Connecting to the Backend

Terms

Axios
Back-end
Effect hook
Front-end
HTTP
HTTP request
HTTP response
Side effects

Summary

- We use the effect hook to perform side effects, such as fetching data or updating the DOM.
- The effect hook takes a function that performs the side effect and an optional array of dependencies. Whenever the dependencies change, the effect hook runs again.
- To clean up any resources that were created by the effect hook, we can include a clean-up function that runs when the component unmounts or the dependencies change.
- React is a library for building front-end user interfaces, but to create complete apps, we also need a back-end server to handle business logic, data storage, and other functionality.
- The communication between the front-end and the back-end happens over HTTP, the same protocol that powers the web. The front-end sends an HTTP request to the back-end, and the back-end sends an HTTP response back.
- Each HTTP request and response contains a header and a body. The header provides metadata about the message, such as the content type and HTTP status code, while the body contains the actual data being sent or received.

- To send HTTP requests to the backend, we can use axios, a popular JavaScript library. axios makes it easy to send requests.

- When we send HTTP requests with the effect hook, we should provide a clean-up function to cancel the request if the component is unmounted before the response is received. This is important to prevent errors, especially if the user navigates to a different page while the request is still pending.

- When sending HTTP requests, we must handle errors properly. This can be done using try-catch blocks or by handling the error in the promise chain using .catch().

- Custom hooks are a way to reuse code logic between multiple components. By encapsulating logic in a custom hook, we can create reusable pieces of code that can be shared across components without duplicating the code. Custom hooks can be used to handle common tasks, such as fetching data, and can help to make our code more organized and easier to maintain.

USING THE EFFECT HOOK

```
function App() {
  useEffect(() => {
    document.title = 'App';
  }, []);
}
```

FETCHING DATA WITH AXIOS

```
const [users, setUsers] = useState<User[]>([]);

useEffect(() => {
  axios.get<User[]>('http://...')
    .then((res) => setUsers(res.data));
}, []);
```

HANDLING ERRORS

```
const [error, setError] = useState('');

useEffect(() => {
  axios.get<User[]>('http://...')
    .then((res) => setUsers(res.data))
    .catch(err => setError(err.message));
}, []);
```


CANCELLING AN HTTP REQUEST

```
useEffect(() => {
  const controller = new AbortController();

  axios.get<User[]>('http://...', { signal: controller.signal })
    .then((res) => setUsers(res.data))
    .catch(err => {
      if (err instanceof CanceledError) return;
      setError(err.message)
    });

  return () => controller.abort();
}, []);
```

DELETING DATA

```
axios.delete('http://...')
```

CREATING DATA

```
axios.post('http://...', newUser)
```

UPDATING DATA

```
axios.put('http://...', updatedUser)
```

