Backend

Backend, и frontend — это варианты архитектуры ПО. Термины возникли в программной инженерии по причине появления **принципа разделения ответственности между внутренней реализацией и внешним представлением**. В результате frontend-разработчик может не знать особенностей работы сервера, а backend-программисту не обязательно вникать в реализацию frontend.

Frontend — это разработка пользовательского интерфейса и функций, которые работают на клиентской стороне веб-сайта или приложения (desktop приложения или мобильного). Это всё, что видит пользователь, открывая веб-страницу, и с чем он взаимодействует.

Backend - это процесс программирования, конечной целью которого является разработка серверной части web-ресурса и ее объединение с пользовательской стороной. Backend — это все то, что происходит на стороне сервера и что остается невидимым пользователю.

Базовый набор требований, который необходим для backend-разработчика

- Знание хотя бы одного «серверного» языка программирования: PHP, Go, ASP.NET (C#), Python, Ruby, Java... JavaScript (Node.js).
- Знание API (REST, SOAP — всё реже).
- Понимание принципов работы серверов Apache, NGINX, IIS и проч.
- Навыки написания unit-тестов и покрытия кода тестами.
- Основы сетевой безопасности и знание инструментов её обеспечения.
- Знание популярных веб-фрейморков, которые способны решать задачи разработки конкретного приложения.
- Проектирования баз данных.
- Знание основ frontend.

REST API (RESTful веб-сервисы)

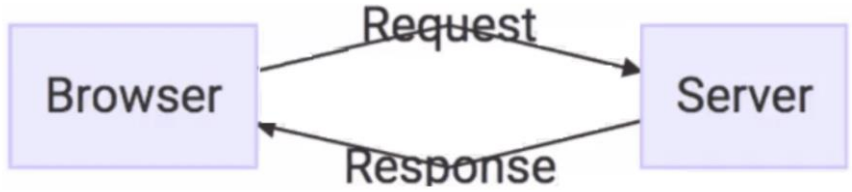
API — это Application Programming Interface, или программный интерфейс приложения, с помощью которого одна программа может взаимодействовать с другой. API позволяет отправлять информацию напрямую из одной программы в другую, минуя интерфейс взаимодействия с пользователем.

Какие бывают? API может быть внутренним, частным — когда программные компоненты связаны между собой и используются внутри системы. А может быть открытым, публичным — в таком случае он позволяет внешним пользователям или другим программам получать информацию, которую можно интегрировать в свои приложения.

Чтобы программам общаться между собой, их API нужно построить по единому стандарту. Одним из них является REST — стандарт архитектуры взаимодействия приложений и сайтов, использующий протокол HTTP.

REST означает REpresentational State Transfer (Википедия: «передача состояния представления»). Это популярный архитектурный подход для создания API в современном мире.

Это был термин, первоначально введен Roy Fielding, который также был одним из создателей протокола HTTP. Отличительной особенностью сервисов REST является то, что они позволяют наилучшим образом использовать протокол HTTP.



Когда вводите в браузере URL-адрес, например, www.google.com, на сервер, указанный по URL-адресу, отправляется запрос на сервер. Затем этот сервер формирует и выдает ответ. Важным является формат этих запросов и ответов. Эти форматы определяются протоколом HTTP — Hyper Text Transfer Protocol.

Когда набираете URL в браузере, он отправляет запрос GET на указанный сервер. Затем сервер отвечает HTTP-ответом, который содержит данные в формате HTML — Hyper Text Markup Language. Затем браузер получает этот HTML-код и отображает его на экране.

Ресурс — это ключевая абстракция, на которой концентрируется протокол HTTP.

Когда вы разрабатываете RESTful сервисы, вы должны сосредоточить свое внимание на ресурсах приложения. Способ, которым идентифицируется ресурс для предоставления, состоит в том, чтобы назначить ему URI — универсальный идентификатор ресурса. Например:

- Создать пользователя: POST /users
- Удалить пользователя: DELETE /users/1
- Получить всех пользователей: GET /users
- Получить одного пользователя: GET /users/1

Чтобы распределенная система считалась сконструированной по REST архитектуре (Restful), необходимо, чтобы она удовлетворяла следующим критериям:

1. Client-Server. Система должна быть разделена на клиентов и на серверов. Разделение интерфейсов означает, что, например, клиенты не связаны с хранением данных, которое остается внутри каждого сервера, так что мобильность кода клиента улучшается. Серверы не связаны с интерфейсом пользователя или состоянием, так что серверы могут быть проще и масштабируемы. Серверы и клиенты могут быть заменяемы и разрабатываться независимо, пока интерфейс не изменяется.
2. Stateless. Сервер не должен хранить какой-либо информации о клиентах. В запросе должна храниться вся необходимая информация для обработки запроса и, если необходимо, идентификации клиента.
3. Cache. Каждый ответ должен быть отмечен является ли он кэшируемым или нет, для предотвращения повторного использования клиентами устаревших или некорректных данных в ответ на дальнейшие запросы.

4. Uniform Interface. Единый интерфейс определяет интерфейс между клиентами и серверами. Это упрощает и отделяет архитектуру, которая позволяет каждой части развиваться самостоятельно.
    - HATEOAS (hypermedia as the engine of application state). Статус ресурса передается через содержимое body, параметры строки запроса, заголовки запросов и запрашиваемый URI (имя ресурса). Это называется гипермедиа (или гиперссылки с гипертекстом).
5. Layered System. В REST допускается разделить систему на иерархию слоев, но с условием, что каждый компонент может видеть компоненты только непосредственно следующего слоя. Например, если вы вызывайте службу PayPal, а он в свою очередь вызывает службу Visa, вы о вызове службы Visa ничего не должны знать.

Методы HTTP-запроса

Метод, используемый в HTTP-запросе, указывает, какое действие вы хотите выполнить с этим запросом.

- GET: получить информацию о ресурсе
  HTTP метод GET используется для получения (или чтения) представления ресурса. В случае “удачного” (или не содержащего ошибок) адреса, GET возвращается представление ресурса в формате XML или JSON в сочетании с кодом состояния HTTP 200 (OK). В случае наличия ошибок обычно возвращается код 404 (NOT FOUND) или 400 (BAD REQUEST).

  GET http://www.example.com/api/v1.0/users (вернуть список пользователей)
  GET http://www.example.com/api/v1.0/users/12345 (вернуть данные о пользователе с id 12345)

- PUT: обновить существующий ресурс
  HTTP метод PUT обычно используется для предоставления возможности обновления ресурса. Тело запроса при отправке PUT-запроса к существующему ресурсу URI должно содержать обновленные данные оригинального ресурса (полностью, или только обновляемую часть).
  Для создания новых экземпляров ресурса предпочтительнее использование POST запроса. В данном случае, при создании экземпляра будет предоставлен корректный идентификатор экземпляра ресурса в возвращенных данных об экземпляре.
  При успешном обновлении посредством выполнения PUT запроса возвращается код 200 (или 204 если не был передан какой либо контент в теле ответа).

  PUT http://www.example.com/api/v1.0/users/12345 (обновить данные пользователя с id 12345)
  PUT http://www.example.com/api/v1.0/users/12345/orders/98765 (обновить данные заказа с id 98765 для пользователя с id 12345)

- POST: создать новый ресурс
  При успешном создании ресурса возвращается HTTP код 201.

POST http://www.example.com/api/v1.0/customers (создать новый ресурс в разделе customers)
POST http://www.example.com/api/v1.0/customers/12345/orders (создать заказ для ресурса с id 12345)

- DELETE: Удалить ресурс
  При успешном удалении возвращается 200 (OK) код HTTP, совместно с телом ответа, содержащим данные удаленного ресурса. Также возможно использование HTTP кода 204 (NO CONTENT) без тела ответа.
  Повторный DELETE запрос к ресурсу часто сопровождается 404 (NOT FOUND) кодом HTTP по причине того, что ресурс уже удален (например из базы данных) и более не доступен.

  DELETE http://www.example.com/api/v1.0/customers/12345 (удалить из customers ресурс с id 12345)
  DELETE http://www.example.com/api/v1.0/customers/12345/orders/21 (удалить у ресурса с id 12345 заказ с id 21)

Код статуса ответа HTTP

Код состояния всегда присутствует в ответе HTTP.

Основные группы кодов состояний:

2xx: Success

- 200: OK
- 201: Created
- 202: Accepted
- 204: No Content

4xx: Client Error

- 400: Bad Request
- 401: Unauthorized
- 403: Forbidden
- 404: Not Found

5xx: Server Error

- 500: Internal Server Error
- 501: Not Implemented
- 502: Bad Gateway
- 503: Service Unavailable
- 504: Gateway Timeout

Примеры RESTFul API вебсервисов

1. OMDb API The Open Movie Database http://www.omdbapi.com/ - информация о фильмах
2. RapidAPI (https://rapidapi.com/marketplace) — самый большой в мире marketplace с более чем 10 000 API (и около 1 млн разработчиков).
   RapidAPI не только предоставляет единый интерфейс для работы со сторонними API, но и дает возможность быстро и без проблем опубликовать ваш собственный API.