Purwadhika
Digital Technology School

Full Stack AI Software Development

Network Requests and Form Validation

Job Connector Program

Outline

Introduction to Network Requests
Understanding how front-end applications communicate with servers, send and receive data, and handle asynchronous operations.

Calling APIs with Fetch and Axios
Learning how to use the native **Fetch API** and **Exploring Axios** as an alternative to Fetch — understanding its configuration options, interceptors, and cleaner syntax for GET/POST requests.

Connecting to Backendless
Integrating a Backendless backend into your React project — setting up API endpoints, connecting the app, and performing CRUD operations with real backend data.

Form Handling with Formik
Building and managing complex forms easily using Formik, including handling input state, submission logic, and form-level validation.

What is Network Call?

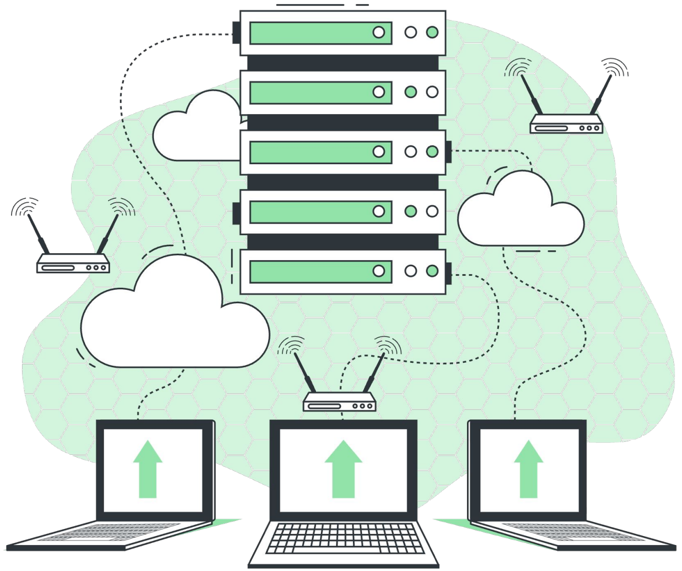
Network calls, in web development, are how the **front-end** (what users see) **communicates** with the **backend** (the server) to exchange data.

Think of it like ordering food at a restaurant — the front-end makes a **request** (asks for data), and the backend sends a **response** (the data).

For example:

- The user clicks "Show Products."
- The front-end sends a **GET request** to the backend API, e.g., *https://api.example.com/products.*
- The backend fetches the data and **sends it back as JSON**.
- The front-end displays the product list on the page.

In short, network calls let your app talk to the server so it can load, send, or update data dynamically.



What is Network Call?

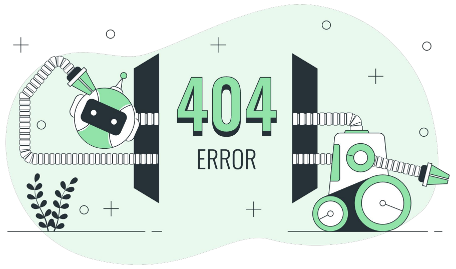
In the context of a React application or any other web application, network calls typically involve making HTTP requests to a server or an external API (Application Programming Interface). These requests can be of various types, such as:

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?

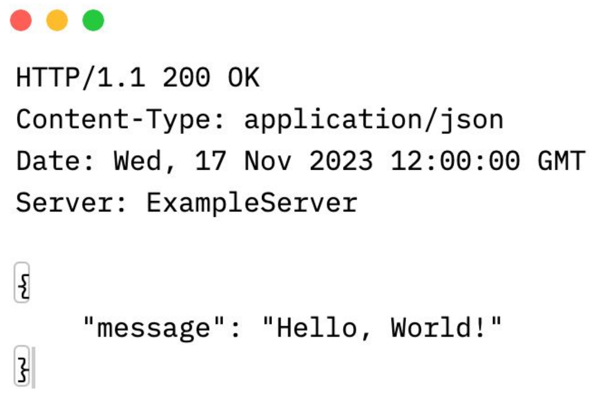
In this example:

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.

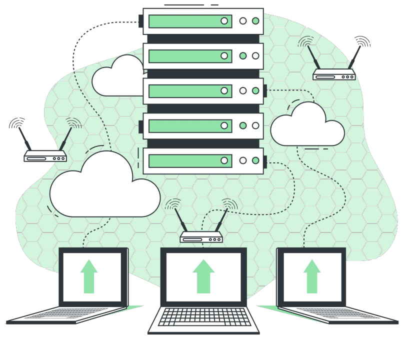


```
HTTP/1.1 200 OK
Content-Type: application/json
Date: Wed, 17 Nov 2023 12:00:00 GMT
Server: ExampleServer

{
    "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



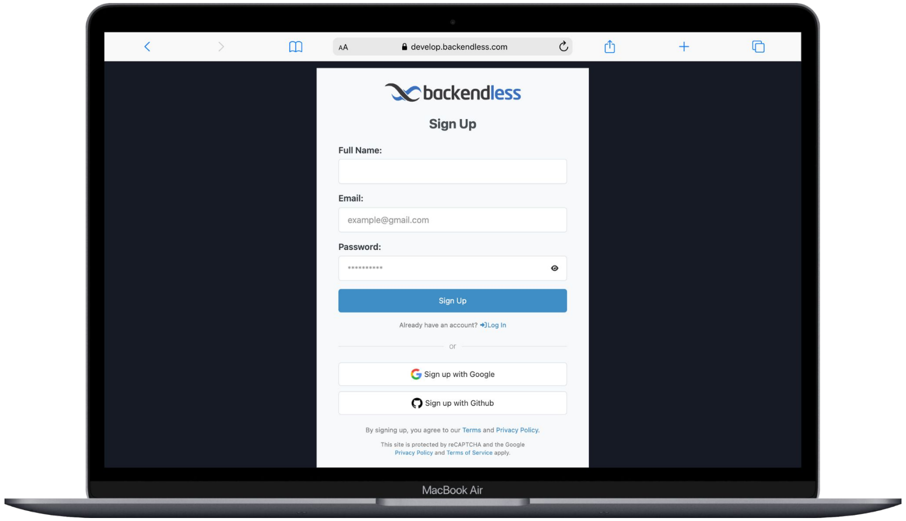
BACKENDLESS

What is Backendless?

- Backendless is a **Backend-as-a-Service (BaaS)** platform that provides various services such as database, hosting, user authentication, and API management.
- It allows developers to focus on application development without having to build a backend from scratch.

Backendless Preparation

- **Sign Up and Set Up Backendless**:
  - Sign up at Backendless.
  - Create a new application and note down the **App ID** or **REST API Key** and **API URL**.



develop.backendless.com

backendless

Sign Up

Full Name:

Email:

example@gmail.com

Password:

Sign Up

Already have an account? Log In

or

Sign up with Google

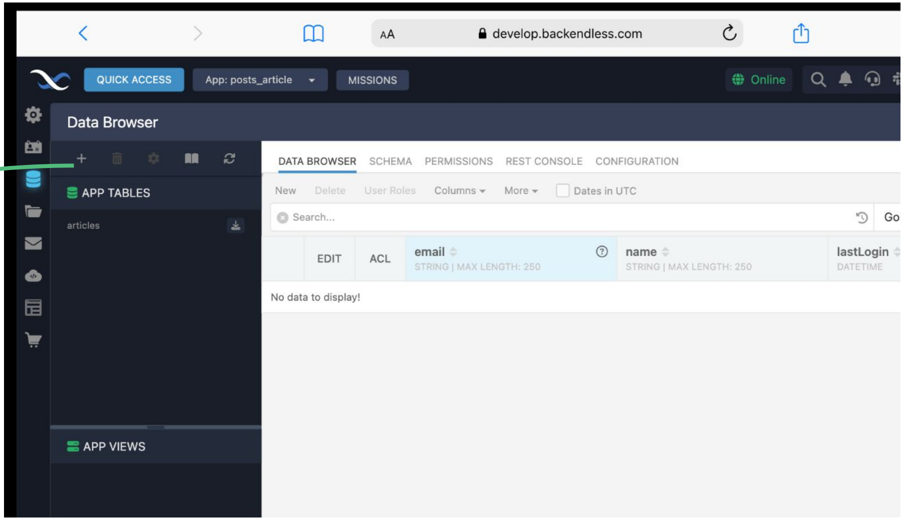
Sign up with Github

By signing up, you agree to our Terms and Privacy Policy.

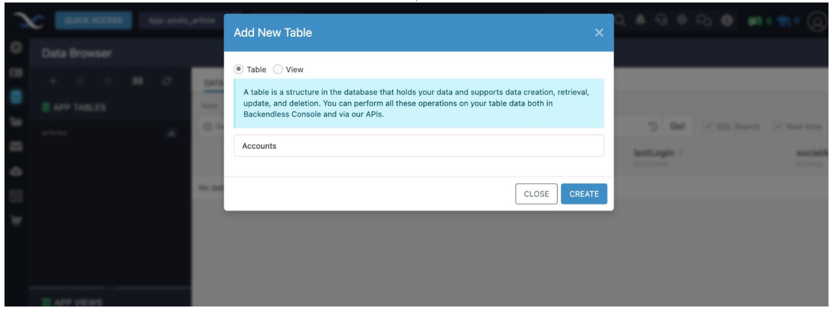
This site is protected by reCAPTCHA and the Google Privacy Policy and Terms of Service apply.

Backendless Preparation

- **Configure Backendless Database**:
  - Add a data table via the Backendless Console (e.g., a "Accounts" table).
  - Add columns as needed (e.g., `id`, `email`, `password`, `isVerified`).



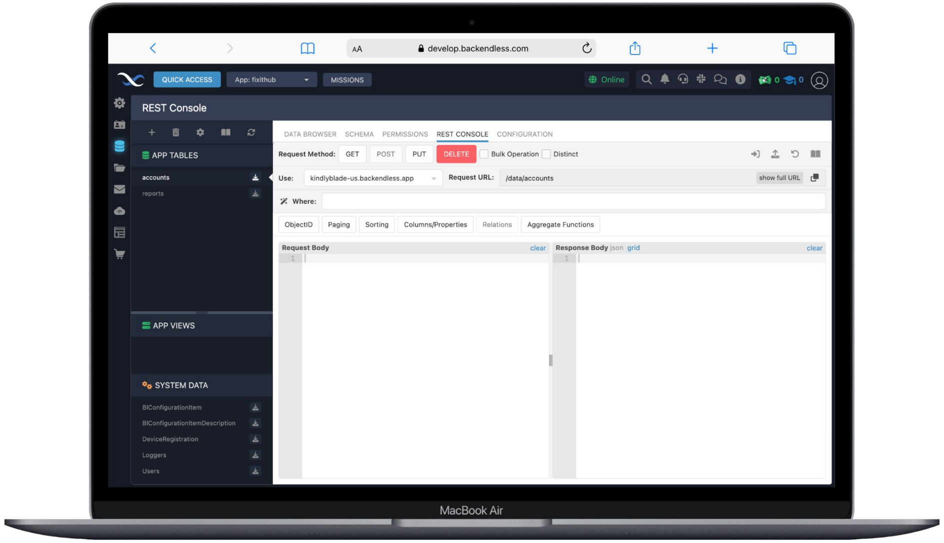
No data to display!



A table is a structure in the database that holds your data and supports data creation, retrieval, update, and deletion. You can perform all these operations on your table data both in Backendless Console and via our APIs.

Backendless REST Console

- We can test backendless API at REST Console tab menu like this

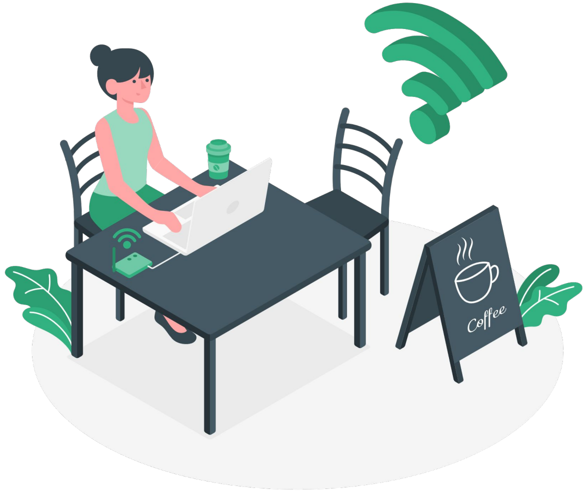


Network Call using Fetch and Axios

Network call can be done by two ways using:

- Fetch
- Axios

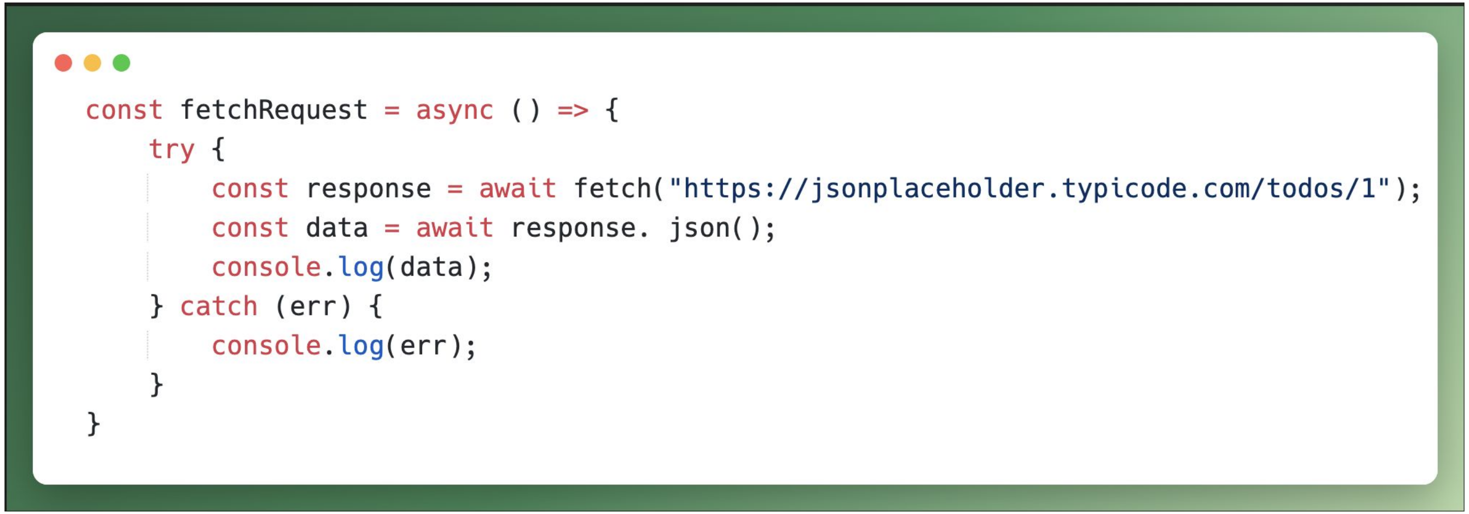
Let’s see how they both work!



Fetch API

Fetch api is a built in promise-based api. Let's look at an example, here we took json placeholder API which is generally used for testing.

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



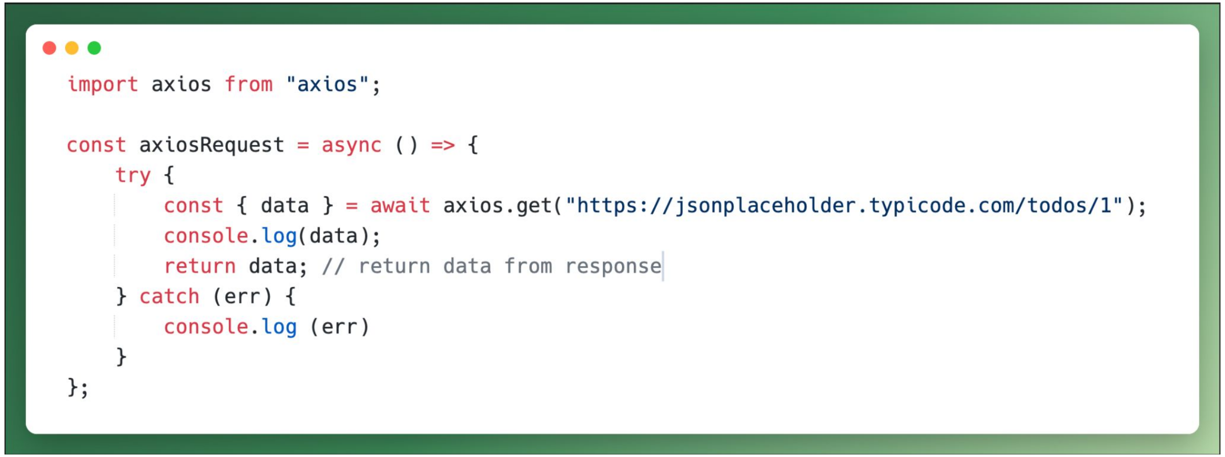
```
const fetchRequest = async () => {
    try {
        const response = await fetch("https://jsonplaceholder.typicode.com/todos/1");
        const data = await response. json();
        console.log(data);
    } catch (err) {
        console.log(err);
    }
}
```

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.

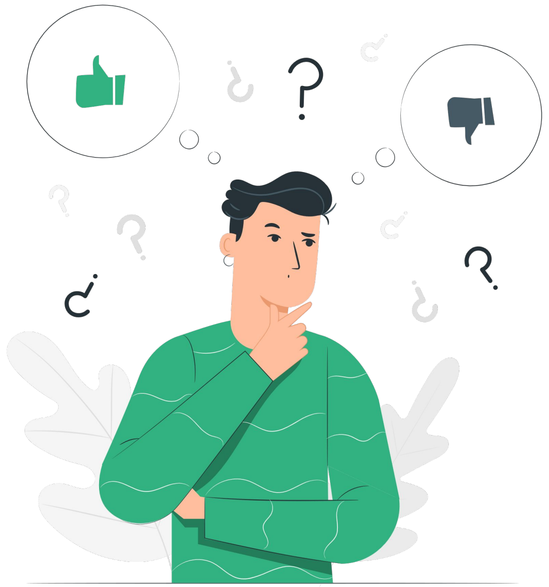


```
import axios from "axios";

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

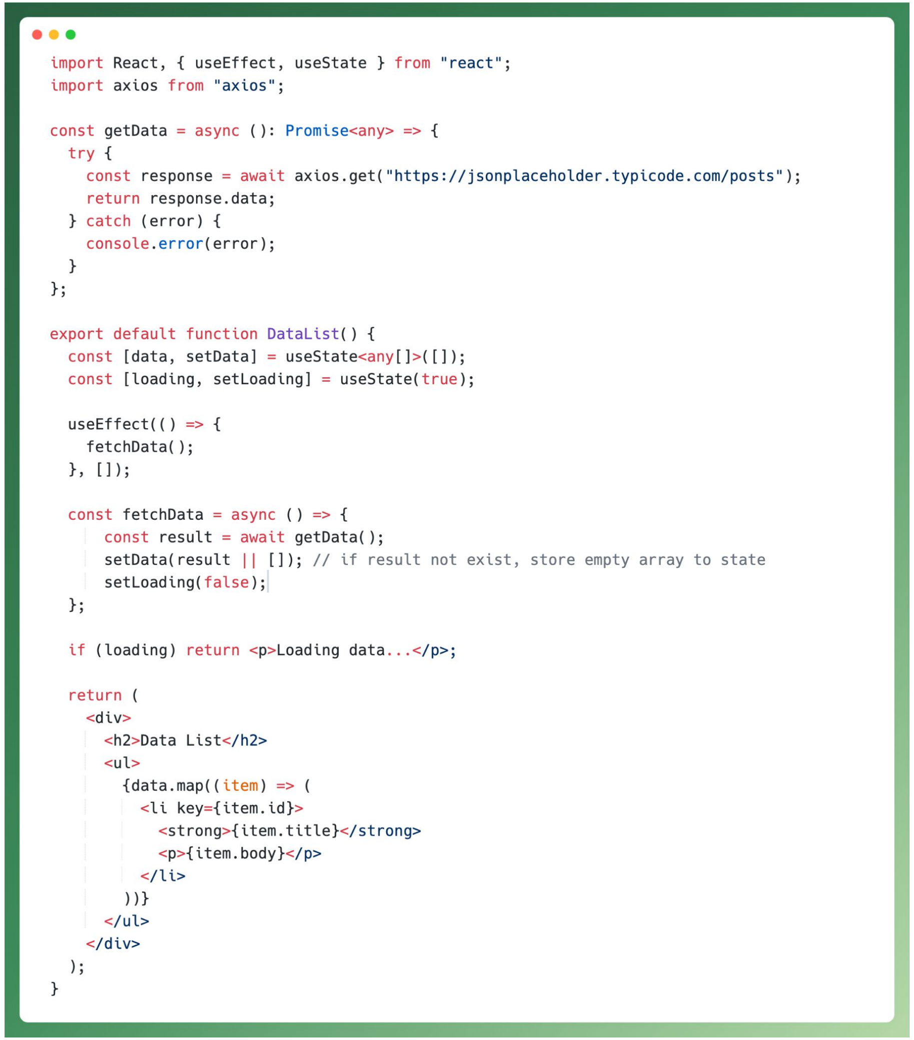
It’s perfectly possible to reproduce the key features of the Axios library using the fetch() method provided by web browsers. Ultimately, whether it’s worth loading a client HTTP API depends on whether you’re comfortable working with built-in APIs.



Network Call using Axios : GET request

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

- **getData()** – an asynchronous function that uses Axios to fetch data from an API and return the result.
- **useState()** – used to store the data and the loading status.
- **useEffect()** – runs once when the component is first rendered to call **getData()** and update the state with the fetched data.
- **Render Section** – while data is loading, it shows a “Loading...” message. Once the data is fetched, it displays a list of items from the API.

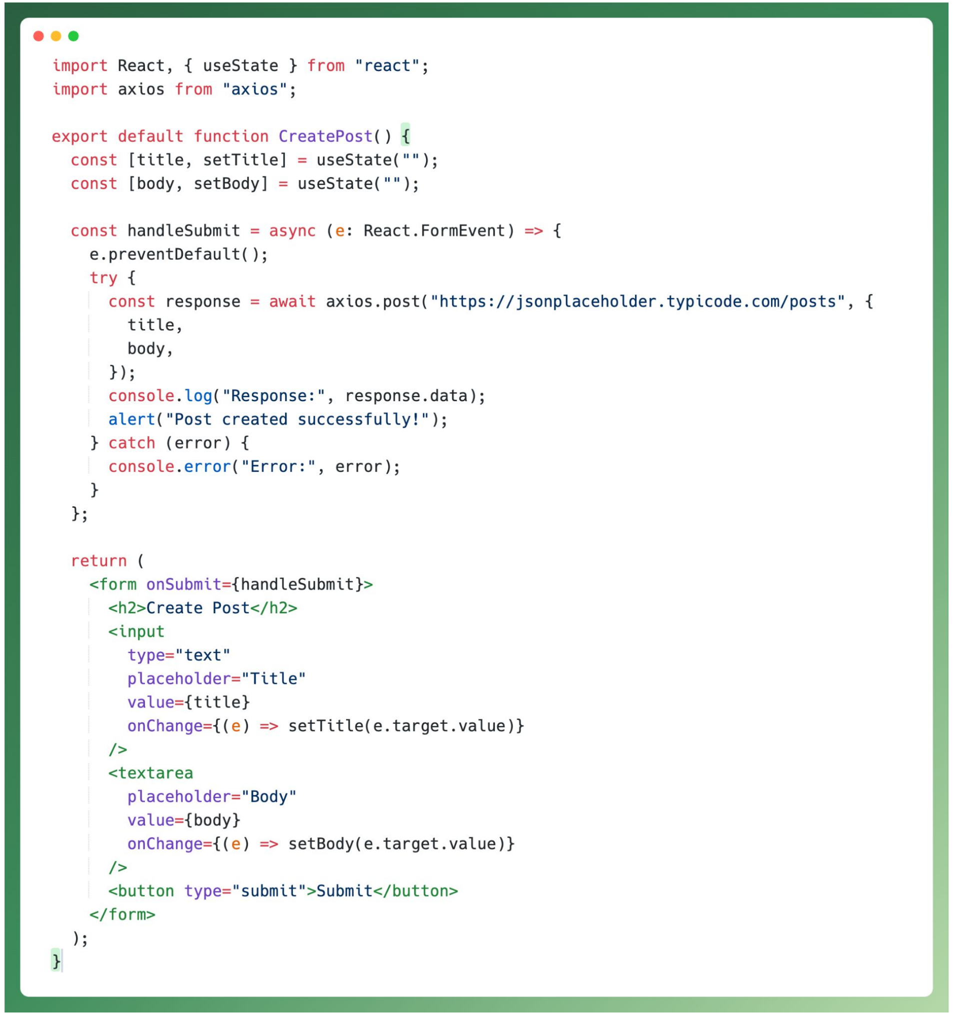


```
import React, { useEffect, useState } from "react";
import axios from "axios";

const getData = async (): Promise<any> => {
  try {
    const response = await axios.get("https://jsonplaceholder.typicode.com/posts");
    return response.data;
  } catch (error) {
    console.error(error);
  }
};

export default function DataList() {
  const [data, setData] = useState<any[]>([]);
  const [loading, setLoading] = useState(true);

  useEffect(() => {
    fetchData();
  }, []);

  const fetchData = async () => {
    const result = await getData();
    setData(result || []); // if result not exist, store empty array to state
    setLoading(false);
  };

  if (loading) return <p>Loading data...</p>;

  return (
    <div>
      <h2>Data List</h2>
      <ul>
        {data.map((item) => (
          <li key={item.id}>
            <strong>{item.title}</strong>
            <p>{item.body}</p>
          </li>
        ))}
      </ul>
    </div>
  );
}
```

Network Call using Axios : POST request

With a POST request, we create a new user.

- Uses **Axios POST** to send form data to the API.
- **useState** stores form inputs.
- When submitted, the data is sent, and a success message appears.

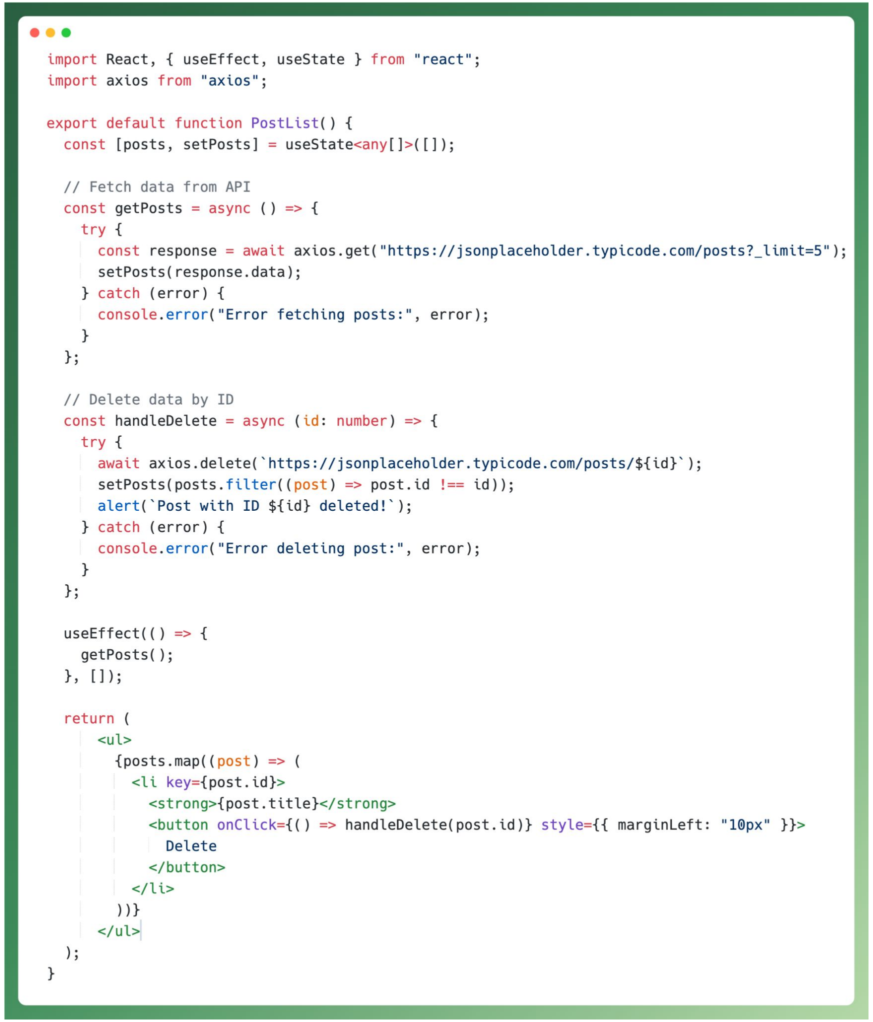


```
import React, { useState } from "react";
import axios from "axios";

export default function CreatePost() {
  const [title, setTitle] = useState("");
  const [body, setBody] = useState("");

  const handleSubmit = async (e: React.FormEvent) => {
    e.preventDefault();
    try {
      const response = await axios.post("https://jsonplaceholder.typicode.com/posts", {
        title,
        body,
      });
      console.log("Response:", response.data);
      alert("Post created successfully!");
    } catch (error) {
      console.error("Error:", error);
    }
  };

  return (
    <form onSubmit={handleSubmit}>
      <h2>Create Post</h2>
      <input
        type="text"
        placeholder="Title"
        value={title}
        onChange={(e) => setTitle(e.target.value)}
      />
      <textarea
        placeholder="Body"
        value={body}
        onChange={(e) => setBody(e.target.value)}
      />
      <button type="submit">Submit</button>
    </form>
  );
}
```

Network Call using Axios : DELETE request

In the following example, we show how to delete a posts by selected ID.

- **getPosts()** → fetches post data when the component loads.
- **handleDelete()** → sends a **DELETE request** to the API and removes the item from the list.
- The UI updates automatically because we remove the deleted post from state (**setPosts**).



```
import React, { useEffect, useState } from "react";
import axios from "axios";

export default function PostList() {
  const [posts, setPosts] = useState<any[]>([]);

  // Fetch data from API
  const getPosts = async () => {
    try {
      const response = await axios.get("https://jsonplaceholder.typicode.com/posts?_limit=5");
      setPosts(response.data);
    } catch (error) {
      console.error("Error fetching posts:", error);
    }
  };

  // Delete data by ID
  const handleDelete = async (id: number) => {
    try {
      await axios.delete(`https://jsonplaceholder.typicode.com/posts/${id}`);
      setPosts(posts.filter((post) => post.id !== id));
      alert(`Post with ID ${id} deleted!`);
    } catch (error) {
      console.error("Error deleting post:", error);
    }
  };

  useEffect(() => {
    getPosts();
  }, []);

  return (
      <ul>
        {posts.map((post) => (
          <li key={post.id}>
            <strong>{post.title}</strong>
            <button onClick={() => handleDelete(post.id)} style={{ marginLeft: "10px" }}>
              Delete
            </button>
          </li>
        ))}
      </ul>
  );
}
```

Form Submission and Validation

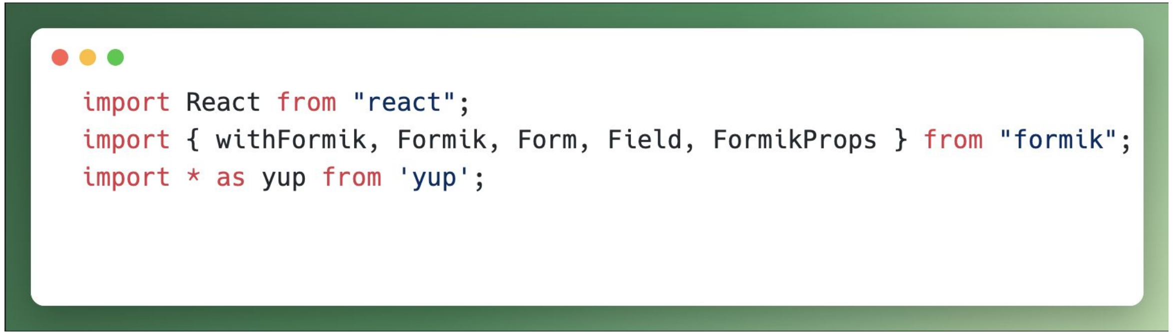
Now we will make a simple form submission with formik and yup as a validator.

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.

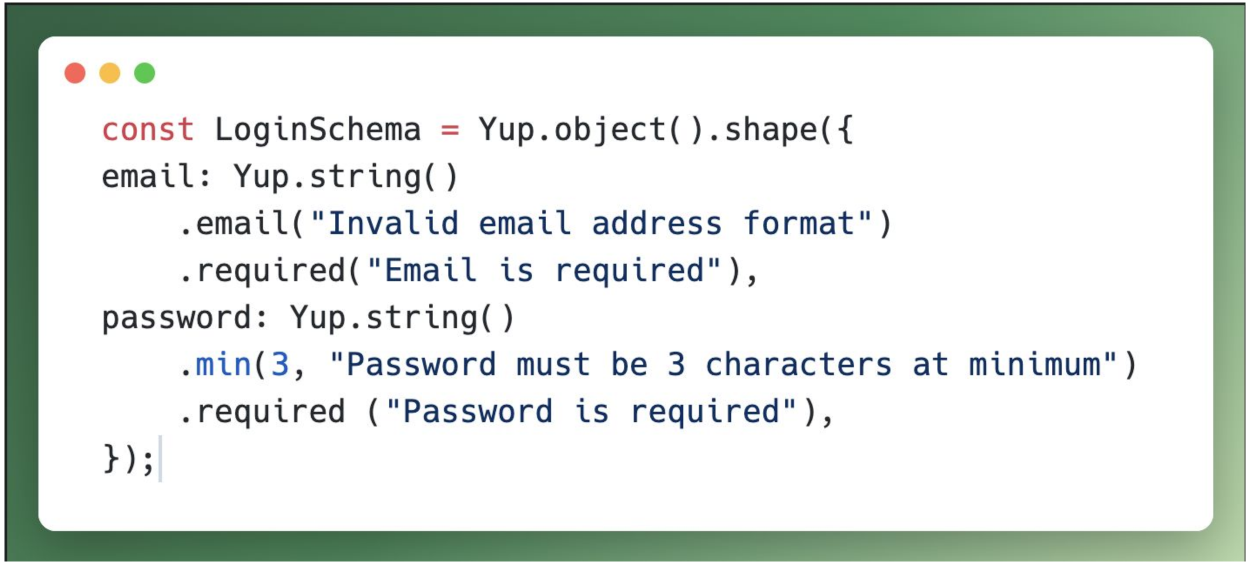


```
import React from "react";
import { withFormik, Formik, Form, Field, FormikProps } from "formik";
import * as yup from 'yup';
```

- 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

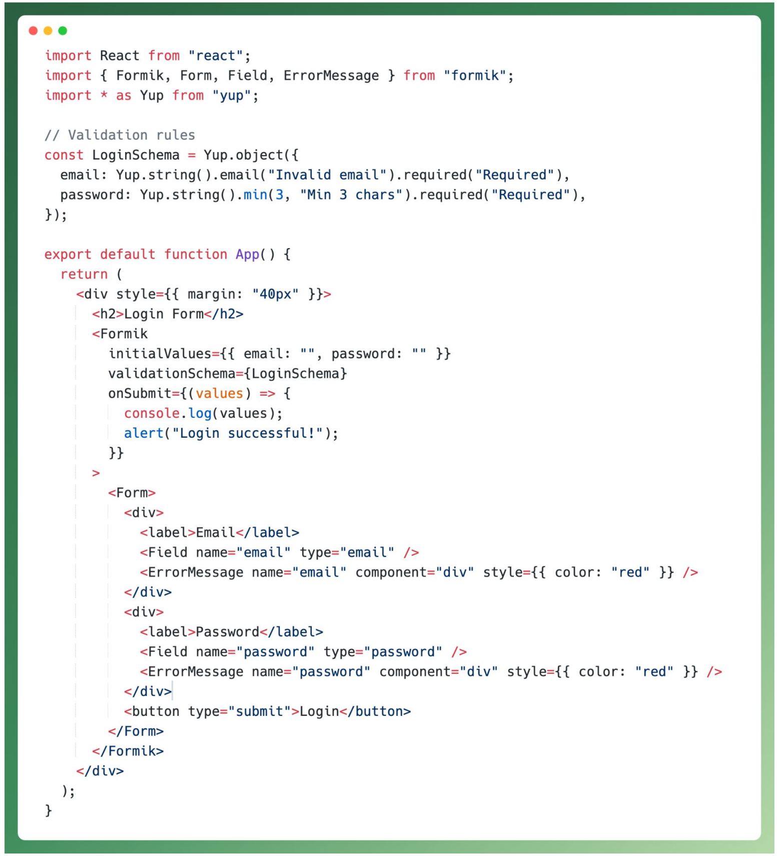
Now, let's create our component as shown in the code. We wrap the entire form inside the **<Formik>** component. Formik helps us handle form state, validation, and submission in a simpler and more declarative way — without manually managing **useState** or event handlers for every input.

Inside <Formik>, we provide several important props:

- **initialValues**: This is an object that defines the initial state of all form fields.
- **validationSchema**: This prop defines the validation rules for each field using a validation library such as Yup.
- **onSubmit**: This function runs when the user submits the form.

By wrapping our form in **<Formik>** and defining **initialValues**, **validationSchema**, and **onSubmit**, we let Formik take care of:

- Managing form state automatically
- Validating inputs according to rules
- Handling submission efficiently

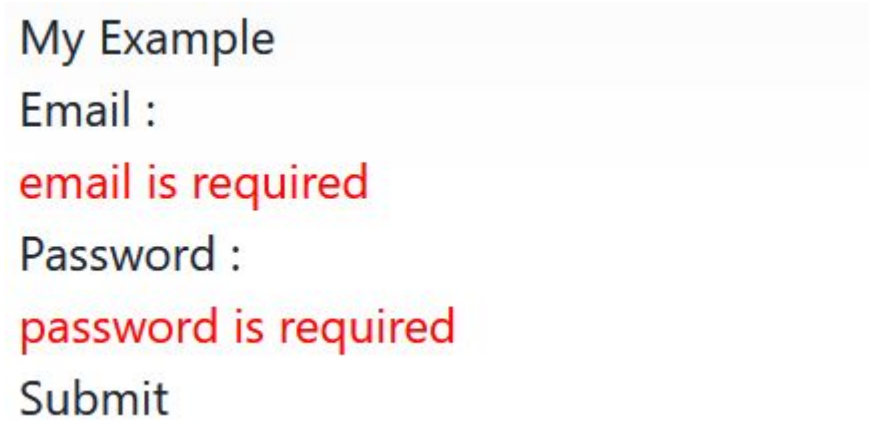


```
import React from "react";
import { Formik, Form, Field, ErrorMessage } from "formik";
import * as Yup from "yup";

// Validation rules
const LoginSchema = Yup.object({
  email: Yup.string().email("Invalid email").required("Required"),
  password: Yup.string().min(3, "Min 3 chars").required("Required"),
});

export default function App() {
  return (
    <div style={{ margin: "40px" }}>
      <h2>Login Form</h2>
      <Formik
        initialValues={{ email: "", password: "" }}
        validationSchema={LoginSchema}
        onSubmit={(values) => {
          console.log(values);
          alert("Login successful!");
        }}
      >
        <Form>
          <div>
            <label>Email</label>
            <Field name="email" type="email" />
            <ErrorMessage name="email" component="div" style={{ color: "red" }} />
          </div>
          <div>
            <label>Password</label>
            <Field name="password" type="password" />
            <ErrorMessage name="password" component="div" style={{ color: "red" }} />
          </div>
          <button type="submit">Login</button>
        </Form>
      </Formik>
    </div>
  );
}
```

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
```

Exercise

- Completing app to have Sign In and Sign Up feature and store data at Backendless
- Don’t forget to use form validation

Thank you