**How to Set Up a Mock Backend with Angular 13 Applications**

In a world of simultaneous front-end and back-end development, it is essential that the Angular developers working on an app should be able to interact with a fake back end in order to create the UI by mocking the back end data.

Thanks to npm packages like **[json-server](https://www.npmjs.com/package/json-server" \t "_blank)** and **[nodemon](https://www.npmjs.com/package/nodemon" \t "_blank)**, the process is made real simple.

**json-server** is a package providing full fake REST API with zero coding and it can be set up without much effort.

**nodemon** is a tool that helps develop Node.js-based applications by automatically restarting the node application when file changes in the directory are detected.

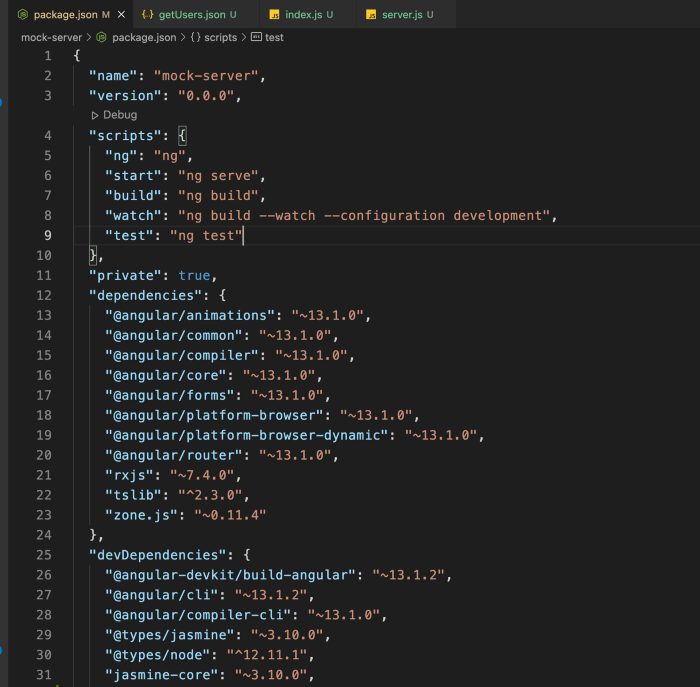
Here is a step-by-step set up of a mock back-end where you can set up a fake REST API endpoint returning mock JSON response from scratch:

**1. Create a new project**

Create a brand new Angular project by using the [**Angular CLI tool**](https://www.npmjs.com/package/@angular/cli) command:

ng new mock-server

This will set up an Angular project by the name ***mock-server*** containing ***package.json***as shown below. At the time of writing this article, the Angular version used was v13. It might change for someone reading at a later point in time, nevertheless, the concept still remains the same and it works on every version of Angular 2+.

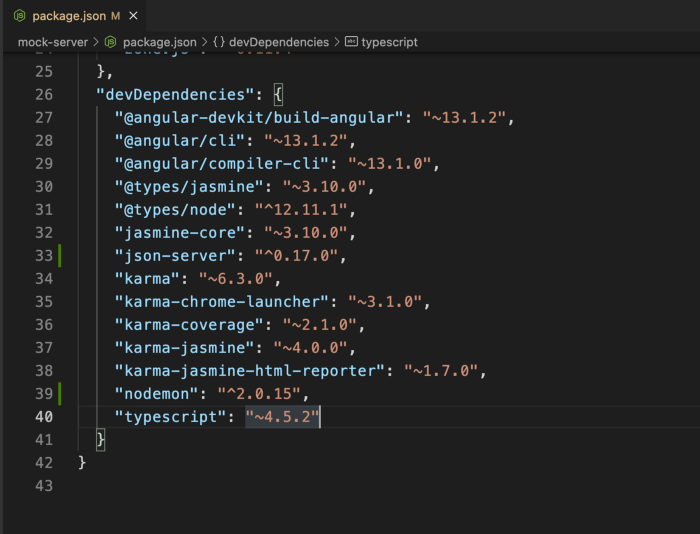


# 2. Installing npm packages

npm install json-server --save-dev  
npm install nodemon --save-dev

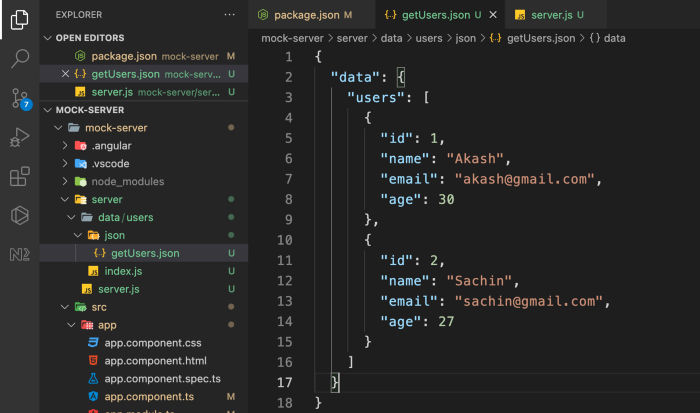
The above commands install the latest version of json-server and nodemon as a dev dependency and would add it to your **package.json**file.

The above commands install the latest version of json-server and nodemon as a dev dependency and would add it to your **package.json**file.



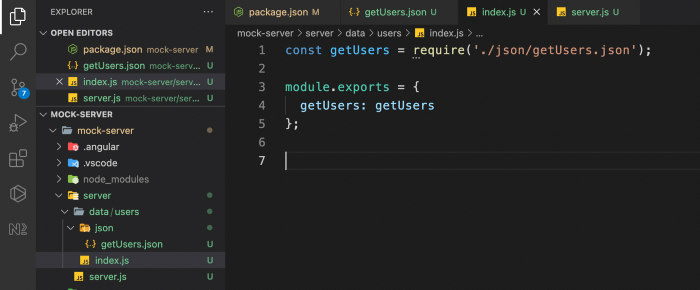
# 3. Add mock data in the JSON file

Create a folder by the name of **server**. The name can be anything based on your choice. Create another folder structure within the **server** folder, **data/users/json** and create a **getUsers.json**filecontaining the list of mock users in the form of a JSON response**.**The above folder structure is created to ensure consistency in case multiple fake API JSON responses are added and they might belong to separate feature modules in the app. The folder structure is created on the basis of feature modules, much like what we do for feature modules in an Angular app.



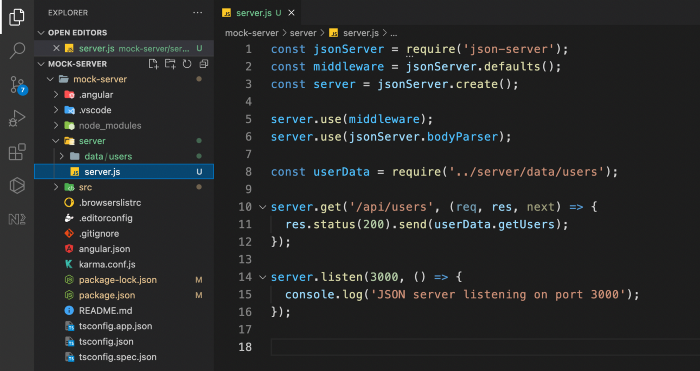
# 4. Add an index.js file

To export the JSON data in the **server.js** file, create an **index.js**file in the **server/data/users** folder which exports the mock JSON response (a list of users) as a module to be used in the **server.js** file. The **module.exports** is a special object which is included in every JavaScript file in the Node.jsapplication by default. The module is a variable that represents the current module, and exports is an object that will be exposed as a module.



# 5. Add a server.js file to plug everything together

Create a **server.js** file which contains the configuration to set up a **json-server** using Node.js with the below configuration:

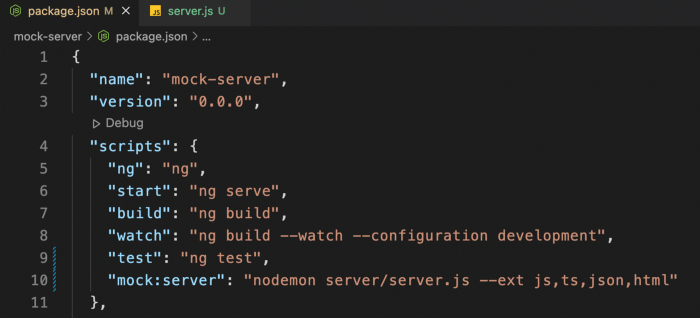


The above Node server runs on port number 3000. It listens for requests at endpoint **/api/users** to return the list of users on [**http://localhost:3000/api/users**](http://localhost:3000/api/users)**.**

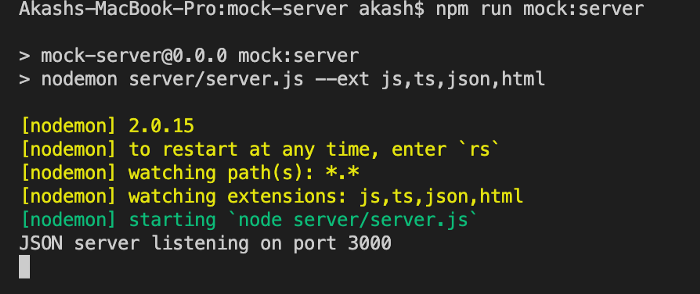
# 6. Add a script in package.json

When the configuration of the server is done in the **server.js** file, create a script in the **package.json** as below:

"mock:server": "nodemon server/server.js --ext js,ts,json,html"



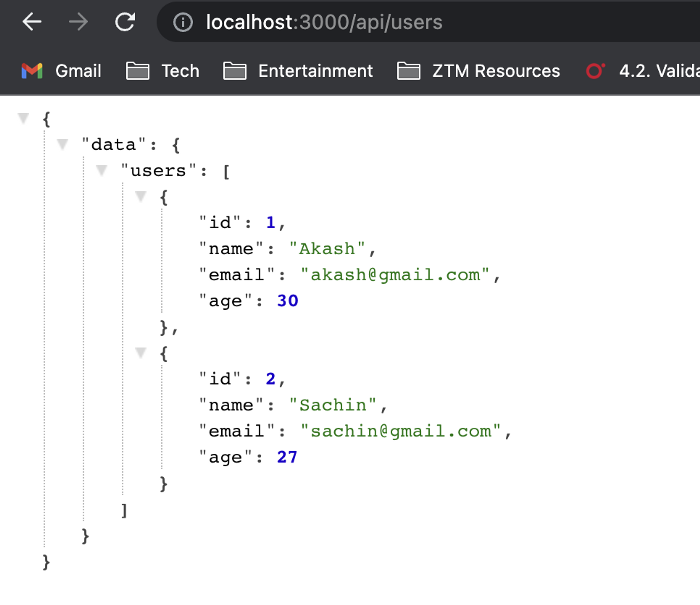
Add **npm run mock:server** on the command line to see the server running with a console message, **“JSON Server running at port 3000”.**The above script runs in watch mode and also listens for any changes to files of type \*.js, \*.ts, \*.json, and \*.html and it restarts the node server automatically.



# 7. Go to the browser and enter the URL

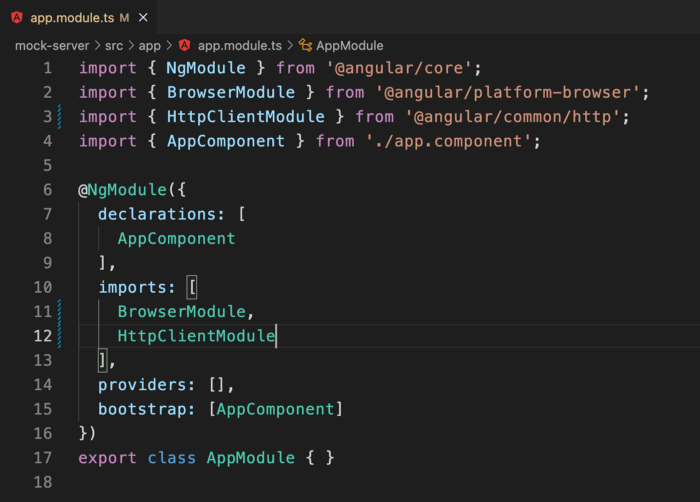
Voila! Our mock server is set up and ready for testing. Go to your browser and hit the <http://localhost:3000/api/users> in the address bar. You will be able to see the list of users on the browser.

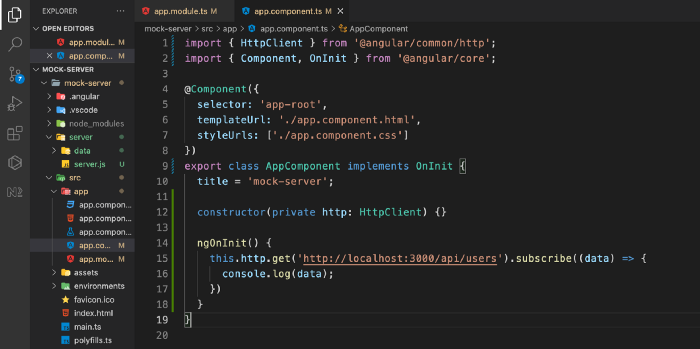
Similarly, you can set up the mock endpoints for POST, PUT, PATCH, and DELETE operations.



# ****8. Testing it on Angular Application UI****

Inject the [**HttpClientModule**](https://angular.io/api/common/http/HttpClientModule) in the **app.module.ts** and inject [**HttpClient**](https://angular.io/api/common/http/HttpClient) as a dependency injection in the **app.component.ts.**Then, in the [**ngOnInit()**](https://angular.io/api/core/OnInit) lifecycle hook, get the list of users from the mock API endpoint <http://localhost:3000/api/users> and print the data in the console or bind it in the class property to view in the UI.





If you were coding along and got stuck anywhere in the process, below is the public link to the GitHub repo: