Что такое API Как это работает?

API (Application Programming Interface), или интерфейс прикладного программирования - это понятие, которое часто упоминается в контексте программирования и веб-приложений. На турецком языке это выражается как "Архитектура программного интерфейса". Интерфейсы API широко используются в самых разных областях и для самых разных целей. API - это интерфейс подключения, который программа использует для доступа к данным, серверному программному обеспечению или другим программам. Он содержит набор правил, определяющих, как две машины общаются. Например, сайт электронной коммерции может позволить совершать покупки, разрешая с помощью API снимать деньги с кредитной карты, выданной банком. Или приложение Linkedin (Android или iOS) может через API извлечь ваши данные с серверов Linkedin и показать их вам. С помощью API мы получаем доступ к ранее разработанной инфраструктуре, и нам не нужно заново создавать эту инфраструктуру. Вкратце, мы можем легко получать множество специальных данных через веб-сайт или наши смартфоны благодаря API и получать некоторые дополнительные функции.

Какие виды API существуют

API могут быть классифицированы различными способами. В общем, API используются в веб-сервисах. Теперь, если хотите, давайте сосредоточимся на классификации по способам использования и архитектурам.

Виды API по назначению

Внутренние API

Внутренний API - это API, который используется только определенными людьми или в определенной области, например, определен для отделов компании.

Открытые API

Это API, которые доступны для использования всеми. Они также называются Open API. Они отличаются от внутренних API, которые определены специально для одной компании или организации.

Партнерские API

Партнерские API позволяют компаниям, работающим с определенным партнерством, проводить совместные операции и обеспечивают координацию их систем.

Виды API по архитектуре

REST API

REST API - это служба API, работающая с использованием протокола HTTP. В этой модели API, созданные с использованием URI (Uniform Resource Identifier),

подключаются к базам данных. Обмен данными осуществляется с помощью HTTP-запросов, таких как GET, POST, PUT, DELETE.

В REST (Representational State Transfer) API данные передаются в формате JSON (JavaScript Object Notation). REST API очень удобны и легки в использовании, поэтому это очень популярная архитектура.

SOAP

SOAP (Simple Object Access Protocol), или Простой протокол доступа к объектам, - это архитектура API, которая обеспечивает передачу данных с более жесткой структурой безопасности. В этой архитектуре обмен данными осуществляется с использованием формата XML. Хотя его конфигурация сложнее, чем у REST API, считается, что эта архитектура обеспечивает более безопасное соединение.

Как работает API?

API - это интерфейс приложения, который служит для открытия безопасных и контролируемых врат между программным обеспечением или базами данных.

Принцип работы API следующий:

• Приемная программа делает запрос (API Call). Этот запрос обрабатывается веб-сервером посредством URI.
• После получения запроса API делает вызов к внешней программе или веб-серверу.
• Сервер отправляет запрошенную информацию API в ответ.
• API передает полученные данные обратно в программу, сделавшую запрос.
• Можно описать принцип работы API следующим образом;

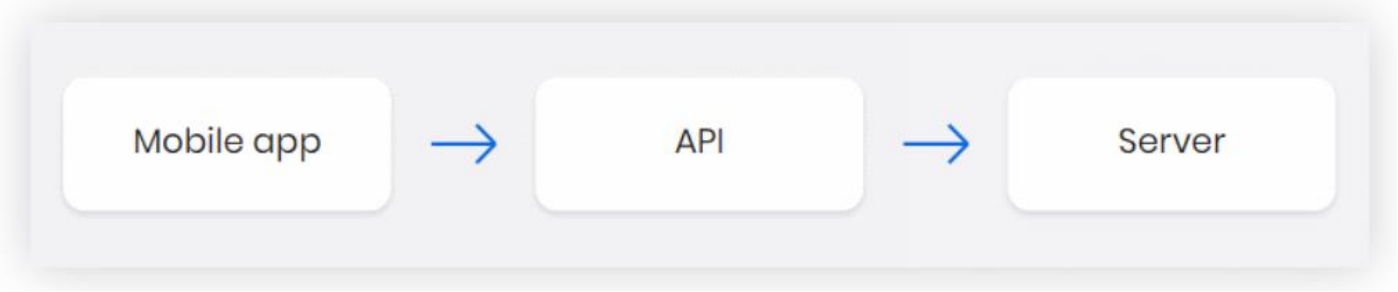
Представьте большую библиотеку с огромным количеством книг и документов. Допустим, для выполнения вашей работы вам регулярно нужно получать оттуда некоторую информацию. Некоторые этажи или комнаты библиотеки не открыты для всех, потому что они содержат особые данные, и только ключ, открывающий соответствующие двери в пределах предоставленных полномочий, предоставляется тем, кто хочет получить информацию.

Библиотечная служба безопасности выпускает ключ в соответствии с вашими правами и передает его вам. С помощью этого ключа вы можете войти в нужные вам помещения или этажи, получить информацию и продолжить свою работу. Здесь предоставленный вам ключ представляет API.

Благодаря этой карте вы можете получить доступ к данным в открытых для вас отделах любым удобным вам способом, не обращаясь снова в службу безопасности. В этом примере каждый посетитель, имеющий ключ, также является клиентским программным обеспечением.

Последнее время активно используемые приложения для интернет-банкинга также работают с помощью API. Приложение, установленное на ваш смартфон, создает мост между вашими банковскими данными с помощью API, определенного банком.

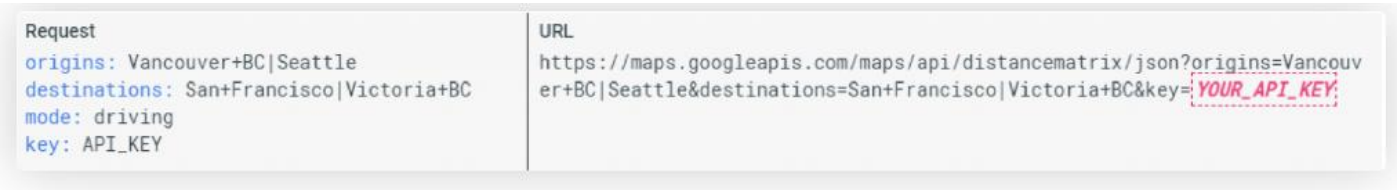
После интеграции API оно разрешает вам совершать операции с вашим банковским счетом. В нормальных условиях операционная система и программное обеспечение банка отличаются от операционной системы и программного обеспечения на вашем телефоне. API позволяет общаться между различными опер



Как использовать API? Как проводится интеграция API?

API можно использовать для упрощения услуг, которые вы предоставляете. Например, интегрированный в вашу систему API, который отслеживает текущий обменный курс, позволяет вам автоматически обновлять предоставляемую вами услугу в соответствии с текущими курсами обмена и обеспечивать автоматическое выполнение платежей на основе актуальной информации о курсах обмена.

Запрос API (вызов), созданный с помощью метода GET, выглядит примерно так.



Пример архитектуры API выглядит следующим образом. Из этого примера видно, для каких целей используется API и какие возможности при этом приобретаются. Здесь также можно увидеть, что можно делать с помощью API Gateway.

Для интеграции API клиенту необходим ключ API, чтобы представить себя. Приложение или сервер, предоставляющий услуги API, требует от клиента ключ API и пароль.

Приложение с аутентификацией (разрешением) с помощью пароля создает постоянный канал связи с сервером с помощью идентифицирующего его ключа. Таким образом, начинается регулярный обмен данными.

Чтобы создать интеграцию API, вам нужны эти ключи API и пароли. Процесс интеграции осуществляется на стороне бэкэнда. Ключ API, предоставляемый клиенту, хранится в созданной в базе данных таблице.

После создания аутентификации с помощью заданного ключа API и пароля, клиент, например, при отправке HTTP-запроса, должен добавить ключ API в заголовок и тело и отправить его.

Благодаря этому ключу идентифицируется личность клиента, и после того, как была обеспечена аутентификация, устанавливается безопасное соединение и запрашиваемые данные открываются для клиента. По желанию, для дополнительной защиты, может быть запрошен IP-адрес клиента, что предотвратит использование его ключей API другими людьми.

Что такое Postman?

Postman - это платформа для сотрудничества по разработке API. Особенности Postman упрощают каждый этап создания API и облегчают сотрудничество, позволяя вам создавать лучшие API быстрее.

Основные особенности:

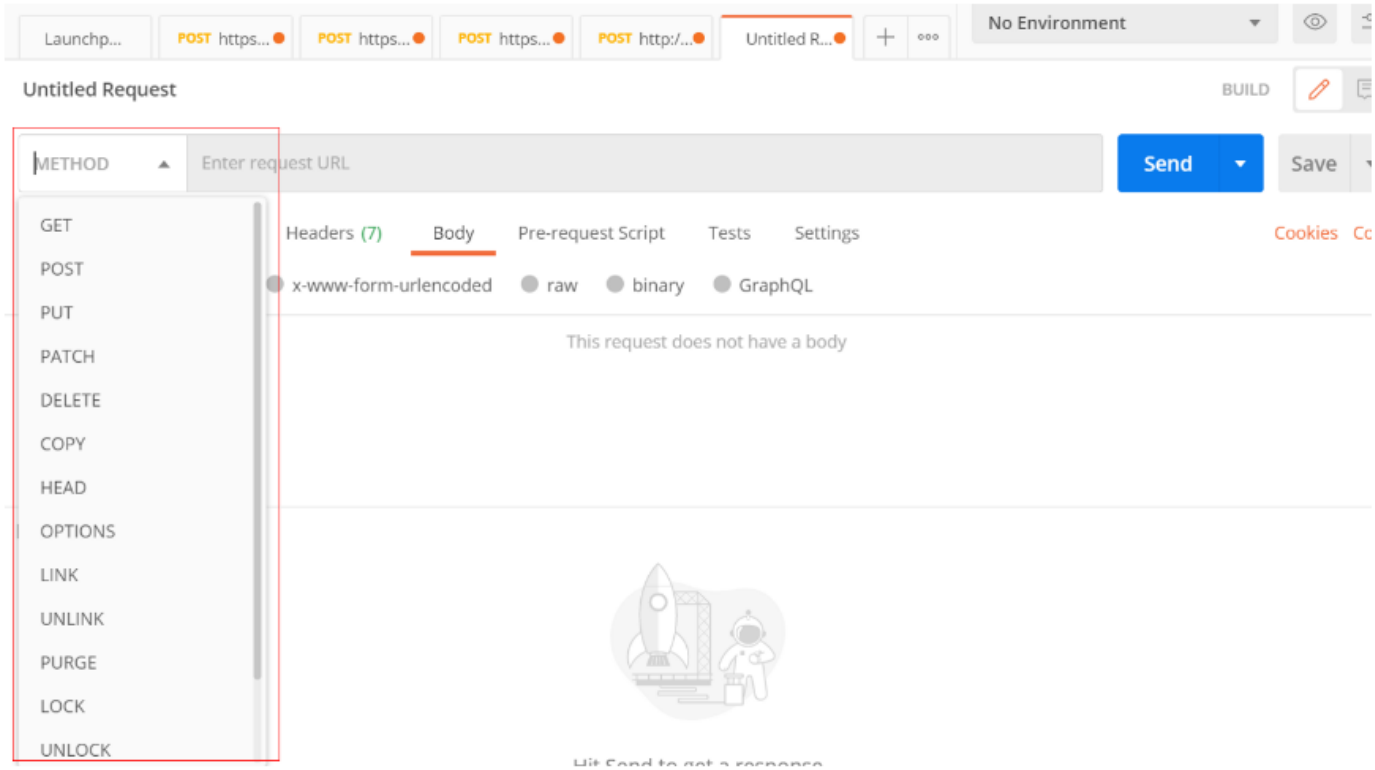
• Api Клиент: С помощью Postman вы можете быстро и легко создавать запросы Rest и Soap. Вы можете использовать его вместо клиента.

• Автоматические тесты: Тесты автоматизируются путем создания групп тестов, которые можно запускать снова и снова. Postman может быть использован для автоматизации многих типов тестирования, включая модульное тестирование, функциональное тестирование, интеграционное тестирование, end-to-end тестирование, регрессионное тестирование и т. д. Автоматическое тестирование устраняет человеческие ошибки и упрощает тестирование.

• Документация: Postman позволяет быстро и легко публиковать ваши документы. Postman автоматически получает ваши образцы запросов, чтобы заполнить вашу страницу документации динамическими примерами и инструкциями, которые можно прочитать машине, чтобы вы могли легко поделиться вашим API с остальным миром.

Postman - это приложение, которое мы можем использовать как расширение Chrome или скачать и установить на наш компьютер. Он также может быть определен как Rest Client. API (Application Programming Interface) позволяет различным приложениям взаимодействовать друг с другом. Общение клиента (Android) и Backend (Java) с помощью Restfull Api является примером этого. С помощью Postman мы можем легко тестировать наши API, вместо того чтобы писать длинные коды. Благодаря его многим функциям, мы можем легко подготовить запрос и использовать возвращаемые значения.

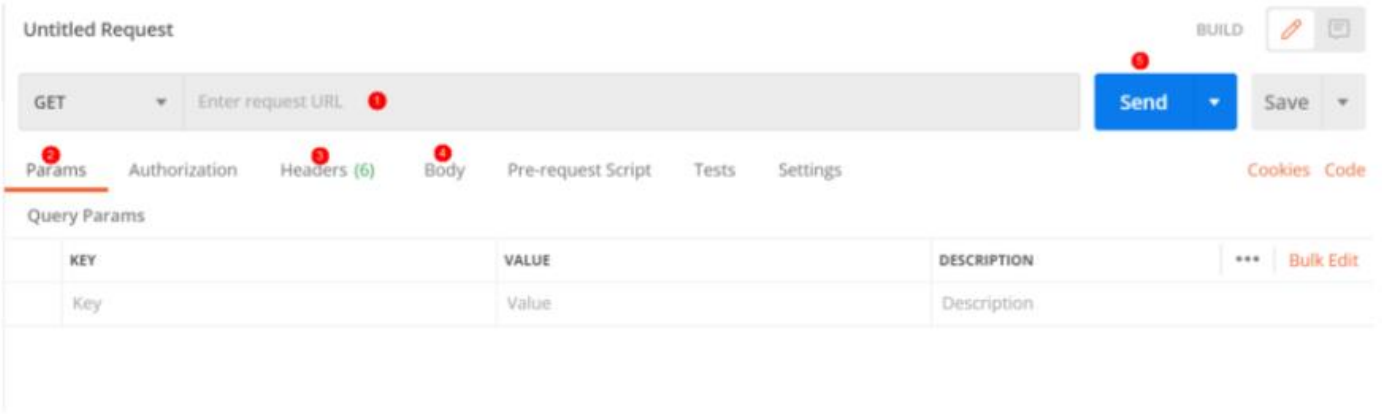
POST-GET-PUT-DELETE



Справа на вышеприведенной картинке, помеченной красным, находятся наши типы методов запросов. Я объясню, когда мы должны использовать наиболее часто используемые методы GET, POST, PUT, DELETE. Замечание: Операции CRUD обозначают Создание, Чтение, Обновление, Удаление.

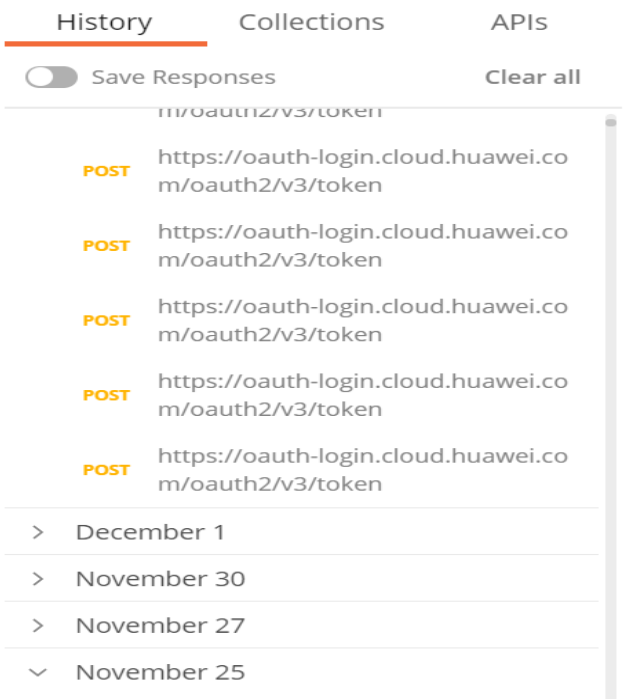
- GET: Если мы хотим просто извлечь (прочитать) данные с сервера, то есть мы не собираемся делать какие-либо изменения (добавление, удаление, модификация) в данных, рекомендуется использовать метод GET. Мы можем сказать, что это соответствует операции Read из CRUD. Пример: при использовании GET /students сервер возвращает нам список студентов.
- POST: Мы можем использовать этот метод, когда хотим изменить данные на сервере, заполнив тело запроса. Изменение включает в себя операции Create и Update из CRUD. Пример: с помощью POST /createUser и вводом информации о пользователе в тело запроса мы можем запросить создание пользователя в базе данных.
- PUT: У этого метода есть свойства запроса POST. То есть, мы используем его, когда хотим выполнить операции Create и Update из CRUD. В отличие от POST, запрос PUT определяется как идемпотентный и не кешируемый.
- DELETE: Это соответствует операции Delete из CRUD. Рекомендуется использовать его, когда мы хотим удалить данные.

PARAMS-HEADER-BODY-PATH-SEND



При подготовке запроса обычно активно используются эти 5 области.
Место, указанное номером 1, это место, где мы вводим URL API.
Место, указанное номером 2, это область, где мы определяем значения Params.
Место, указанное номером 3, это область, где мы определяем значения Headers.
Место, указанное номером 4, это область, где мы заполняем тело запроса, доступны такие варианты, как raw, binary.
Место, указанное номером 5, это место, где мы можем отправить наш запрос, нажав Send после подготовки нашего запроса.

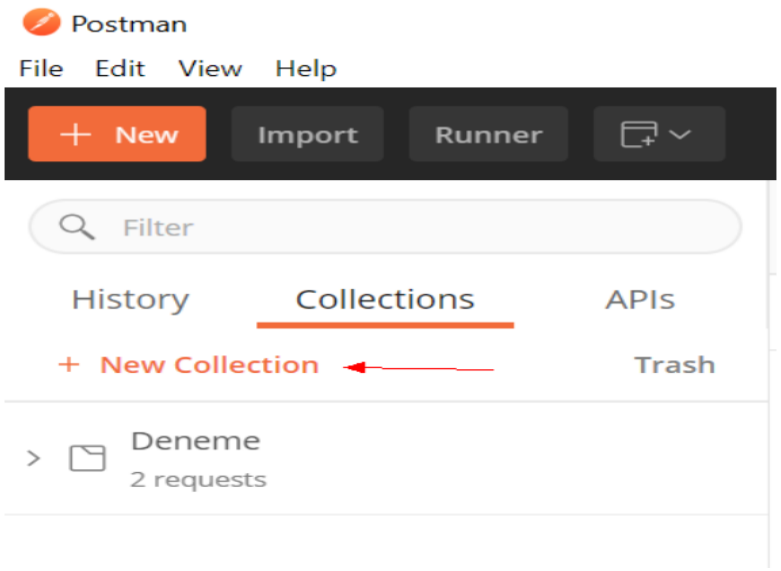
HISTORY



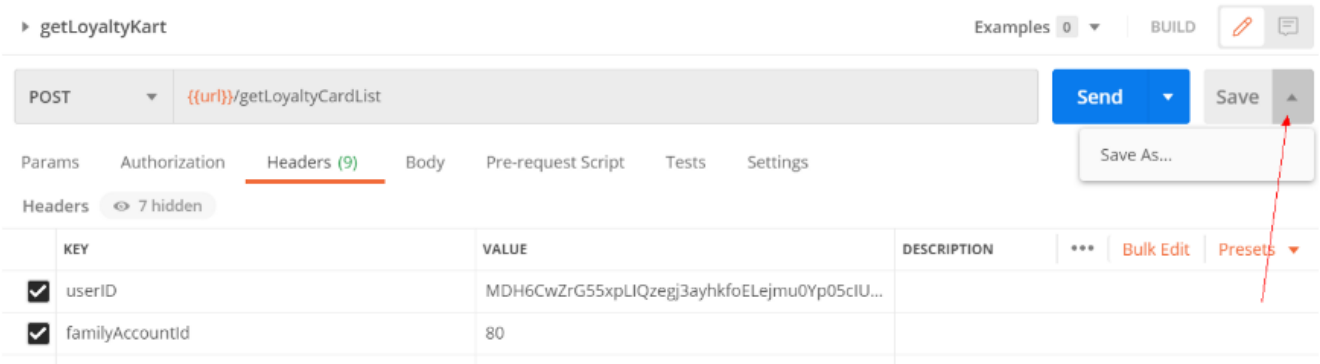
Вкладка "History" на левой стороне Postman сохраняет все наши предыдущие запросы по дням. Если вы кликните по любому из них, запрос откроется на новой вкладке.

COLLECTİON

Collection позволяет нам сохранять несколько запросов в группе и делиться ими с другими при необходимости. Это позволяет нам удобно хранить запросы, которые мы уже использовали в проекте. Сначала мы создаем новую коллекцию, как показано ниже, через вкладку "New Collection". Достаточно просто ввести название коллекции и нажать "Создать".



Мы подготавливаем запрос для добавления в только что созданную коллекцию "Тест", как показано на изображении ниже, и нажимаем на стрелку рядом с кнопкой "Сохранить", затем выбираем "Сохранить как...".



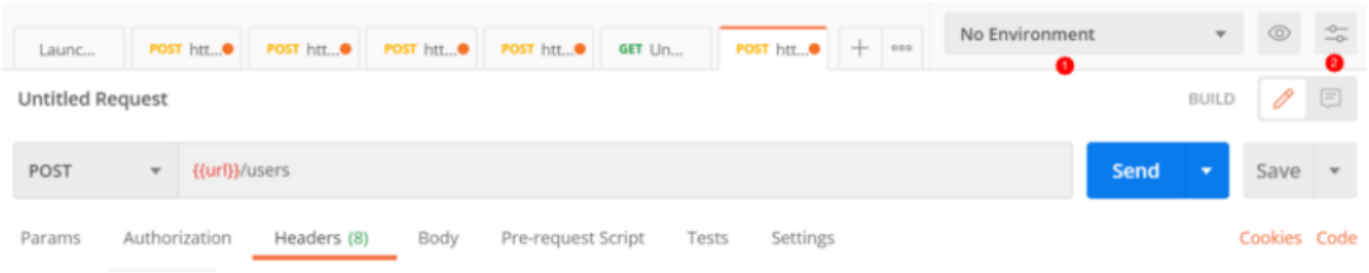
Мы вводим имя запроса и, при необходимости, его описание на экране "Сохранить как...". Затем выбираем в какую коллекцию сохранить запрос. После этого мы можем увидеть добавленный запрос внутри коллекции "Тест". Кроме того, полезно создавать папки под названиями GET, POST, DELETE и т.д. внутри коллекции "Тест" и располагать запросы в соответствии с этой структурой файлов.

Функция Импорта

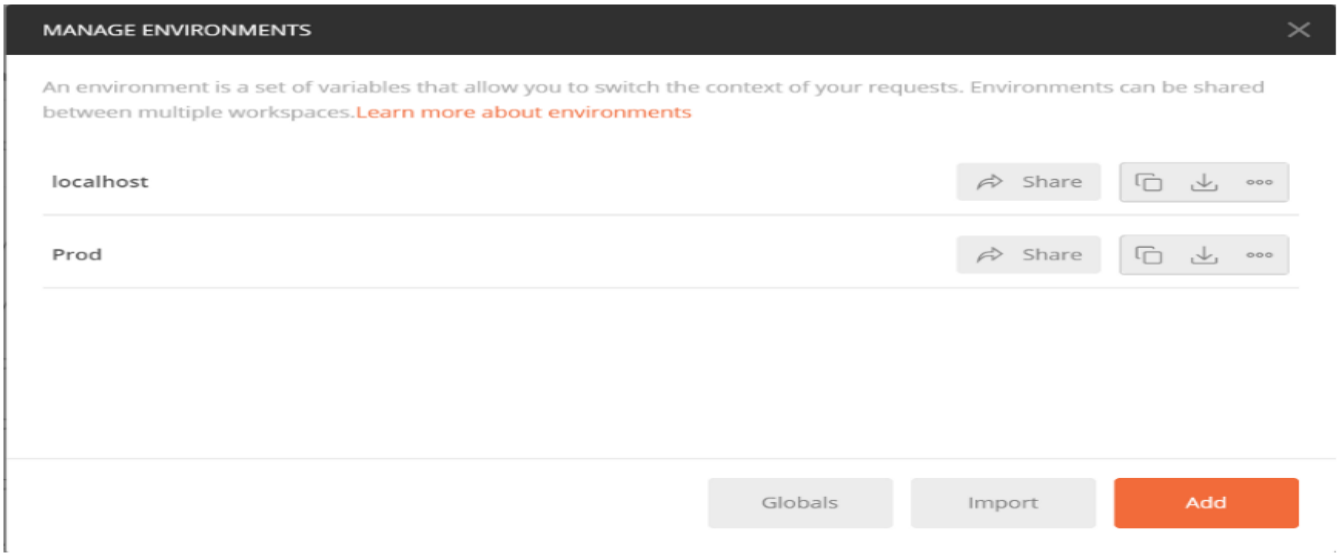
Она обычно используется для загрузки готовой, сохраненной коллекции Postman. Но если у вас есть API, который вы хотите протестировать, и он был преобразован в документ с помощью инструментов, таких как Swagger, вы можете быстро сохранить все конечные точки в Postman с помощью ссылки в поле импорта. Например, я использую ссылку

https://petstore.swagger.io/v2/swagger.json для тестового API, предоставляемого Swagger. Затем я завершаю операцию импорта в соответствии с моими предпочтениями, выбирая требуемые параметры во вкладке "Confirm". Конечно, помимо этих опций, вы можете создавать новые запросы с помощью кнопки "New".

Environment



Для определения переменных используется функция Environment. Чтобы создать Environment, мы переходим на вкладку, обозначенную цифрой 2, и выбираем "Управление окружениями" (Manage Environments).



Мы нажимаем "Add" и определяем наши окружения, как показано в примере ниже.

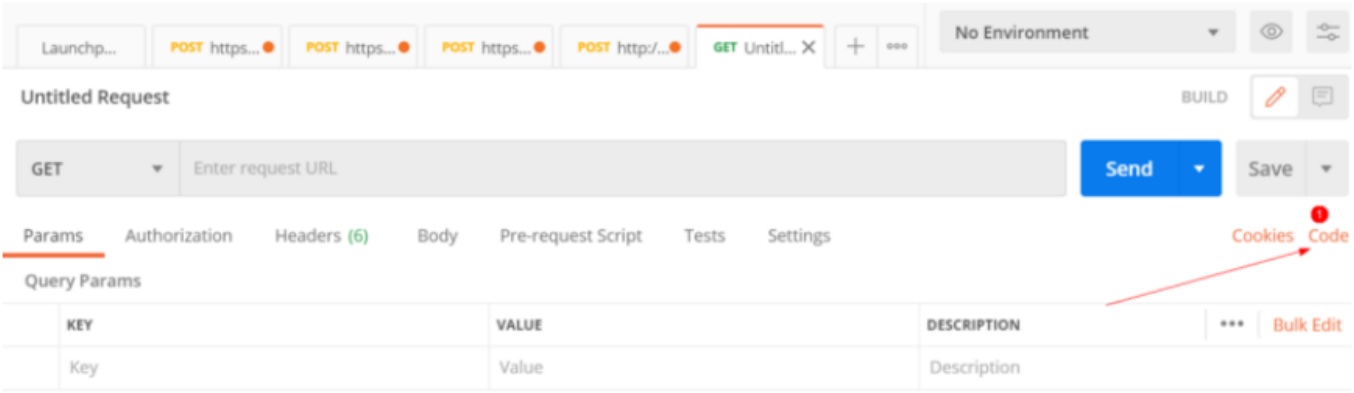
Например, мы определили окружение с именем "Localhost", переменной "url" со значением "http://localhost:8080".

Мы также определили еще одно окружение с именем "Prod", переменной "url" со значением "http://159.159.59.159".

Теперь, когда я указываю в поле API URL "{{url}}/users" и выбираю "Localhost" в области, обозначенной цифрой 1, "{{url}}/users" будет вести себя как "http://localhost:8080/users".

Аналогичным образом, если я выбираю "Prod", он будет восприниматься как "http://159.159.59.159/users".

Code



Да, на правой стороне Postman есть опция "Code". Эта опция позволяет нам преобразовать подготовленный запрос в код.

Да, в левом верхнем углу изображения, обведенного красным цветом, мы можем выбрать язык программирования для генерации кода. В примере

показано, как с использованием библиотеки OkHttp на языке Java можно сгенерировать код, соответствующий нашему подготовленному запросу. Эта функция помогает нам ответить на вопрос о том, как выполнить запрос на стороне кода после подготовки и запуска запроса в Postman.