

Postman Server Mock

Author	Сергей Погребняк
Last changes	19.12.2019
Swift	5.2
Xcode	11.3
Postman	7.14.0

-
- Введение
 - Мануал
 - 1 Что это? Зачем оно мне нужно?
 - 2 Создание новой коллекции и добавление нового запроса
 - 3 Добавление Mock Server-а
 - 4 Настраиваем Mock-запрос для проверки
 - 5 Добавление нового Example - нового mock-api-call
 - 6 Проверяем созданный mock-endpoint
 - Дополнительные плюшки
 - 1 Test Application
 - 2 Проверка лимита запросов и смена плана
 - 3 Ссылка на тестовый проект и исходники
-

Введение

Данный мануал предназначен для людей обладающими минимальными знаниями об программе Postman, которые могут ответить на ряд вопросов:

- что такое Postman и для чего он нужен
- что такое коллекция
- что такое запрос
- что такое environment
- как добавляются запросы и из чего они состоят

Если вы можете ответить на все эти вопросы можно перейти сразу к мануалу, если нет вот несколько ссылок которые помогут Вам разобраться и дать ответы на эти вопросы:

- <https://learning.getpostman.com/docs/postman/launching-postman/sending-the-first-request/>
- <https://learning.getpostman.com/docs/postman/launching-postman/creating-the-first-collection/>
- <https://learning.getpostman.com/docs/postman/collections/creating-collections/>
- <https://learning.getpostman.com/docs/postman/environments-and-globals/intro-to-environments-and-globals/>
- <https://qwiki.nixsolutions.com/display/IPT/Postman>

Мануал

1 Что это? Зачем оно мне нужно?

Postman server mock - инструмент приложения postman который позволяет разворачивать сервер для API запросов на серверах postman-а. Он позволяет сгенерировать base URL сервера и добавляя адрес API кола мы можем получить нужный нам mock API-call. Mock server полезен в случае:

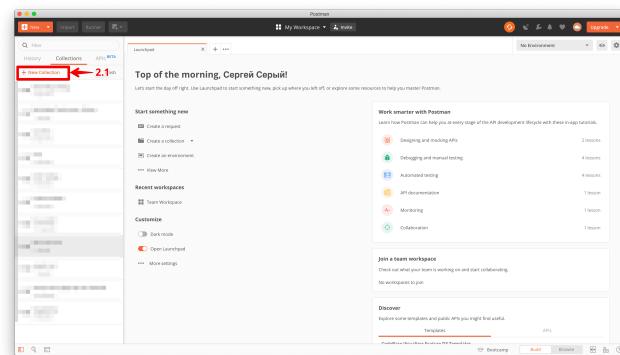
- Сервер находится не на одной стороне с клиентом, и нужно протестировать поведение клиента на специфические ответы (фидлер и подобные приложения отпадают в случае множественных API колов, отладка приложения будет очень затруднится в таком случае и займет больше времени)
- Для дальнейшей разработки нужен API call который в данный момент сломан или не реализован еще
- Для написания тестов

Рассмотрим использование этого инструмента на примере – задача: реализовать авторизацию юзеров в приложении которое должно было быть отдано заказчику вчера, и в виду того, что сроки уже горят нужно срочно реализовывать это и на клиенте, и на сервере.

После встречи с серверной частью и принятия решения о структуре ответа мы можем реализовать наш mock API call. И так, для того, чтобы это сделать нам нужна существующая коллекция или создать ее.

2 Создание новой коллекции и добавление нового запроса

2.1- Создать новую коллекцию



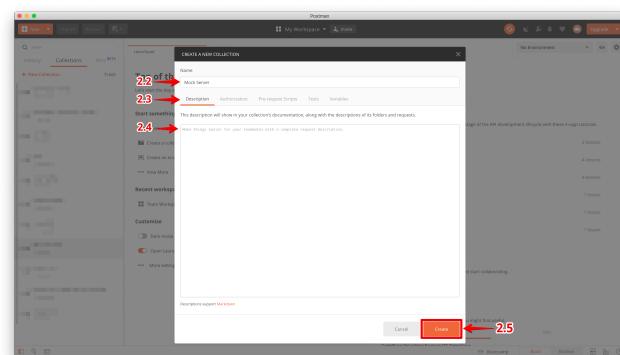
2.2- Ввести название новой коллекции

2.3- Выбрать для добавления (опционально):

- описание
- тесты на всю коллекцию
- Pre-scripts
- т.д.

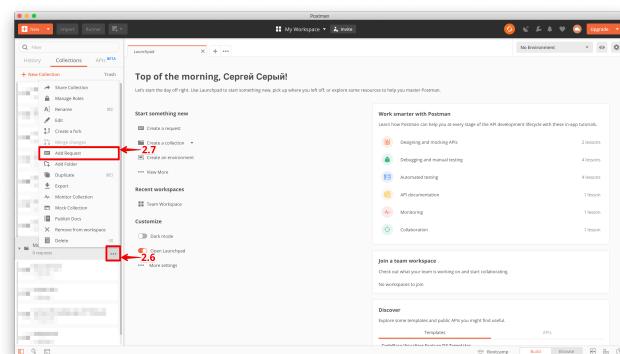
2.4- Заполнить нужными данными выбранными в п.3 (опционально)

2.5- Сохраняем созданный запрос



2.6- Нажав на «...» откроется меню с возможными опциями коллекции

2.7- Добавляем новый запрос в коллекцию

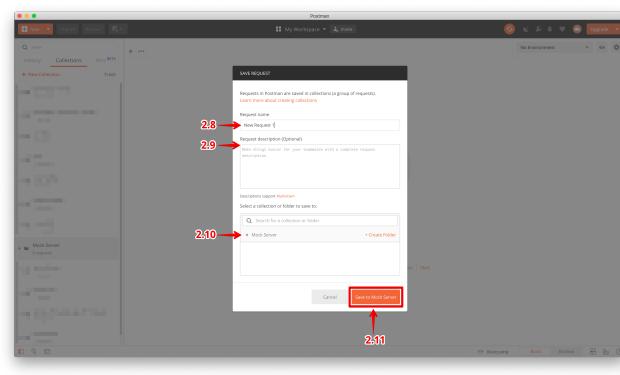


2.8- Ввести название нового запроса

2.9- Добавить описание к запросу (опционально)

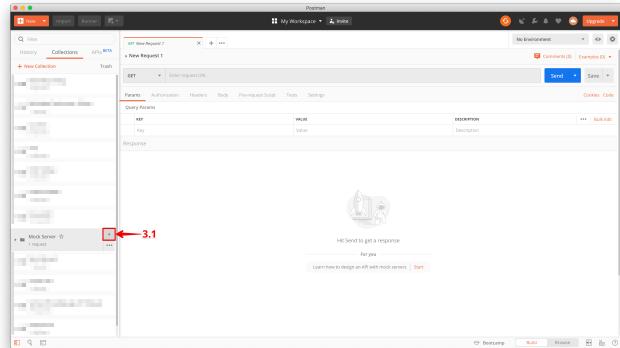
2.10- Убедиться, что новый запрос добавляется в нужную коллекцию

2.11- Сохранить созданный запрос



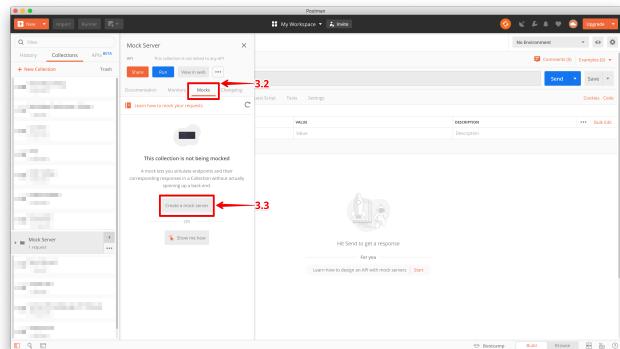
3 Добавление Mock Server-a

3.1- Разворачиваем весь список действий для коллекции



3.2- Переходим на вкладку «Mock»

3.3- Нажимаем на кнопку «Create a mock server»

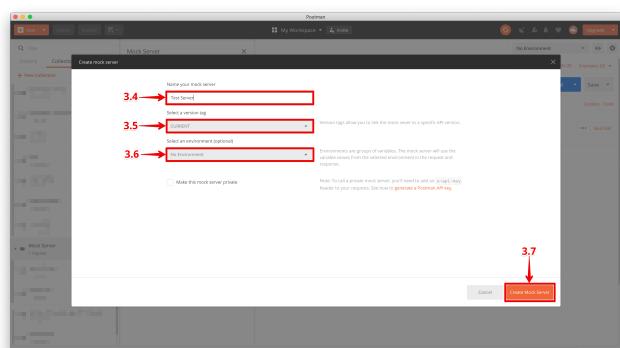


3.4- Вводим название нового Mock server-a

3.5- Выбираем версию апи (опционально)

3.6- Выбираем окружение (опционально)

3.7- Сохраняем созданный мок сервер

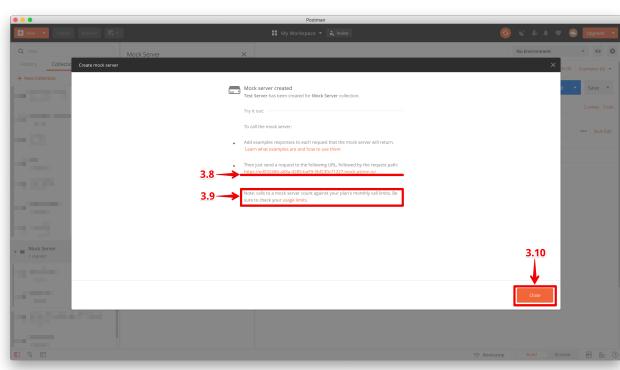


Ознакомливаемся с результатами созданного Mock сервера:

3.8- это и есть наш base URL только что созданного сервера

3.9- **ATTENTION** это ссылка, где можно проверить количество доступных вызовов API, так как в бесплатной версии оно **ограничено!** (дальнейнее)

3.10- закрываем окно

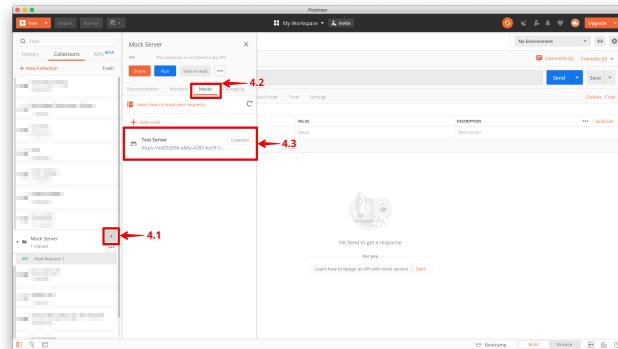


4 Настраиваем Mock-запрос для проверки

4.1- разворачиваем список

4.2- выбираем вкладку «Mock»

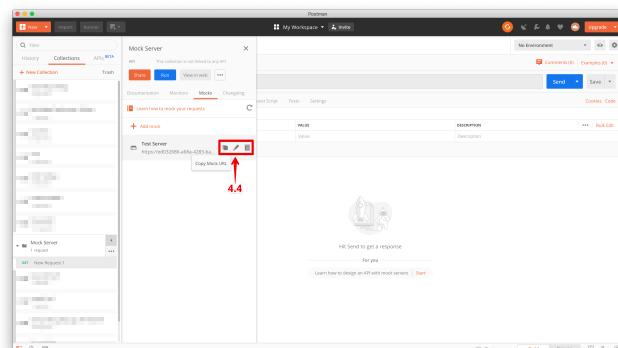
4.3- наводим на тестовый сервер и увидим доп. действия



4.4- после наведения на тестовый сервер появятся доп. действия

- копировать base URL server-a
- редактировать mock server
- удалить mock server

Копируем base URL.



4.5- Выбираем созданный запрос

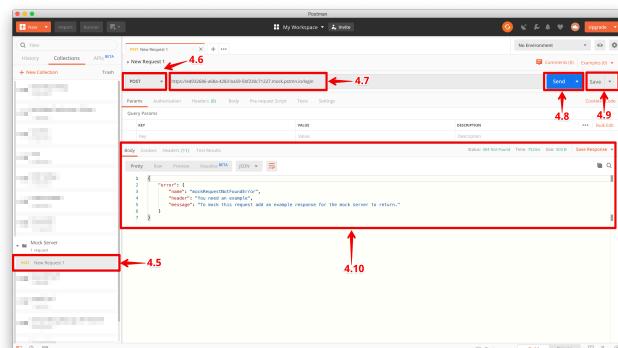
4.6- Выбираем тип запроса (post так как в запросе на авторизацию отправляются данные – логин и пароль)

4.7- Вставляем скопированный Base URL и добавляем – “/login” так как мы создаем апи кол для авторизации

4.8- Жмем кнопку отправить и пока обрабатывается запрос

4.9- Жмем кнопку «Save» для сохранения внесенных изменений

4.10- Если все сделали верно мы увидим результат следующего вида (см скрин ниже), из него нам становится ясно что endpoint не найден... Пока все логично) мы создали только сам сервер, теперь создаем конкретный endpoint.



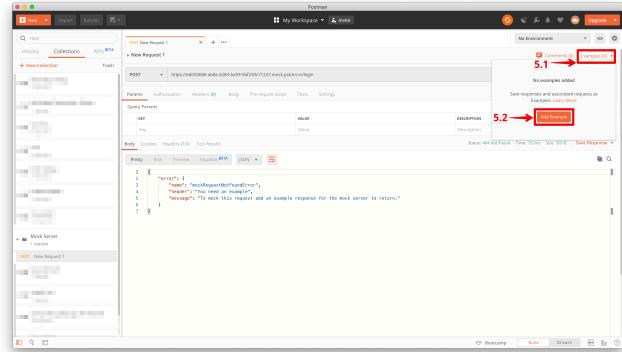
На данный момент (2019) Mock server не имеет полного функционала обычного сервера, то есть - Postman НЕ имеет функционала который позволял бы реагировать на данные отправленные в body или header секциях разными способами. Например: при отправке в теле запроса на получение данных о юзере с id = 1 - отправить токс данные юзера вымышленного а при отправке id = -999 отправить ответ что юзера не существует, на данный момент такое можно реализовать только посредством создания двух разных примеров соответственно и двух разных API колов.

5 Добавление нового Example - нового mock-api-call

Для того чтобы создать мок конкретного endpoint-a:

5.1- Нажимаем на «Example» и в открытом меню выбираем

5.2- «Add Example»

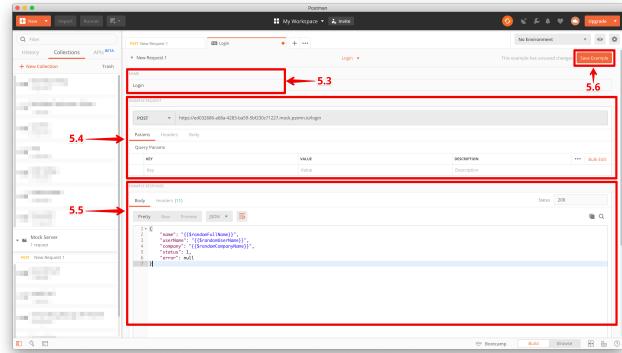


5.3- Вводим название нового примера

5.4- Блок запроса (подробнее описан ниже)

5.5- Блок ответа (подробнее описан ниже)

5.6- После заполнения выше указанных полей не забываем сохраняться



Блок запроса – это блок который отвечает за то что сервер получает и по какой URL к нему обращаться, в нем:

5.7- тип запроса который осуществляется на сервер

5.8- endpoint по которому производится запрос (в нашем случае base URL сгенерированный постменом и наша добавка – «/login»)

5.9- поля которые мы можем выбрать для заполнения

5.10- место редактирования выбранных полей из пункта 34.3

Блок ответа – это блок который сервер нам должен вернуть, в нем:

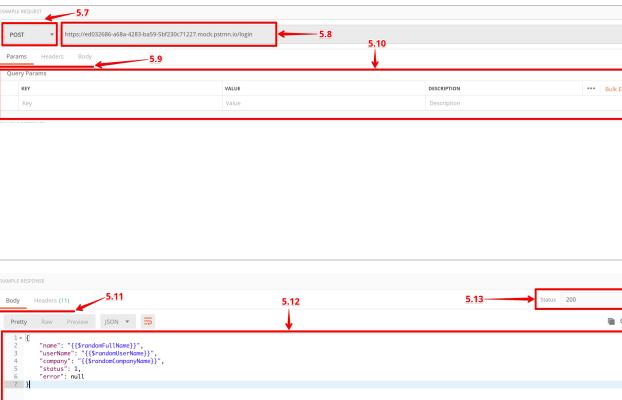
5.11- выбираем тип данных которые мы будем заполнять

5.12- заполняем выбранные данные

5.13- вводим статус ответа

В нашем случае нас интересует тип данных Body и тип будет JSON, статус код - 200. Предположим что мы договорились с серверной частью что во всех API колах у нас будет присутствовать 2 обязательных поля:

- status
- error



Если все успешно то статус = 1(true) и error = nil, если что-то не так то статус = 0(false) и в error содержится текст ошибки. После этого заполняем сам Body следующими данными:

```
{
  "name": "{$randomFullName}",
  "userName": "{$randomUserName}",
  "company": "{$randomCompanyName}",
  "status": 1,
  "error": null
}
```

Итак мы создали body со следующими данными name, username, company – которые Postman сгенерирует самостоятельно (Postman умеет генерировать некоторые данные весь список можно найти вот тут https://learning.getpostman.com/docs/postman/scripts/postman_random_box_api_reference/#dynamic-variables) и обязательными данными status и error. После этого ставим статус код 200 (означает что запрос прошел успешно) и сохраняем mock api call.

6 Проверяем созданный mock-endpoint

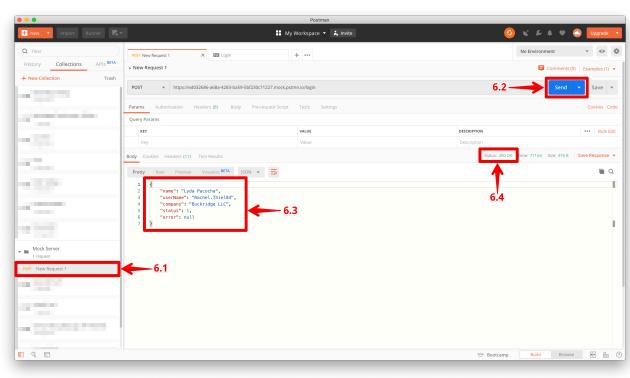
И так пришло время проверить наш созданный mock api call, для этого:

6.1- Переходим на наш API запрос

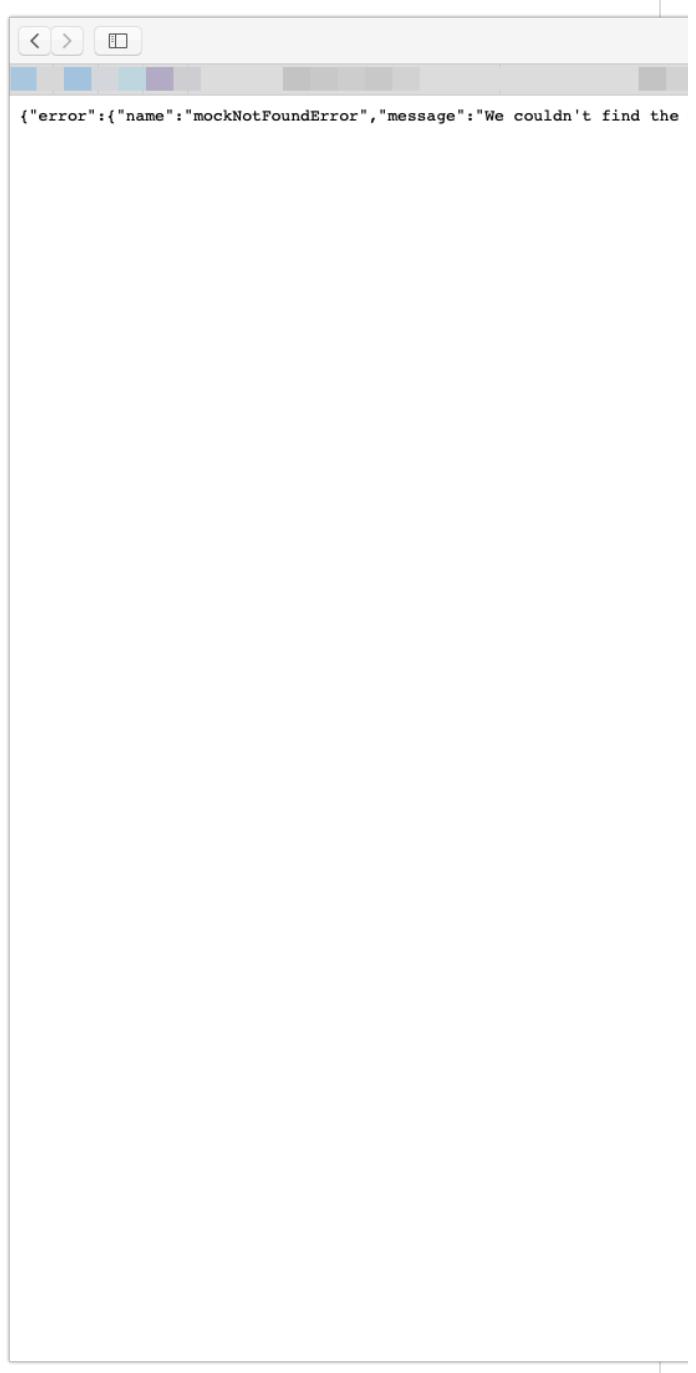
6.2- Жмем кнопку «Send»

6.3- Если все сделали верно то тут мы увидим случайные данные которые будут меняться каждый раз при нажатии кнопки send, неизменны останутся только status и error.

6.4- Убеждаемся что наш статус 200



Если все верно значит наш mock api call готов)
Также его можно проверить скопировав ссылку апи кола и
вставив ее в браузере, мы увидим следующий результат:



Наш mock api call готов и можно перейти к коду, но перед этим я предлагаю сразу создать еще один ответ сервера, успешный мы уже
создали а теперь давайте создадим запрос когда мы отправили данные на авторизацию юзера которого не существует, по аналогии
создаем второй пример только после base url добавим - /errorLogin и body следующего вида:

```
{  
  "status": 0,  
  "error": "User Not Found"  
}
```

Вставив нашу новую ссылку (base URL+errorLogin) в сафари убеждаемся что все хорошо и мы получаем ошибку - юзер не найден,
можем перейти к написанию кода.

Дополнительные плюшки

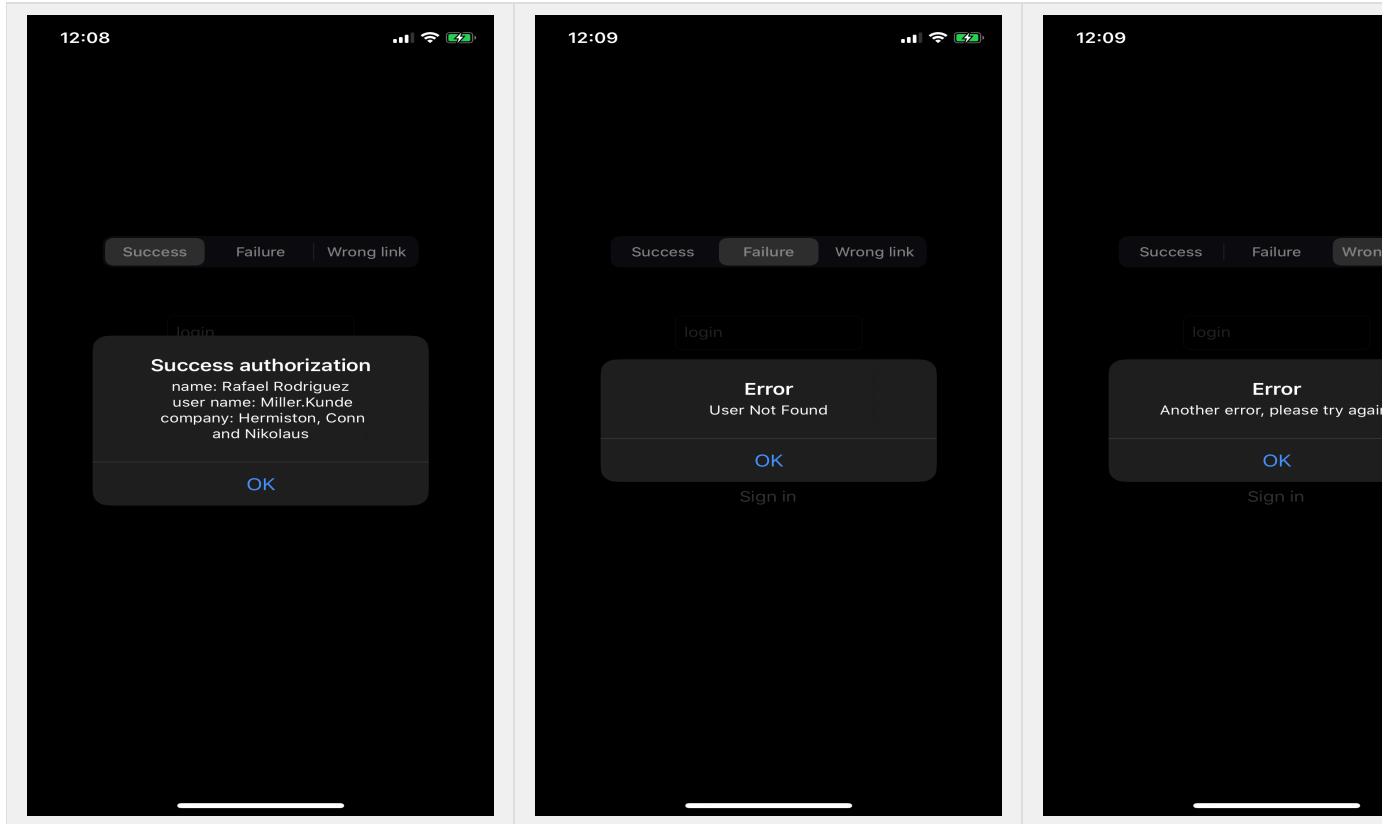
1 Test Application

В тестовом примере нужно заменить base URL, на свой, для этого переходим в ServerMock->APIManager находим property baseURL и вставляем наш mock base URL и запускаем приложение.

APIManager.swift

```
enum LoginLinks: String, CaseIterable {
    case login // endpoint success API
    case errorLogin // endpoint failure API
    case wrong // endpoint-
}
class APIManager: NSObject, URLSessionDelegate {
...
private let baseURL = "YOUR BASE URL"
...
}
```

Запустив тестовое приложение мы увидим пример экрана для входа в приложение, единственное что будет отличать от реального это сегмент контроллер для выбора ендпоинта на который мы будем обращаться, после тапа на кнопку мы увидим разные текстовки соответственно можем моделировать дальнейшее поведение приложения, после того как закончим и нам дадут реализованный api call нашего сервера просто заменим мок апи кол на реальный и проверим что все работает корректно.



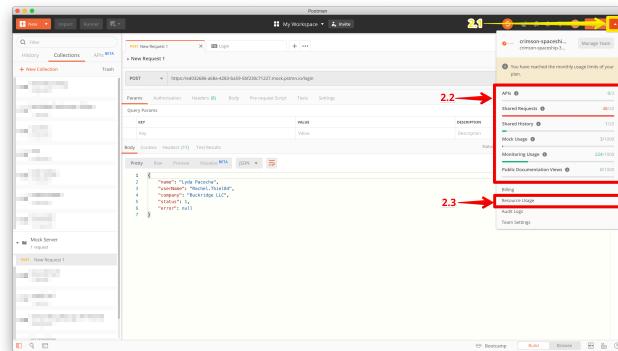
2 Проверка лимита запросов и смена плана

В бесплатном тарифном плане количество запросов ограничено 1 000, проверить оставшееся количество можно в приложении, для этого:

2.1- Разворачиваем меню своего work space

2.2- Отображаются данные о количестве использованных ресурсов в 30 дневный период

2.3- Можем перейти на сайт для смены плана или просмотра данных в веб версии

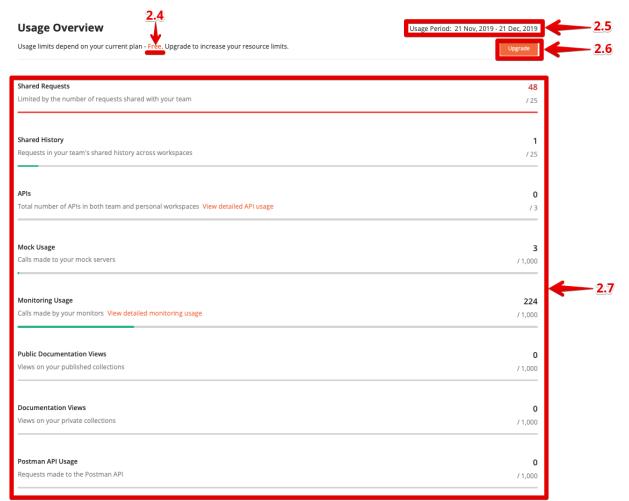


2.4- Тип текущего плана

2.5- Период использования данных (каждый месяц некоторые данные сбрасываются и доступны снова)

2.6- Переход на новый тарифный план

2.7- Таблица использованных данных



3 Ссылка на тестовый проект и исходники

https://gitlab.nixdev.co/nix_iphone_skillup/postman-server-mock