

Full Stack AI Software Development

Intro to Back-End Development & Node JS

Job Connector Program

Outline

Backend Development

Learn how backend systems work, how software runs on servers, and how code instructs computers.

Basic API with Node JS

Build simple APIs using JavaScript & TypeScript, and explore Node.js as a backend runtime.

Intro to API

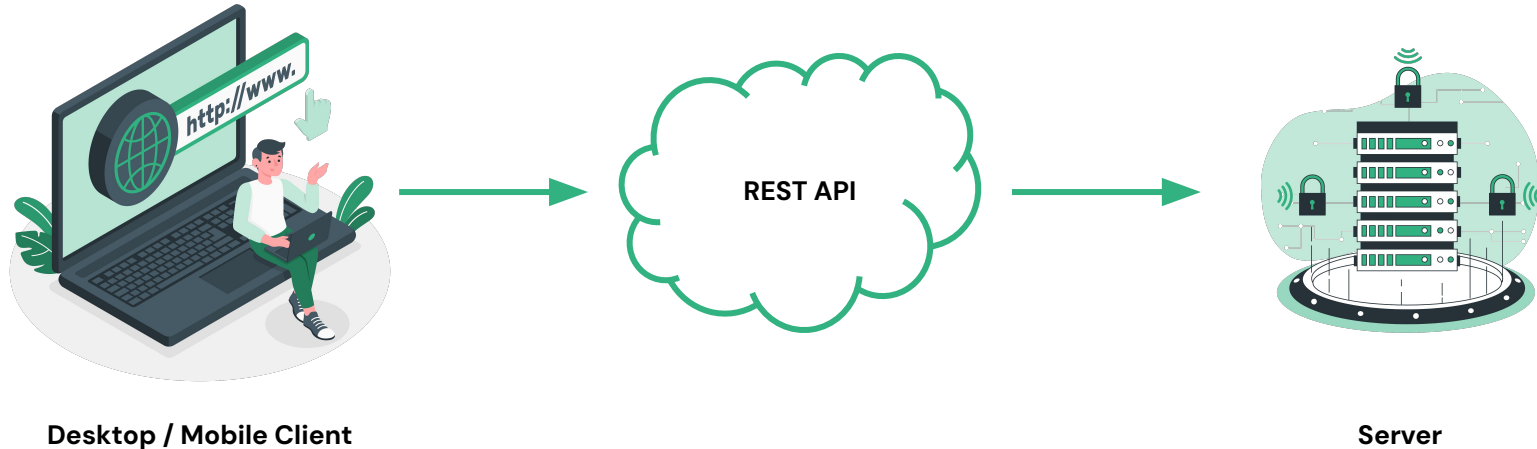
Understand how APIs connect systems and design clear, logical solutions using HTTP methods.

REST API with Express

Create RESTful APIs with Express, using routing, middleware, and CRUD operations.

What Is Back End Development?

Backend development means working on server-side application, which focuses on everything you can't see on a website. Back-end developers ensure the website performs correctly, focusing on databases, back-end logic, application programming interface (APIs), architecture, and servers. They use code that helps browsers communicate with databases, store, understand, and delete data.

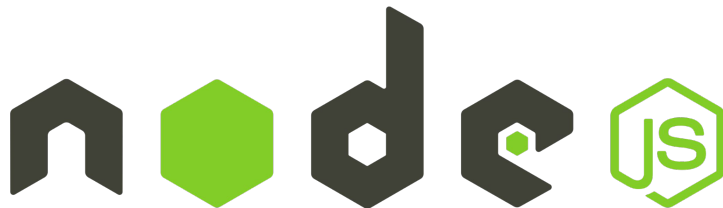


Introduction to NodeJs

Node.js is a JavaScript runtime built on Chrome's V8 JavaScript engine (written in C++). Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient. Node.js' package ecosystem, npm, is the largest ecosystem of open source libraries in the world.

- A platform which allows us to run Javascript on computer/server.
- Read, delete & update files.
- Easily communicate with database.

We use NodeJS as a backend in web development.



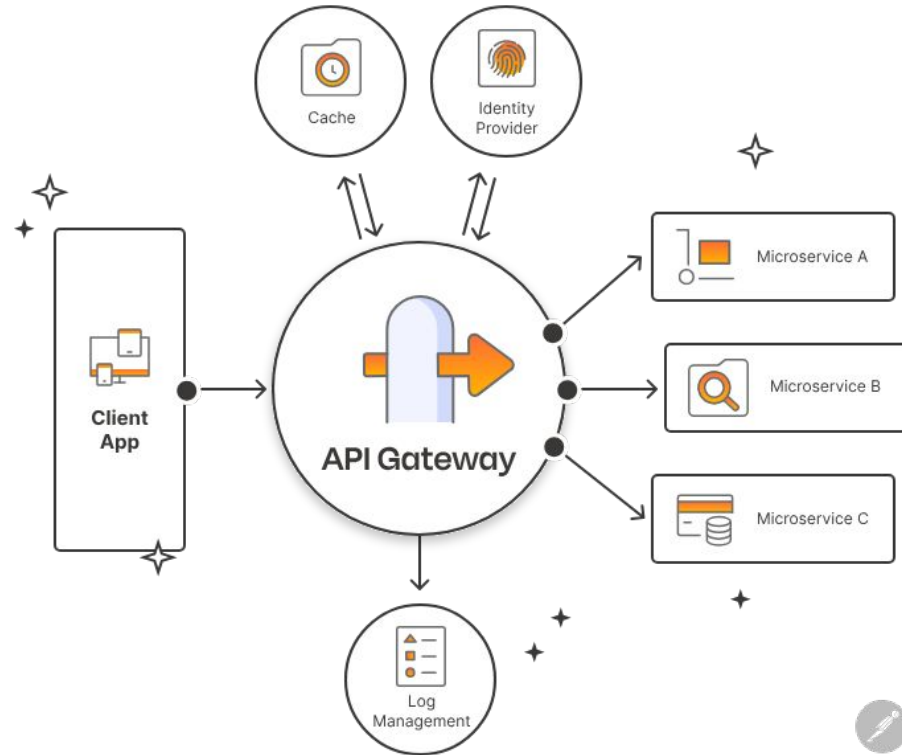
Src: <https://nodejs.org/en/learn/getting-started/introduction-to-nodejs>

What is API ?

API stands for Application Programming Interfaces. APIs are mechanisms that enable two software components to communicate with each other using a set of definitions and protocols.

For example, the weather bureau's software system contains daily weather data. The weather app on your phone "talks" to this system via APIs and shows you daily weather updates on your phone.

You can think of an API as a gateway between clients and resources on the web.



API Architectures

API architecture is usually explained in terms of client and server. The application sending the request is called the client, and the application sending the response is called the server. So in the weather example, the bureau's weather database is the server, and the mobile app is the client.

There are many different ways that APIs can work depending on when and why they were created.

API Architectures

SOAP

These APIs use Simple Object Access Protocol. Client and server exchange messages using XML. This is a less flexible API that was more popular in the past.

RPC

These APIs are called Remote Procedure Calls. The client completes a function (or procedure) on the server, and the server sends the output back to the client.

Websocket

Websocket API is another modern web API development that uses JSON objects to pass data. A WebSocket API supports two-way communication between client apps and the server. The server can send callback messages to connected clients, making it more efficient than REST API.

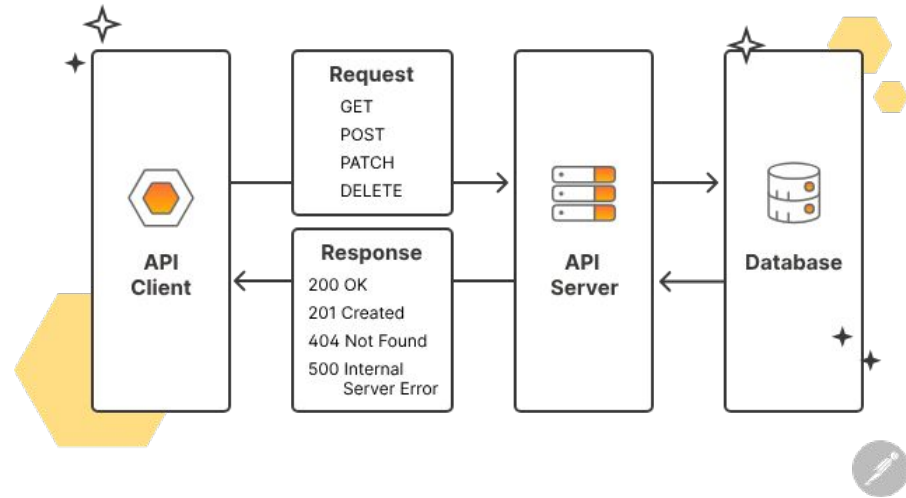
REST

These are the most popular and flexible APIs found on the web today. The client sends requests to the server as data. The server uses this client input to start internal functions and returns output data back to the client.

What is REST API ?

REST stands for Representational State Transfer. REST defines a set of functions like GET, PUT, DELETE, etc. that clients can use to access server data. Clients and servers exchange data using HTTP.

The main feature of REST API is statelessness. Statelessness means that servers do not save client data between requests. Client requests to the server are similar to URLs you type in your browser to visit a website. The response from the server is plain data, without the typical graphical rendering of a web page.



Create Simple REST API with NodeJS

To create our first simple REST API using typescript and NodeJS, let's first run the command like the picture below. The command will do a couple of things :

1. The first line will generate a package.json for us.
2. The third line will generate tsconfig.json for us.



```
npm init --y  
npm i -D typescript @types/node  
tsc --init
```

Create Simple REST API with NodeJS

Inside our tsconfig.json find the option called “outDir”, uncomment the line and change it from “.” to “./dist”. What this do is it will tell typescript where to put the compiled javascript files.

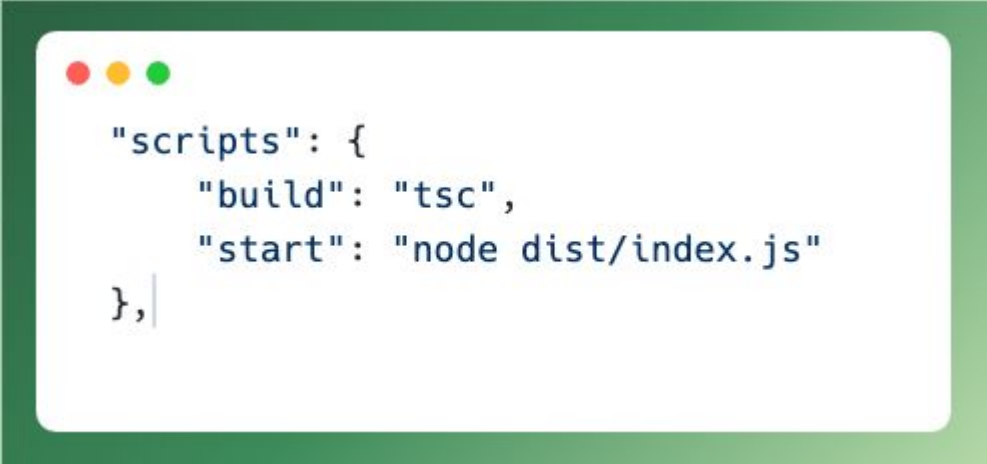
You will also find other option inside **tsconfig.json** but for now let's ignore the others.



Create Simple REST API with NodeJS

Now let's move on to our package.json and add a script like the picture below.

The build script will compile our typescript into a javascript files, it will create a dist folder inside our project where the compiled javascript files will reside, and the start script will first run our build script and run the compiled javascript files inside our dist folder.



```
"scripts": {  
  "build": "tsc",  
  "start": "node dist/index.js"  
},
```

Create Simple REST API with NodeJS

With the setup done, let's create a file called **index.ts**.



```
import http from "http";

const PORT = 8000;
const server = http.createServer(async (req, res) => {
  // define route in next slide
});

server.listen(PORT, () => {
  console.log("application run on port ", PORT)
});
```

Create Simple REST API with NodeJS

```
import http from "http";

const PORT = 8000;
const server = http.createServer(async (req, res) => {
  if(req.url === "/api" && req.method === "GET"){
    // response headers
    res.writeHead(200, { "Content-Type": "application/json" });
    // set the response
    res.write("Hi there, This is a Vanilla Node.JS API");
    // end the response
    res.end();
  }
});

server.listen(PORT, () => {
  console.log("application run on port ", PORT)
});
```

Function / Method	Description
<code>import http from "http"</code>	Import Node.js built-in HTTP module.
<code>http.createServer((req, res) => { ... })</code>	Create an HTTP server to handle requests.
<code>req.url & req.method</code>	Check the request path and method (e.g. GET /api).
<code>res.writeHead(200, { "Content-Type": "application/json" })</code>	Set response status and headers.
<code>res.write(...)</code>	Send response data (e.g. JSON or text).
<code>res.end()</code>	End the response process.
<code>server.listen(PORT, ...)</code>	Start server and listen on a specific port.

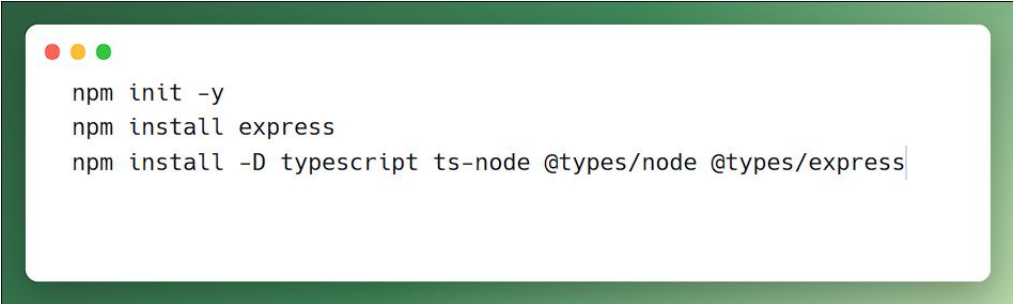
Introduction to ExpressJS

Express is a minimal and flexible Node.js web application framework.

The Express philosophy is to provide small, robust tooling for HTTP servers, making it a great solution for single page applications, websites, hybrids, or public HTTP APIs.

Create Simple REST API with ExpressJS

- First, create a project directory then
- Initialize library:

A terminal window with a dark green border and a light green background. It contains three lines of text: 'npm init -y', 'npm install express', and 'npm install -D typescript ts-node @types/node @types/express'. The text is in a monospaced font, and there is a cursor at the end of the third line.

```
npm init -y  
npm install express  
npm install -D typescript ts-node @types/node @types/express
```

Create Simple REST API with ExpressJS

```
import express, { Request, Response } from "express";

const PORT: number = 8000;

const app = express();

app.get("/", (req: Request, res: Response) => {
  res.status(200).send({ message: "hello world" });
});

app.listen(PORT, () => {
  console.log(`Application running on port ${PORT}`);
});
```

Function / Method	Description
<code>import express from "express"</code>	Import Express framework (simplifies Node.js HTTP server).
<code>const app = express()</code>	Create an Express application instance.
<code>app.get("/", (req, res) => { ... })</code>	Define a GET route for path /. Handles incoming requests.
<code>res.status(200).send({ message: "hello world" })</code>	Send a JSON response with HTTP status 200 (OK).
<code>app.listen(PORT, ...)</code>	Start the server and listen on the defined port.

Testing with client app

1. Download and open POSTMAN <https://www.postman.com/product/rest-client/>
2. Write down your web server port (ex: localhost:8000/api) in url field.
3. Hit enter to see the response.

Network Protocols

A protocol is a set of rules. The network protocols are formal standards and policies, made up of restrictions, procedures, and formats, that define the exchange of data packets to achieve communication between two servers or more devices over a network.

Types of protocols :

- Transmission Control Protocol (TCP)
- Internet Protocol (IP)
- User Datagram Protocol (UDP)
- Post office Protocol (POP)
- Simple mail transport Protocol (SMTP)
- File Transfer Protocol (FTP)
- Hyper Text Transfer Protocol (HTTP)
- Hyper Text Transfer Protocol Secure (HTTPS)
- Etc ...

HTTP Request Methods

HTTP defines a set of request methods to indicate the desired action to be performed for a given resource. Although they can also be nouns, these request methods are sometimes referred to as HTTP verbs.

GET

- The GET method requests a representation of the specified resource. Requests using GET should only retrieve data.

POST

- The POST method submits an entity to the specified resource, often causing a change in state or side effects on the server.

PUT

- The PUT method replaces all current representations of the target resource with the request payload.

DELETE

- The DELETE method deletes the specified resource.

PATCH

- The PATCH method applies partial modifications to a resource.

Exercise

Create REST API using ExpressJS with all HTTP request methods.

Goal:

Build a simple REST API using Express.js, store data in a local todos.json file.

Notes:

- Use fs module for file read/write
- Send responses in JSON
- Use proper status codes (200, 201, 404, etc.)
- Handle invalid ID or file errors

Method	Endpoint	Description
GET	/todos	Get all todos
GET	/todos/:id	Get todo by ID
POST	/todos	Add new todo
PUT	/todos/:id	Update todo
DELETE	/todos/:id	Delete todo

Thank you

