

Fullstack Web Development

Network Call & Form Validation

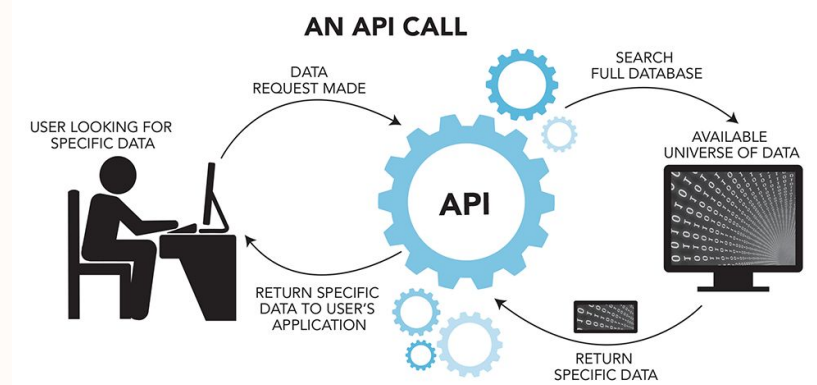
Outline

- Network call
- Create local server using *json-server*
- Fetch and Axios
- React Query
- Form submission & validation



What is Network Call?

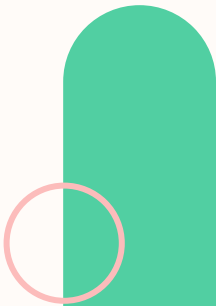
Network calls, in the context of web development, refer to the process of a software application communicating with a server or another part of the network to exchange data. This communication is often necessary for fetching data, submitting data, or performing other operations that involve interactions between



What is Network Call?

In the context of a React application or any other web application, network calls typically involve making HTTP requests to a server or an external API (Application Programming Interface). These requests can be of various types, such as:

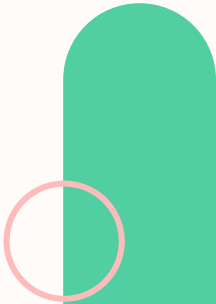
1. **GET** requests: Used to retrieve data from a server.
2. **POST** requests: Used to submit data to be processed to a specified resource.
3. **PUT and PATCH** requests: Used to update existing data on the server.
4. **DELETE** requests: Used to request the removal of a resource on the server.



What is Responses in Network Call?

When you make a network call, the server typically responds with an HTTP (Hypertext Transfer Protocol) response. This response contains information about the status of the request and may include data, headers, and other relevant details. In the context of a React application or any web development, there are several components to a network call response:

- Status Code (200, 201, 204, 400, 404, 500)
- Headers
- Body



What is Responses in Network Call?

In this example:

Status Code: 200 OK indicates a successful response.

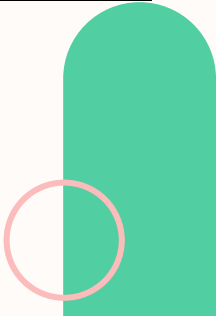
Headers: Content-Type: application/json, Date: Wed, 17 Nov 2023 12:00:00 GMT, Server: ExampleServer.

Body: The JSON data {"message": "Hello, World!"} is the actual content returned by the server.

In a React application, you would typically handle these responses in your code, parsing the data from the response body and taking appropriate actions based on the status code and content. This is commonly done using functions like `fetch` or third-party libraries like `Axios`.

```
HTTP/1.1 200 OK
Content-Type: application/json
Date: Wed, 17 Nov 2023 12:00:00 GMT
Server: ExampleServer

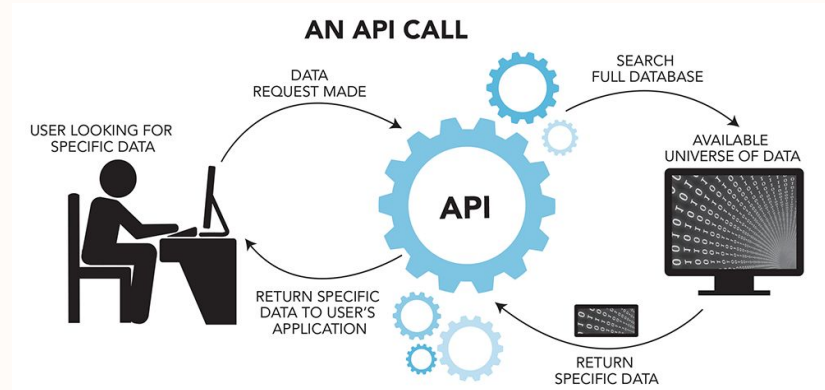
{
  "message": "Hello, World!"
}
```



Why Network Call is Important?

Network calls are a fundamental aspect of web development, enabling communication between a client-side application and a server or external services to exchange data and perform various operations. Here is several reason:

- Data Retrieval
- API Integration
- User Interaction
- Dynamic Content
- Efficient Resource Management



JSON-SERVER

The json-server is a JavaScript library to create testing REST API with form of HTTP request (Network call).

For installation you can install json-server globally.

npm install -g json-server

Now we can create server locally with json-server or the other name is fake API because we don't have to create API.

To create API, we can learn that in module Back-end.



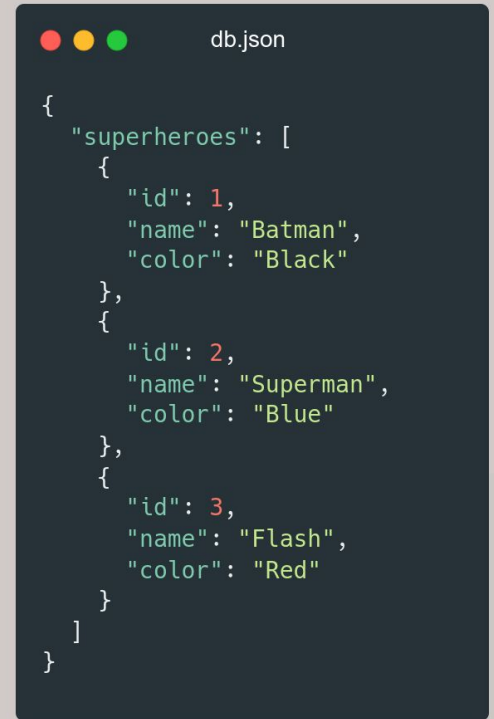
```
{
  "superheroes": [
    {
      "id": 1,
      "name": "Batman",
      "color": "Black"
    },
    {
      "id": 2,
      "name": "Superman",
      "color": "Blue"
    },
    {
      "id": 3,
      "name": "Flash",
      "color": "Red"
    }
  ]
}
```


JSON-SERVER

Now, in the root of project you have to create a file with ".json" extension, for example you can create a file with the name db.json which contains data like in the side. Remember, since file extension in json file, you need to write down the data like the sample in the right.

In this sample of data, we named it as superheroes that contains array of objects. Each objects contain id, name, and color key and value.

Just imagine this file would be your **fake storage data, since you have not learn about how to develop an API**



```
{
  "superheroes": [
    {
      "id": 1,
      "name": "Batman",
      "color": "Black"
    },
    {
      "id": 2,
      "name": "Superman",
      "color": "Blue"
    },
    {
      "id": 3,
      "name": "Flash",
      "color": "Red"
    }
  ]
}
```

JSON-SERVER

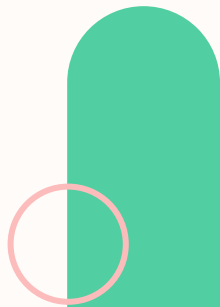
To run our json-server with its data, we have to access its directory in terminal, then we can use this command:

```
json-server --watch jsonFileName => json-server --watch db.json
```

This command will run our server in port default which is 3000. But if you want to run it on different port, you can use -p like this

```
json-server -pPortYouWant jsonFileName => json-server -p 2000 db.json
```

Let's check this port on your browser in order to look after the data in db.json. Try to open <http://localhost:3000/superheroes> if you are using default port (adjust the port if using custom port)



JSON-SERVER

You just completed to create fake API with JSON Server and performing network call using GET method to get the superheroes list from your browser. Next, we will talk about how to perform network call from your React Projects.

```
[
  {
    id: 1,
    name: "Batman",
    color: "Black"
  },
  {
    id: 2,
    name: "Superman",
    color: "Blue"
  },
  {
    id: 3,
    name: "Flash",
    color: "Red"
  }
]
```

Network Call using Fetch and Axios

Network call can be done by two ways using:

- Fetch
- Axios

Let's see how they both work!



Fetch API

Fetch api is a built in promise-based api. Let's look at an example, here we took [json placeholder](https://jsonplaceholder.typicode.com/todos/1) API which is generally used for testing.

The beside code explains the basic syntax of fetching data from an api



```
const fetchRequest = async () => {  
  try {  
    const response = await fetch("https://jsonplaceholder.typicode.com/todos/1");  
    const data = await response.json();  
  
    console.log(data);  
  } catch (err) {  
    console.log(err)  
  }  
}
```

Axios

Axios is a promise-based HTTP Client for node.js and the browser. It is isomorphic (it can run in the browser and nodejs with the same codebase). Unlike Fetch api, axios is not a build-in api. So, we need to install it.

npm install axios

Then, import the axios in your file where you are going to fetch data.

```
import axios from "axios";

const axiosRequest = async () => {
  try {
    const { data } = await axios.get("https://jsonplaceholder.typicode.com/todos/1");

    console.log(data);
  } catch (err) {
    console.log(err)
  }
}
```

Axios or Fetch?

Axios provides an easy-to-use API in a compact package for most of your HTTP communication needs. However, if you prefer to stick with native APIs, nothing stops you from implementing Axios features.

It's perfectly possible to reproduce the key features of the Axios library using the `fetch()` method provided by web browsers. Ultimately, whether it's worth loading a client HTTP API depends on whether you're comfortable working with built-in APIs.



Network Call using Axios and JSON Server GET request

GET request is used when we want to get data from server



```
import axios from "axios";

const getData = async () => {
  try {
    const { data } = await axios.get("http://localhost:3000/users");
  } catch (err) {
    console.log(err);
  }
};
```


Network Call using Axios and JSON Server GET request

Every single data in json-server has an unique id. When we want to retrieve data with specific id, we can get by id like this:



```
import axios from "axios";

const getData = async (id: number) => {
  try {
    const { data } = await axios.get(`http://localhost:3000/users/${id}`);
  } catch (err) {
    console.log(err);
  }
};
```

Network Call using Axios and JSON Server POST request

With a POST request, we create a new user.

```
import axios from "axios";

interface IUser {
  id: number,
  first_name: string,
  last_name: string,
  email: string
}

const postData = async ({ first_name, last_name, email } : IUser) => {
  try {
    const { data } = await axios.put("http://localhost:3000/users", {
      first_name,
      last_name,
      email
    });
  } catch (err) {
    console.log(err);
  }
}
```

Network Call using Axios and JSON Server PUT request

In the following example we modify data with a PUT request to modify the user's email address with id 6.

```
import axios from "axios";

interface IUser {
  id: number,
  first_name: string,
  last_name: string,
  email: string
}

const putData = async ({ id, first_name, last_name, email } : IUser) => {
  try {
    const { data } = await axios.put(`http://localhost:3000/users/${id}`, {
      first_name,
      last_name,
      email
    });
  } catch (err) {
    console.log(err);
  }
}
```

Network Call using Axios and JSON Server DELETE request

In the following example, we show how to delete a user with id 1 with a DELETE request.



```
import axios from "axios";

const deleteData = async (id: number) => {
  try {
    const { data } = await axios.delete(`http://localhost:3000/users/${id}`);
  } catch (err) {
    console.log(err);
  }
};
```

Other Operation Json-server

That's actually some basic and most used feature in json-server, but there are few other features that you can use like sorting, operators, full text search etc. For further explanation you can read it at it's documentation or at <https://zetcode.com/javascript/jsonserver/>



Form Submission and Validation

Now we will make a simple form submission with formik and yup as a validator.

First install formik and yup into your react project

```
npm i formik yup
```

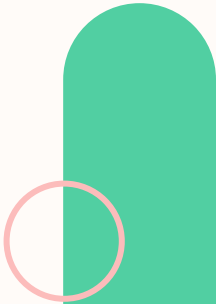


Form Submission and Validation

First, import all things that we need to make a form and validation.

```
import React from "react";
import { withFormik, Formik, Form, Field, FormikProps } from "formik";
import * as yup from 'yup'
```

- Formik needed as a wrap for our form
- Form needed as a wrap for input field
- Field needed to make an input for user
- ErrorMessage needed to show error message from validation
- Yup needed to make a validation schema



Form Submission and Validation

Make our validation schema using Yup. What we are doing in code below is making validation for email and password with some rules. For email, must be in email format and required (can't be empty). For password, must be 3 characters at minimum and required (can't be empty). For more information, you can go directly to the documentation from yup <https://www.npmjs.com/package/yup>

```
const LoginSchema = Yup.object().shape({
  email: Yup.string()
    .email("Invalid email address format")
    .required("Email is required"),
  password: Yup.string()
    .min(3, "Password must be 3 characters at minimum")
    .required("Password is required"),
});
```


Form Submission and Validation

Now make our component like code beside. Wrap our form inside `<Formik>`, with some props for Formik, like `initialValues`, `validationSchema`, and `onSubmit`.

```
import React from "react";
import { Formik, Form, Field, FormikProps } from "formik";
import * as Yup from "yup";

const LoginSchema = Yup.object().shape({
  email: Yup.string()
    .email("Invalid email address format")
    .required("email is required"),
  password: Yup.string()
    .min(3, "Password must be 3 characters at minimum")
    .required("password is required"),
});

interface MyFormValues {
  email: string;
  password: string;
}

const App: React.FC<object> = () => {
  const initialValues: MyFormValues = { email: "", password: "" };

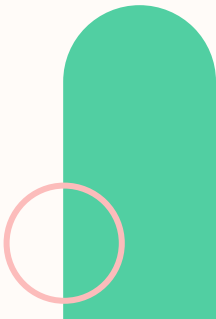
  return (
    <div style={{ margin: "40px" }}>
      <h1>My Example</h1>
      <Formik
        initialValues={initialValues}
        validationSchema={LoginSchema}
        onSubmit={values => {
          console.log(values);
        }}
      >
        {(props: FormikProps<MyFormValues>) => {
          const { values, errors, touched, handleChange } = props;

          console.log(props);
          return <Form></Form>;
        }}
      </Formik>
    </div>
  );
};

export default App;
```

Form Submission and Validation

- `initialValues`, is for making our field have default value, in our case we make our field, email and password, have default value an empty string.
- `validationSchema` is a schema for validation.
- `onSubmit`, is an event handler for what we want to do when we submit our form. In our case for make it simple, we just want to `console.log()` the value from email and password field.



Form Submission and Validation

Take a look at red square.

So inside `<Formik>` we will fill with a function that will return our form, This function has a params (we name it props) which contains an object that has many properties that we can use. You can `console.log(props)` for further information about it.

```
import React from "react";
import { Formik, Form, Field, FormikProps } from "formik";
import * as Yup from "yup";

const LoginSchema = Yup.object().shape({
  email: Yup.string()
    .email("Invalid email address format")
    .required("email is required"),
  password: Yup.string()
    .min(3, "Password must be 3 characters at minimum")
    .required("password is required"),
});

interface MyFormValues {
  email: string;
  password: string;
}

const App: React.FC<object> = () => {
  const initialValues: MyFormValues = { email: "", password: "" };

  return (
    <div style={{ margin: "40px" }}>
      <h1>My Example</h1>
      <Formik
        initialValues={initialValues}
        validationSchema={LoginSchema}
        onSubmit={(values) => {
          console.log(values);
        }}
      >
        {(props: FormikProps<MyFormValues>) => {
          const { values, errors, touched, handleChange } = props;

          console.log(props);
          return <Form></Form>;
        }}
      </Formik>
    </div>
  );
};

export default App;
```

Form Submission and Validation

This is the content of the function, where we will fill in our form component.

```
<Form>
  <div>
    <label htmlFor="email">Email :</label>
    <Field
      type="email"
      name="email"
      onChange={handleChange}
      value={values.email}
    />
    {touched.email && errors.email ? (
      <div style={{ color: "red" }}>{errors.email}</div>
    ) : null}
  </div>
  <div>
    <label htmlFor="password">Password :</label>
    <Field
      type="password"
      name="password"
      onChange={handleChange}
      value={values.password}
    />
    {touched.password && errors.password ? (
      <div style={{ color: "red" }}>{errors.password}</div>
    ) : null}
  </div>
  <button type="submit">Submit</button>
</Form>
```

Form Submission and Validation

And that is our simple form submission and validation using formik dan yup. Just run our application by typing npm start in terminal.

When we fill in the email or password form with a format that does not match the schema we created earlier, an error message will appear.

My Example

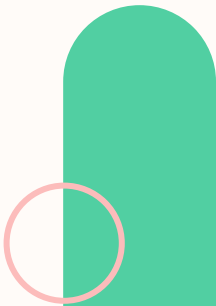
Email :

email is required

Password :

password is required

Submit



Thank You!

