Module 3-17

Consuming Web Services (Part 2)

Review of POST & PUT requests

- Unlike GET requests, POST and PUT requests require a body.
- Axios greatly simplifies this process by allowing the inclusion of a JavaScript object into the respective methods.

Setting up the Axios Service

```
import axios from 'axios';

const http = axios.create({
  baseURL: "http://localhost:3000"
});

export default {

  addCard(card) {
    return http.post('/cards', card);
  },

  updateCard(card) {
    return http.put(`/cards/${card.id}`, card);
  },
}
```

Note how a **body** will be sent with a put or a post request.

Let’s Modify the Axios Service

Calling the Methods in the Service

```
boardsService
.addCard(newCard)
.then(response => {
    // After the promise has been resolved
})
```

```
boardsService
.updateCard(newCard)
.then(response => {
    // After the promise has been resolved
})
```

The methods described on the previous slides are now being called within our VUE components.

We still use **promise chaining** to help us define code that will be run only after the promise is resolved.

Error Handling

```
boardsService
.addCard(newCard)
.then(response => {
     ...
})
.catch( error=> {
     if (error.response) {...}
     else if (error.request) {...}
     else {...}
}
)
```

We can handle errors by chaining a catch section.

If something went wrong, we can inspect the error object.

Any actions that are taken due to the request erroring out should go inside this anonymous function.

Error Handling

- If there is an unsuccessful response (defined as the status code not being in the 200 range), we can identify this occurrence with **error.response**.
- If the response was never received, we can identify this occurrence with **error.request**.

Let’s call the Axios service