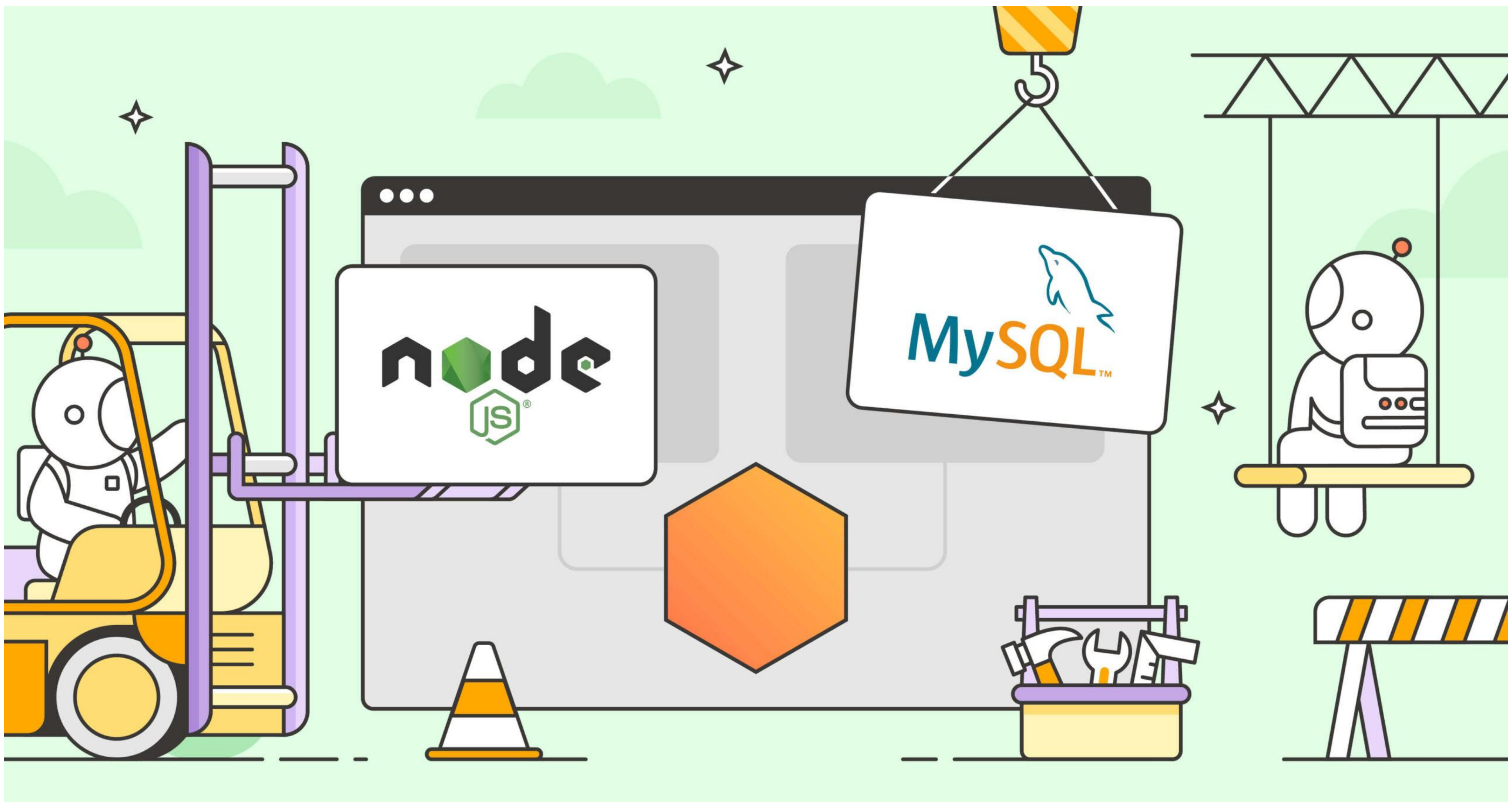


Build an API with Postman, Node.js, and MySQL

Guest Author
June 6, 2023 · 7 mins

This guest post was written by Greg Bulmash, author of Hell on $5 A Day, blogger at LetMyPeopleCode, and content creator for some of the biggest names in tech.

If you're looking to build an API that's highly performant, scalable, and flexible, using technologies that have strong developer communities, Node.js and MySQL make a great pair. Node.js's asynchronous, event-driven architecture can handle a large number of concurrent requests, so scaling your API is no problem. MySQL is highly scalable as well, and its large community of users and extensive documentation make it a strong choice for a relational database.

In this Postman Quickstarts tutorial, we'll walk you through creating a ToDo API with an OpenAPI 3 definition using Postman's API Builder. You'll learn how to generate code for your API, add business logic, and finally, generate a collection from the API definition so that you can send requests to the API.

Related: How to create a REST API with Node.js and Express

Before you begin

For this tutorial, you should have some familiarity with Postman Collections, and you should have a workspace ready to use. As long as you have at least a beginner's understanding of Node.js and MySQL, you're ready to get started.

Make sure you have the following:

- Node.js installed and in your path

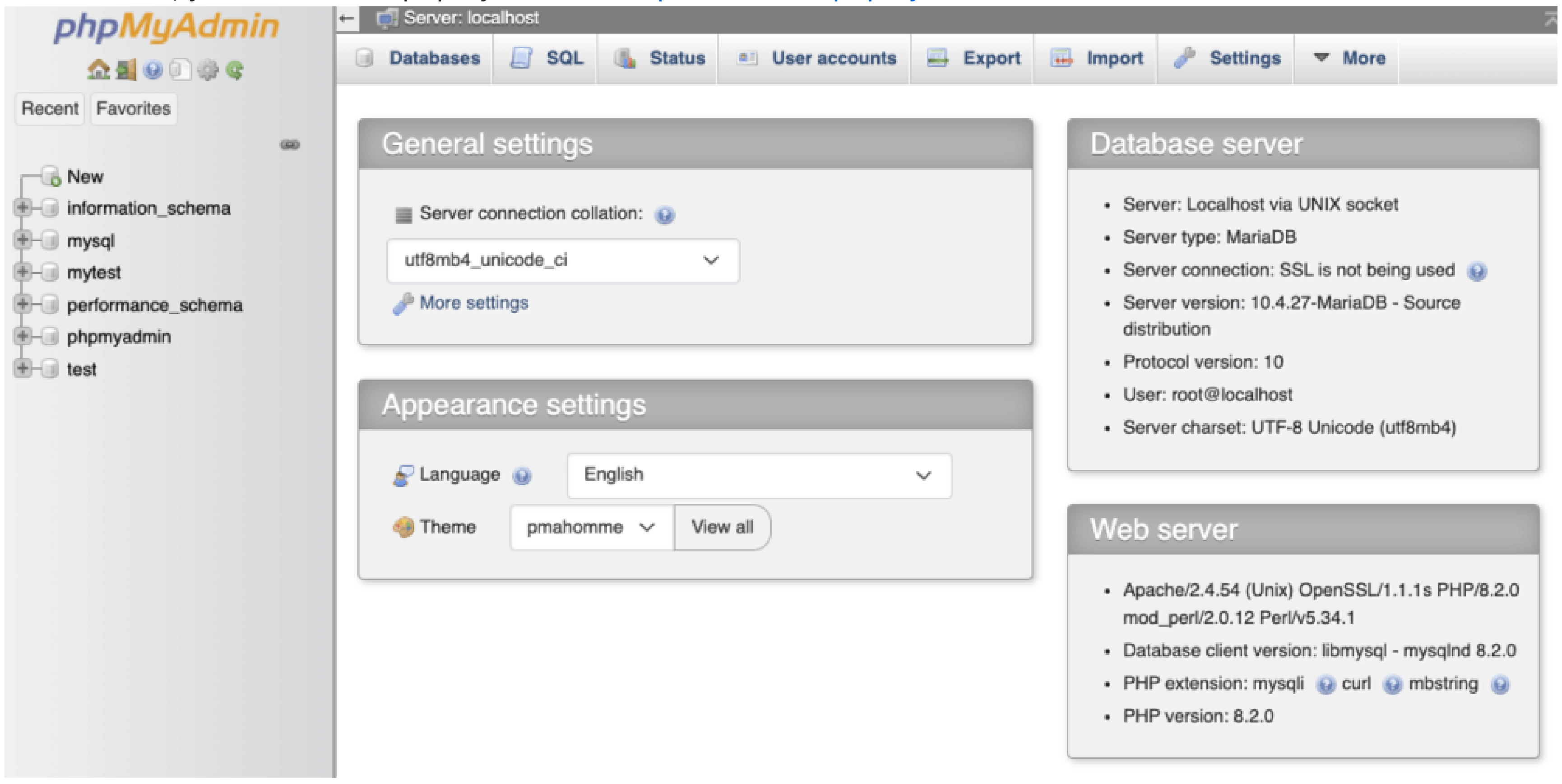
- XAMPP installed (or another tool you prefer for installing and administering MySQL)
- Visual Studio Code (recommended)

If you need more information about installing any of the items above, see the Install the Prerequisites step for this Postman Quickstart.

Step 1: Create the database

First, let's create the MySQL database in XAMPP:

1. Open XAMPP's Control Panel app.
2. Select the **Manage Servers** tab.
3. Select the **MySQL Database**, followed by the **Start** button.
4. Select the **Apache Web Server**, followed by the **Start** button. Starting the web server allows you to access phpMyAdmin, which is a browser-based GUI tool for managing your MySQL server. When both Apache and MySQL have started, you can access phpMyAdmin at http://localhost/phpmyadmin:



Server connection collation: utf8mb4_unicode_ci

Language: English

Theme: pmahomme

- Server: Localhost via UNIX socket
- Server type: MariaDB
- Server connection: SSL is not being used
- Server version: 10.4.27-MariaDB - Source distribution
- Protocol version: 10
- User: root@localhost
- Server charset: UTF-8 Unicode (utf8mb4)

- Apache/2.4.54 (Unix) OpenSSL/1.1.1s PHP/8.2.0 mod_perl/2.0.12 Perl/v5.34.1
- Database client version: libmysql - mysqlnd 8.2.0
- PHP extension: mysqli curl mbstring
- PHP version: 8.2.0

5. In phyMyAdmin, select the **SQL** tab and add the following SQL to the text box:

```
CREATE DATABASE todo;
CREATE TABLE todo.todos (
        id_code varchar(36) NOT NULL UNIQUE,
        to_do varchar(255) NOT NULL,
        completed boolean
);
CREATE USER 'todo_admin'@'localhost' IDENTIFIED BY 'leelu_dallas_multipass-6';
GRANT SELECT, INSERT, UPDATE ON todo.* TO 'todo_admin'@'localhost';
use todo;
INSERT INTO todos (id_code, to_do, completed) VALUES ('todo1','Get something done', TRUE);
INSERT INTO todos (id_code, to_do, completed) VALUES ('todo2','Get another thing done', FALSE);
```

6. Select **Go** to run the query. The query creates a database and adds a table to hold your records, and then it creates a user with privileges to access and update the table. It also creates some records so that we have something to query later on.

Don't panic if you see a few `Error: #1046 No database selected` warnings while the query runs—this is expected. When the query's done, you can turn off the Apache Web Server in the XAMPP app. Only the MySQL server is needed from here.

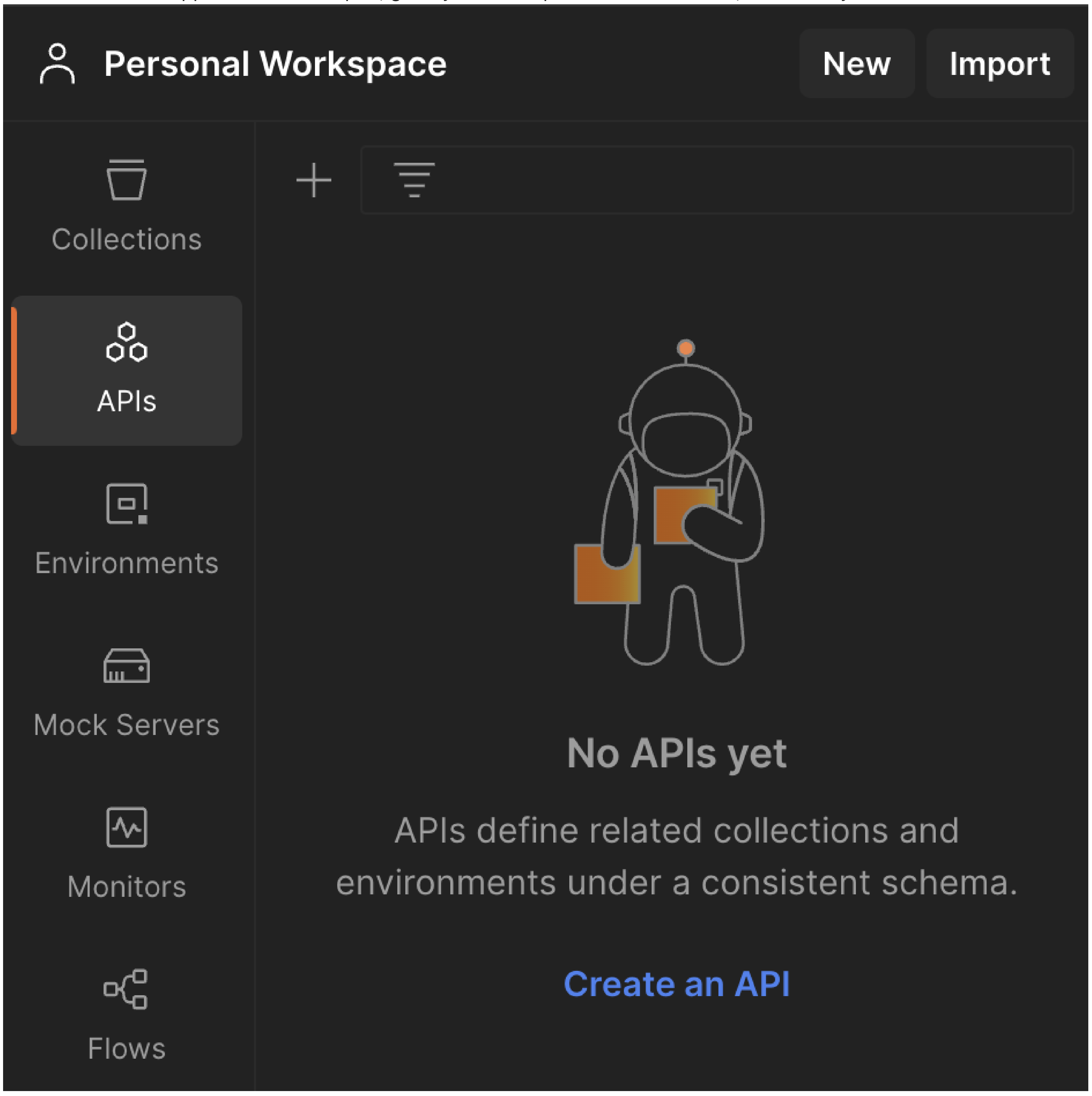
Step 2: Define the API

We'll use OpenAPI 3 to define the ToDo API. You can download the full API definition for this tutorial from GitHub. If you want more details about this definition, see Define the API in the Postman Quickstart for a closer look at each section.

Step 3: Add the API to Postman

Let's add the API to Postman:

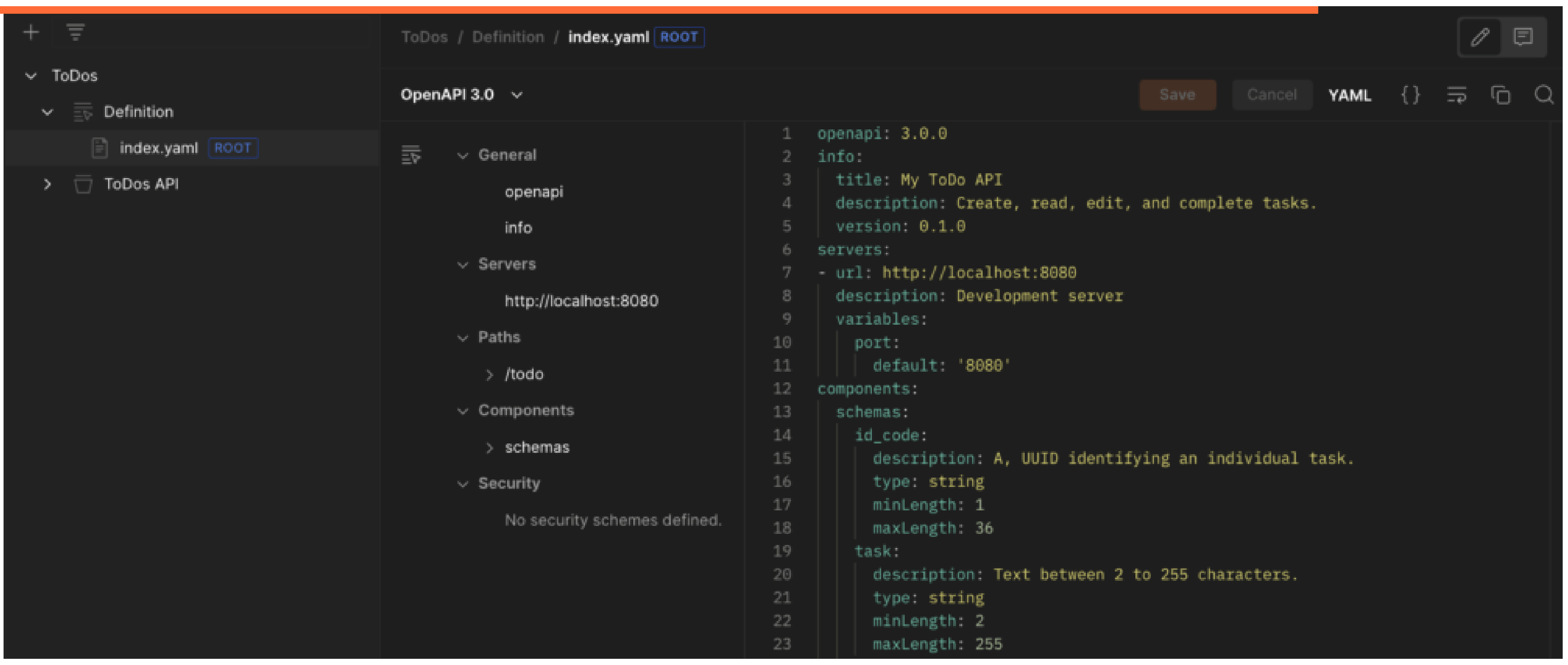
1. With the Postman app or web client open, go to your workspace and select **APIs**, followed by **Create an API:**



No APIs yet

APIs define related collections and environments under a consistent schema.

2. Select the pencil icon that appears when you hover over the API's name, and name it "ToDos".
3. Select **+** to the right of the **Definition** option, then select **Author from scratch**.
4. Select or accept **OpenAPI 3.0** as the **Definition type** and **YAML** as the **Definition Format**.
5. Select **Create Definition**. This opens a code editor.
6. Copy the full ToDo API definition from GitHub and paste it into the code editor:



```yaml
openapi: 3.0.0
info:
  title: My ToDo API
  description: Create, read, edit, and complete tasks.
  version: 0.1.0
servers:
- url: http://localhost:8080
  description: Development server
  variables:
    port:
      default: '8080'
components:
  schemas:
    id_code:
      description: A, UUID identifying an individual task.
      type: string
      minLength: 1
      maxLength: 36
    task:
      description: Text between 2 to 255 characters.
      type: string
      minLength: 2
      maxLength: 255
```

7. Select **Save**.

Step 4: Code the API

You might already know that Postman can generate code to scaffold your API for multiple programming languages. If you want to code the API yourself, see Code the API in the Postman Quickstart. Otherwise, follow these steps to have Postman generate Node.js code for you:

1. Select the top level of the ToDos API.
2. Select **</>** in the context bar.
3. Under **Language and framework**, select **NodeJs – Express**. Leave **Only generate routes and interfaces** unchecked.
4. Select **Generate Code** and download the ZIP file.

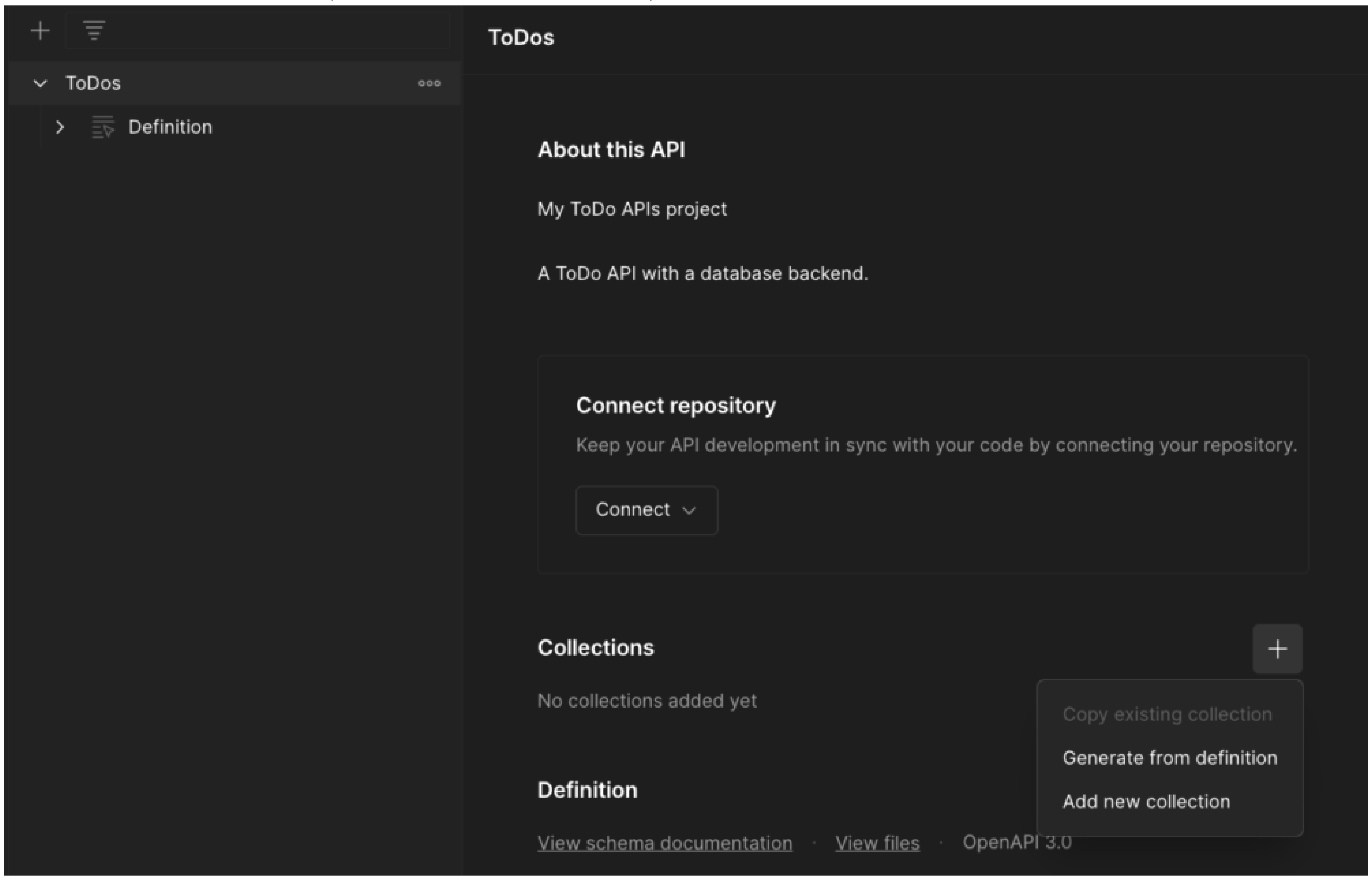
Step 5: Run requests in Postman

In order to run requests in Postman, we'll need to generate a collection that creates `GET`, `POST`, and `PUT` queries, and then start the ToDo server to connect to the MySQL database.

Generate a Postman Collection

First, let's generate a collection in Postman:

1. With the ToDos API selected, select **+** under **Collections**, and then select **Generate from definition:**



About this API

My ToDo APIs project

A ToDo API with a database backend.

Connect repository

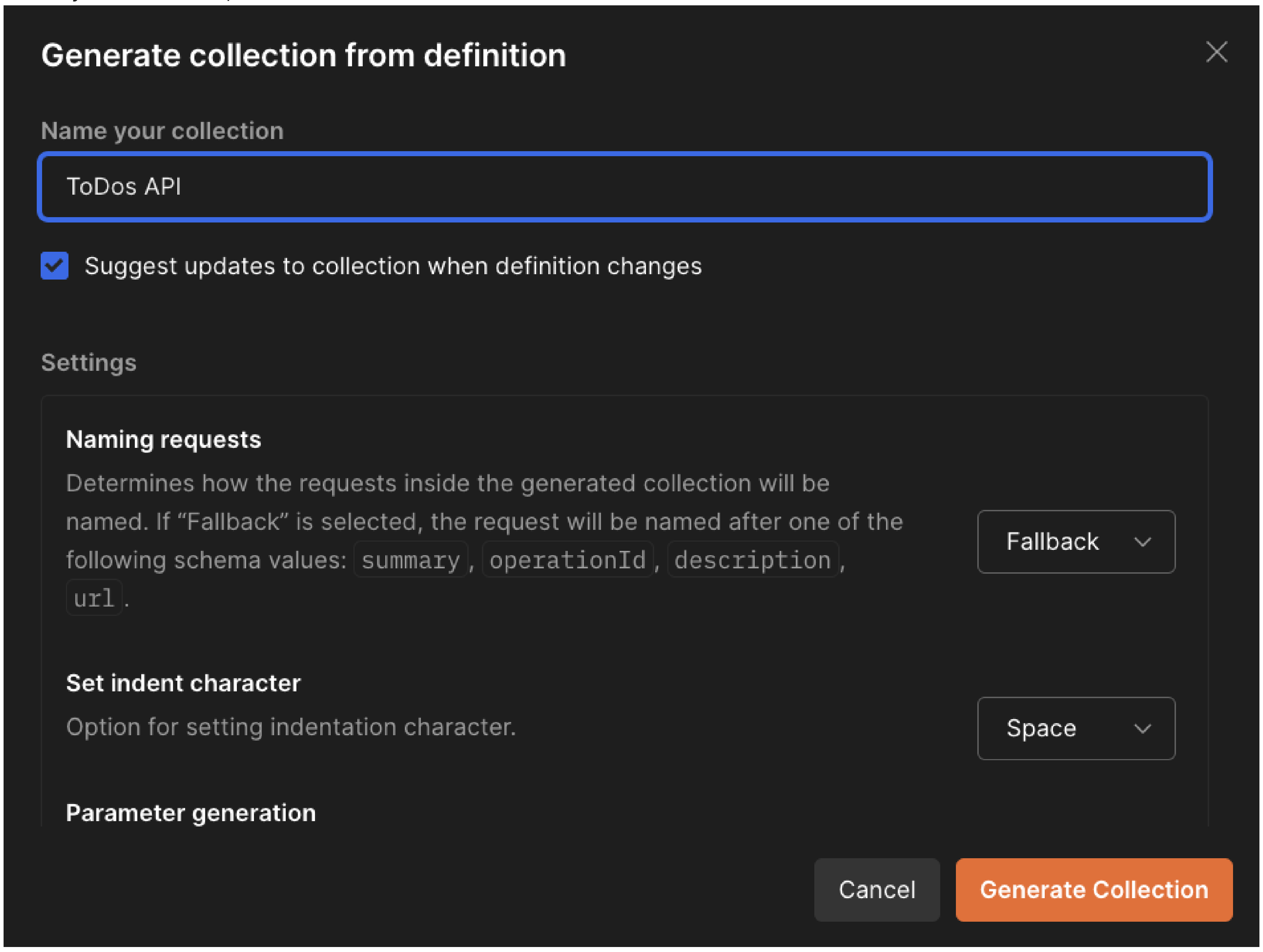
Keep your API development in sync with your code by connecting your repository.

Collections

No collections added yet

Definition

2. Name your collection, and then select **Generate Collection:**


Generate collection from definition

Name your collection: ToDos API

✓ Suggest updates to collection when definition changes

Settings

Naming requests

Determines how the requests inside the generated collection will be named. If "Fallback" is selected, the request will be named after one of the following schema values: `summary`, `operationId`, `description`, `url`.

Fallback

Set indent character

Option for setting indentation character.

Space

Parameter generation

Run the ToDO server

With the collection generated, it's time to get the ToDo server up and running:

1. In the XAMPP Control Panel, make sure that your MySQL database is running.

2. Clone the ToDo API project repository from GitHub.

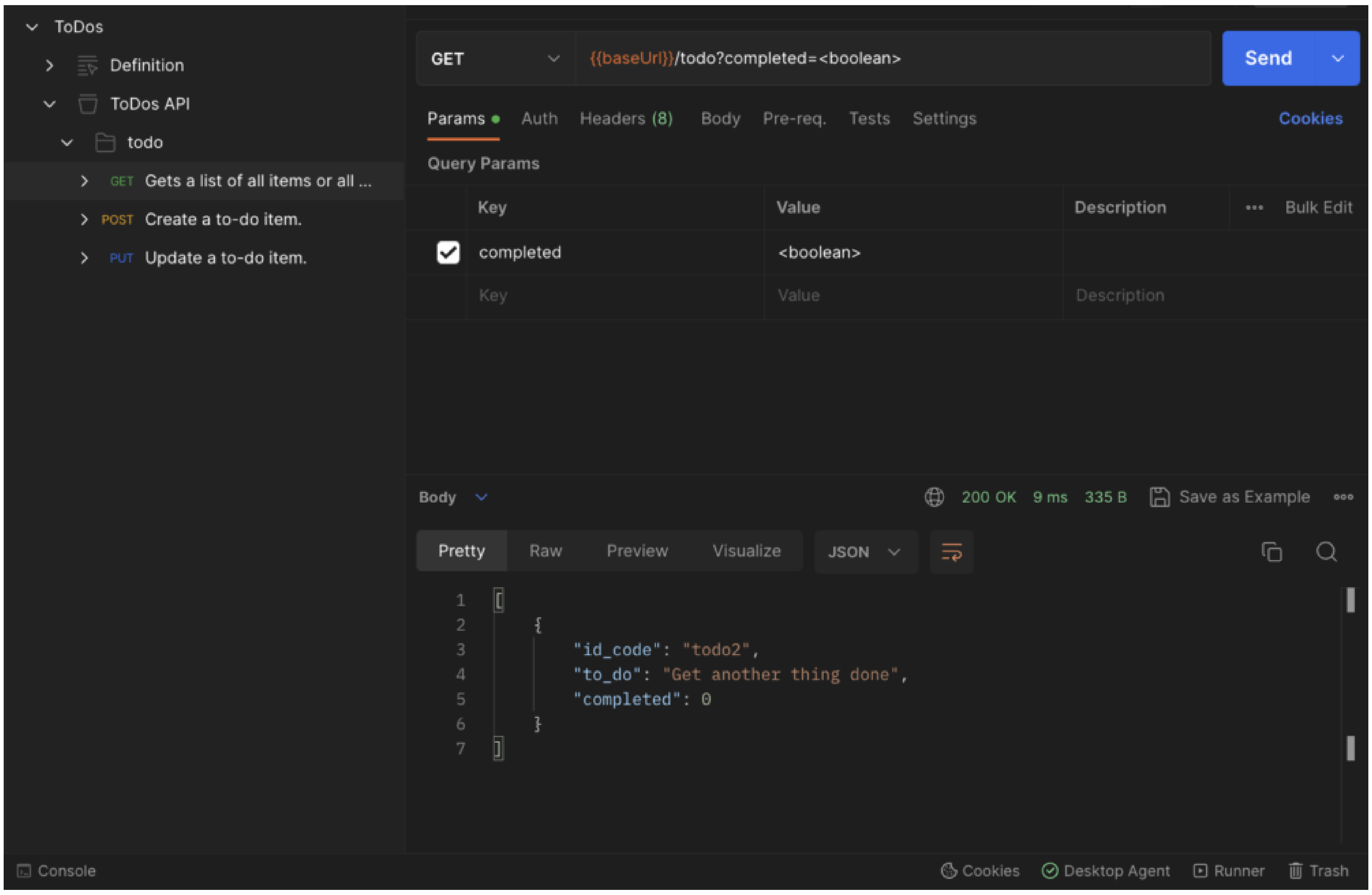
3. In a terminal window, navigate to the top level of the repo.

4. In the terminal, enter `npm install`.

5. When the installation is complete, enter `node run start`.

Try a GET request

From your generated collection in the sidebar, select the `GET` request and then select **Send:**



```
{{baseUrl}}/todo?completed=<boolean>
```

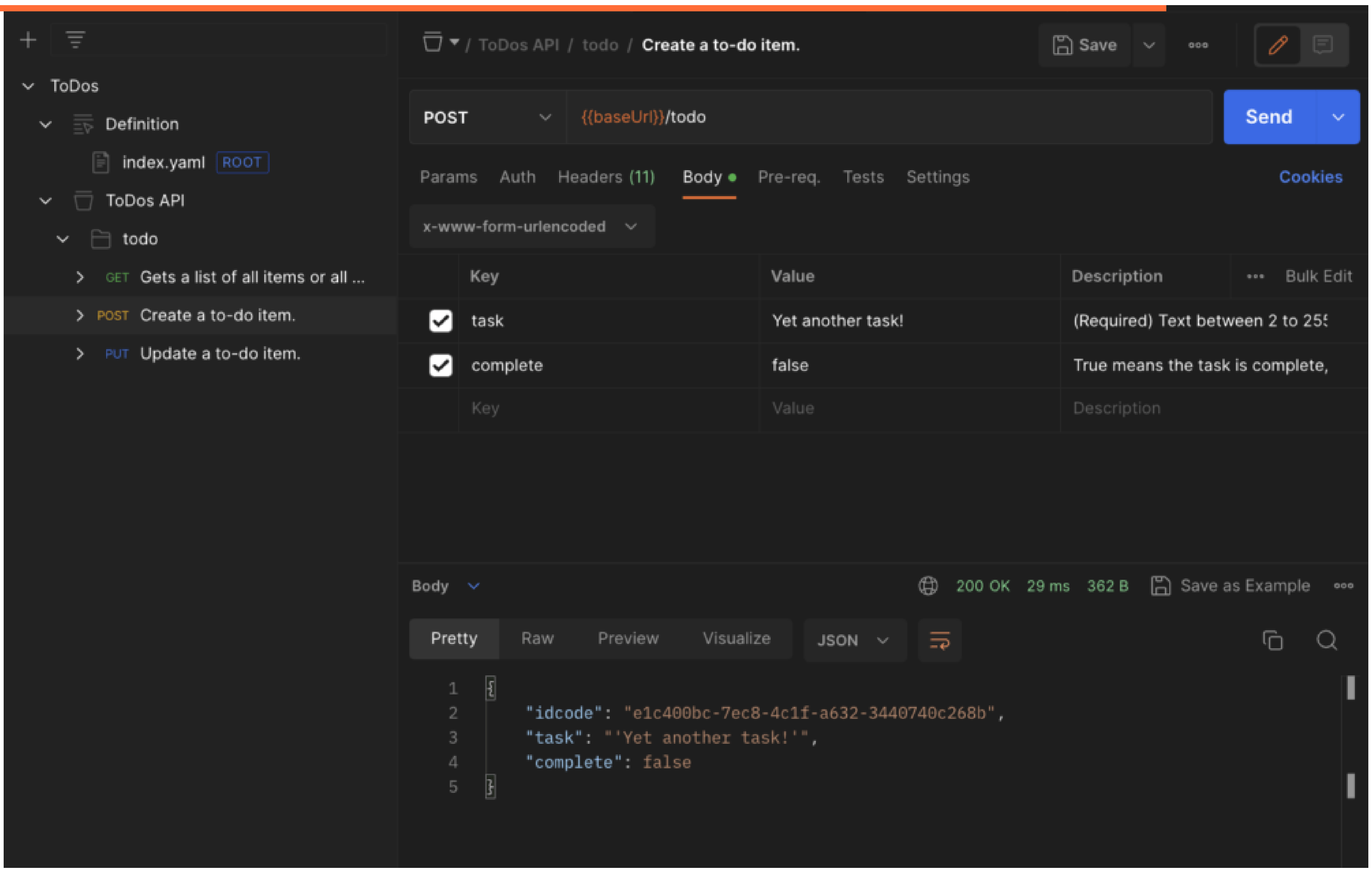
Key	Value	Description
✓ completed	<boolean>	
Key	Value	Description

```
[
    {
        "id_code": "todo2",
        "to_do": "Get another thing done",
        "completed": 0
    }
]
```

In the parameters, you can try changing `<boolean>` to `true` to receive a different response.

Try a POST request

Select the `POST` request and then select the **Body** tab:


ToDos
Definition
index.yaml ROOT
ToDos API
todo
GET Gets a list of all items or all ...
POST Create a to-do item.
PUT Update a to-do item.
ToDos API / todo / Create a to-do item.
Save
Send
Params Auth Headers (11) Body Pre-req. Tests Settings
Cookies
Pretty Raw Preview Visualize JSON
Save as Example

POST {{baseUrl}}/todo

Body: x-www-form-urlencoded

	Key	Value	Description
✓	task	Yet another task!	(Required) Text between 2 to 25!
✓	complete	false	True means the task is complete,
	Key	Value	Description

Body 200 OK 29 ms 362 B

```
{
    "idcode": "e1c400bc-7ec8-4c1f-a632-3440740c268b",
    "task": "'Yet another task!'",
    "complete": false
}
```

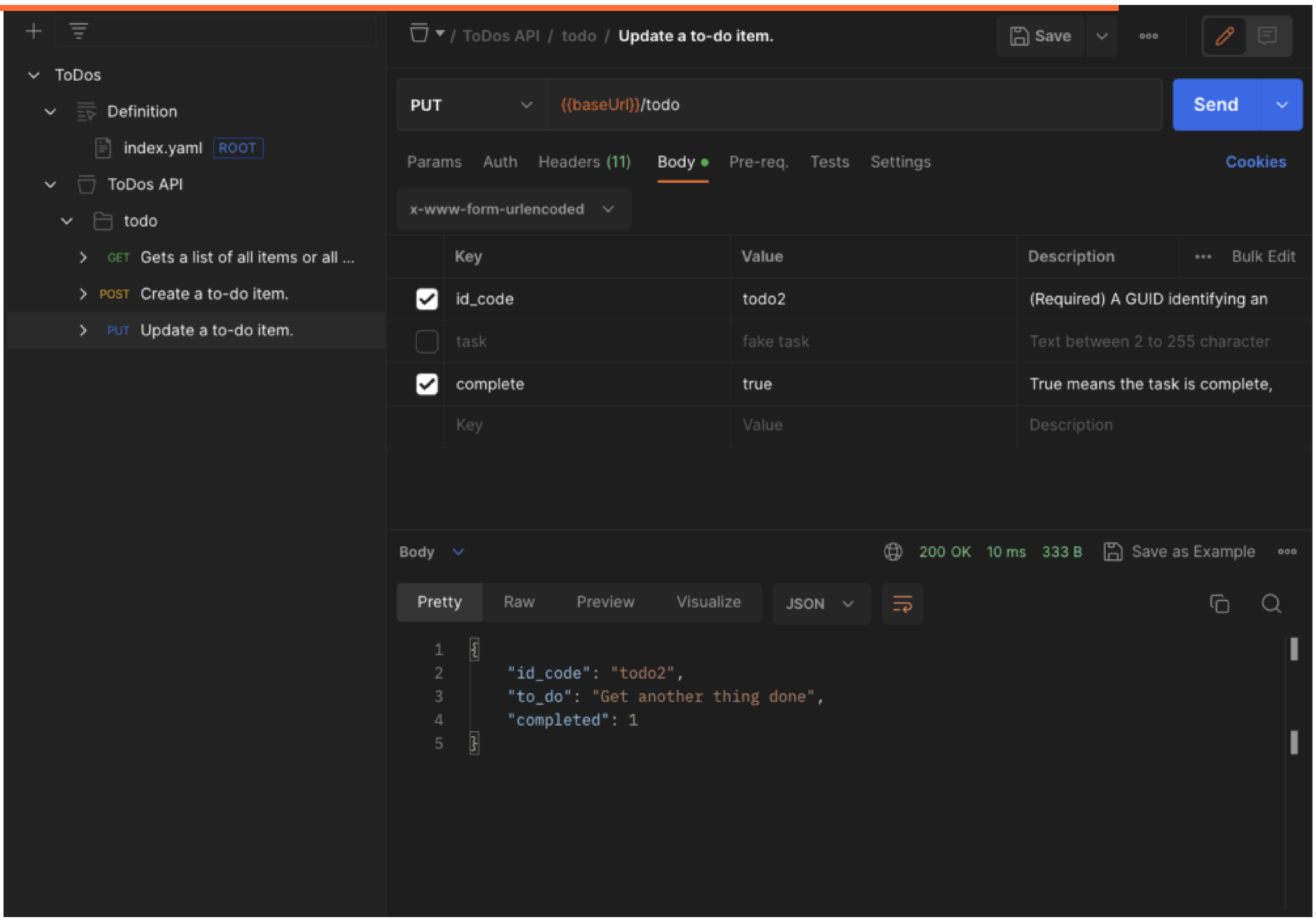
Try replacing the default values with some tasks of your own. Use the following table as a reference when you create your tasks:

Key	Value	Description
task	Title of the task	A description of the task, between 2 and 255 characters
complete	`true`: Task is complete `false`: Task is incomplete	Completion status of the task

If you want to verify that your new tasks were added to the database, run another `GET` request.

Try a PUT request

Select the `PUT` request and then select the **Body** tab:


ToDos
Definition
index.yaml ROOT
ToDos API
todo
GET Gets a list of all items or all ...
POST Create a to-do item.
PUT Update a to-do item.
ToDos API / todo / Update a to-do item.
Save
PUT
{{baseUrl}}/todo
Send
Params Auth Headers (11) Body Pre-req. Tests Settings
Cookies
x-www-form-urlencoded
Bulk Edit
Body
200 OK 10 ms 333 B
Save as Example
Pretty Raw Preview Visualize JSON

	Key	Value	Description
✓	id_code	todo2	(Required) A GUID identifying an
	task	fake task	Text between 2 to 255 character
✓	complete	true	True means the task is complete,
	Key	Value	Description

```
{
    "id_code": "todo2",
    "to_do": "Get another thing done",
    "completed": 1
}
```

The only required parameter for this request is `id_code`. Follow these steps to set it up:

1. Set `id_code` to `todo2`. This is a value we pre-populated when creating the database.
2. Set `completed` to `true`, and then clear the `task` field.
3. Select **Send**.

Sending this request updates the server to show the task is complete, then queries the database for the item and returns it to confirm the database was updated.

Go back and run the `GET` request again to see that the task shows it's complete.

Taking it further

Congratulations! You've defined an API using OpenAPI 3, generated Node.js server code in Postman, created a collection to send requests to the server, and updated the MySQL database. If you want to keep going, here are some other things you can try:

- Add a `DELETE` method to the API definition.
- Update the MySQL server permissions for `todo_admin` and Node.js server code to add a `DELETE` method, then regenerate your collection to test the `DELETE` method.
- Try another Postman Quickstart, or contribute your own!

+9

Tags: Guest Post MySQL Node.js Tutorials

Guest Author

This is a special guest author post for the Postman blog. View all posts by Guest Author.

What do you think about this topic? Tell us in a comment below.

Comment

Your name

Your email

Write a public comment

Post Comment

4 thoughts on "Build an API with Postman, Node.js, and MySQL"

cod Kamp
June 24, 2023

If you are following the tutorial and building the App, at the end why do you need to clone the repo to run it. why not just run the app you've been building a long the way... it doesn't work for some reason!

0

The Postman Team
June 26, 2023

Hi, the blog post is intended to be a big-picture overview. Instead of going through each step of the server code setup, we wanted to show how everything works together when building an API. You can reach out to our support team at https://support.postman.com/support and they can help you if you still need more assistance.

0

Chris Smith
July 26, 2023

0

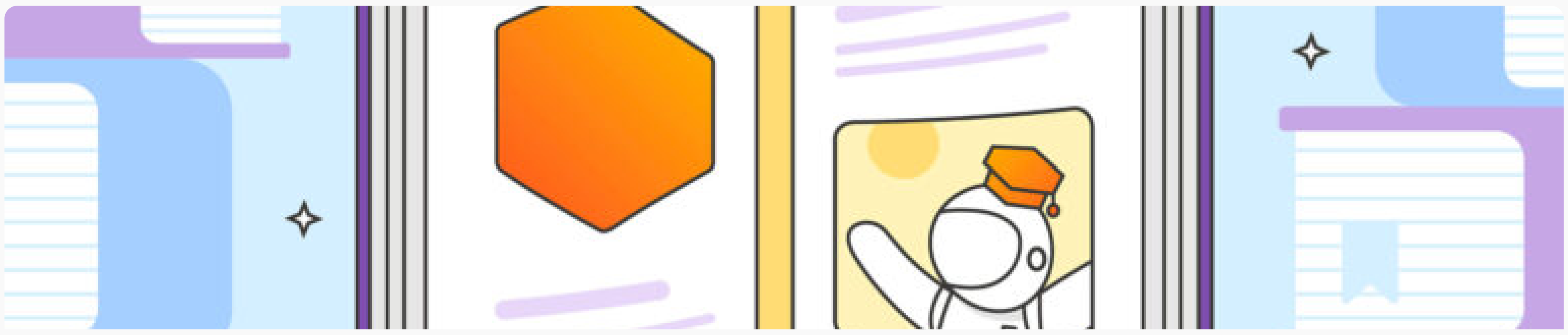
great guide, followed it through to the end. Believe there may be a couple of mis types around the “complete” vs “completed” keys though? unless im mistaken.

The Postman Team
July 31, 2023

Thanks for pointing that out, Chris! We’ve updated the post to say “completed” instead of “complete.”

0

You might also like

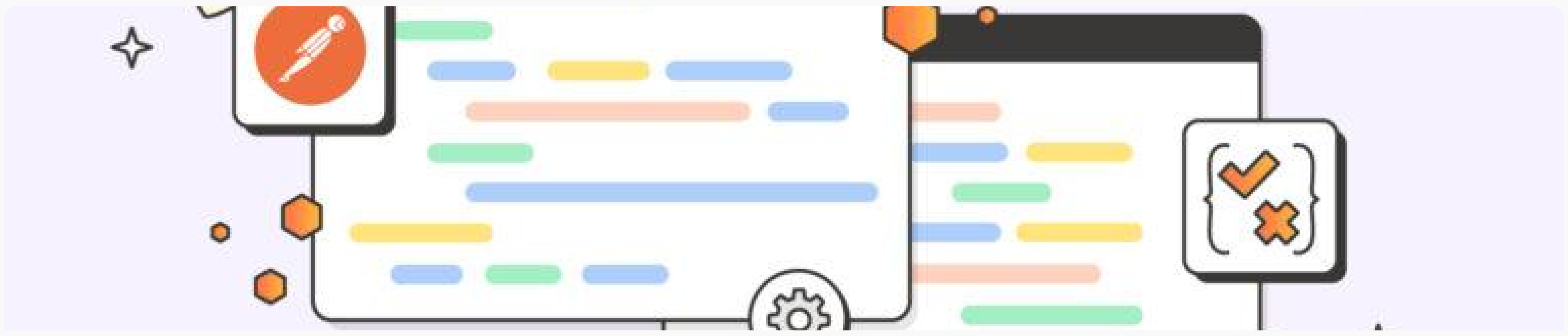


Introducing Project-Based Learning in Postman Academy

Som Nath · 3 mins

As part of our ongoing commitment to promoting API literacy, the Postman Student Programs team is thrilled to announce the launch of...

Read more →

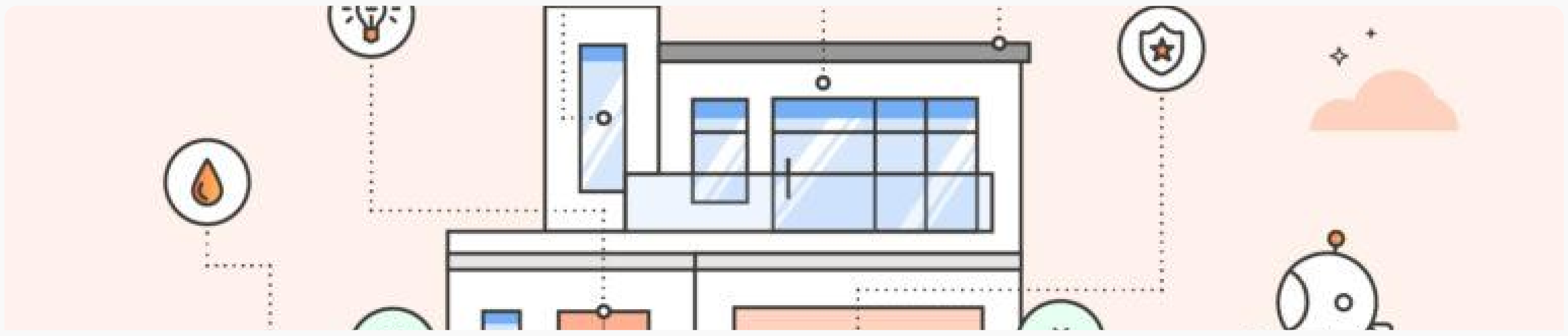


How Postman uses JSON Schema

Juan Cruz Viotti · 5 mins

The Postman API Platform offers a rich set of solutions for every step of the API lifecycle. Through the years, we have...

Read more →



Powering home automation with WebSocket APIs

Joyce · 4 mins

In Part 1 of this series, we learned about the WebSocket protocol and how to set up our own WebSocket server in...

Read more →

Postman named Best API Platform

Postman is the #1 place where developers come to work with APIs. See why we're top-ranked in G2's first-ever evaluation of API Platforms.

Read more →

Product

What is Postman?

API Repository

Tools

Governance

Workspaces

Integrations

Enterprise

Company

About

Careers and culture

Press and media

Contact us

Partner program

Plans and pricing

Download the app

Support Center

Legal and Security

Terms of Service

Trust and Safety

Privacy policy

Cookie notice

Privacy choices

API Categories

App Security

Payments

Financial Services

DevOps

Developer Productivity

Data Analytics

Communication

Artifical Intelligence

Social

_Twitter

_LinkedIn

_GitHub

_YouTube

_Twitch