Lecture 3.2

Working with Data

Topics

- Handling Data
  - Axios API
  - Axios Config Options
  - Fetch Revisted

Working with Data

React is great at displaying your data in a hierarchical component view. But how do your components get the data?

Without question, some developers prefer Axios over built-in APIs for its ease of use. But many overestimate the need for such a library.

The fetch() API is perfectly capable of reproducing the key features of Axios, and it has the added advantage of being readily available in all modern browsers.

Axios

Axios != Delta Force

Axios - REST API

- Axios is a lightweight HTTP client based on the $http service within Angular.js v1.x and similar to Fetch API
- Axios is promise-based, and we can take advantage of async and await for asynchronous code.
- We can intercept and cancel requests, and there is built-in CORS (Cross Site Request forgery) protection
- We can include Axios in our React project by installing it with npm

```
npm install axios --save
```

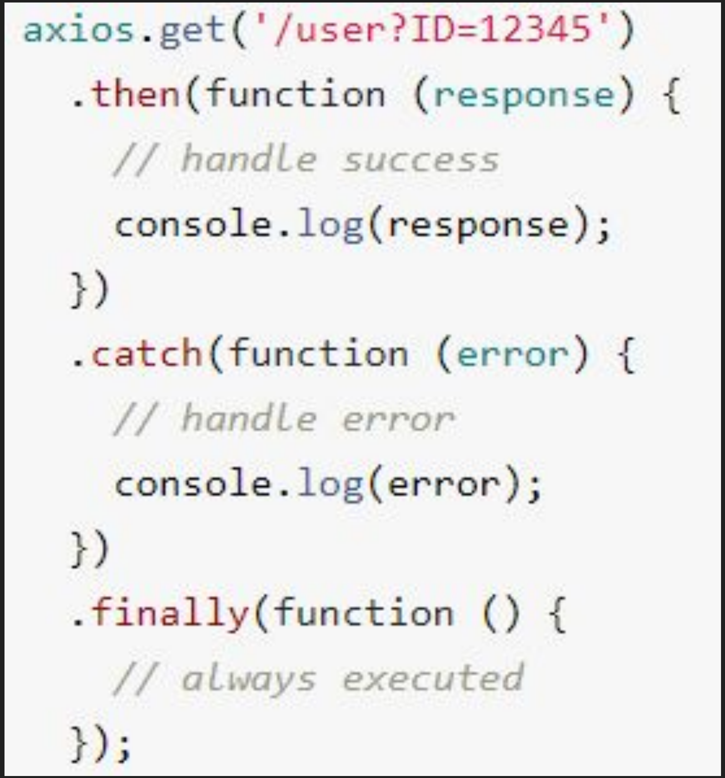
Axios HTTP Verbs

- Axios offers methods for all the HTTP verbs, which are less popular but still used:
  - axios.get()
  - axios.put()
  - axios.delete()
  - axios.patch()
  - axios.options()
  - axios.head()

- If we do not want to use async-await, we can make use of the promise to resolve or handle the error

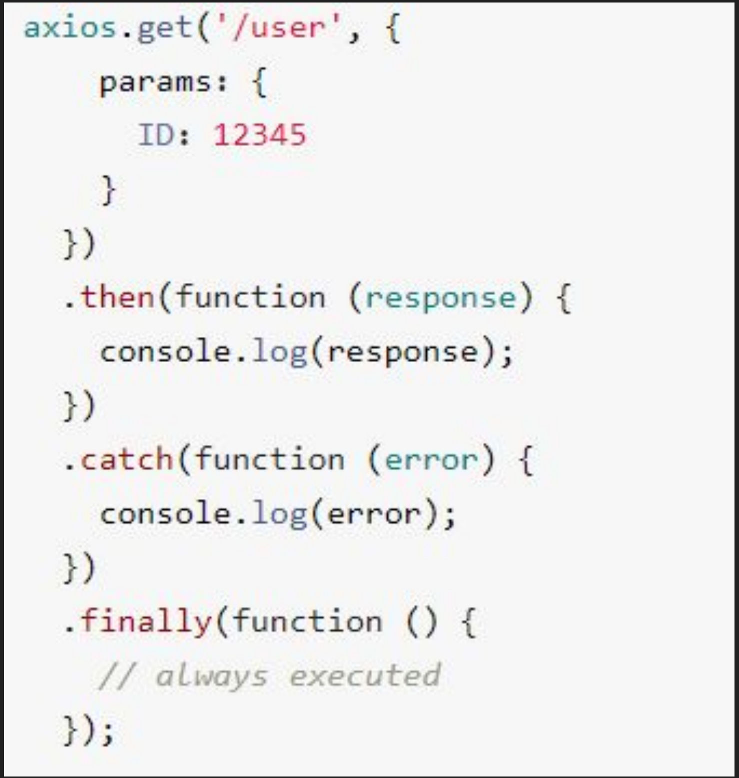
Axios GET Request

Request with ID in Query String



```
axios.get('/user?ID=12345')
  .then(function (response) {
    // handle success
    console.log(response);
  })
  .catch(function (error) {
    // handle error
    console.log(error);
  })
  .finally(function () {
    // always executed
  });
```

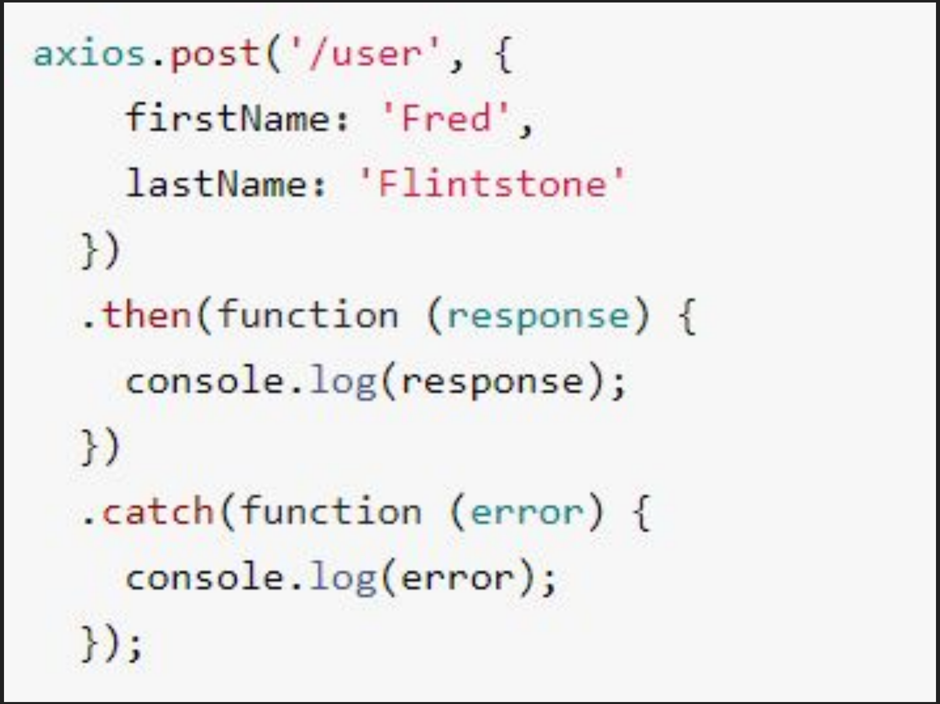
Request with ID as params



```
axios.get('/user', {
    params: {
      ID: 12345
    }
  })
  .then(function (response) {
    console.log(response);
  })
  .catch(function (error) {
    console.log(error);
  })
  .finally(function () {
    // always executed
  });
```

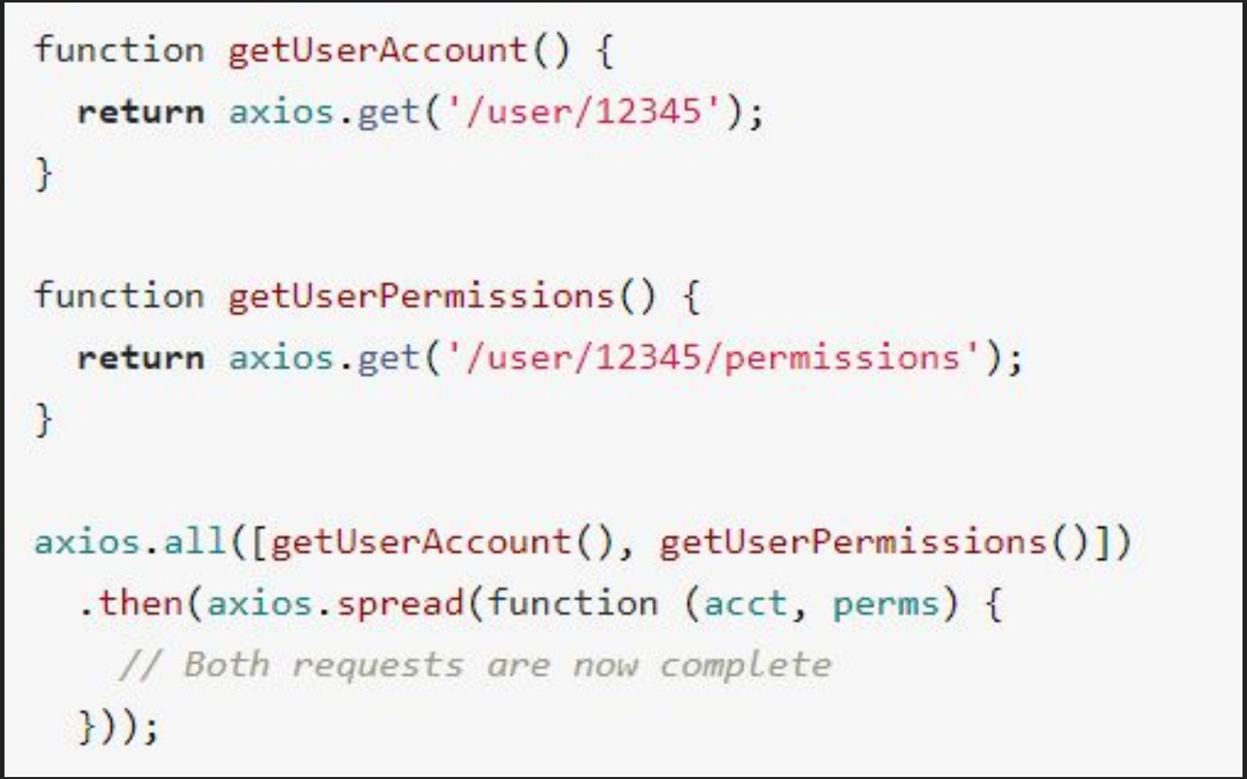
Axios POST Request

Axios Post Request with JSON data object



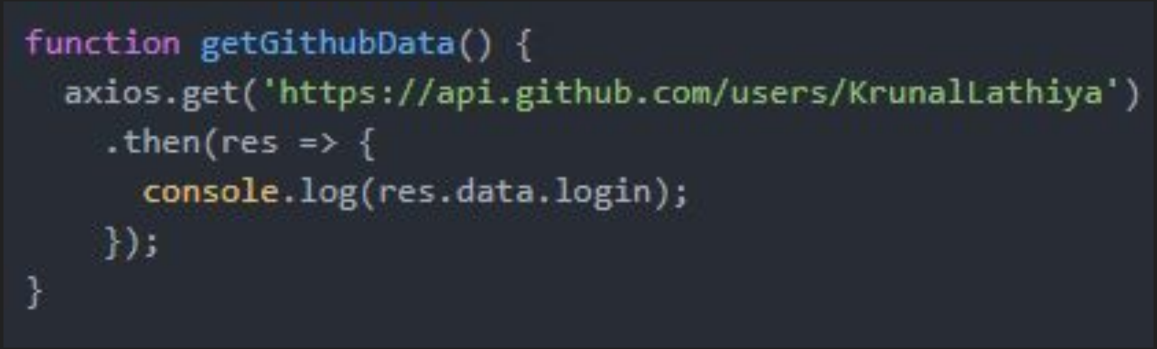
```
axios.post('/user', {
    firstName: 'Fred',
    lastName: 'Flintstone'
  })
  .then(function (response) {
    console.log(response);
  })
  .catch(function (error) {
    console.log(error);
  });
```

Axios Multiple Concurrent Requests



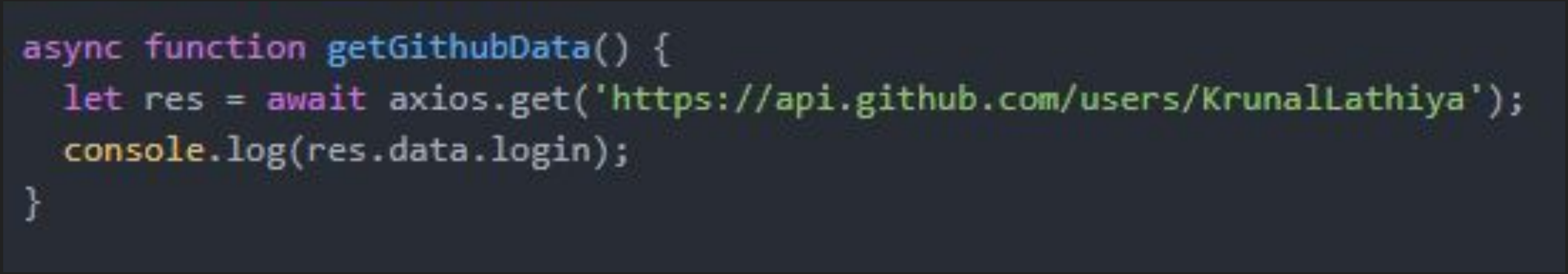
```
function getUserAccount() {
  return axios.get('/user/12345');
}

function getUserPermissions() {
  return axios.get('/user/12345/permissions');
}

axios.all([getUserAccount(), getUserPermissions()])
  .then(axios.spread(function (acct, perms) {
    // Both requests are now complete
  }));
```

Axios Promises vs Async

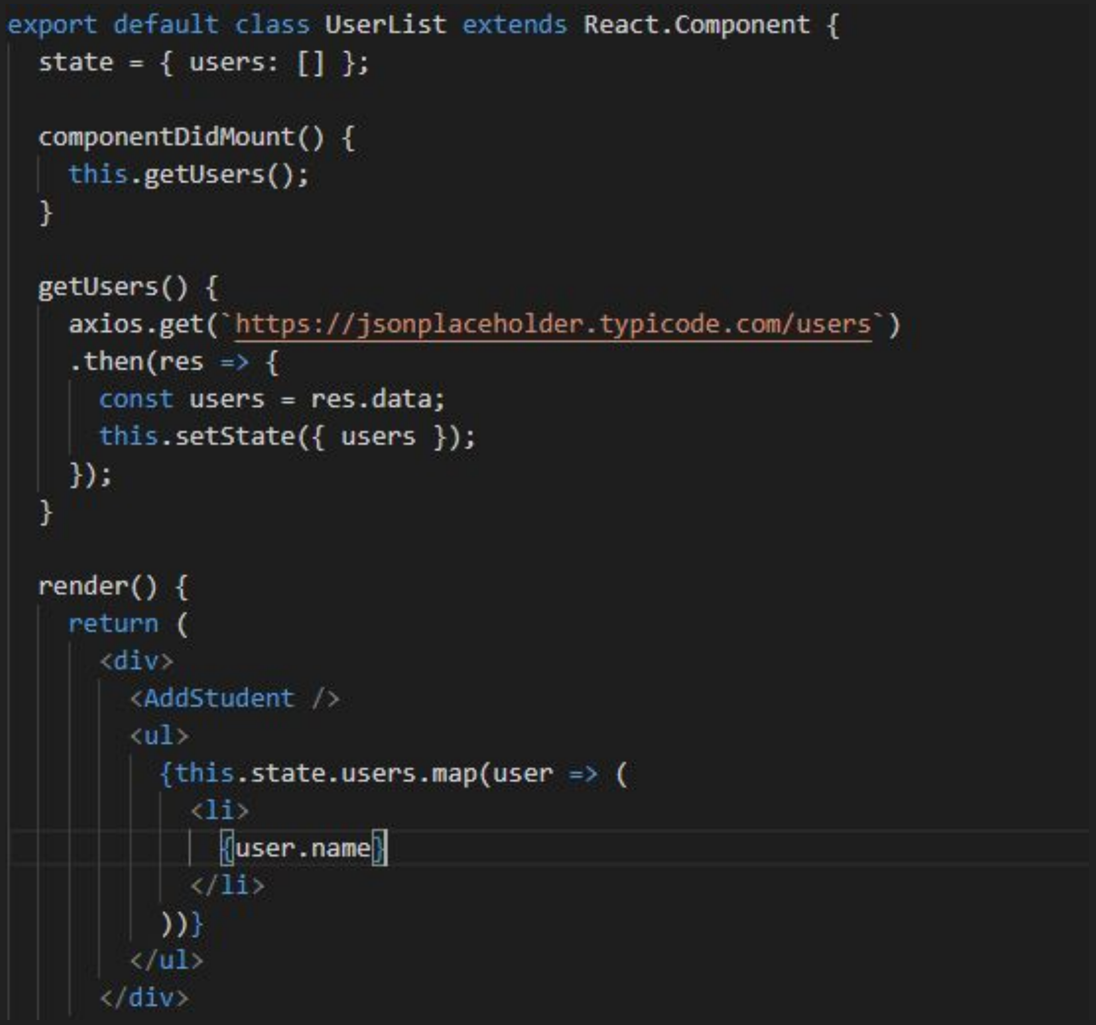


```
function getGithubData() {
  axios.get('https://api.github.com/users/KrunalLathiya')
    .then(res => {
      console.log(res.data.login);
    });
}
```

Axios async/await



```
async function getGithubData() {
  let res = await axios.get('https://api.github.com/users/KrunalLathiya');
  console.log(res.data.login);
}
```



```
export default class UserList extends React.Component {
  state = { users: [] };

  componentDidMount() {
    this.getUsers();
  }

  getUsers() {
    axios.get(`https://jsonplaceholder.typicode.com/users`)
    .then(res => {
      const users = res.data;
      this.setState({ users });
    });
  }

  render() {
    return (
      <div>
        <AddStudent />
        <ul>
          {this.state.users.map(user => (
            <li>
              {user.name}
            </li>
          ))}
        </ul>
      </div>
```

Integration of Axios with React

- Use component life cycle event to call GET Request
- Updated state with response data
- Using array.proto.map to display the state data in render()

Axios Config Options

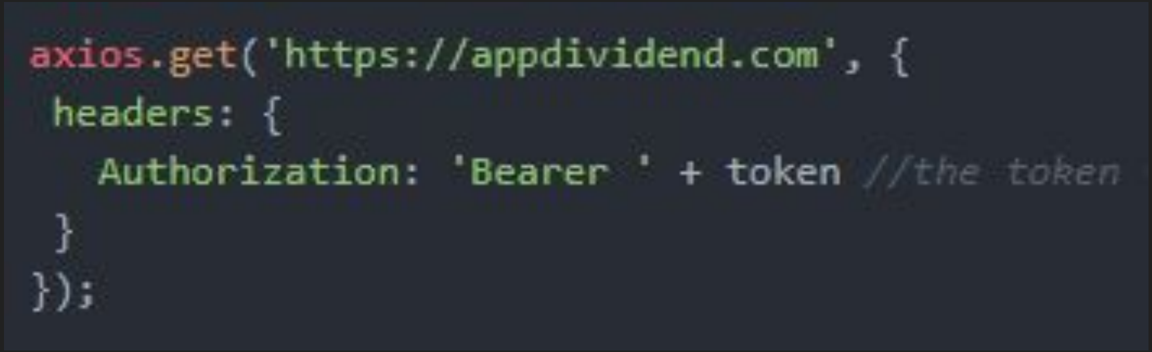
Axios Request Config/Options

The following options can be passed in the request:

- headers: It is an object of key-value pairs to be sent as headers.
- params: an object of key/value pairs that will be serialized and appended to the URL as a query string.
- responseType: if we want to get the response in a format other than JSON then we can set this property to arraybuffer, blob, document, text, or stream.
- auth: Auth option is used when we need to pass an object with the username and password credentials for HTTP Basic authentication on the request.

Axios GET Request with JWT Token

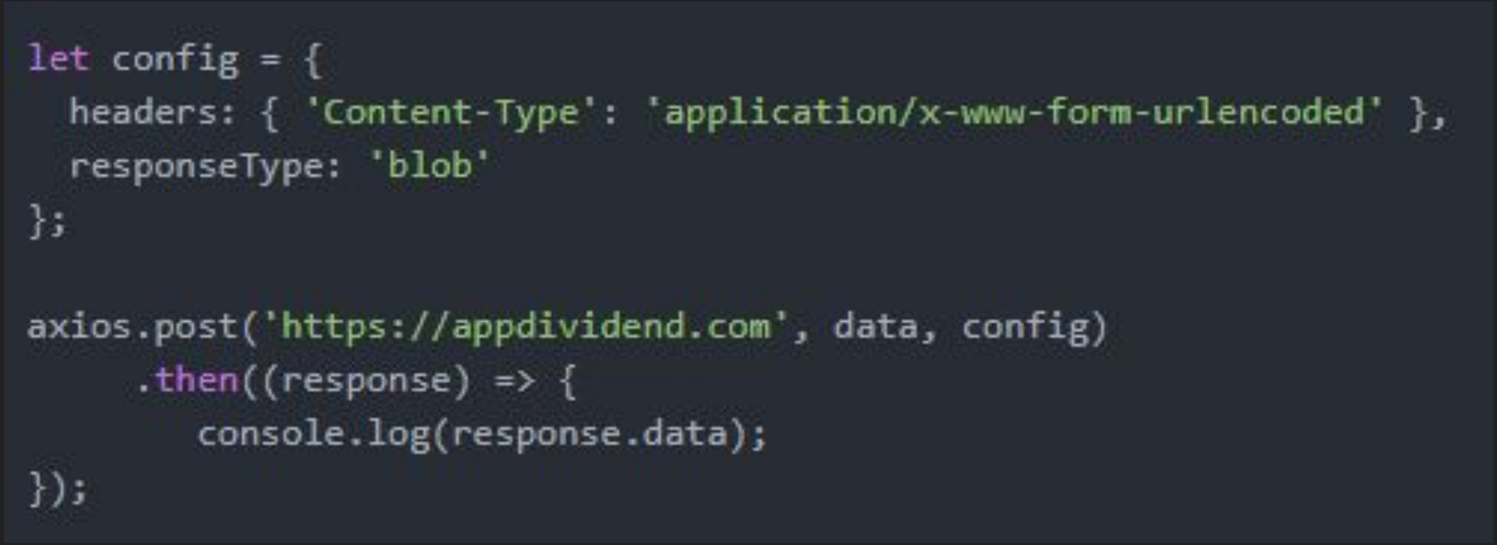
Axios with Authorization JWT Token in the GET Request



```
axios.get('https://appdividend.com', {
 headers: {
   Authorization: 'Bearer ' + token //the token
 }
});
```

Axios POST Request with Config Options

Axios POST Request with the following config options



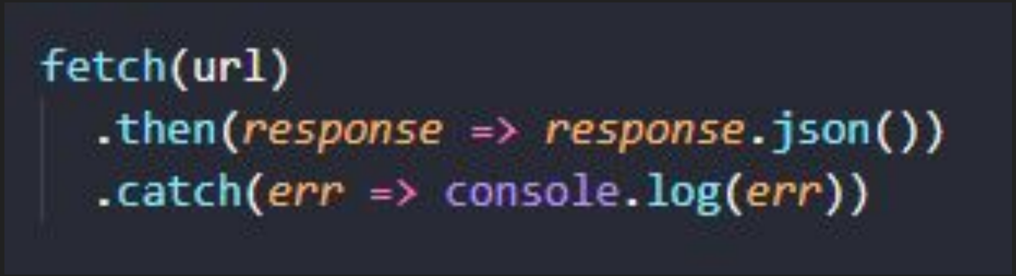
```
let config = {
  headers: { 'Content-Type': 'application/x-www-form-urlencoded' },
  responseType: 'blob'
};

axios.post('https://appdividend.com', data, config)
     .then((response) => {
        console.log(response.data);
});
```

Fetch Revisited

Fetch

Executing fetch() starts a request and returns a promise. When the request completes, the promise is resolved with the Response object. If the request fails due to some network problems, the promise is rejected.

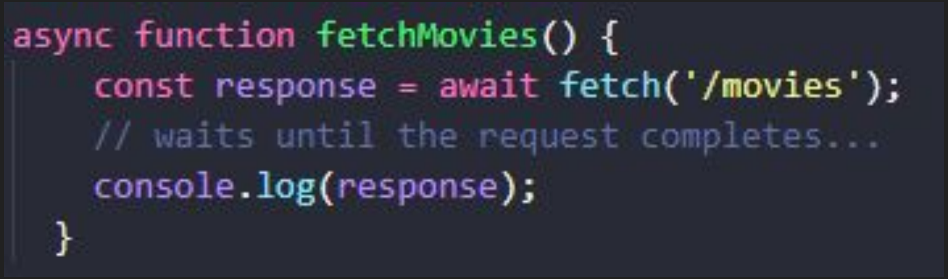
If there is no problem connecting to the server and the server responded a status code. This status code could be 200s, 400s or 500s.



```
fetch(url)
  .then(response => response.json())
  .catch(err => console.log(err))
```

1. async/await with fetch()

async/await syntax fits great with fetch() because it simplifies the work with promises..



```
async function fetchMovies() {
    const response = await fetch('/movies');
    // waits until the request completes...
    console.log(response);
  }
```

fetchMovies() is an asynchronous function since it’s marked with the async keyword.

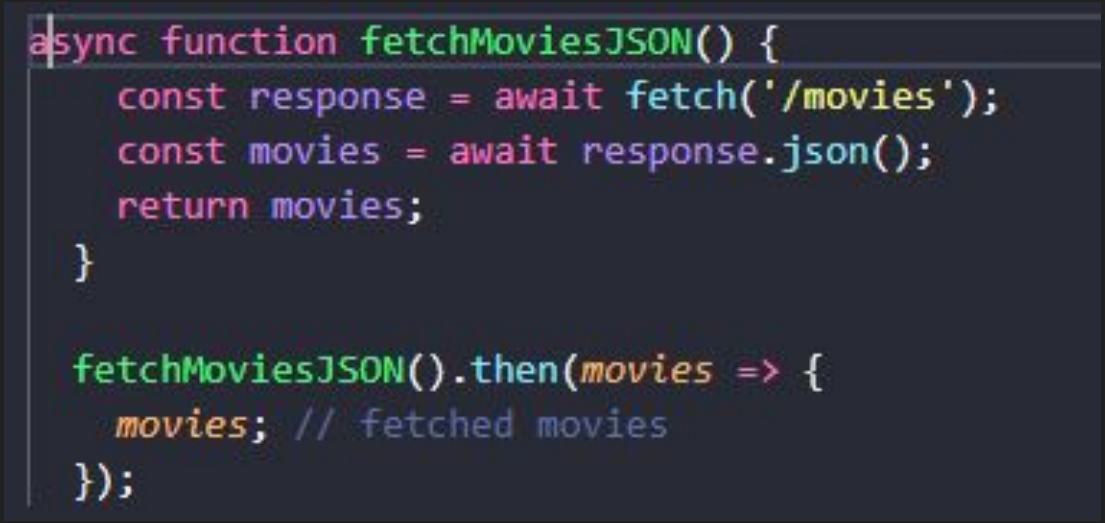
await fetch('/movies') starts an HTTP request to '/movies' URL. Because the await keyword is present, the asynchronous function is paused until the request completes.

When the request completes, response is assigned with the response object of the request.

2. Fetching JSON with fetch()

The Response object, returned by the await fetch(), is a generic placeholder for multiple data formats.

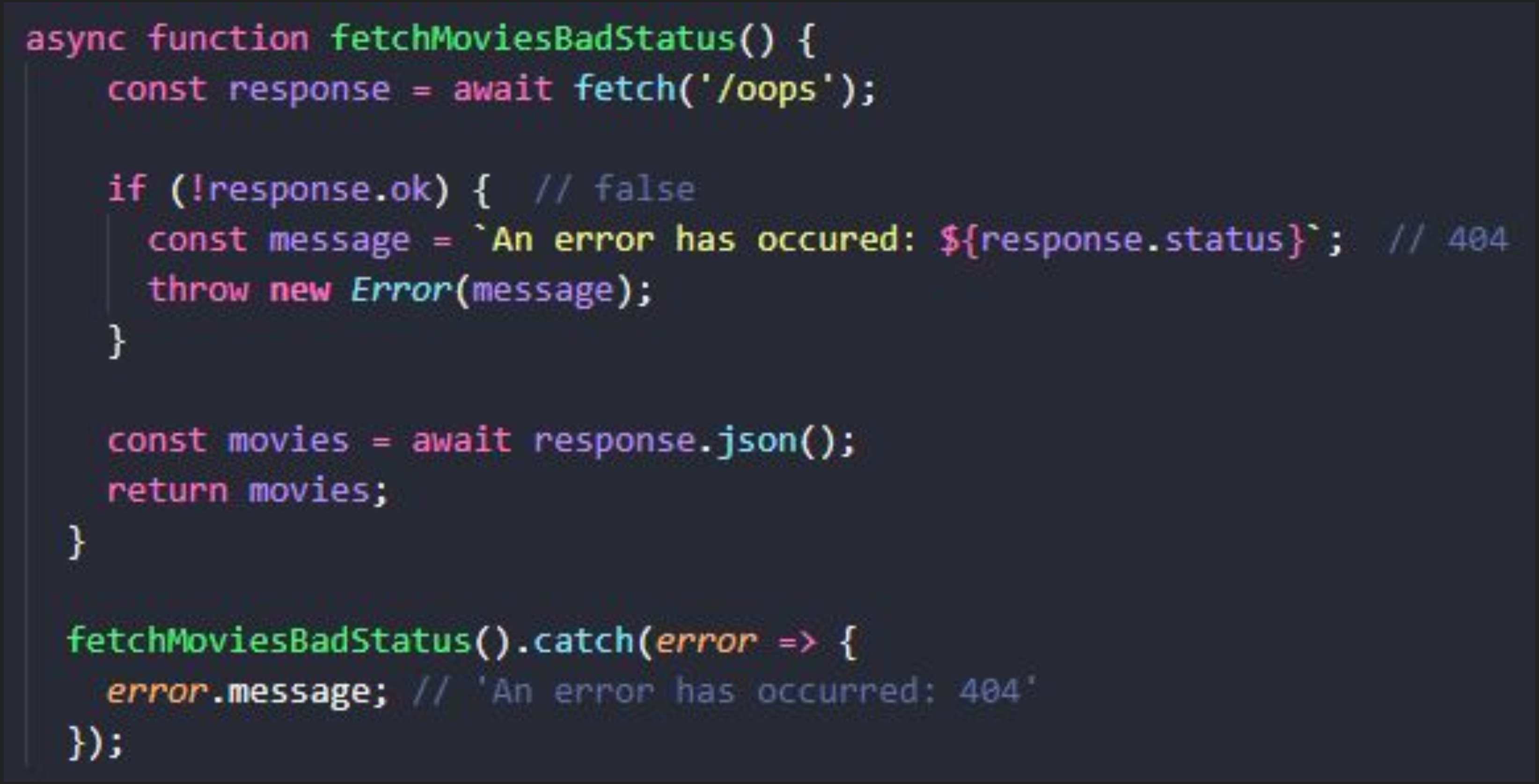
Here's how you can extract the JSON object from a fetch response:



```
async function fetchMoviesJSON() {
    const response = await fetch('/movies');
    const movies = await response.json();
    return movies;
  }

  fetchMoviesJSON().then(movies => {
    movies; // fetched movies
  });
```

response.json() is a method on the Response object that lets you extract a JSON object from the response. The method returns a promise, so you have to wait for the JSON: await response.json().

3. Handling errors with fetch()



```
async function fetchMoviesBadStatus() {
    const response = await fetch('/oops');

    if (!response.ok) {  // false
      const message = `An error has occured: ${response.status}`;  // 404
      throw new Error(message);
    }

    const movies = await response.json();
    return movies;
  }

  fetchMoviesBadStatus().catch(error => {
    error.message; // 'An error has occurred: 404'
  });
```

Axios	Fetch
Cleaner Syntax	Modern JavaScript
Promise Based	Promise Based
Wide Browser Support	New Interfaces
Upload Progress	Request Modes
Transformers & Interceptors	Built-in with Browsers
Built-in XSRF protection	Less Browser Support
Config for All Requests	Lacks features