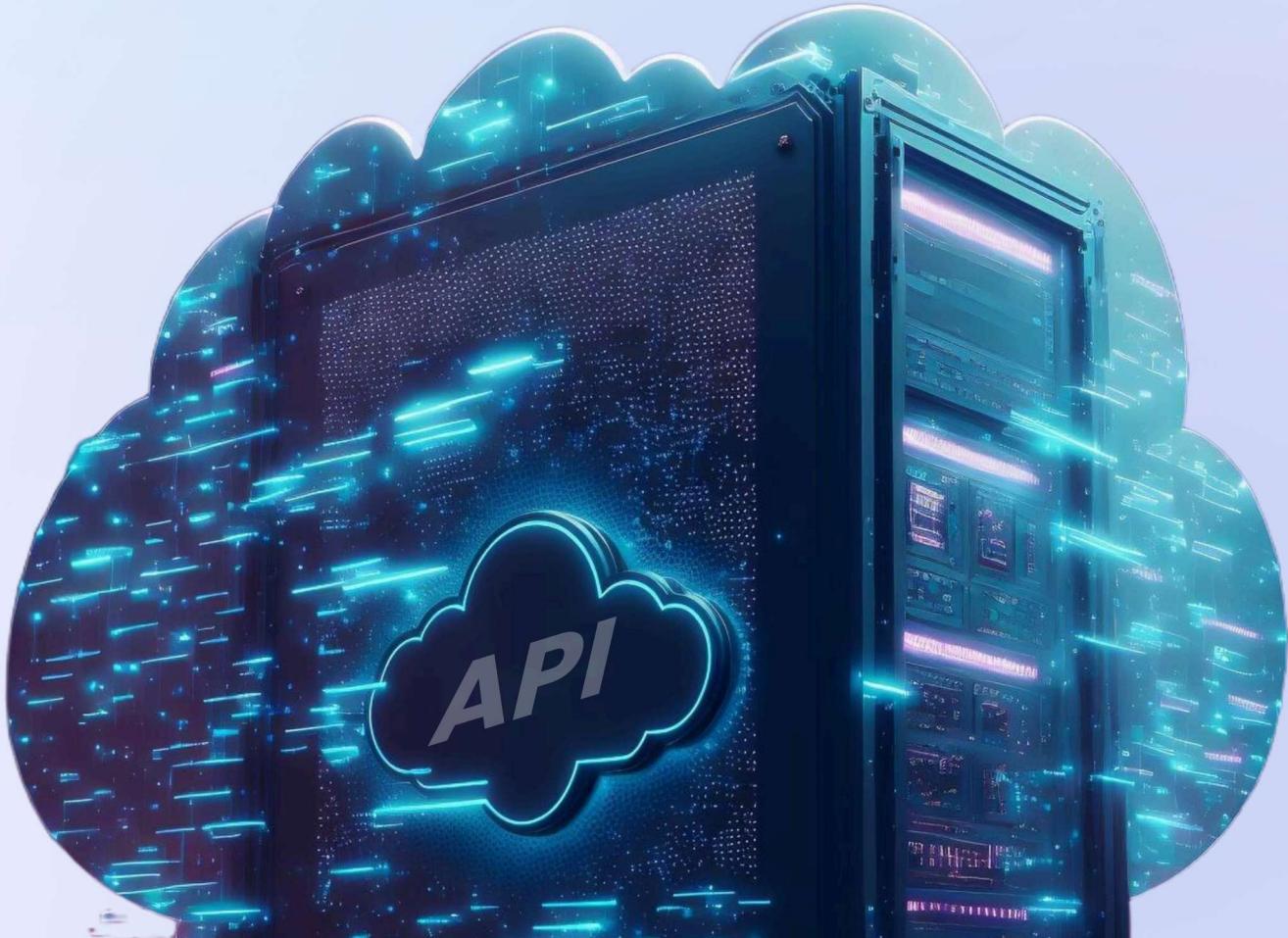




# Backend HandBook

Created by **JS Mastery**

Visit *jsmastery.pro* for more



# What's in the guide?

Welcome! If you've ever felt overwhelmed by backend development or API concepts, this guide is here to make things simple.

If you're just starting out, we'll walk through everything you need to know—step by step.

We'll start with a structured Backend Dev Roadmap, giving you a clear path to mastering backend skills.

From there, we'll dive into core API concepts, breaking down REST API architecture, CRUD operations, HTTP verbs, and common status codes in a way that actually makes sense. And project ideas in the end.

By the time you reach the end, you'll have a solid grasp of backend development and the confidence to build and interact with APIs like a pro. Let's dive in! 

# Choose a Programming language



JavaScript



Python



PHP



Ruby



Rust



Java



Go Lang

# Then choose a Runtime according to your programming language

If you're working with JavaScript, selecting an appropriate runtime environment is essential.

 **JavaScript**

**JavaScript Runtime**



# Node.js Popular Framework

---

ex      Express.js

 Nest.js Fastify Sails Hapi Restify Koa



# Python Popular Framework

---



Django



Flask



PHP Popular Framework

---



Laravel



Symfony



Ruby

Popular Framework

---



Rails



Sinatra



Popular Framework

---



Spring



Google Web Toolkit



# Rust

# Popular Framework

---



## Rocket.rs



# Popular Framework

---



## Gin

---



## ASP .NET

---

asp.net is also a popular framework of C#

# Backend concepts you should learn

📌 HTTP / HTTPS

📌 REST

📌 GraphQL

📌 Web Sockets

📌 CORS

📌 MVC Architecture

📌 CI / CD

📌 Serverless

# Learn about databases

## 2 Types of Databases

SQL



MySQL

NoSQL



MongoDB



PostgreSQL

CouchDB



SQL Lite

DynamoDB



# Learn Caching



Redis



Memcached

## ORM / ODM



Mongoose (MongoDB)



Prisma (SQL)



Drizzle (SQL)

# Essential things to learn



Authentication



OAuth



Hashing (bcrypt)



Rate Limiting



Reverse proxy



Load balancer



Documentation (Swagger)

# Other Important Topics



**Linux**



**Terminal (CLI)**



**Git**



**Basic Networking**



**Web Security**



**Payment Gateways**



**Testing**

# Learn to deploy and learn any cloud service



Google Cloud



Digital Ocean



AWS



Azure



Heroku



Linode



Vercel

# Optional things to learn



Docker



Performance



Scalability



Kubernetes

# Tools to learn as a backend developer



NPM



Yarn



Postman



Insomnia



GitHub



# Complete Guide to APIs + Cheatsheet



APIs have become the center of software development, connecting and transferring data to millions of applications every day.

The guide covers simple explanations of all basic concepts of APIs as well as the most common HTTP status code and, more importantly, a list of popular APIs that you can use to improve your applications.

# What is API?

API stands for

**Application Programming Interface.**

It's a connection between computers or between computer programs. It is a type of software interface, offering a service to other pieces of software.

Basically, it enables apps to exchange data and functionality easily and securely.

It let your product or service communicate with other products or services without having to know how they're implemented.

# A great example to understand API

Imagine you're sitting at a table in a restaurant with a menu of choices to order from. The kitchen is the part of the "system" that will prepare your order.

What is missing is the critical link to communicate your order to the kitchen and deliver your food back to your table.

That's where the waiter or API comes in. The waiter is the messenger – or API – that takes your request or order and tells the kitchen – the system – what to do. Then the waiter delivers the response back to you; in this case, it is the food.

# How an API works?

01

APIs sit between an application and the web server, acting as an intermediary layer that processes data transfer between systems.

**01.** A client application initiates an API call to retrieve information also known as a request. This request is processed from an app to the web server via the API's Uniform Resource Identifier (URI) and includes a request verb, headers, and sometimes, a request body.

**02.** After receiving a valid request, the API makes a call to the external program or web server.

# How an API works?

02

03. The server sends a response to the API with the requested information.

04. The API transfers the data to the initial requesting application.

While the data transfer will differ depending on the web service being used, this process of requests and response all happens through an API.

Whereas a user interface is designed for use by humans, APIs are designed for use by a computer or application.

# REST API Architecture

01

REST stands for

**REpresentational State Transfer.**

The REST architecture is the most popular approach to build APIs.

REST relies on a client and server approach which separates front and back ends of the API, & provides considerable flexibility in development and implementation.

This means the implementation of the client and the implementation of the server can be done independently without each knowing about the other.

# REST API Architecture 02

APIs that adhere to REST principles are called RESTful.

A Restful system consists of a

- client who requests for the resources
- server who has the resources

In the REST architecture, clients send requests to retrieve or modify resources, and servers send responses to those requests.

By using a REST interface, different clients hit the same REST endpoints, perform the same actions, and receive the same responses.

# REST API Architecture 03

REST requires that a client make a request to the server in order to retrieve or modify data on the server.

**A request generally consists of:**

- an HTTP verb, which defines what kind of operation to perform
- a header, which allows the client to pass along information about the request
- a path to a resource
- an optional message body containing data

CRUD Stands for

## Create Read Update Delete

In a REST environment, CRUD often corresponds to the HTTP methods GET, POST, PUT/PATCH, and DELETE.

Create → POST

Read → GET

Update → PUT / PATCH

Delete → DELETE

In regards to its use in RESTful APIs, CRUD is the standardized use of HTTP Action Verbs.

This means that if you want to create a new record you should be using “**POST**”.

If you want to read a record, you should be using “**GET**”. To update a record use “**PUT**” or “**PATCH**”, And to delete a record, use “**DELETE**.”

Let’s learn about HTTP verbs, There are many HTTP verbs we use in requests to interact with resources in a REST system:

# HTTP Verbs

01

HTTP defines a set of request methods to indicate the desired action to be performed for a given resource.

These request methods are referred to as **HTTP Verbs**.

Below are the various types of HTTP Verbs

## GET

The GET method is used to retrieve specific resource. Requests using GET should only retrieve data and should have no other effect on the data.

## HEAD

Same as GET, but doesn't have a message-body in the response. The HEAD method is useful in recovering meta-data that is written according to the headers, without transferring the entire content.

## POST

A POST request is utilized to send data to a server to create a resource, for example, customer information, file upload, etc. usually using HTML Forms.

## PUT

PUT is similar to POST as it is used to send data to the server to create or update a resource. The difference between it replaces all current representations of the target resource with the uploaded content.

## DELETE

As it sounds, the DELETE request method is used to delete resources indicated by a specific URI. Making a DELETE request will remove the targeted resource.

## CONNECT

CONNECT request establishes a tunnel to the server identified by a specific URI. A good example is SSL tunneling.

## OPTIONS

The OPTIONS method requests permitted communication options for a given URL or server. A client can specify a URL with this method, or an asterisk (\*) to refer to the entire server.

## TRACE

The TRACE method performs a message loop-back test along the path to the target resource, to provide a useful debugging mechanism.

It allows clients to view whatever message is being received at the other end of the request chain so that they can use the info for testing or diagnostic functions.

## PATCH

The PATCH method is used for making partial changes to an existing resource.

The PATCH method provides an entity containing a list of changes to be applied to the resource requested using the URI.

## PUT vs PATCH

PUT method uses the request URI to supply a modified version of the requested resource which replaces the original version of the resource, whereas the PATCH method supplies a set of instructions to modify the resource.

# HTTP Status Codes

01

Status codes are issued by a server in response to a client's request made to the server.

The first digit of the status code specifies one of five standard classes of responses.

## Standard Classes

**1xx** Informational responses

**2xx** Successful responses

**3xx** Redirection responses

**4xx** Client error responses

**5xx** Server error responses

## 1xx - Informational responses

It indicates that the request was received and understood by the server and its continuing the process.

**100** Continue

**101** Switching Protocols

**102** Processing

**103** Early Hints

## 2xx - Successful responses

It indicates that the action requested by the client was received, understood, and accepted

**200** OK

**201** Created

**202** Accepted

**203** Non-Authoritative Information

**204** No Content

## 3xx – Redirection responses

Many of these 3xx status codes are used in URL redirection or it indicates the client must take additional action to complete the request.

**301** Moved Permanently

**302** Found

**304** Not Modified

**305** Use Proxy

**307** Temporary Redirect

**308** Permanent Redirect

## 4xx - Client error responses

This status code is intended for situations in which the error seems to have been caused by the client.

**400** Bad Request

**401** Unauthorized

**403** Forbidden

**404** Not Found

**406** Not Acceptable

**408** Request Timeout

## 5xx – Server error responses

It indicates that the server has encountered a situation where it doesn't know how to handle a request.

**500** Internal Server Error

**501** Not Implemented

**502** Bad Gateway

**503** Service Unavailable

**504** Gateway Timeout

**505** HTTP Version Not Supported

# HTTP Status Codes

07

These were not all the status codes that exists, there are many more status codes that indicates different things.

You can find and learn more about http status codes on this [website](#), with detailed information about that stauts code.

# Backend Project Ideas



Social Media Web App



Chat App



Video Chat App



REST API



Amazon Scraper API



Web Scraper



URL Shortner



CRUD Blog App

# Backend Project Ideas



Discord Clone with webRTC



Travel Log App



File Sharing App



Users System



Multiplayer Game



Gist Clone



Weather Logger



Ecommerce API

# Backend Project Ideas



AI-Powered Resume Analyzer



Dynamic API Generator



Custom Webhook System



Time Capsule API



AI-Customer Support Chatbot



Leaderboard API for Games



URL Expiry & Redirection Service

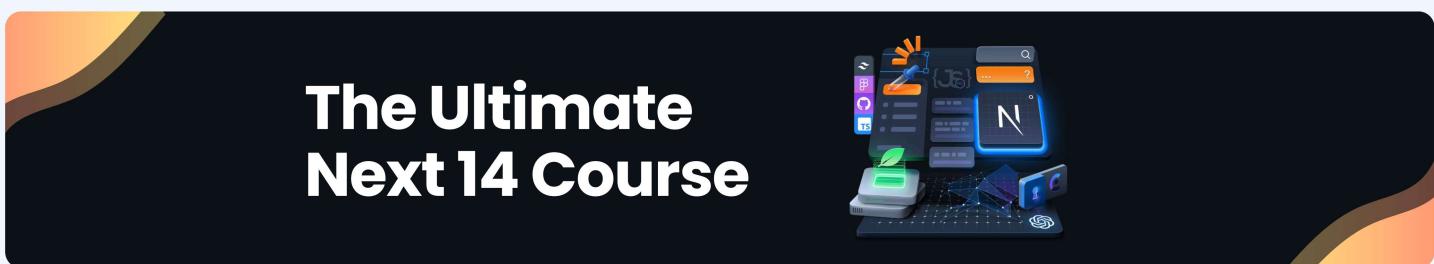


JSON to SQL Converter API

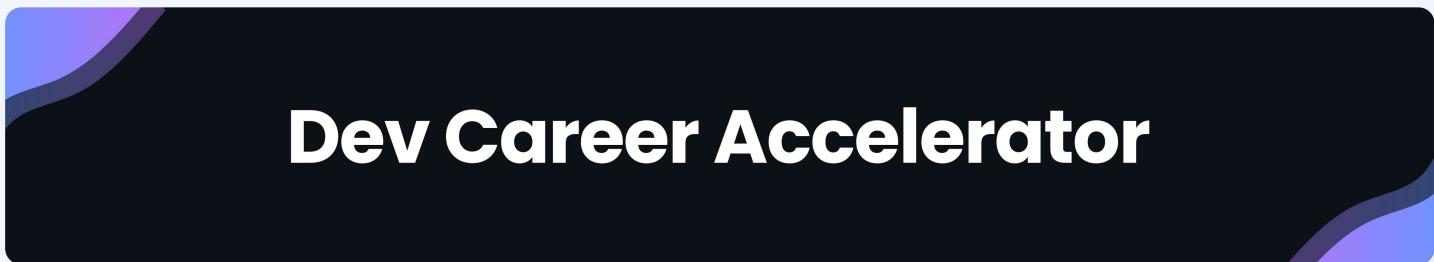
# The End

Congratulations on reaching the end of our guide! But hey, learning doesn't have to stop here.

If you're eager to dive deep into something this specific and build substantial projects, our **special course on Next.js** has got you covered.



If you're craving a more personalized learning experience with the guidance of expert mentors, we have something for you – [Dev Career Accelerator](#).



If this sounds like something you need, then don't stop yourself from leveling up your skills from junior to senior.

Keep the learning momentum going. Cheers! 🚀