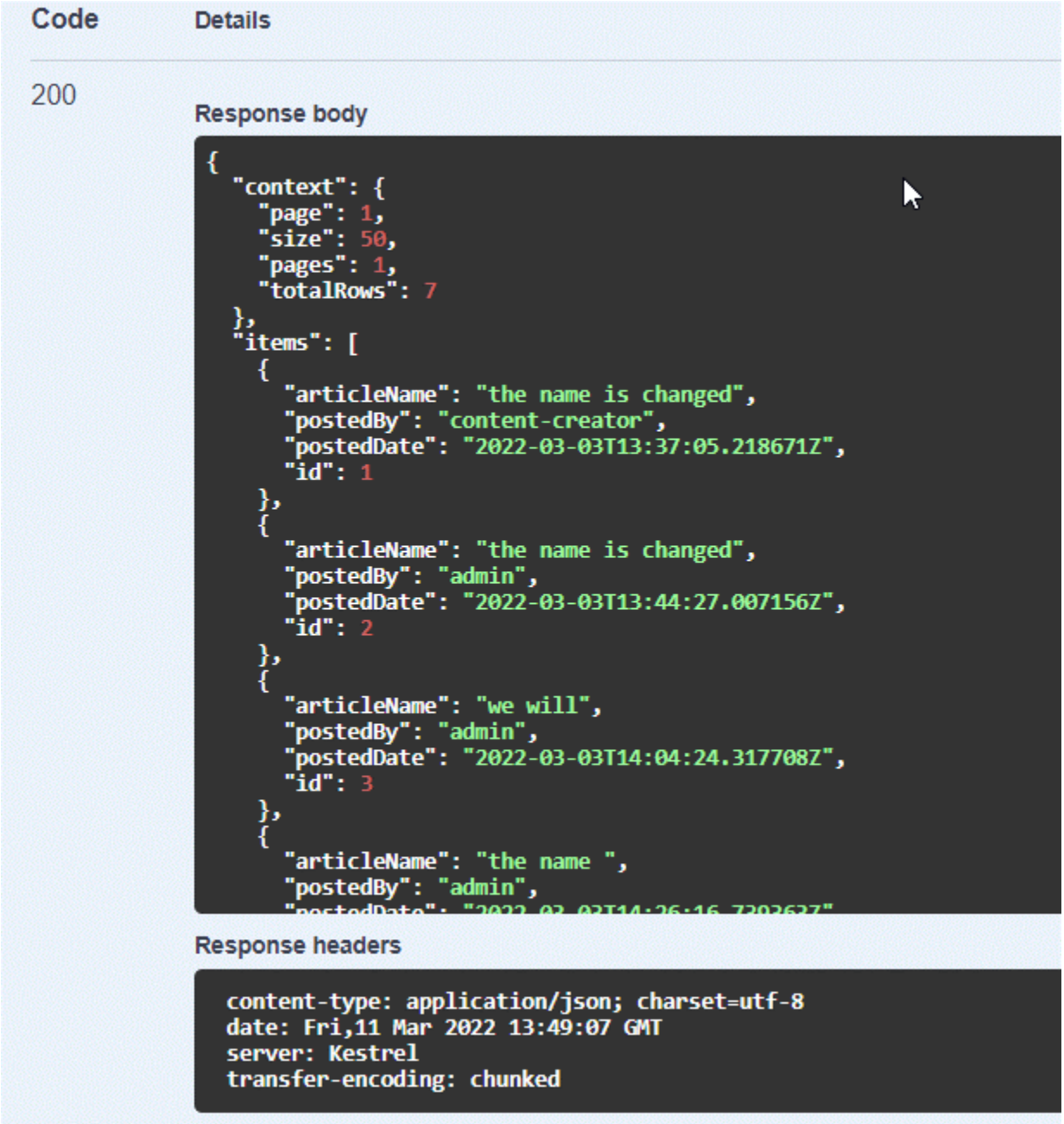


Дальше – например, нам нужно протестировать первый апи запрос:

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:

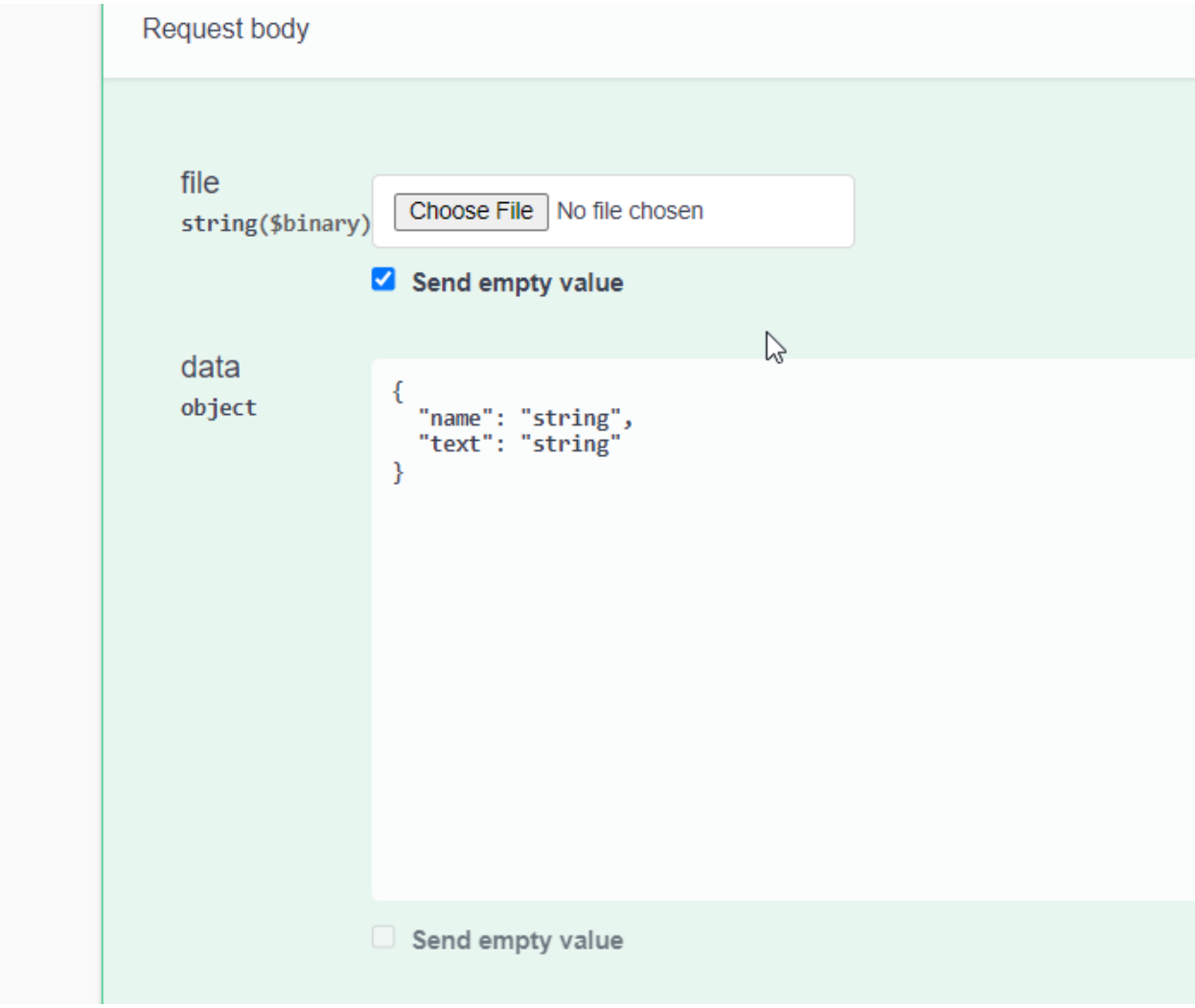


На нем мы видим, что запрос прошел успешно: статус код 200 ОК и в Body находится JSON-файл, в котором содержится запрашиваемая информация, а именно – данные по статьям (имя, кем опубликована, дата публикации и id).

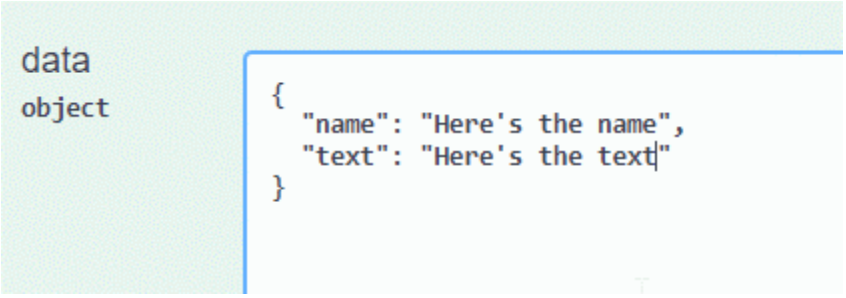
Это – успешный вариант тестирования. Какие могут быть ошибки посмотрим дальше.

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

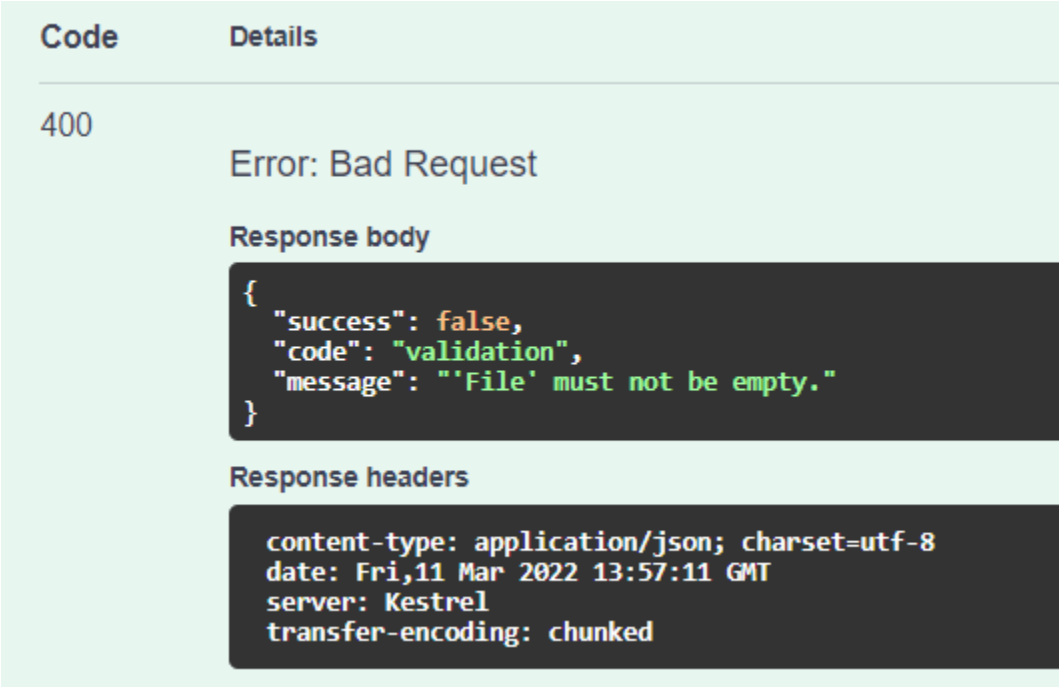
Чтобы отправить его, нам нужно добавить в запрос JSON файл собственно с информацией в статье, которую мы хотим опубликовать.



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.

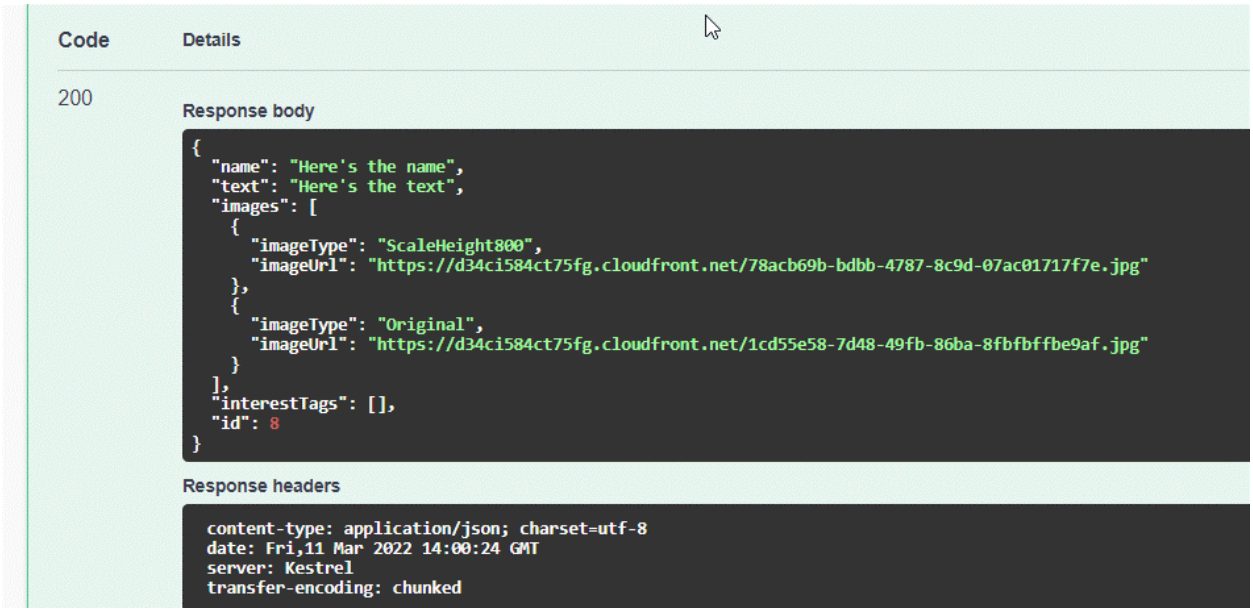


Сервер нам присылает ответ:



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

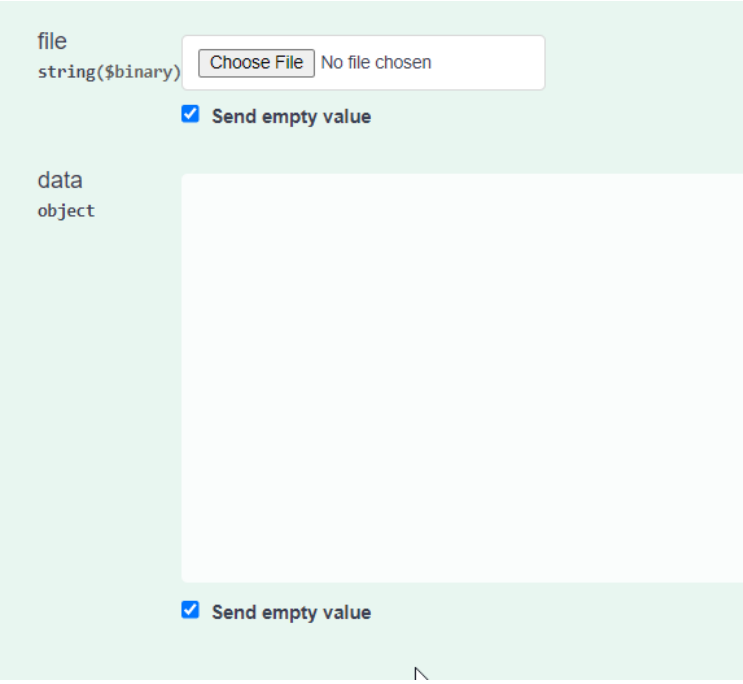
В данном случае нашими программистами стоит ограничение на создаваемые статьи – они должны идти только с картинками присоединенными. Поэтому мы снова идем в запрос и присоединяем картинку в поле File, после чего запрос отправляем.



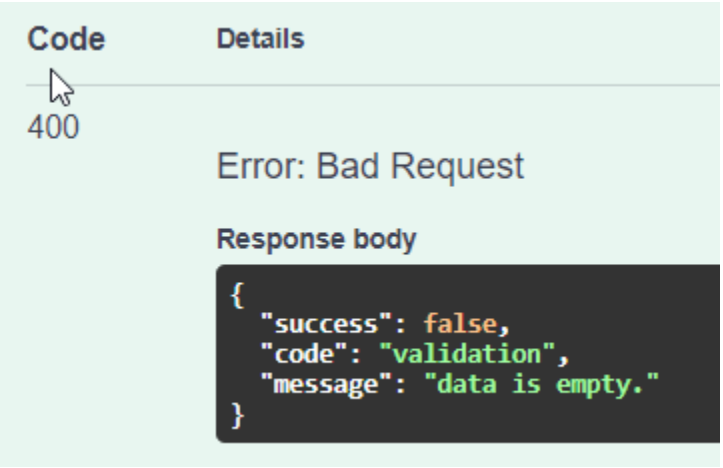
Видим, что нам прилетел замечательный ответ от сервера – 200 ОК, а в response body информация по созданной статье – имя, текст, данные по приложенной картинке, id. Так же есть параметр

“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 обработаются адекватно. Поэтому берем и убираем вообще всю информацию из request body:

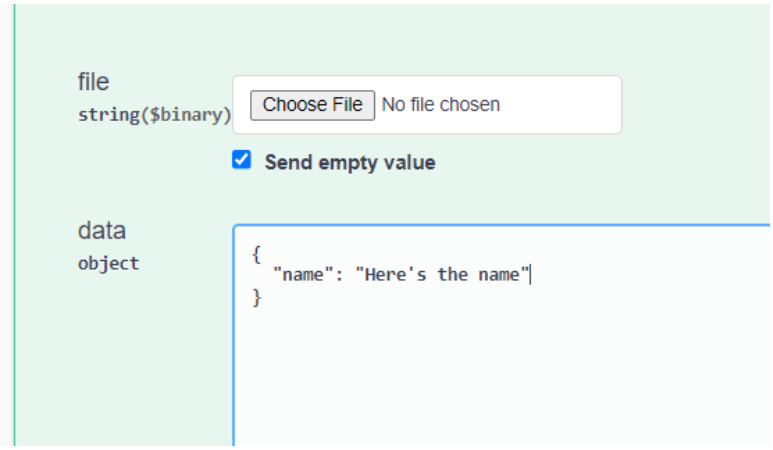


Сервер присылает в ответ

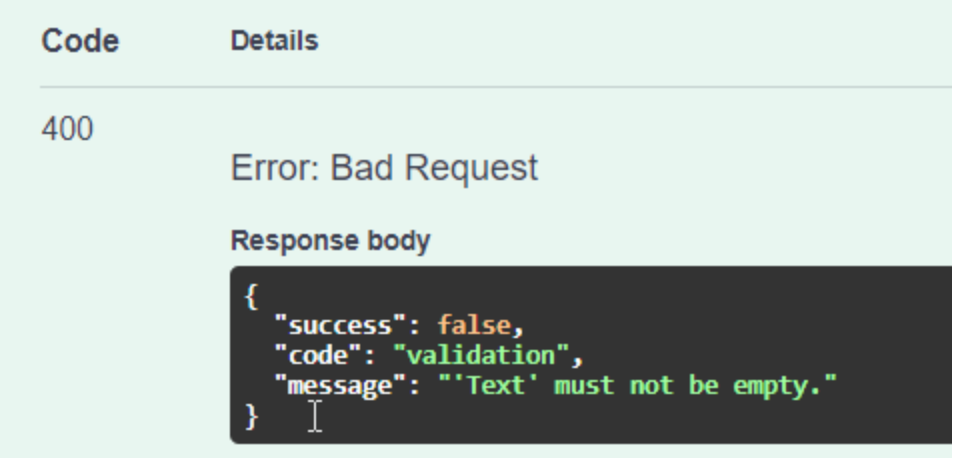


Значит все ок. Сигнализирует о том, что данные пусты.

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



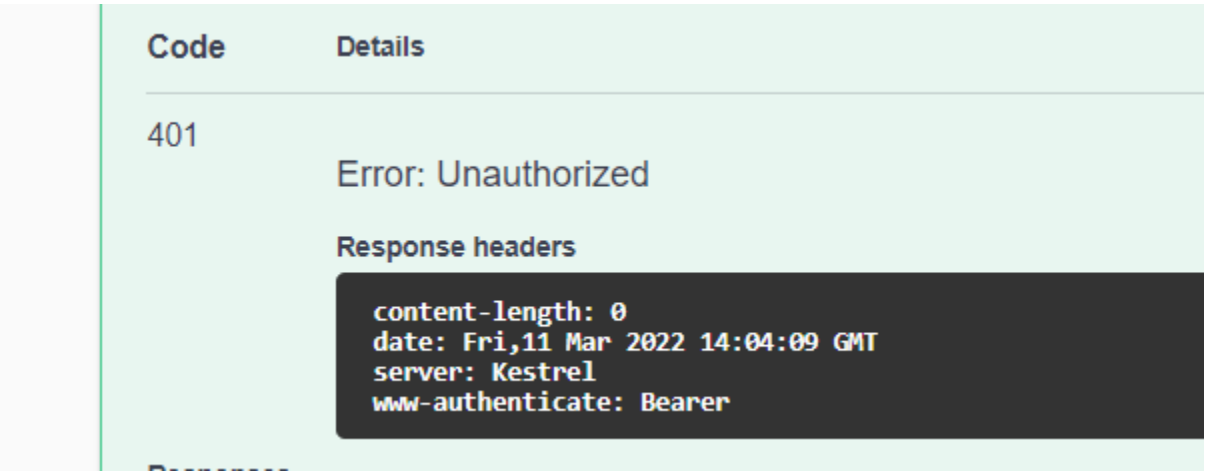
Что присылает сервер в ответ:



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



401 – не авторизован.

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

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