Calling Web Services with Vue Part 2: POST/PUT/DELETE

Review: Request Types

HTTP Request Types:

- **GET:** Ideally suited for simply reading data
- **POST**: Ideally suited for inserting new data into the data source.
- **PUT**: Ideally suited for updating an existing record within a data source.
- **DELETE**: Ideally suited for removing an existing record from the data source.

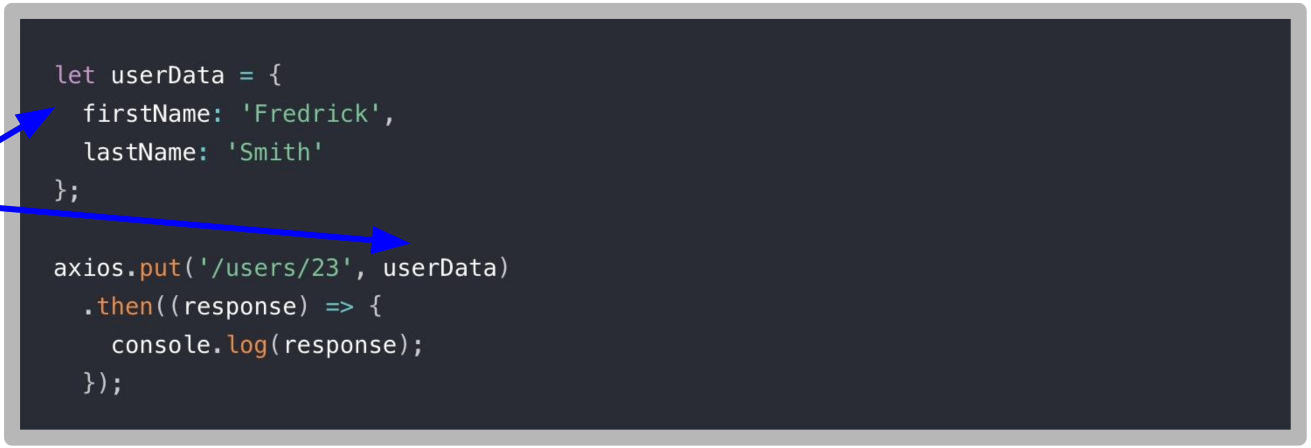
Request Types that have a body:

- **POST**
- **PUT**

Using Axios to send POST, PUT, DELETE requests

Sending **POST** and **PUT** requests with Axios is similar to sending **GET** requests, but you are able to pass data to send in the payload (body).

The **`userData`** object is passed as a param and will be sent as the payload of the **POST/PUT** request.

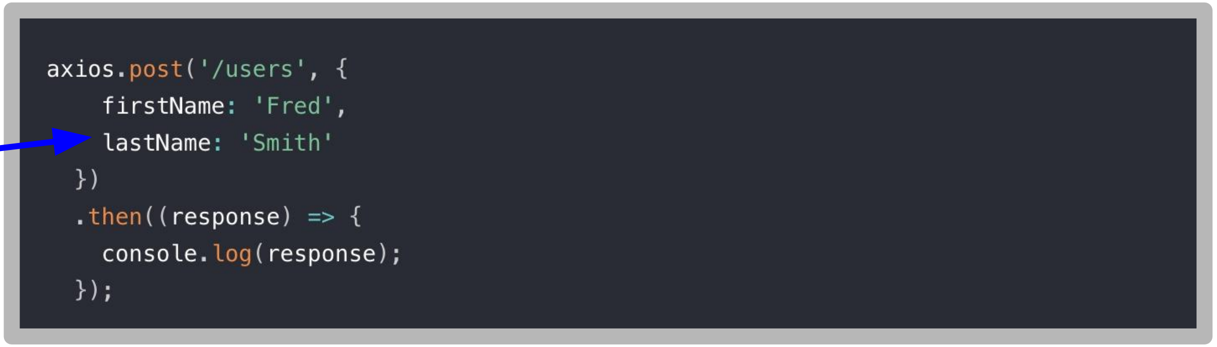


```
let userData = {
  firstName: 'Fredrick',
  lastName: 'Smith'
};

axios.put('/users/23', userData)
  .then((response) => {
    console.log(response);
  });
```

Using Axios to send POST, PUT, DELETE requests

We can even define the payload data in the request.

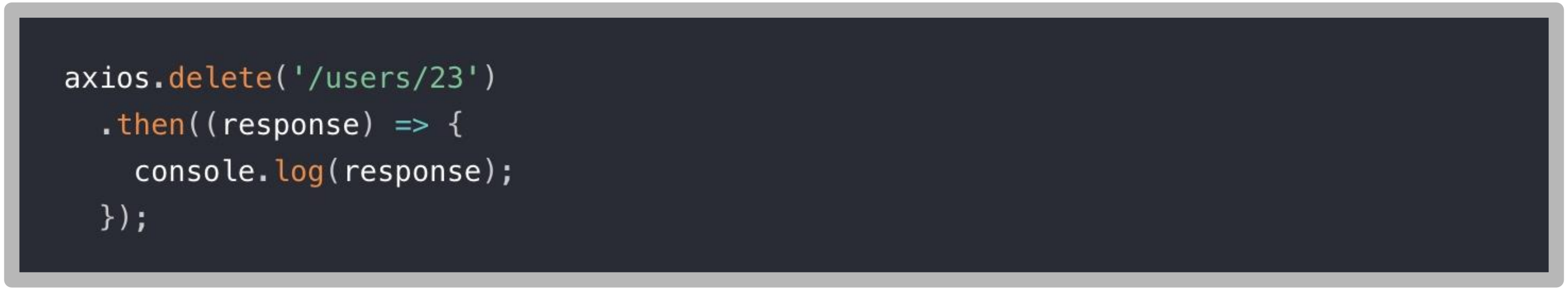
The payload data is declared IN the request.



```
axios.post('/users', {
    firstName: 'Fred',
    lastName: 'Smith'
  })
  .then((response) => {
    console.log(response);
  });
```

Using Axios to send POST, PUT, DELETE requests

Sending **DELETE** requests with Axios is almost the same as sending **GET** requests, but you use **`axios.delete()`** rather than **`axios.get`**

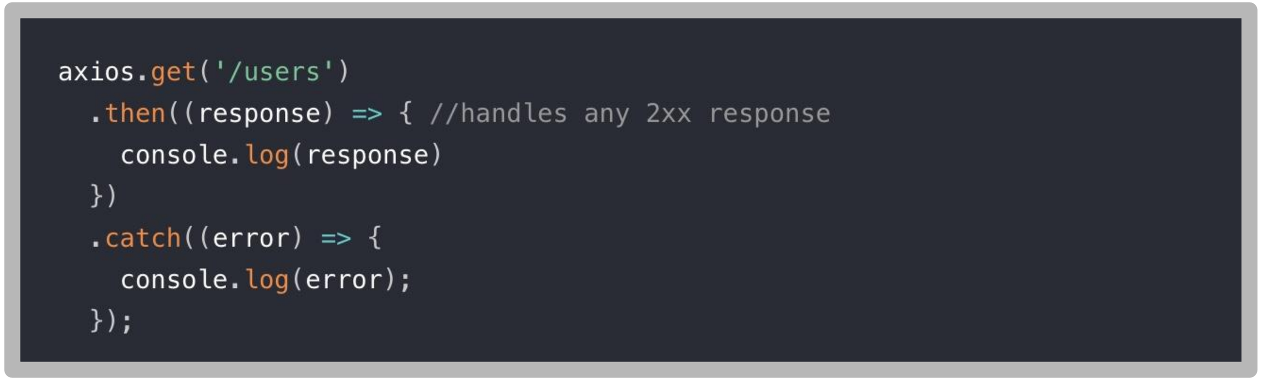


```
axios.delete('/users/23')
  .then((response) => {
    console.log(response);
  });
```

Handling errors with Axios

You can chain a `.catch()` method after your `.then()` method. The `.catch()` method runs if

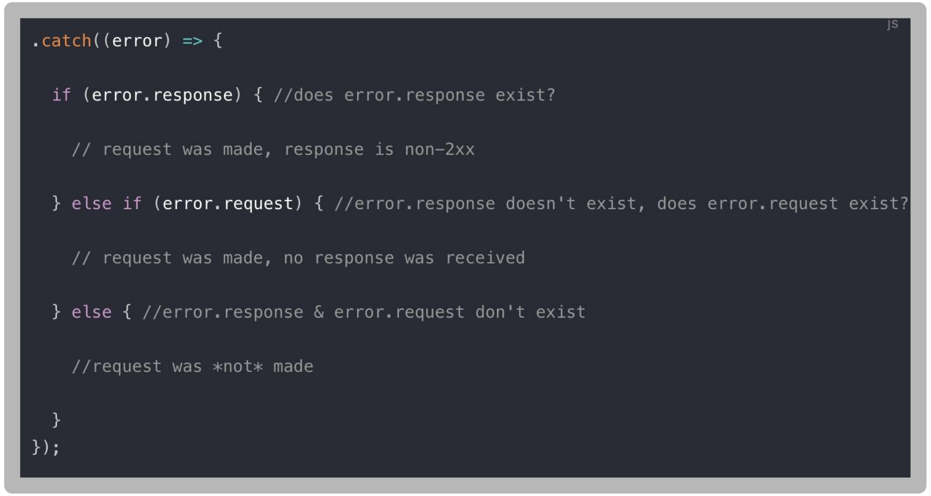
- The server responds with a non-**`2xx`** response code—remember that **`2xx`** codes are "success" messages.
- The server fails to respond due to an incorrect domain/port/protocol or network error.
- Something happened while setting up the request that triggered an error.



```
axios.get('/users')
  .then((response) => { //handles any 2xx response
    console.log(response)
  })
  .catch((error) => {
    console.log(error);
  });
```

Handling errors with Axios

You can distinguish between these situations with an **`if-else`** block that tests for the **`.response`** and **`.request`** properties of the error object:



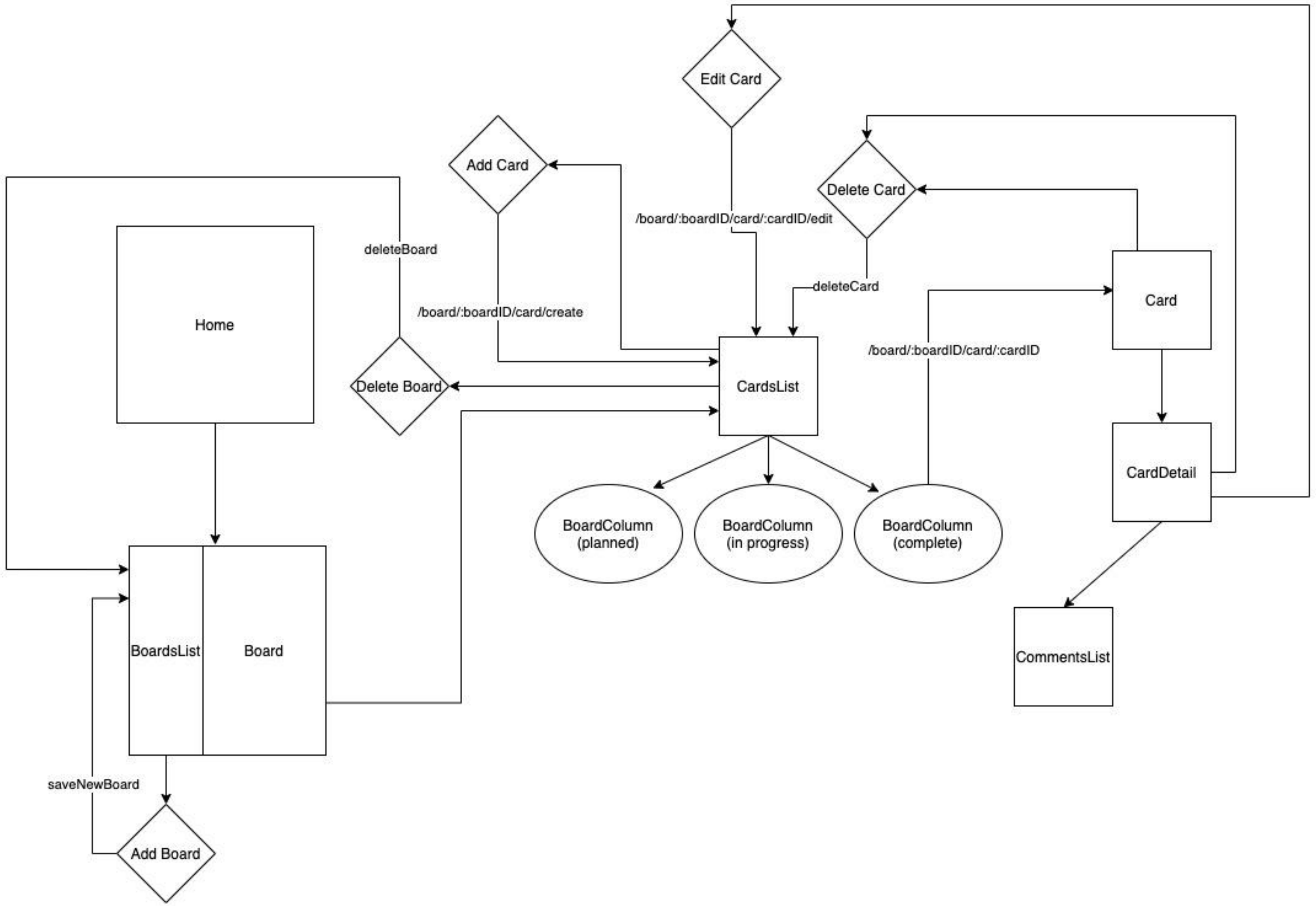
```js
.catch((error) => {

  if (error.response) { //does error.response exist?

    // request was made, response is non-2xx

  } else if (error.request) { //error.response doesn't exist, does error.request exist?

    // request was made, no response was received

  } else { //error.response & error.request don't exist

    //request was *not* made

  }
});
```

Handling errors with Axios

With this process, you can handle all three situations that the .catch() method fires on. The .response object contains some more properties that can help you determine what happened:

- **`error.response.status`** is the response status code, like **`401`** or **`503`**. The description of the status code can be found in **`error.response.statusText`**.
- **`error.response.data`** contains information sent back from the server that might help you diagnose the error. This isn't a guarantee, but it's worth looking at if you do run into an issue.

Let's Write Some Code


Edit Card
Add Card
Delete Card
/board/:boardID/card/:cardID/edit
deleteBoard
Home
deleteCard
Card
/board/:boardID/card/create
/board/:boardID/card/:cardID
Delete Board
CardsList
CardDetail
BoardColumn (planned)
BoardColumn (in progress)
BoardColumn (complete)
BoardsList
Board
CommentsList
saveNewBoard
Add Board