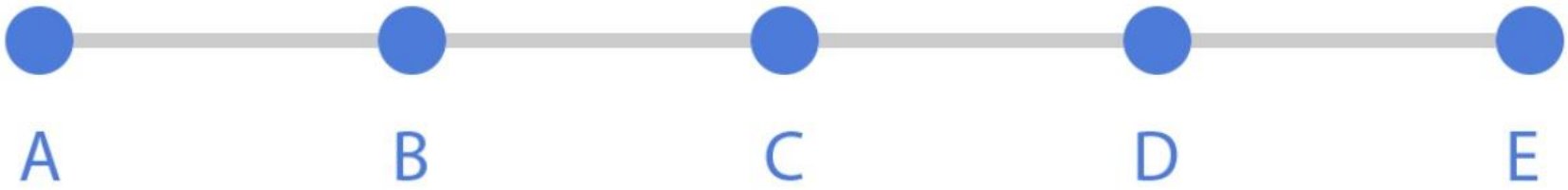
Calling Web Services with Vue

Part 1: GET

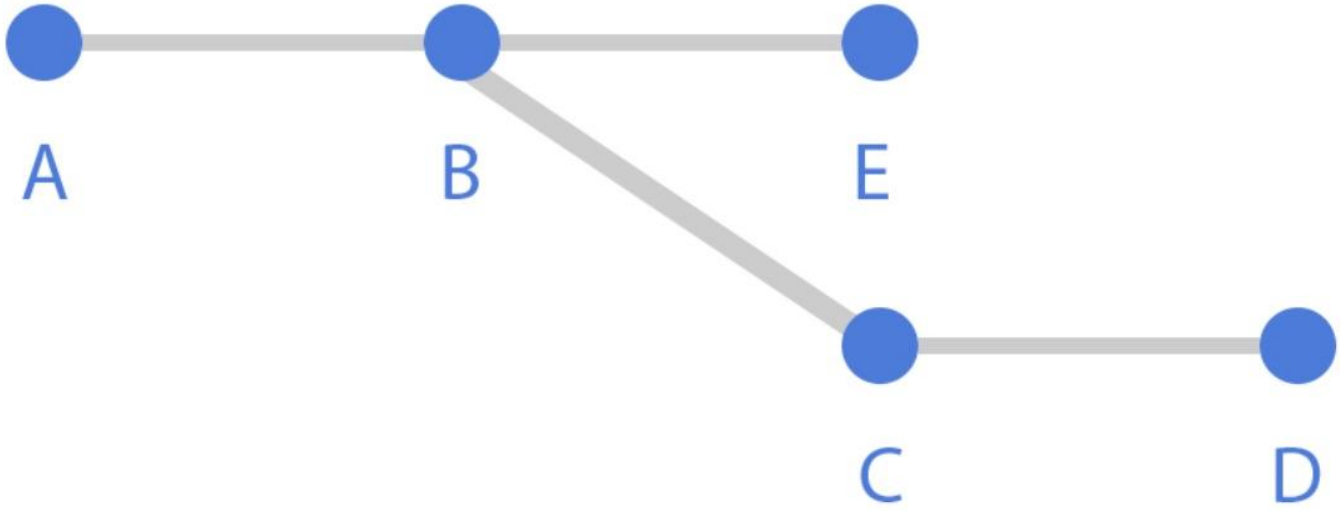
Synchronous vs. Asynchronous Programming

- **<u>Synchronous Programming:</u>**
  - When calling a function or method, the code expects to get the result before the flow of execution moves on to the next line of code.
- **<u>Asynchronous Programming:</u>**
  - When calling a function or method, the call returns right away but the called code continues to run until it completes. If the calling code expected a result, the result data will be resolved once the called code completes and the result is available. Using this approach for web service calls, which can be very slow, helps make code more efficient and responsive.

Synchronous Calls



Asynchronous Calls



Let's Review Using Postman & JSON for API Calls

Using Axios to Make Web API Calls

Axios is a library that is used to make calls to Web API services from a JavaScript front-end application. It provides many easy ways to process the request and response used in the HTTP interaction.

We can use the axios.get method to make an HTTP GET request:

```
axios.get('/users');
```

What does this call return? Does it return the value of the HTTP response that is returned from the backend?

Introducing: Promises

In JavaScript, one of the most common ways to handle the results of asynchronous methods is using an object called a **Promise**. When a method returns a Promise, you can think of it as saying, "I don't have your answer now, but I promise to get back to you when I do."

A Promise can be in one of three states:

- **Pending**: initial state, neither fulfilled nor rejected.
- **Fulfilled**: meaning that the asynchronous operation completed successfully.
- **Rejected**: meaning that the asynchronous operation failed.

Using Axios to Make Asynchronous Web API Calls

```js
axios.get('/users')                    // sends an HTTP request to '/users' and returns a
    .then( (users) => {                // here you're dealing with the Promise returned by

        ... // some code that deals with the JSON data
    });
```

When the HTTP request completes, the **`then`** portion of the call executes..

Using Axios to Make Asynchronous Web API Calls

```js
axios.get('/users')                    // sends an HTTP request to '/users' and returns a
    .then( (users) => {                // here you're dealing with the Promise returned by

        ... // some code that deals with the JSON data
    });
```

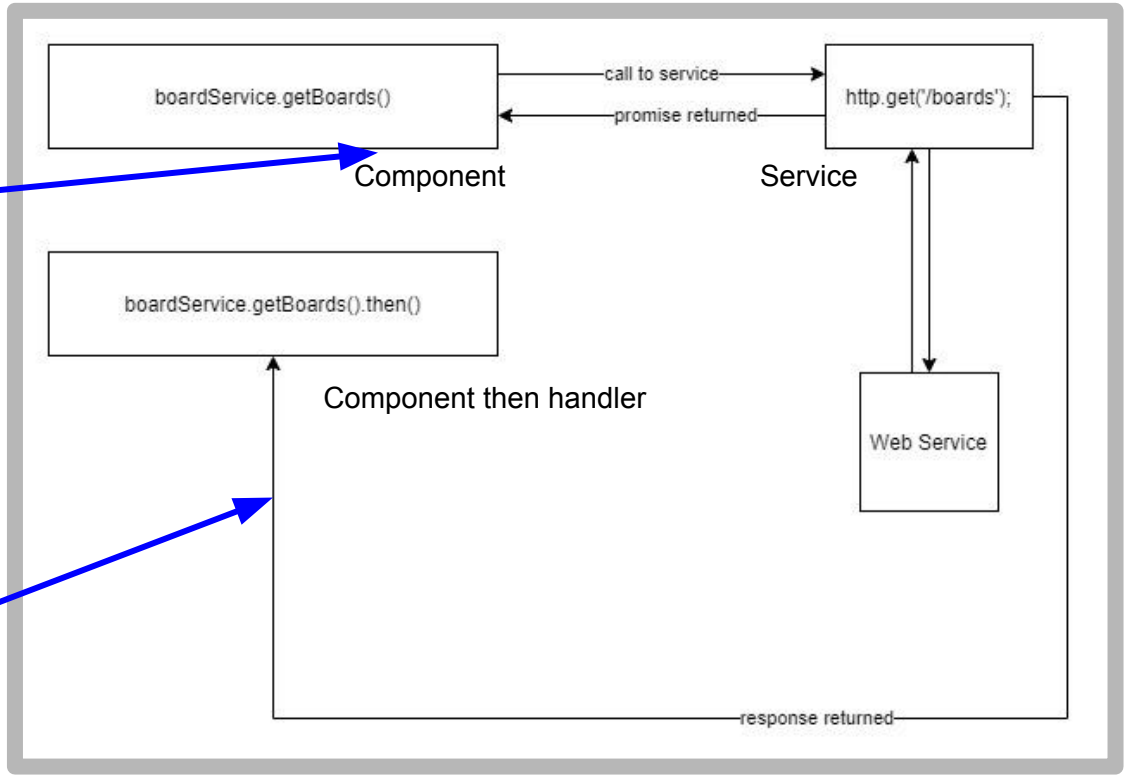
When the HTTP request completes, the **`then`** portion of the call executes..

The Promise returned by **`get()`** resolves, **`users`** contains the information from the response, and the code in the arrow function now has access to do what it needs with it.

Axios Asynchronous Flow

`boardService.getBoards()` code continues to run once the promise is returned.

Once web call completes, the response is sent to the **`then`** handler in **`boardService.getBoards()`** and the promised data will be in the **`data`** property of the response if call was successful.



Service Objects

Service Objects encapsulate the logic of a service into one JavaScript object that can then be used in multiple places in your application. Separating that logic into its own object also makes it more testable and replaceable, if needed. So, you want to keep all of your API calls in a Service Object in your applications.

Service Objects are regular `.js` files. For example, there might be an `apiService.js` file in a `/service` directory under the `/src` of your application

Imports the **`axios`** library.

```
import axios from 'axios';

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

export default {

  list() {
    return http.get('/users');
  },

  get(id) {
    return http.get(`/users/${id}`);
  }

}
```

Imports the **`axios`** library.

Creates the a configured version of the Axios object and assigns it to an **`http`** variable. Here we set the base URL for calls made using the Axios object.

```
import axios from 'axios';

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

export default {

  list() {
    return http.get('/users');
  },

  get(id) {
    return http.get(`/users/${id}`);
  }

}
```

Imports the **`axios`** library.

Creates the a configured version of the Axios object and assigns it to an **`http`** variable. Here we set the base URL for calls made using the Axios object.

```
import axios from 'axios';

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

export default {

  list() {
    return http.get('/users');
  },

  get(id) {
    return http.get(`/users/${id}`);
  }

}
```

`export` the functions that make up the service object.

Imports the **`apiService`**

```
<script>
    import apiService from '@/services/apiService.js';

    // You now have access to the service in your code
    export default {
        name: 'user-list',
        methods: {
            loadUsers() {
                apiService.list().then( (response) => {
                    this.users = response.data;
                });
            }
        }
    }
</script>
```

Imports the **`apiService`**

Call the list method in the **`apiService`**

```
<script>
    import apiService from '@/services/apiService.js';

    // You now have access to the service in your code
    export default {
        name: 'user-list',
        methods: {
            loadUsers() {
                apiService.list().then( (response) => {
                    this.users = response.data;
                });
            }
        }
    }
</script>
```

Imports the **`apiService`**

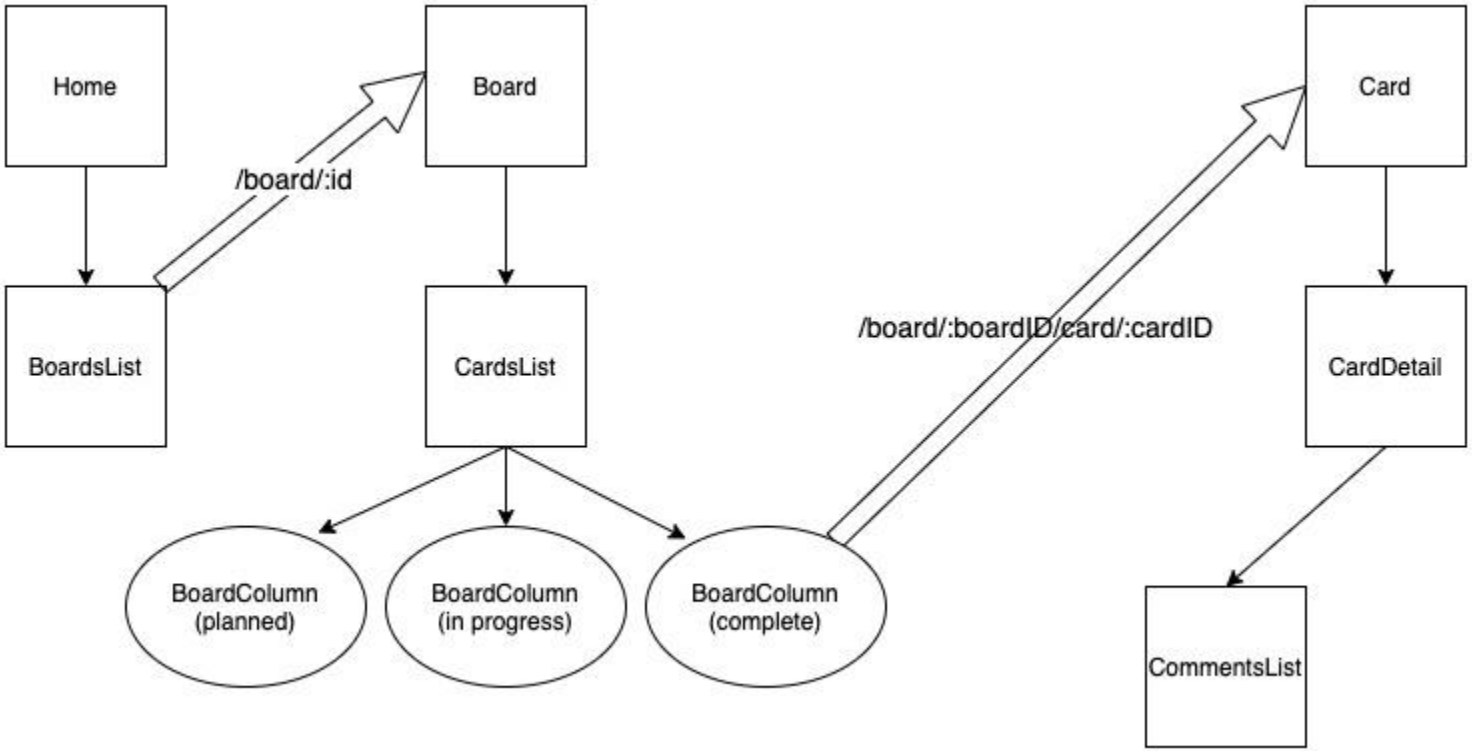
Call the list method in the **`apiService`**

```
<script>
    import apiService from '@/services/apiService.js';

    // You now have access to the service in your code
    export default {
        name: 'user-list',
        methods: {
            loadUsers() {
                apiService.list().then( (response) => {
                    this.users = response.data;
                });
            }
        }
    }
</script>
```

The **`then`** portion of the code will execute when the http request completes.

Let's Implement the Card Detail Code...


Home
Board
Card
/board/:id
BoardsList
CardsList
/board/:boardID/card/:cardID
CardDetail
BoardColumn
(planned)
BoardColumn
(in progress)
BoardColumn
(complete)
CommentsList