Mockbee Notes

Why do we need Mockbee?

frontend needs a backend to fetch data and for that, a backend has to be created. As a frontend developer, we do not have to create a backend so you can use a premade backend called Mockbee.

How it is helpful?

Mockbee provides us with a premade backend that already contains controllers, APIs, and data.

The data present within Mockbee is customizable i.e. you can edit the backend files according to you.

A backend has 3 basic parts to understand -

1. The API
2. The request-response cycle
3. The database

1. The API

API is a means that helps in the communication of the front end and server. The main role of API is to call or transfer data on different requests such as `GET`, `POST`, or `DELETE` from the server.

2. Request-response cycle

Whatever request is made to the backend is called the request cycle. The request is demanded by the front end.

In response, the request in return from the backend is called the response cycle. The response from the database can be data or an error message.

3. Database

A database is something where the actual data is stored. Mockbee provides us with a way to store data in the local system that is your PC.

Mockbee Features:

- Mockbee gives fake backend API
- Easily configurable and highly customizable
- Provides auth
- Can be extended to have more features
- If you want to create a new data API and you can just use static data

How to install a mockBee template?

- Mockbee template is provided with create-react-app. We can download the mockbee template for e-commerce with the help of create-react-app.

```
npx create-react-app my-app --template mockbee-[project-name-here]
```

COPY

- After the successful installation, a folder named backend can be seen which is created by Mockbee.
- Within this backend folder, the db folder can be seen which will have 3 files that will provide data to the front end.
- The data in these 3 files are customizable and can be changed according to the theme of our project.

API section

There are 2 types of APIs provided by mockBee -

- General API - `auth API`
- App-specific APIs (such as `E-Commerce`, `Video-Library`, etc.)- that have public routes as well as private routes.

auth API-

How does auth API work in mockBee:

- A JWT Token is generated whenever a user logs in that has the encrypted information that includes the user's data.
- Now, when the User tries to access private routes, the token is passed with the HTTP REQUEST as a `requestHeader`.
- The Server, with the `JWT_SECRET`, verifies the token and if the token is valid, which means that the user is authenticated. Now, the server can let users access the routes that are requested.

How to setup auth `JWT_SECRET`-

- Outside the src folder, create a .env file.

- Inside the .env file, create a variable with the name `REACT_APP_JWT_SECRET` which takes a secret string value. Any value can be given to this variable. Given below:

```
REACT_APP_JWT_SECRET = "any";
```

COPY

- Without `REACT_APP_JWT_SECRET`, the authentication system will not work.

How to test the API routes?

- To test the APIs of mockbee we will make use of mockman.js.
- Mockman.js is a package that helps in testing the fake mock backend APIs.

How to install mockman.js?

- Install Mockman by running:

```
npm i mockman-js
```

COPY

- Import the component in your App.

```
import React from "react";
import Mockman from "mockman-js";
function MockAPI() {
  return (
    <div className="MockAPI">
      <Mockman />
    </div>
  );
}

export default MockAPI;
```

COPY

- In your `index.html`, add the following:

```
<link
  rel="stylesheet"
  href="https://unpkg.com/mockman-js@latest/dist/style.css"
/>
```

COPY

- To make work easy, we can create a separate route for `<Mockman/>` component with route `/mockman`. And test the API requests at this route.

```
<Route path='/mockman' element={<Mockman />} />
```

COPY

- The mockman interface provides a dropdown to select the request type, which can be `GET`, `POST`, or `DELETE`, and an input field to enter the API route.

- Let's understand the API testing with the help of `cart` API routes.

How to test a GET request?

- Navigate to `/mockman`.
- From the dropdown menu select `GET` request and add the API route as `/api/user/cart`.
- In the headers section, check the `accept` checkbox.
- In response, the data present inside the cart component will be obtained.

How to test a POST request?

- Navigate to `/mockman`.
- From the dropdown menu select `POST` request and add the API route as `/api/user/cart`.
- In the headers section, check the `accept` checkbox.
- Add another header with name `authorization` and pass the `encodedToken` returned from the auth login route.
    - To get this encoded token you can follow the below Topic named Storing encodedToken
- Make sure the body of the request is in the following format :

```
{
  "product" : // data to be added;
}
```

COPY

How to test a DELETE request?

- From the dropdown menu select `DELETE` request and add the API route as `/api/user/cart/:productId`.
- In the URL, pass the id of the item that is to be deleted in place of the `productId`.
- In the headers section, check the `accept` checkbox.

- Add another header with name `authorization` and pass the `encodedToken` returned from the auth login route.
- There is no need of providing the `body` in a `DELETE` request.

How to call the API?

- Let's create a GET request for the `products` API route.
- The fetch function can be used to fetch the `products` API route.

```
const getData = async () => {
try {
  const response = await fetch("api/products")
  console.log(await response.json())
 }
catch(e){
 console.log(e)
 }
}
```

COPY

Storing the encodedToken

- The Login and Sign Up APIs return `encodedToken` as one of the responses. Since, for accessing the private routes, this has to be passed as `authorization` header; you have to store it somewhere on the client.
- You can store this token in the client's `localStorage`:

```
const signupHandler = async () => {
  try {
    const credentials = {
            email: "adarshbalika@neog.camp",
            password:"adarshBalika"
          }

          const res = await fetch("/api/login",{
                  method: "POST",
                  body: JSON.stringify(credentials)
                })

                const {encodedToken} = await res.json();
                // to store the token in localStorage
                localStorage.setItem("token", encodedToken);
                // to get the token on the console
                console.log(localStorage.getItem("token"));

  } catch (error) {
    console.log(error);
  }
};
```

COPY

JSON parsing

When we want to pass data as JSON, we have to use `JSON.stringify()` function. The data will be converted to JSON format.

When we get data from the server in JSON format, we will use JSON to convert it into object notation.`parse()` function.

Reference:

- For E-commerce app Documentation: https://mockbee.netlify.app/docs/api/apps/e-commerce/
- For Auth Documentation: https://mockbee.netlify.app/docs/api/general/auth