Setting Up the REST API Project

Steve Buchanan

DevOps Architect

@buchatech | www.buchatech.com

Overview

Exploring the REST API

REST API Design Principles

Planning & Structuring your REST API

Storing the REST API Data

Exploring the REST API

Overview of the REST API

REST or RESTful is the internet's language it stands for REpresentational State Transfer.

REST is an architectural style & an approach for communications often used in web dev & for systems to communicate.

REST API is an interface used by two or more computer systems to exchange data securely via HTTP requests GET, PUT, POST & DELETE over the internet.

Reasons for REST APIs

Limited bandwidth	Ease of coding	Easy of integration	Cache
HTTP calls used by with REST APIs are fast, & lightweight making them efficient when bandwidth is limited	Coding & implementation of a REST API service is easier than that of a SOAP service	Any client can use REST API apps, even if the apps were not designed specifically for the client using it.	REST makes caching easier since the server is stateless & requests can be processed individually making them easily cacheable

REST API Concepts

Client

- software that runs on a user's computer or smartphone and initiates communication

Server

- offers a REST API as a means of access to the app data or features

Stateless

- Client sends all info necessary to understand the request. The server does have any stored context or session state & can not reuse info from any previous requests

Header

- store information relevant to both the client & server such as API key, name, IP address, & info about the response format

Body

- used to convey additional information to the server such as data you want to add or replace

Endpoint

- a Uniform Resource Identifier (URI) indicating where and how to find the resource on the Internet such as a Unique Resource Location (URL)

Cacheable

- require that the data within a response to a request be implicitly or explicitly labeled as cacheable or non-cacheable

Uniform Interface

- a consistent interface across all APIs that functions as a contract between the client & the service, & it is shared by all REST APIs

Layered System

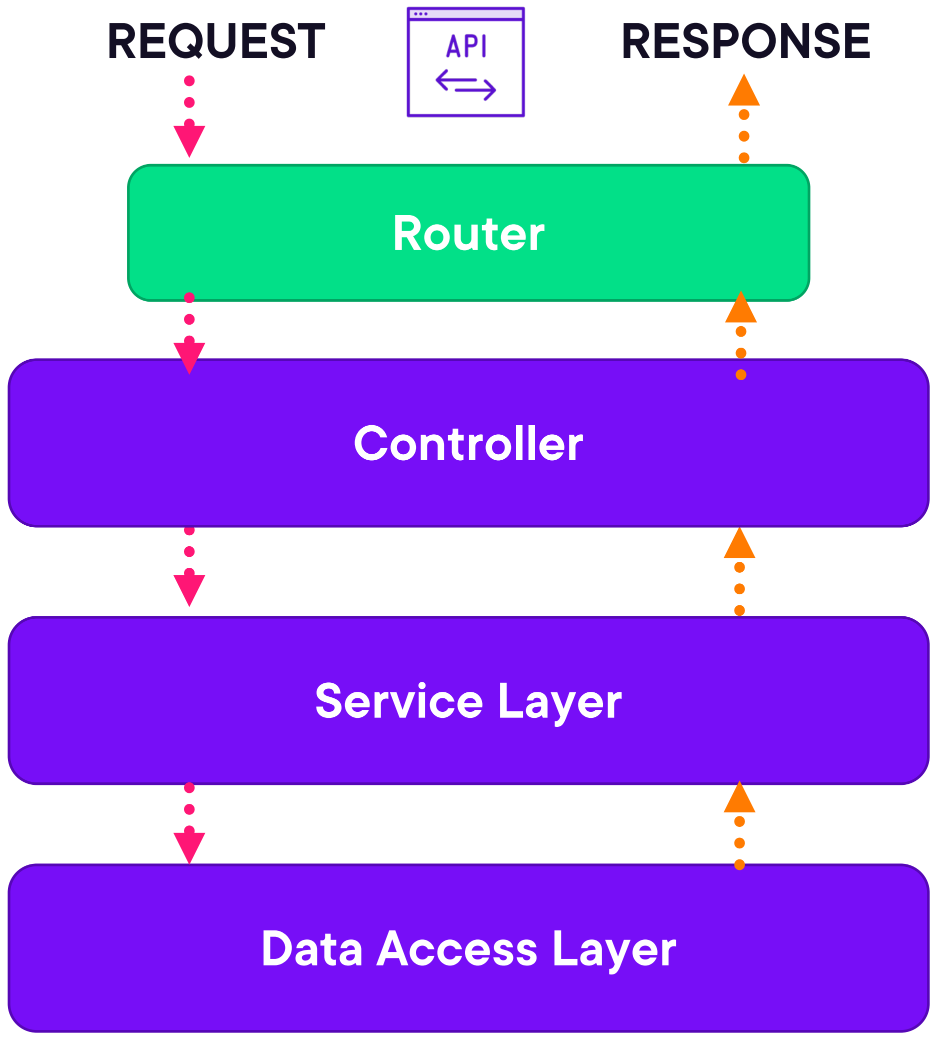
- Every REST-enabled component has no access to components other than the one with whom it is communicating

Code On Demand (CoD)

- allows client functionality to be extended by downloading and executing code in the form of applets or scripts

REST API Architecture

- Router from Express that passes requests to the corresponding controller
- Controller handles all stuff related to HTTP/S
- Service Layer has the business logic & exports services (methods) used by the controller
- Data Access Layer where interacting with the Database happens



REST API Benefits

Independent - client and server are independent. In other words, the REST protocol separates the data storage and the UI from the server

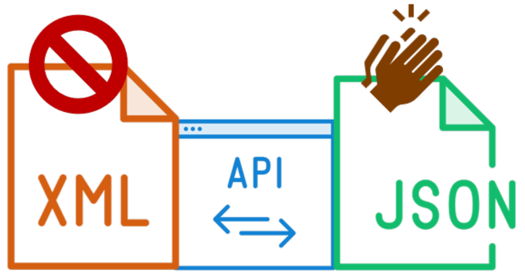
Scalability - REST APIs can be scaled by a dev team without much difficulty due to the separation between the client and the server.

Easy Integration - the use of ubiquitous standards, coupled with support across many languages, the use of HTTP protocol, & use by many devs with their apps are a recipe for ease of integration among many platforms.

Flexibility - it is easy for devs to understand & Implement, it is format-agonistic with the ability to work with XML, JSON, HTML, etc., & easy to modify.

REST API Design Principles

REST API Design Principles



Accept & respond with JSON

JavaScript has built-in methods to encode & decode JSON.

Almost every networked

☹

/getAllStudents

☺

/students

Use nouns instead of verbs in endpoint paths

Our HTTP request method already has the verb

It is better to let the HTTP

/students?limit=10&offset=25

Allow API to filter, order, & paginate data

Limit the amount of data the REST API will return
This helps conserve bandwidth & the potential of overloading the REST API server

https://schooldirectory.com/v2/teachers

Version the REST API

Versioning indicates a REST APIs features & exposed resources

Versioning is typically done using /v1/, /v2/, etc. at the start of the REST APIs path.

REST API Design Principles

```
const express =
require('express');

const bodyParser =
require('body-parser');

const redis = require('redis');

const app = express();

let cache = redisClient;

app.use(cache('5 minutes'));
```

Use Caching

Caching data will improve performance

We can add caching to return data from the local memory cache instead of querying the database to get the data every time

middleware such as redi,, apicache etc is needed to handled the caching

Some Frequently used error codes in REST APIs:

200-Ok HTTP response for every success for GET, PUT or POST.

201-Created HTTP response for success and resource had created and not response body or response body is empty.

404 Not Found HTTP response for the requested resource is not available to access.

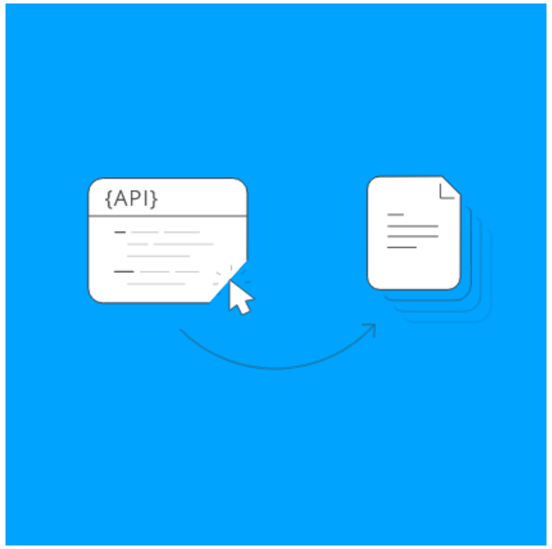
500 INTERNAL SERVER ERROR — HTTP response indicates that there might be a system failure.

```
app.post('/students', (req, res) => {

  const { email } = req.body;

  const userExists = users.find(st =>
u.email === email);

  if (studentExists) {

    return res.status(404).json({ error:
'Student already exists' })

}
```

Proper HTTP error handling

REST APIs use HTTP calls if an error/exception is thrown from the server side, it will give HTTP error

The REST API should handle errors gracefully & return HTTP response codes to the invoked client



Document

Document your API

Start documentating your API from its design phase

You can use a tool to help build your API documentation

Swagger is a set of open-source tools built around the OpenAPI Spec that can help build the documentation of your API

Planning & Structuring your REST API

Planning a REST API

Step 1	**Understand WHY you are building an API** • Who is your target user for this API? • What products/services & data do you want them to access? • What technologies / systems will need to integrate with the API?
Step 2	**List out the user FUNCTIONALITY of your API** • List out what your API needs to do. • Build out user stories from this list. • Breakdown the work and build a backlog from your list.
Step 3	**Determine the TYPE of API you are building** • Will this be a REST API, SOAP based, RPC etc.?
Step 4	**Build initial roadmap for your API** • Build out the long-term strategy and direction for your API. It is never too early to start planning for this. This roadmap can change over time.

Authentication Methods with a REST API

It is recommended to secure REST APIs access using SSL & TLS

There are four fundamental approaches to authentication for REST APIs

The easiest way to handle authentication with a REST API in JavaScript is to use HTTP basic auth. It sends a username & password alongside every API call.

This creates unique keys for devs & passes them alongside every request. The API generates a secret key that is a long, difficult-to-guess string of letters & numbers.

This orchestrates approvals automatically between an API owner & the service.

You can also authorize devs to obtain access on their own.

Another approach is to have no auth. Requests can be made to a specific URL & get a response without an API key or any credentials.

This is not recommended but is sometimes used for internal only APIs.

Structuring Your REST API

project folder

proj1

src folder

Src

- v1

sub folders

controllers

routes

services

database

```
/proj1
    /src
        /v1/
            controllers
            routes
            services
            database
            ...
```

Demo

Demo: Create Directories for a REST API project

Storing the REST API Data

Using an Array or Database to Store REST API Data

There are many ways to access data from JavaScript. You would use Node.js. The most common way is to connect from your JavaScript code to a database. To access the database from Node.js, you first need to install drivers for the database

JavaScript supports relational and NoSQL databases. JavaScript supports connecting to the following databases:

Relational	NoSQL
MS SQL	MongoDB
PostgreSQL	Cassandra
MySQL	LevelDB
SQLite	Redis
Oracle	CouchDB

We also can store data in other types such as an array or even a JSON file.

For production using a database to store data is recommended.

Using an Array or Database to Store REST API Data

Storing Data in a Database with a REST API

database.js

```
const mysql = require('mysql');
const dbconnection = mysql.createConnection({
 host: "localhost",
 user: “user",
 password: “p@ssw0rd145",
 database: "schooldir",
});

dbconnection.connect();
```

← npm install mysql

← Create JS code to connect to a MySQL database utilizing the MySQL createConnection method.

← You can pass host, username, password, and database name as required parameters to the createConnection method.

← Add *“const dbconnection = require("../database");”*

Using an Array or Database to Store REST API Data

Storing Data in Array with a REST API

index.js

```
const express = require('express')
const bodyParser = require('body-parser');
const cors = require('cors');

const app = express();
const port = 3000;

const teachers = [
    "John Smith",
    "Deshan Jackson",
    "Susan Johnson",
    "Alexis Frederickson"
    ];
```

← An array is a special variable, which can hold more than one value like a list.

← const teachers shows an example of array. The data from the array can be used in our REST API.

Demo

Demo: Build an Array in JavaScript to simulate a database

Summary

In this module we covered:

- What REST APIs are
- Design Principles when building REST APIs
- How to plan and structure your REST API
- Ways to store data for your REST API

Why this is important?

- It is important to understand what REST APIs are and how they differ from a traditional server-side apps.
- As you build out a REST API it is good to understanding how to design and plan for it.
- Last it is critical to know how to properly structure your REST API project in JavaScript.

Up Next:

Building and Testing the REST API