What is a REST API?

In this lesson, you will get an insight into the REST API

We'll cover the following

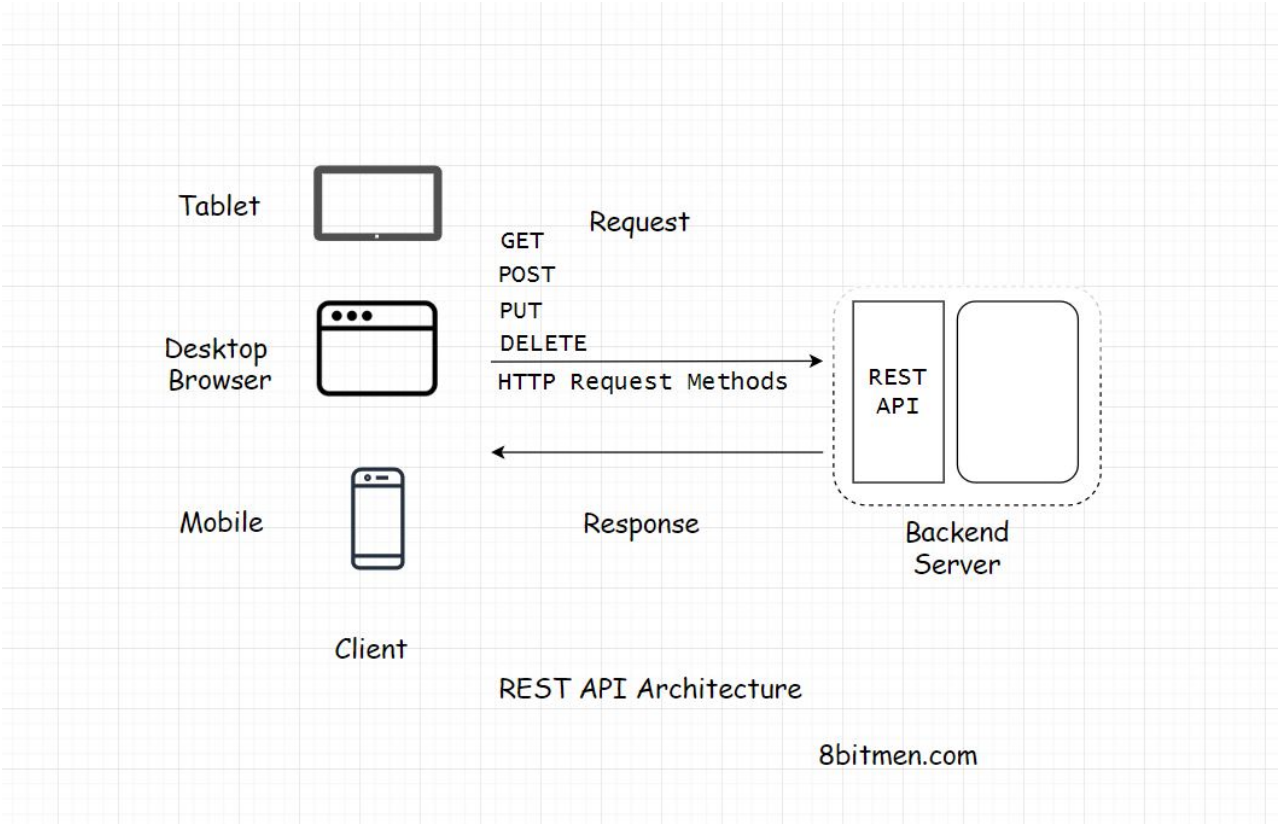
What is REST?#

> REST stands for Representational State Transfer. It's a software architectural style for implementing web services. Web services implemented using the REST architectural style are known as the RESTful Web services.

REST API#

A *REST API* is an *API* implementation that adheres to the REST architectural constraints. It acts as an interface. The communication between the client and the server happens over *HTTP*. A *REST API* takes

advantage of the *HTTP* methodologies to establish communication between the client and the server. *REST* also enables servers to cache the response that improves the application's performance.



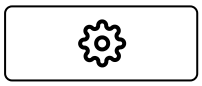
The communication between the client and the server is a stateless process. By that, I mean every communication between the client and the server is like a new one.

There is no information or memory carried over from the previous communications. So, every time a client interacts with the backend, the client has to send the authentication information to it as well. This enables the backend to figure out whether the client is authorized to access the data or not.

When implementing a REST API the client communicates with the backend endpoints. This entirely decouples the backend and the client code.

Let's break down what this means.

REST endpoint#



An *API/REST/Backend* endpoint means the *URL* of a service. For example, `https://myservice.com/users/{username}` is a backend endpoint for fetching the user details of a particular user from the service.

The *REST-based* service will expose this *URL* to all its clients to fetch the user details using the above stated *URL*.

Decoupling clients and the backend service#

With the availability of the endpoints, the backend service does not have to worry about the client implementation. It just calls out to its multiple clients and says "*Hey everyone! Here is the URL address of the resource/information you need. Hit it when you need it. Any client with the required authorization to access a resource can access it*".

Developers can have different implementations with separate codebases, for different clients, on a mobile browser, a desktop browser, a tablet or an API testing tool. Introducing new types of clients or modifying the client code has no effect on the functionality of the backend service.

This means the clients and the backend service are decoupled.

Application development before the REST API#

Before the *REST-based API* interfaces became mainstream in the industry, we often tightly coupled the backend code with the client. *Java Server*

Pages (JSP) is one example of this.

We would always put business logic in the *JSP* tags. This made code refactoring and adding new features difficult because the logic got spread across different layers.

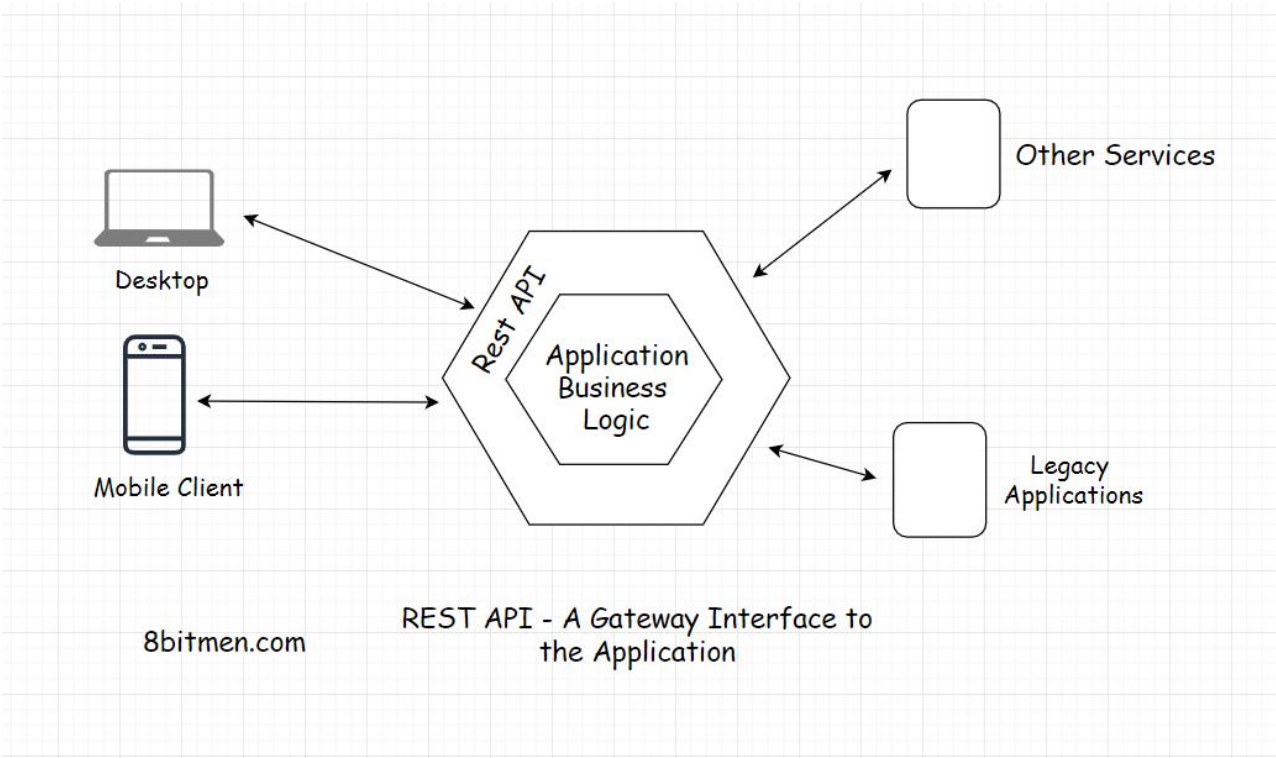
Also, in the same codebase, we had to write separate code/classes for handling requests from different types of clients. We needed a different servlet for a mobile client and a different one for a web-based client.

After *REST APIs* became widely used, there was no need to worry about the type of the client. Just provide the endpoints and the response will generally contain data in the *JSON* or any other standard data transport format. Additionally, the client will handle the data in whatever way they want.

This cut down a lot of unnecessary work for us. Also, adding new clients became a lot easier. Now, we can introduce multiple types of new clients without considering the backend implementation.

In today's industry landscape, there are hardly any online service without a *REST API*. Want to access the public data of any social network? Use their *REST API*.

API gateway#



The *REST-API* acts as a gateway, or a single-entry point into the system. It encapsulates the business logic and handles all the client requests, taking care of the authorization, authentication, sanitizing the input data, and other necessary tasks before providing access to the application resources.

So, now you are aware of the client-server architecture and we know what a *REST API* is. It acts as the interface, and the communication between the client and the server happens over HTTP.

Let's look into the HTTP Pull and Push-based communication mechanism.

← Back
Web Architecture Quiz - Part 1

Next →
HTTP Push and Pull - Introduction

Completed

Report an Issue