REST - архитектурный стиль, с помощью которого описывается структура передачи информации у веб-сервисов

REST API - определенная архитектура (набор ограничений, которым нужно следовать)

Принципы REST API:

1) REST основан на доступе к ресурсам

На Сервере четко обозначен ресурс, который отвечает за обработку запросов

Пример: https://swapi.dev/api/film, где /film - это ресурс(эндпоинт)

2) У ресурса должен быть идентификатор

URI = URL + URN

Пример: https://wikipedia.org/wiki/Canada, где wikipedia.org - это URL, wiki/Canada - URN, а все вместе - URI

3) Клиент и Сервер общаются неким представлением о ресурсе

Тип данных - любой(JSON, XML, HTML...). Делая запрос, приходит представление

4) REST API управляются гиперссылками в представлении

Представление, это например JSON файл, который пришел в ответе, а в ответе будет вся информация, куда и по какой ссылке мы можем перейти

5) Клиент и Сервер независимы друг от друга

К Серверу может обращаться множество Клиентов и ему не важно кто это, как и Клиенту не важно, что за Сервер

Пример: к Серверу могут обратиться браузер, мобильное приложение, Postman и т.д.

6) Единый интерфейс для всех (для всех Клиентов со стороны Сервера)

В независимости от того, какой Клиент обращается к Серверу, Сервер использует один и тот же интерфейс

Интерфейс в API - URI, на который мы обращаемся, эндпоинт

7) Сервер не хранит состояние, это может делать только Клиент

Ответ, который вернул Сервер может быть закэширован

Сервер может как разрешить кэширование, так и запретить его (Cache-Control в хедерах)

cache-control : private, max-age = 0, no-cache
cache-control : no-transform, public, max-age = 300

private - личная информация, которая относится лично к пользователю
max-age - максимальное время жизни (например время жизни 300 неких единиц)
no-cache - кэшировать не надо
public - публичная информация
no-transform - нельзя изменять информацию (кодировать, шифровать)

8) Многоуровневая система, где компоненты одного уровня могут общаться только со следующим уровнем

Клиент -> Балансировщик -> Сервер -> Балансировщик -> БД

Клиент не может перескочить сразу на уровень Сервера

*Необязательно все принципы REST должны соблюдаться, так появилось понятие RESTful, т.е. когда приложение полностью придерживается принципов REST архитектуры

*Катенация используется для того, чтобы не загружать ответ большим количеством информации, поэтому идет разветвление [ключ : ссылка]

Передача данных через GET:

https://swapi.dev/api/people/?page=2

https://swapi.dev/api/film - ресурс, на который мы обращаемся
? - разделитель
page - параметр
2 - значение параметра

Уровни зрелости REST API:

Уровень 0 - значит, что у нашего REST API есть один URI на все операции и все это работает на POST запросах (вся информация передается через тело)

Уровень 1 - есть отдельные URI для разных ресурсов

Уровень 2 - ресурсы работают с различными методами и готовы выполнять различные операции в зависимости от метода

Уровень 3 - в ответе есть гиперссылки для навигации по ресурсу

*У SOAP есть документация - WSDL, у REST API - Swagger

Swagger: 1) документация REST API

2) обычно генерируется разработчиком

3) разработчик генерирует Swagger файл, который можно забрать и генерировать в Postman и на основании этого файла появится вся документация по API(что в ней есть, тип данных, ресурсы, как к ним обращаться)

4) форматы YAML, JSON

Swagger UI - визуализирует ресурсы OpenAPI и взаимодействия с ними без отображения логики реализации. Выполняет запросы для тестирования API