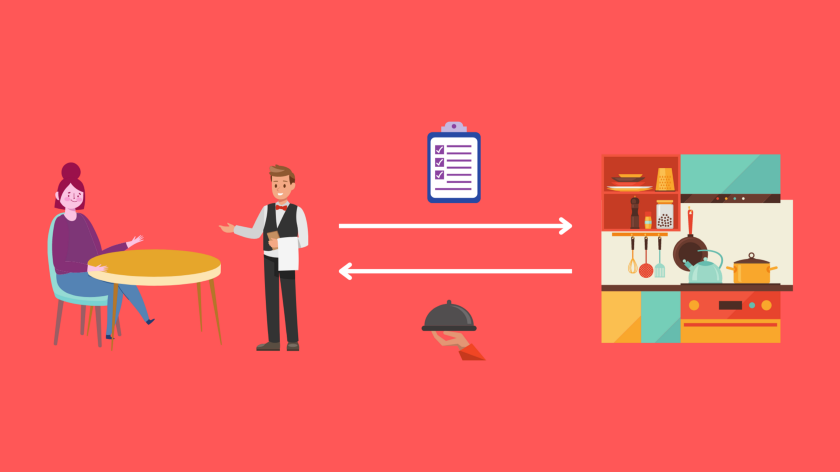
If you are in tech, then you may have heard this popular term called “API”. Some people use APIs for fun, some for money and some for their applications. There are N ways you can use APIs. In this blog, let’s learn what exactly is an API, how you can build your own API and how you can monetize your API.

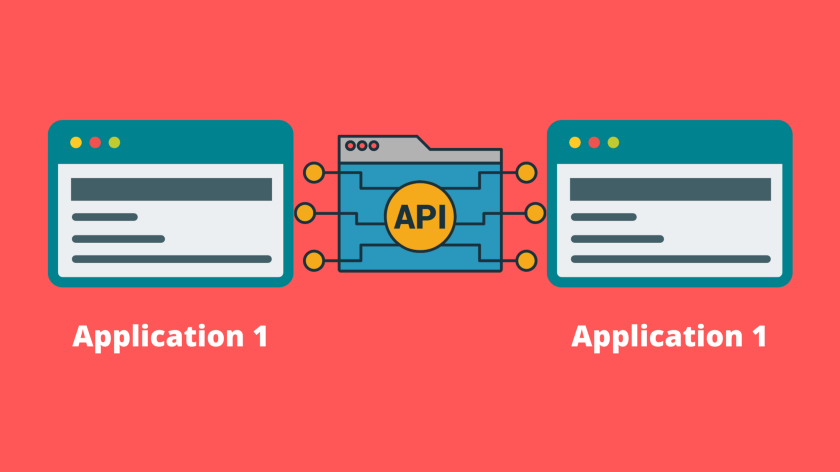
**Let’s get started**

**What is an API?**

I am taking a popular example to explain this. Imagine you go to a restaurant to eat some food. Now you don’t directly go to the kitchen and cook yourself and then eat it, right 😂? (Of course, they don’t allow you to do so). You call a waiter and order your food. Then the waiter goes to the kitchen and brings your food.

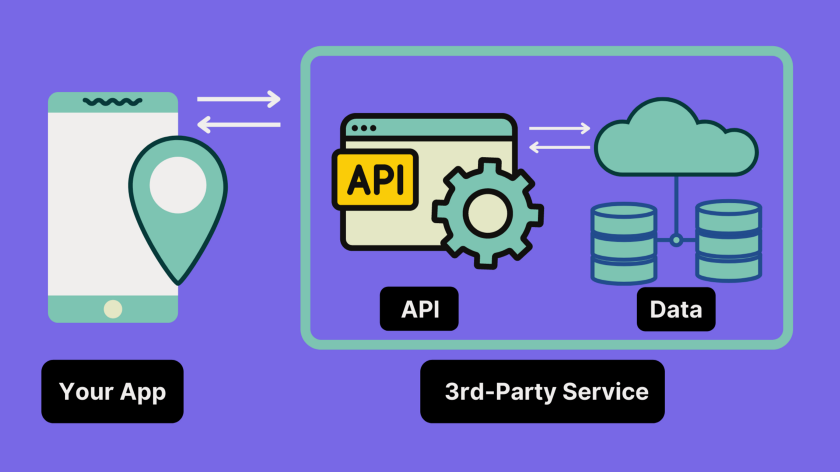


Here you can compare API with the waiter. So, API is an intermediary between two applications and makes it possible for those two applications to communicate with each other. If we put this in our example, one application is you the customer, another application is the restaurant kitchen where the food is prepared and the waiter is an API who acts as an intermediary between you and the kitchen.



**Why do we need APIs?**

Imagine you have data and you want to share that data to allow developers to build software with your data. Now you need some sort of way you can make this possible. That’s where APIs can help you. You can build an API to share your data and other resources so that developers can use your API to build services or software. Let’s understand this with an example, Let’s say you are building an app that suggests the vehicle take the route with less traffic. For this, you need traffic data of different routes so that you can train a machine learning model and build your app. It’s not an easy task to count the number of vehicles travelling on different routes and prepare data. So what you can do is, use a 3rd party service that provides their data with APIs.



**How to build an API?**

Another thing you need to know about API is not just about data, it can be a set of functions, objects, and commands. For example, [browser API](https://developer.mozilla.org/en-US/docs/Web/API) provides various functions, objects etc to use in your applications to interact with the browser.

Before building our own API let’s use an API. We will be using a [JokeAPI](https://sv443.net/jokeapi/v2/).

Before that let’s learn some terms of API:

**Endpoint** — An endpoint is an API server URL where you can access all the different resources that API provides. Endpoints are actions like GET, POST, DELETE, etc.., which you can perform on different routes. For example,

* GET https://api.github.com/ - is an API endpoint
* POST https://api.github.com/user - is another endpoint
* and so on…

**Paths** — Paths are different URLs of an API.

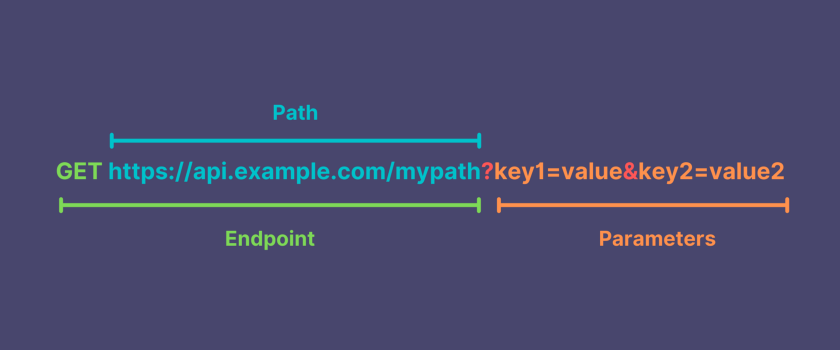
For example:

* https://api.github.com/user - is a path/route

**Parameter** — All the paths are pre-defined in the API server. If you have a path that can’t be pre-defined in the server, then you can use parameters. Parameters are key-value pairs and start after ? from the end of a path.

For example,

* https://api.github.com/user?userId=12335235235 - here userId is a parameter. If you have more than one parameter, then you can append them by adding & after each parameter.
* For example,
* <https://api.github.com/user?userId=12335235235&api_key=yu67432ffu6f2t446>

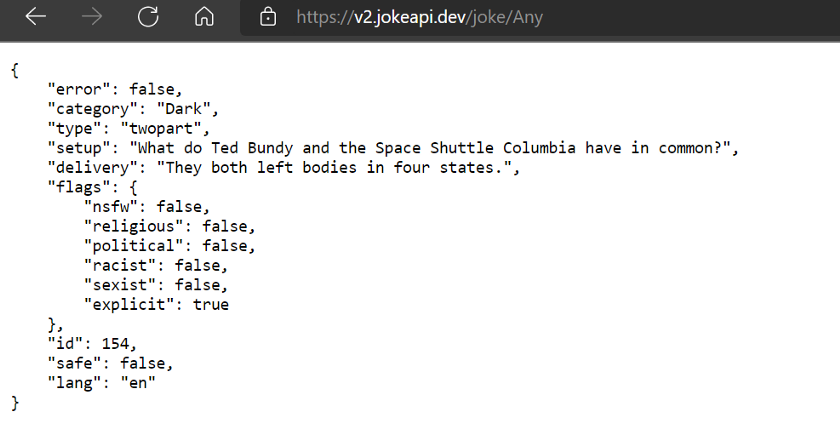


**Let’s use an API**

* Open a new browser tab, paste this URL and see,

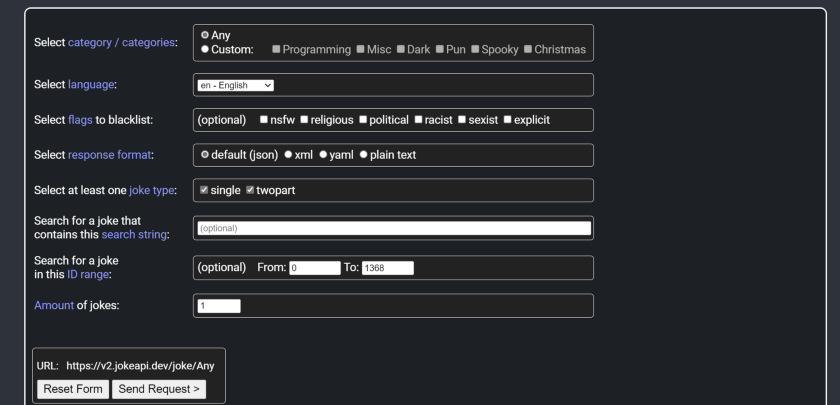
<https://v2.jokeapi.dev/joke/Any>

You will receive something like this,



This is called a “response” you got from JokeAPI for your request. And the format of the response is “JSON”. [JSON](https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Objects/JSON) is a popular output format for APIs.

* If you visit JokeAPI [documentation](https://sv443.net/jokeapi/v2/#try-it), you can try out different categories and filters.



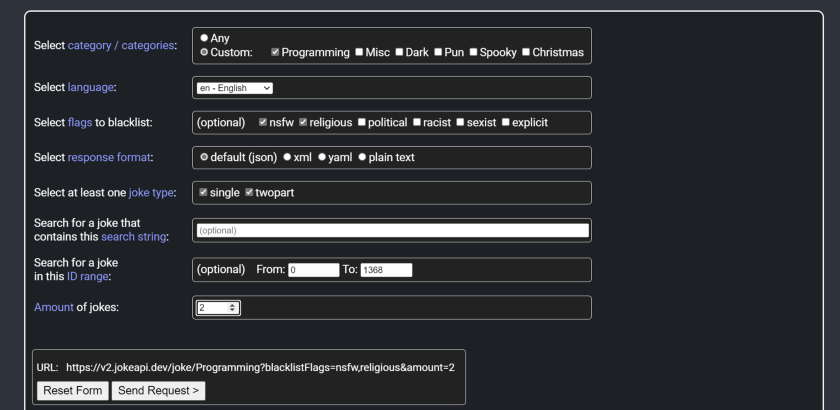
In the above options, each category is a different route/path, like

* <https://v2.jokeapi.dev/joke/Programming>
* <https://v2.jokeapi.dev/joke/Miscellaneous>
* <https://v2.jokeapi.dev/joke/Dark>

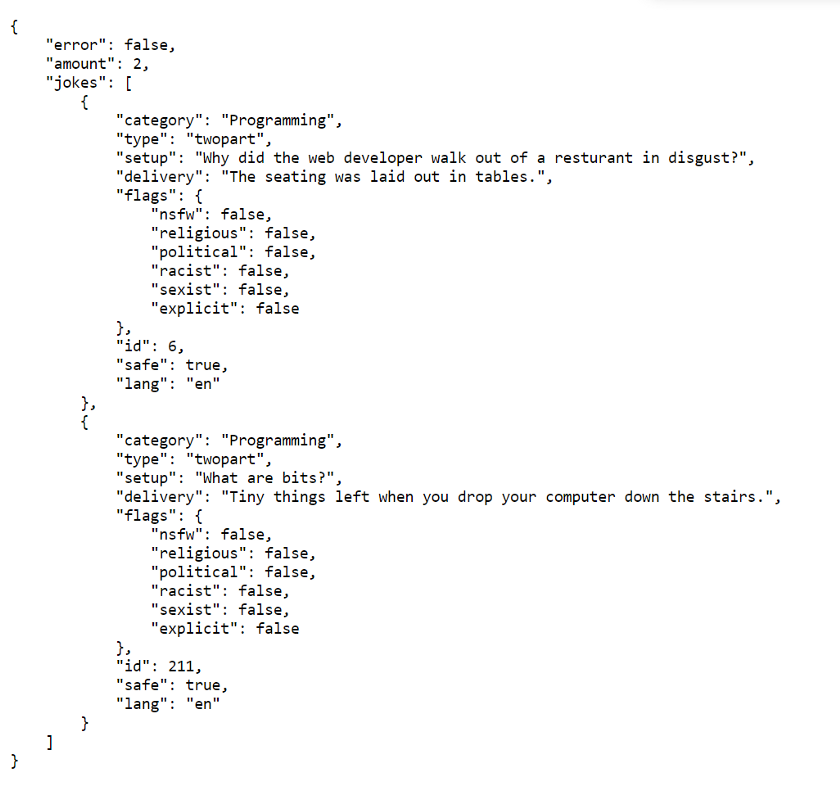
and all the options below category can be appended as parameters, like

* <https://v2.jokeapi.dev/joke/Programming?blacklistFlags=nsfw&type=twopart&amount=2>

Let’s try to tweak the options,



After tweaking the options, copy the URL and paste it into the browser,



Now you will get a response with all the filters applied.

**Let’s build our own API**

You can build two types of APIs:

1. Software — As mentioned somewhere above a software API is just a set of functions, objects and commands, it doesn’t require a database. For example, [jQuery API](https://api.jquery.com/), [Browser API](https://developer.mozilla.org/en-US/docs/Web/API), etc.
2. API service — An API service gives people access to their data through API. For example, [JokeAPi](https://sv443.net/jokeapi/v2), [The Movie Database](https://developers.themoviedb.org/3), [Open Weather API](https://openweathermap.org/api), etc.

Let’s build an API service to add, delete, edit and get your daily tasks. We need a database and a server to create an API service. Let’s use MongoDB as our database and NodeJs, and ExpressJs for creating a server.

* Open your IDE or code editor. Create a folder and name it something like todo-api.
* Before we start make sure you have these dev tools installed,
* NodeJs
* MongoDB
* Initialize npm with,

npm init

* Install express, mongoose, and axios packages as we use them for the project.

npm i express mongoose axios

* Install nodemon as a dev dependency. (Nodemon restarts the server every time we make changes to the code so that we don't need to manually restart)

npm i nodemon --save-dev

* Add a script to start the server with nodemon.

[*todo-api/package.json*](https://github.com/itsrakeshhq/todo-api/blob/6a37ad9739af4bcf8c6738aa247be01247181c88/package.json#L9)

"scripts": {  
 ...  
 "dev": "nodemon server.js"  
 ...  
},

* Next, create a file called server.js in the root and paste this boilerplate code.

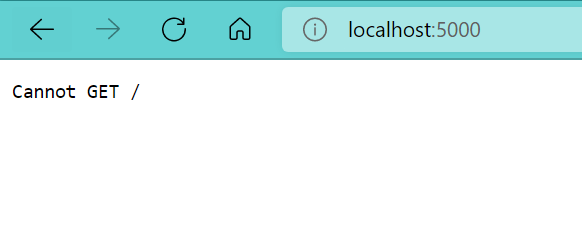
[*todo-api/server.js*](https://github.com/itsrakeshhq/todo-api/blob/6a37ad9739af4bcf8c6738aa247be01247181c88/server.js)*(Snippet)*

const express = require("express");  
const mongoose = require("mongoose");  
  
const app = express();  
const PORT = process.env.PORT || 5000;  
const MONGODB\_URI = process.env.MONGODB\_URI || "mongodb://localhost/todoapiDB";  
  
app.use(express.json());  
  
mongoose  
 .connect(MONGODB\_URI, { useNewUrlParser: true })  
 .then(() => {  
 app.listen(PORT, console.log("Server stated on port 5000"));  
 })  
 .catch((err) => {  
 console.log(err);  
 });

* Now, start the server with this command,

npm run dev

* Visit http://localhost:5000/ in your browser and see the response.



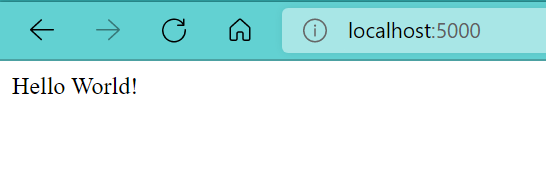
You should see this in your browser. What it is telling you is there is no endpoint like GET http://localhost:5000/ defined in the server.

* So let’s add an endpoint. As we are using expressjs we can register a route like this.

[*todo-api/server.js*](https://github.com/itsrakeshhq/todo-api/blob/6a37ad9739af4bcf8c6738aa247be01247181c88/server.js#L10-L12)*(Snippet)*

app.get("/", (req, res) => {  
 res.send("Hello World!");  
});

* Now visit the URL again and you will see the response.



So this is a simple “GET” request we created on our server.

* Next, create a simple model in our database to store our tasks.

[*todo-api/models/tasks.model.js*](https://github.com/itsrakeshhq/todo-api/blob/b007217eafc0c54e159fc2ef6064a79e6b230e8f/models/tasks.model.js)*(Snippet)*

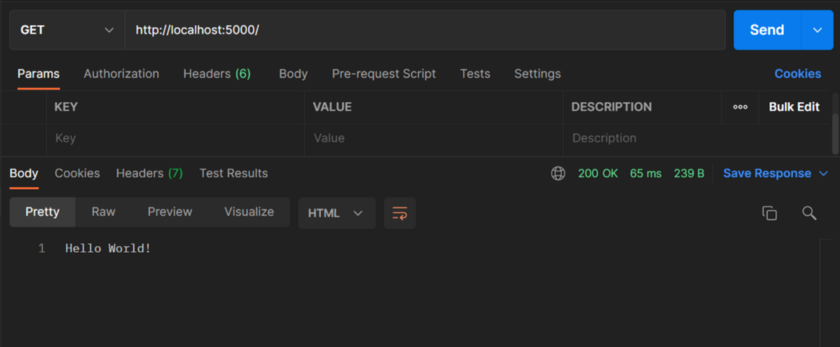
const mongoose = require("mongoose");  
  
const Schema = mongoose.Schema;  
  
const taskSchema = new Schema({  
 name: {  
 type: String,  
 required: true,  
 },  
});  
  
module.exports = mongoose.model("Task", taskSchema);

and require the model in server.js

[*todo-api/server.js*](https://github.com/itsrakeshhq/todo-api/blob/b007217eafc0c54e159fc2ef6064a79e6b230e8f/server.js#L4)*(Snippet)*

const Task = require("./models/tasks.model");

Before we move further, it’s not possible to do everything from the browser so let’s use an API tool called “Postman”. Download it from [here](https://www.postman.com/downloads) (Free). After downloading, test it by entering the URL http://localhost:5000/ and clicking **Send**.



* Now define a route that gets all the tasks.

[*todo-api/server.js*](https://github.com/itsrakeshhq/todo-api/blob/7915108a543b5c2051419411c1addd1279a9b0bd/server.js#L17-L24)*(Snippet)*

// GET http://localhost:5000/getTasks  
  
app.get("/getTasks", async (req, res) => {  
 try {  
 const response = await Task.find();  
 res.json(response);  
 } catch (err) {  
 res.json({ message: err });  
 }  
});

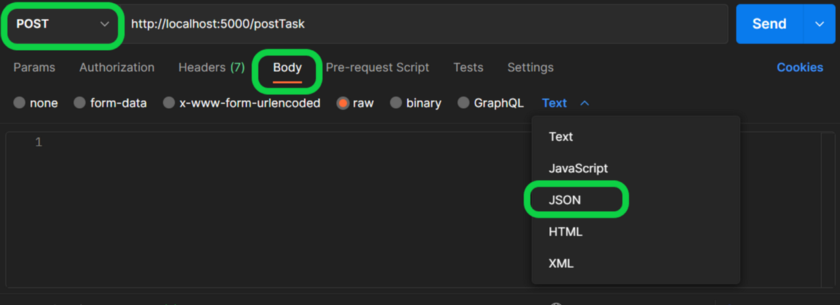
If you test this now you will get an empty response as we have not added any tasks to our database.

* So let’s create a route to add tasks to our database. To send data in our request we need to make a “POST” request.

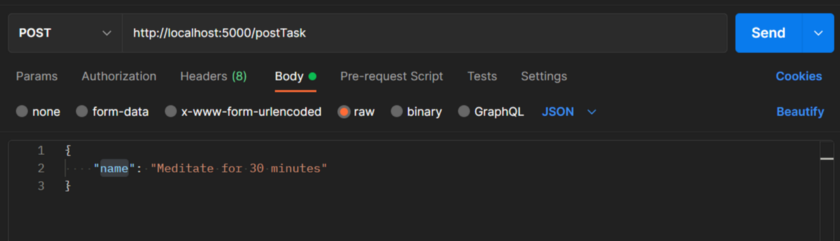
[*todo-api/server.js*](https://github.com/itsrakeshhq/todo-api/blob/7915108a543b5c2051419411c1addd1279a9b0bd/server.js#L27-L34)*(Snippet)*

// POST http://localhost:5000/postTask  
  
app.post("/postTask", async (req, res) => {  
 try {  
 const response = await Task.create(req.body);  
 res.json(response);  
 } catch (err) {  
 res.json({ message: err });  
 }  
});

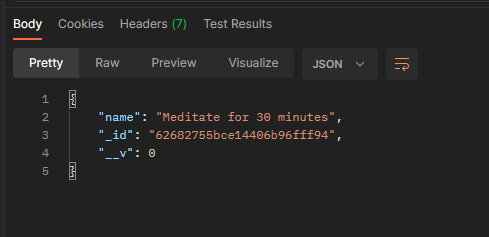
Now in postman change the GET request to POST. Then go to the **Body** tab and select **raw** -> **JSON** from dropdown.



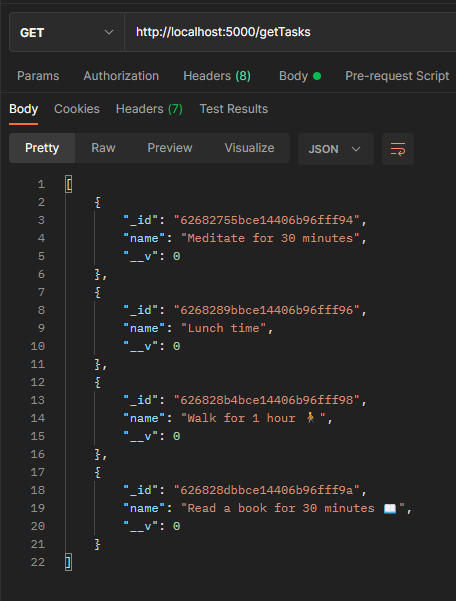
Write a JSON object in the body field and make a POST request to <http://localhost:5000/postTask.>



You will receive a response back containing name- the name of the task, \_id - the unique id of the task generated by MongoDB.



* Add a few more tasks and make a GET request to http://localhost:5000/, you will see all your tasks.



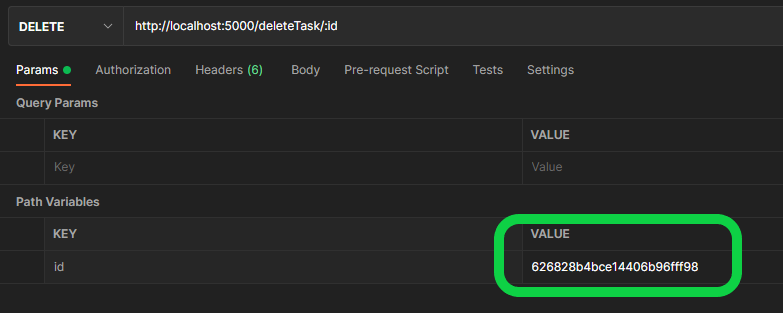
* Now let’s add a route to delete a task.

[*todo-api/server.js*](https://github.com/itsrakeshhq/todo-api/blob/7915108a543b5c2051419411c1addd1279a9b0bd/server.js#L37-L44)*(Snippet)*

// DELETE http://localhost:5000/deleteTask/:id  
  
app.delete("/deleteTask/:id", async (req, res) => {  
 try {  
 const response = await Task.remove({ \_id: req.params.id });  
 res.json(response);  
 } catch (err) {  
 res.json({ message: err });  
 }  
});

In the above route http://localhost:5000/deleteTask/:id :id is called a **Path Variable**. It is used when we can't pre-define a route. You can also use **Query Parameter** for our case.

So, change the request method to DELETE in postman and copy one of your tasks id and paste in the path variable value and click **Send**.



If you now make a GET request to /getTasks you won't see the deleted task. That means you successfully deleted the Task.

* Now let’s edit a task. We all make mistakes so we need an edit button(I Hope Elon Musk adds an edit button to Twitter). To edit data we have to make a PATCH request. Let's create a route for that. You can make use PUT request to edit a document. But PATCH request is better if we want to edit partial data.

[*todo-api/server.js*](https://github.com/itsrakeshhq/todo-api/blob/7915108a543b5c2051419411c1addd1279a9b0bd/server.js#L47-L57)*(Snippet)*

// PATCH http://localhost:5000/editTask/:id  
  
app.patch("/editTask/:id", async (req, res) => {  
 try {  
 const response = await Task.updateOne({ \_id: req.params.id }, { $set: req.body });  
 res.json(response);  
 } catch (err) {  
 res.json({ message: err });  
 }  
});

Same as the POST request, add body to your PATCH request. Copy the id of the task that you wish to edit and paste it into the path variable value field and click **Send**. Now make a GET request to /getTasks you will see Task updated.

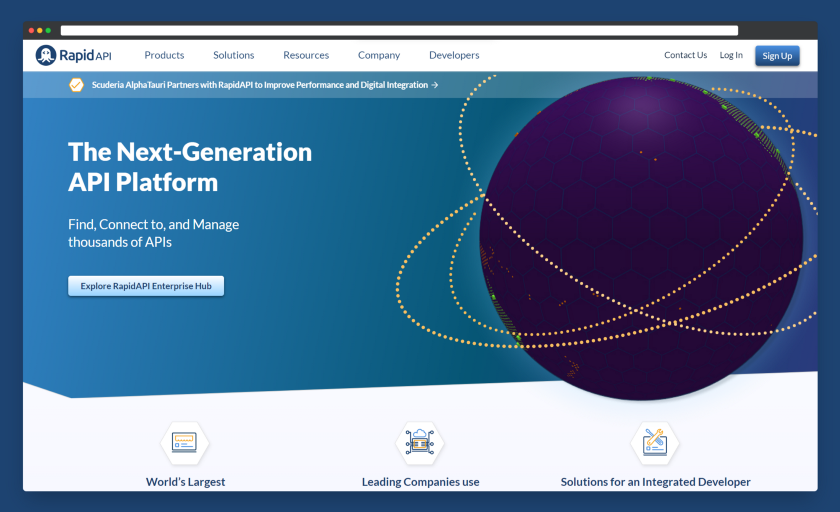
So that’s it! We learned 4 important RESTAPI methods while building our small “todo application”.

* Here is the postman collection containing the four requests — [Link](https://www.getpostman.com/collections/9daee5e447906e58744b)
* Here is the GitHub repository for this tutorial — [Link](https://github.com/itsrakeshhq/todo-api)

**How to sell/monetize an API?**

“Data is the new oil” — a popular quote of the 21st century and it is 100% true. If you have data, then you can make loads of $$$. API is one great way to sell/monetize your data. Let’s see how we can monetize our API.

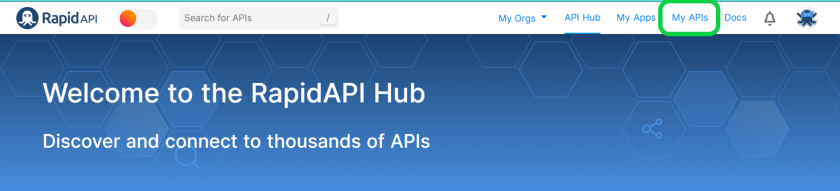
To monetize our API, we are going to use [RapidAPI](https://rapidapi.com/)



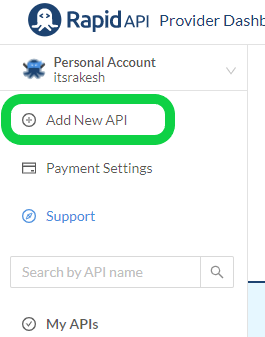
**Rapid API** is the world’s largest API hub where you can explore different APIs, and create and manage your own APIs.

Before continuing, host your API server somewhere like **Heroku** because you know “localhost” doesn’t work outside of your computer :). And replace all http://localhost:5000/ with https://yourdomain.com/ in your postman collection.

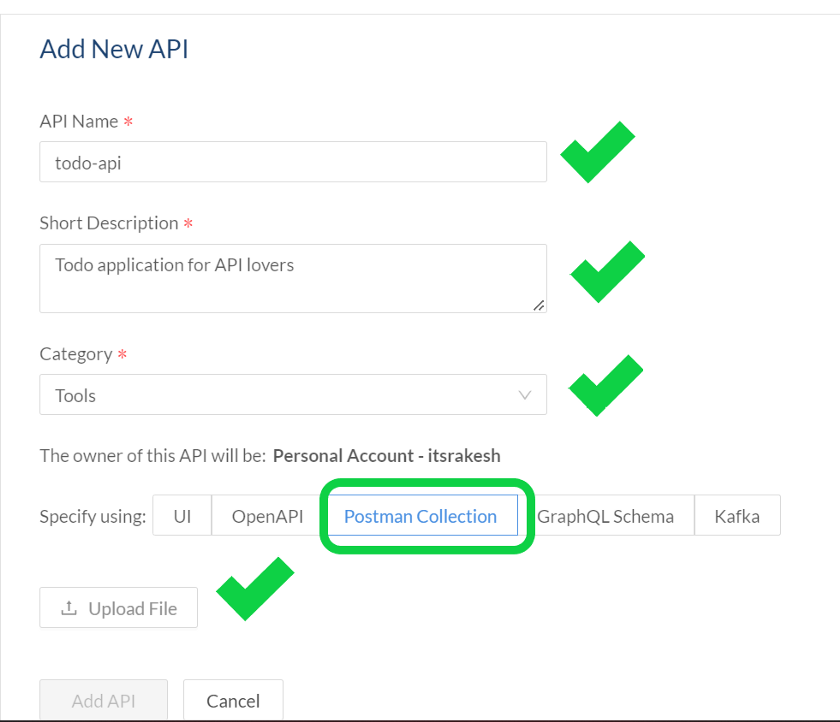
* Let’s start by creating an account if you don’t have one already.
* After creating an account, click on **My APIs** on the top-right.



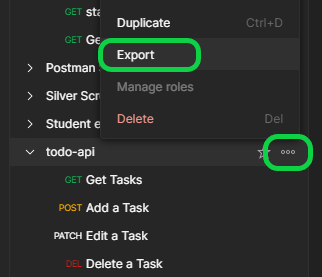
* Click on **Add New API** on the left panel.



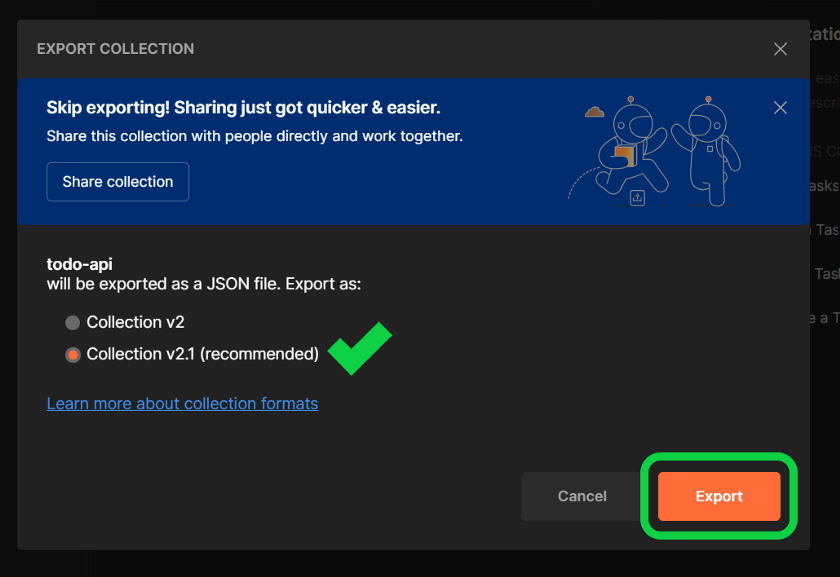
* Fill in the details for **API Name**, **Short Description** and **Category**. For **Specify using**, select “Postman Collection”. And then upload the collection file.



You can download your postman collection by exporting the collection as a JSON file. To do so, open your postman collection, and click three dots -> **Export**.

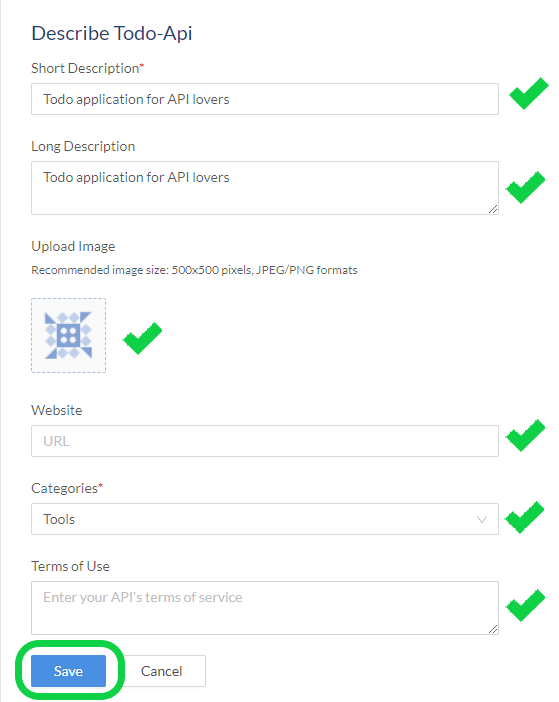


Export

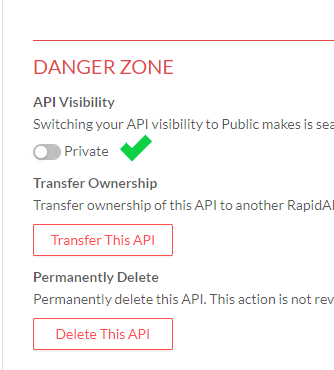


Or you can download the JSON file from this [tutorial GitHub repository](https://github.com/itsrakeshhq/todo-api/blob/main/todo-api.postman_collection.json). Make sure to change the domain name.

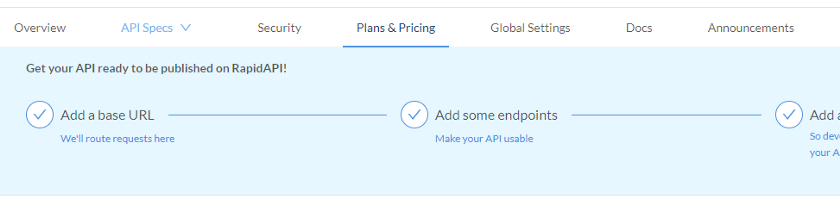
* After uploading the file and clicking **Add API**. Fill in the details for “Describe and click on **Save**.



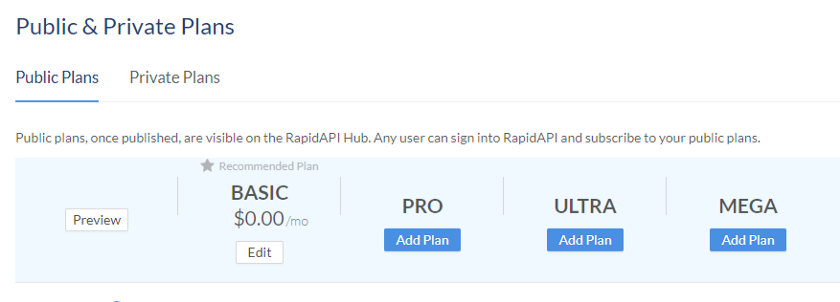
* Next, add a base URL.
* Finally, make your API public, so every on the internet can see your API. If they like it they can subscribe to your API.



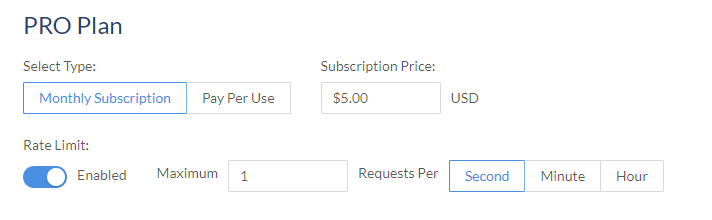
* Let’s actually monetize our API by adding Plans & Pricing. So go to the **Plans & Pricing** tab.



* Here you can choose different plans and set the number of requests for different plans.



* Let’s add a **PRO** plan. Choose “Montly Subscription” or “Pay Per Use”. Set a price. Choose rate limit — number of requests per second/minute/hour.



* Explore more on Rapid API [docs](https://docs.rapidapi.com/).

That’s it! I hope you learned something new from this article. Feel free to ask any questions or doubts or anything in the comments.

Follow me for more stuff like this. Also if you want to receive my articles directly to your Inbox, subscribe to my Medium newsletter. Thank you.