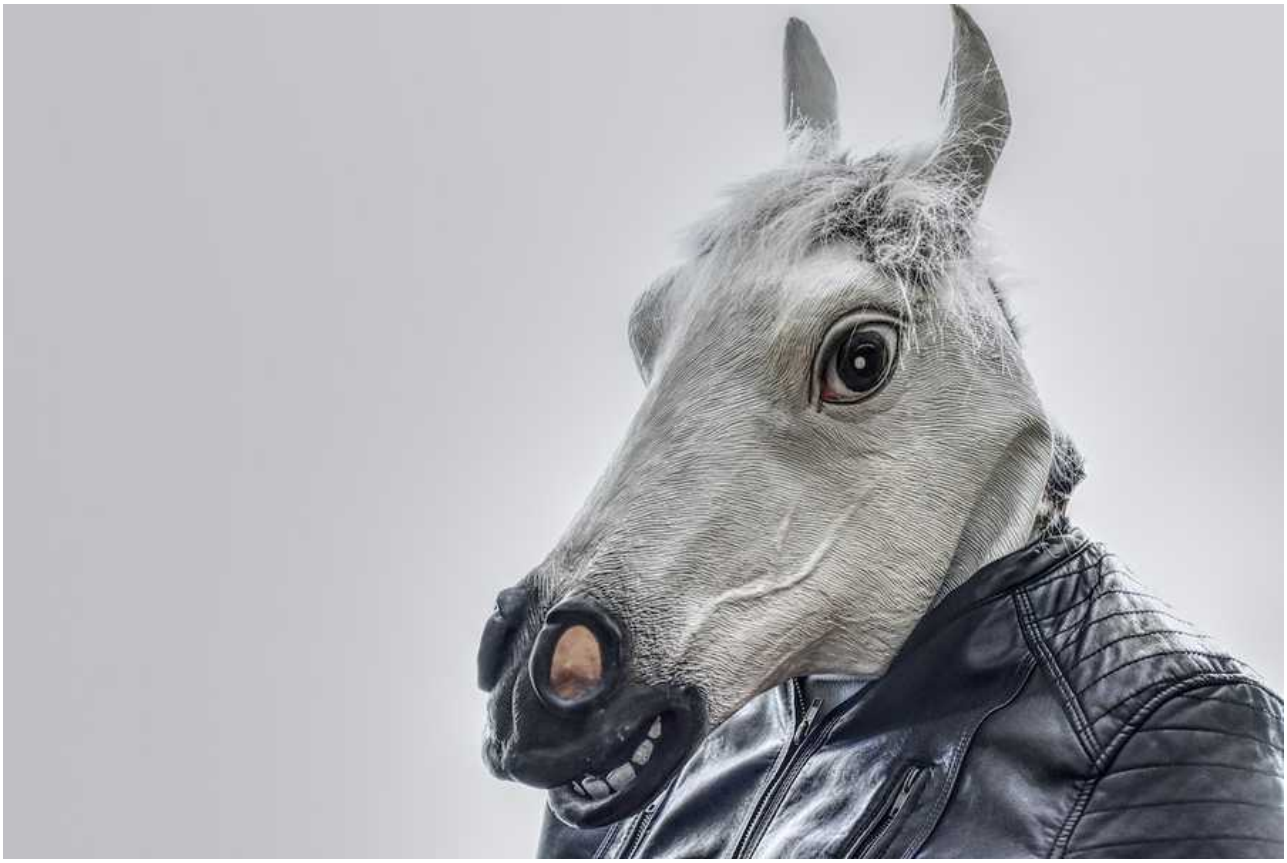
JavaScript fake API with Mock Data

SEPTEMBER 13, 2020 BY ROBIN WIERUCH - EDIT THIS POST



In this tutorial we will implement a JavaScript fake API. Often this helps whenever there is no backend yet and you need to implement your frontend against some kind of realistic data. Fake it till you make it!

JAVASCRIPT FAKE API

Let's get started. First of all, we need some data which would normally come from our backend's database, but which will just come from a JavaScript file in our case:

```
// pseudo database
let users = {
```

```
  1: {
    id: '1',
    firstName: 'Robin',
    lastName: 'Wieruch',
    isDeveloper: true,
  },
  2: {
    id: '2',
    firstName: 'Dave',
    lastName: 'Davddis',
    isDeveloper: false,
  },
};
```

Next we need unique identifiers, which aren't important with only two items in the fake database, but which are important for creating more items eventually:

```
import { v4 as uuidv4 } from 'uuid';

const idOne = uuidv4();
const idTwo = uuidv4();

let users = {
  [idOne]: {
    id: idOne,
    firstName: 'Robin',
    lastName: 'Wieruch',
    isDeveloper: true,
  },
  [idTwo]: {
    id: idTwo,
    firstName: 'Dave',
    lastName: 'Davddis',
    isDeveloper: false,
  },
};
```

You can install the library with `npm install uuid` from over here. Our fake database is complete now.

Now we will move on with our fake API. Therefore we will follow the CRUD pattern for creating, reading, updating and deleting entities from our fake database via our fake API. First, we need to retrieve all items from the database with one pseudo API request:

```
const getUsers = () =>
  Object.values(users);

// usage
```

```
// usage
const result = getUsers();
console.log(result);
```

This function returns our object of items as an converted array. However, it's just a function which returns data synchronously. In order to fake an API, it would need to be asynchronous. Therefore, we will wrap it into a JavaScript promise:

```
const getUsers = () =>
  Promise.resolve(Object.values(users));

// usage (1)
getUsers()
  .then(result => {
    console.log(result);
  });

// usage (2)
const doGetUsers = async () => {
  const result = await getUsers();
  console.log(result);
};

doGetUsers();
```

Instead of using the previous shorthand promise version, we will use the longer version:

```
const getUsers = () =>
  new Promise((resolve) => {
    resolve(Object.values(users));
  });
```

The longer promise version enables us to handle errors too:

```
const getUsers = () =>
  new Promise((resolve, reject) => {
    if (!users) {
      reject(new Error('Users not found'));
    }

    resolve(Object.values(users));
  });

// usage (1)
getUsers()
  .then((result) => {
    console.log(result);
```

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

// usage (2)
const doGetUsers = async () => {
  try {
    const result = await getUsers();
    console.log(result);
  } catch (error) {
    console.log(error);
  }
};

doGetUsers();
```

Last but not least, we want to introduce a fake delay to make our fake API realistic:

```
const getUsers = () =>
  new Promise((resolve, reject) => {
    if (!users) {
      return setTimeout(
        () => reject(new Error('Users not found')),
        250
      );
    }

    setTimeout(() => resolve(Object.values(users)), 250);
  });
```

That's it. Calling this function feels like a real API request, because it's asynchronous (JavaScript promise) and has a delay (JavaScript's `setTimeout`). After we went through this first API step by step, we will continue with the other CRUD operations now.

JAVASCRIPT FAKE REST API

A traditional REST API can be seen to analogous to CRUD operations very well. That's why we will implement the following API with REST in mind, by offering API endpoints for reading item(s), creating an item, updating an item, and deleting an item. Before we already implemented reading multiple items:

```
const getUsers = () =>
  new Promise((resolve, reject) => {
```

```
    if (!users) {
      return setTimeout(

        () => reject(new Error('Users not found')),
        250
      );
    }

    setTimeout(() => resolve(Object.values(users)), 250);
  });
```

Next, we will implement the equivalent for reading a single item; which is not much different from the other API:

```
const getUser = (id) =>
  new Promise((resolve, reject) => {
    const user = users[id];

    if (!user) {
      return setTimeout(
        () => reject(new Error('User not found')),
        250
      );
    }

    setTimeout(() => resolve(users[id]), 250);
  });

// usage
const doGetUsers = async (id) => {
  try {
    const result = await getUser(id);
    console.log(result);
  } catch (error) {
    console.log(error);
  }
};

doGetUsers('1');
```

Next, creating an item. If not all information is provided for the new item, the API will throw an error. Otherwise a new identifier for the item is generated and used to store the new item in the pseudo database:

```
const createUser = (data) =>
  new Promise((resolve, reject) => {
    if (!data.firstName || !data.lastName) {
      reject(new Error('Not all information provided'));
    }
```

```
    }

    const id = uuidv4();

    const newUser = { id, ...data };

    users = { ...users, [id]: newUser };

    setTimeout(() => resolve(true), 250);
  });

// usage
const doCreateUser = async (data) => {
  try {
    const result = await createUser(data);
    console.log(result);
  } catch (error) {
    console.log(error);
  }
};

doCreateUser({ firstName: 'Liam', lastName: 'Wieruch' });
```

Next, updating an item. If the item is not found, the API will throw an error. Otherwise the item in the object of item will be updated:

```
const updateUser = (id, data) =>
  new Promise((resolve, reject) => {
    if (!users[id]) {
      return setTimeout(
        () => reject(new Error('User not found')),
        250
      );
    }

    users[id] = { ...users[id], ...data };

    return setTimeout(() => resolve(true), 250);
  });

// usage
const doUpdateUser = async (id, data) => {
  try {
    const result = await updateUser(id, data);
    console.log(result);
  } catch (error) {
    console.log(error);
  }
};

doUpdateUser('1', { isDeveloper: false });
```

And last but not least, deleting an item. Same as before, if the item cannot be found, the API returns an error. Otherwise we only get the confirmation that the item has been removed from the object of items:

```
const deleteUser = (id) =>
  new Promise((resolve, reject) => {
    const { [id]: user, ...rest } = users;

    if (!user) {
      return setTimeout(
        () => reject(new Error('User not found')),
        250
      );
    }

    users = { ...rest };

    return setTimeout(() => resolve(true), 250);
  });

// usage
const doDeleteUser = async (id) => {
  try {
    const result = await deleteUser(id);
    console.log(result);
  } catch (error) {
    console.log(error);
  }
};

doDeleteUser('1');
```

We have implemented the entire fake API for a RESTful resource (here user resource). It includes all CRUD operations, has a fake delay and returns an asynchronous result. For the write operations, the API only returns an acknowledgment (boolean), however, you could also decide to return an identifier (e.g. identifier of the removed item) or an item (e.g. the created/updated item).

This tutorial is part 1 of 2 in this series.

Part 2: How to mock data in React with a fake API

Show Comments

KEEP READING ABOUT REACT›

SETUP POSTGRESQL WITH SEQUELIZE IN EXPRESS

Eventually every Node.js project running with Express.js as web application will need a database. Since most server applications are stateless, in order to scale them horizontally with multiple server...

HOW TO MOCK DATA IN REACT WITH A FAKE API

In this tutorial we will implement use JavaScript fake API with mock data from a pseudo backend to create our frontend application with React. Often this helps whenever there is no backend yet and you...

TAKE PART

NEVER MISS AN ARTICLE ABOUT WEB DEVELOPMENT AND JAVASCRIPT.



- ✔ Join 50.000+ Developers
- ✔ Learn Web Development with JavaScript
- ✔ Tips and Tricks
- ✔ Access Tutorials, eBooks and Courses
- ✔ Personal Development as a Software Engineer

Your email address SUBSCRIBE

View our Privacy Policy.

Online Courses
Open Source
Tutorials
About me
What I use
How to work with me
How to support me

Contact Me Privacy & Terms