HTTP

В клиент-серверной модели клиенты и серверы обмениваются сообщениями по принципу «запрос-ответ»: клиент отправляет запрос, а сервер возвращает ответ.

Хранить трек из этих сообщений сложнее, чем звучит, поэтому клиент и сервер придерживаются общего языка и набора правил. Этот «язык», или протокол, называется **HTTP**.

Протокол HTTP определяет синтаксис (формат и кодировку данных), семантику (значение, связанное с синтаксисом) и тайминг (скорость и последовательность). Каждый HTTP-запрос и ответ, которыми обмениваются клиент и сервер, рассматривается как одна HTTP-транзакция.

HTTP: ОБЩАЯ ИНФОРМАЦИЯ

Во-первых, HTTP текстовый протокол, что означает, что сообщения, которыми обмениваются клиент и сервер, являются битами текста. Каждое сообщение содержит две части: заголовок и тело.

Во-вторых, HTTP - это протокол прикладного уровня, то есть это просто абстракционный уровень, который стандартизирует взаимодействие хостов. Сам HTTP не передает данные. Получение запроса и ответа от одной машины к другой по-прежнему зависит от базового протокола **TCP/IP**.

TCP/IP - это двухкомпонентная система, которая функционирует как фундаментальная «система управления» Интернета.

Простой HTTP-запрос или ответ не зашифрован и уязвим для различных типов атак. HTTPS, напротив, является более безопасной протоколом связи, которая использует **TLS/SSL** шифрование для обеспечения безопасности.

SSL - это протокол безопасности, который позволяет клиенту и серверу взаимодействовать по сети безопасным способом - чтобы предотвратить сниффинг и подмену во время передачи сообщений по сети.

Клиент обычно указывает, требуется ли ему подключение TLS/SSL, используя специальный номер порта 443. Как только клиент и сервер соглашаются использовать TLS/SSL для обмена данными, они согласовывают соединение с отслеживанием состояния, выполняя так называемое «квитирование TLS». Затем клиент и сервер устанавливают секретные сеансовые ключи, которые они могут использовать для шифрования и дешифрования сообщений, когда они разговаривают друг с другом.

Многие крупные веб-сайты, такие как Google и Facebook, используют HTTPS - в конце концов, это то, что сохраняет ваши пароли, личную информацию и данные кредитных карт в безопасности.

ЗАГОЛОВОК ЗАПРОСА HTTP

Заголовки HTTP обычно содержат метаданные (данные о данных). Метаданные включают тип запроса (**GET**, **POST**, **PUT** или **DELETE**), путь, код состояния, тип содержимого, используемый браузер, cookie, текст сообщения (иногда) и многое другое.

ТЕЛО HTTP

Выше мы узнали, что сервер содержит большинство важных «метаданных» (данные о данных), которые необходимы для связи с клиентом.

Теперь поговорим о теле HTTP запроса.

Тело – это основная часть сообщения. В зависимости от типа запроса он может быть и пустым.

МЕТОДЫ HTTP

Команды или методы HTTP указывают серверу, что делать с данными, определенными по URL. URL-адреса всегда идентифицируют определенный ресурс. Когда клиент использует

URL-адрес в сочетании с командой HTTP, это сообщает серверу, какое действие необходимо выполнить с указанным ресурсом.

Для работы с WEB-сервисами HTTP использует методы. То же самое касается и RESTful API.

Возможные сценарии в реальной жизни описываются термином **CRUD — Create, Read, Update, Delete**.

Вот список наиболее популярных методов HTTP, реализующих CRUD:

- HTTP GET
- HTTP POST
- HTTP PUT
- HTTP DELETE
- HTTP PATCH

Методы также называются **глаголами**, поскольку указывают на то, какое действие производится. Примеры URL-адресов:

- **GET** http://www.example.com/users (получить всех пользователей)
- **POST** http://www.example.com/users/a-unique-id (создание нового пользователя)
- **PUT** http://www.example.com/comments/a-unique-id (обновить комментарий)
- **DELETE** http://www.example.com/comments/a-unique-id (удалить комментарий)

Когда клиент делает запрос, он указывает тип запроса, используя одну из этих команд. Наиболее важными являются **GET**, **POST**, **PUT** и **DELETE**. Есть и другие методы, такие как **HEAD**, **OPTIONS**, **CONNECT, TRACE, PATCH**, но они используются редко.

GET

GET является наиболее часто используемым методом. Он используется для чтения информации по данному URL-адресу с сервера.

Запросы GET доступны только для чтения, что означает, что данные никогда не должны быть изменены на сервере - сервер должен просто извлечь данные без изменений. Таким образом, запросы GET считаются безопасными операциями, поскольку сколько бы не вызывай его, ответ будет одинаковым.

Кроме того, запросы GET являются идемпотентными. Это означает, что отправка нескольких запросов GET на один и тот же URL-адрес должна привести к тому же эффекту, что и один запрос GET, поскольку запрос GET просто запрашивает данные с сервера, а не изменяет их.

Запросы GET отвечают кодом состояния 200 (OK), если ресурс был успешно найден, и 404 (NOT FOUND), если ресурс не был найден. (Отсюда термин «404 page» для сообщений об ошибках при посещении несуществующих или неправильно набранных URL-адресов.)

POST

POST используется для создания нового ресурса, например, через форму регистрации. Функция POST используется при необходимости создания дочернего ресурса (например, нового пользователя) для какого-либо родительского ресурса (http://example.com/users). Родительский ресурс запроса на создание новой сущности определяется по URL-адресу, и сервер обрабатывает новый ресурс и связывает его с родительским ресурсом.

POST не является ни безопасным, ни идемпотентным. Это связано с тем, что выполнение двух или более идентичных запросов POST приведет к созданию двух новых идентичных ресурсов.

Запросы POST отвечают кодом состояния 201 (CREATED) вместе с заголовком местоположения со ссылкой на вновь созданный ресурс.

PUT

PUT используется для обновления ресурса, идентифицированного по URL, с использованием информации в теле запроса. PUT также может использоваться для создания нового ресурса. Запросы PUT не считаются безопасными операциями, поскольку они

изменяют данные на сервере. Однако он является идемпотентным, поскольку несколько идентичных запросов PUT на обновление ресурса должны иметь тот же эффект, что и первый.

Запросы PUT отвечают кодом состояния 200 (OK), если ресурс был успешно обновлен, и 404 (NOT FOUND), если ресурс не был найден.

DELETE

DELETE используется для удаления ресурса, определенного по URL-адресу. Запросы DELETE являются идемпотентным, поскольку если УДАЛИТЬ ресурс, он будет удален, и даже если сделать несколько идентичных запросов DELETE, результат будет одинаковым: удаленный ресурс.

Скорее всего, просто получите сообщение об ошибке 404, если отправить запрос DELETE для одного и того же ресурса несколько раз, поскольку сервер не сможет найти его после удаления.

Запросы DELETE отвечают кодом состояния 200 (OK) в случае успешного удаления или 404 (NOT FOUND), если не удалось найти удаляемый ресурс.

Все вышеуказанные запросы возвращают значение 500 (ВНУТРЕННЯЯ ОШИБКА СЕРВЕРА), если обработка завершается неуспешно и сервер выдаёт ошибку.

REST

HTTP-методы не что иное, как часть REST.

REST расшифровывается как **Representational State Transfer** (Передача состояния представления). Это архитектурный стиль проектирования приложения.

Основная идея заключается в том, что для выполнения вызовов между машинами используется протокол «без состояния», «клиент-сервер», «кэшируемый» - и чаще всего этот протокол HTTP. В общем, REST это согласованный набор ограничений для проектирования приложения. Эти ограничения помогают сделать систему более производительной, масштабируемой, простой, изменяемой, видимой, портативной и надежной.

Два наиболее важных из них:

1. **Унифицированный интерфейс - Uniform interface**: это ограничение позволяет определить интерфейс между клиентом и сервером путь, чтобы упростить и разъединить архитектуру

2. **Отсутствие состояния - Stateless**: это ограничение говорит о том, что все данные о состоянии, необходимые для обработки запроса клиента, должны содержаться в самом запросе (URL, параметры запроса, тело HTTP или заголовки HTTP), а сервер должен отправить все необходимые данные о состоянии клиенту через сам ответ (заголовки HTTP, код состояния и тело ответа HTTP).

Это означает, что для каждого запроса пересылаем информацию о состоянии туда и обратно, так что сервер не должен поддерживать, обновлять и отправлять состояние.

Наличие системы без сохранения состояния делает приложения намного более масштабируемыми, потому что ни один сервер не должен беспокоиться о поддержании одного и того же состояния сеанса на протяжении нескольких запросов. Все необходимое для получения данных о состоянии доступно в самом запросе и ответе.