November 2, 2021

**Back End 2**

**Learning Objectives**

**What We Will Cover**

* Sending form data
* POST
* PUT
* DELETE

**A Quick Note**

This is a shorter lecture, however, it if full of important info. Please come back and re-read this content if you need to.

**Form Data**

**What is form data?**

In many web applications it is common to gather information from users. This is typically done with a form.

Table

Description automatically generated with medium confidence

**<Form>**

There are many different ways to collect the data of a form on the front-end. We will not be covering that today. However, once the data is collected on the front-end, it will typically be in object form, like this:

let body = {

username: "RingBearer",

firstName: "Frodo",

lastName: "Baggins",

age: 51

}

This is what will ultimately be sent to the back-end, so it can work its magic.

**POST**

**POST - from the Front-End**

**let** user = {

username: 'Dragon',

firstName: 'Joey',

lastName: 'Tribiani'

}

axios.post('/api/users', user).then(...)

In the above example, the front-end will make a POST request to the back-end, giving it the ***user*** object. The back-end will then handle the object, and send a response back to the front-end.

**POST - from the Back-End**

Now, we need to setup our code to handle the info passed in from the front-end. **Note: The back-end typically dictates what info the front-end should be sending.**

app.post('/api/users', (req, res) => {

console.log(req.body) *// { username: 'Dragon", firstName: 'Joey', lastName: 'Tribiani' }*

**let** username = req.body.username

**let** firstName = req.body.firstName

**let** lastName = req.body.lastName

res.status(200).send(`Welcome, *${*username*}*! It is nice to meet you *${*firstName*}* *${*lastName*}*.`)

})

**Destructuring**

Destructuring can save us a lot of time when dealing with objects.

app.post('/api/users', (req, res) => {

**let** { username, firstName, lastName } = req.body

res.status(200).send(`Welcome, *${*username*}*! It is nice to meet you *${*firstName*}* *${*lastName*}*.`)

})

**Handling Errors**

app.post('/api/users', (req, res) => {

userDatabase.push(req.body) *// Adds the user that was sent from the front-end to our database.*

res.status(200).send('User successfully added.')

})

Let’s say we wanted to add a user to our database of uses (like in the above example), but something went wrong and the data didn’t come through properly. We might have added an object that looks like this { } to our database, and then still sent a message back to the front end saying ‘User successfully added.’ To avoid this, we need to write code that protects against this.

app.post('/api/users', (req, res) => {

**let** { username, firstName, lastName } = req.body

**if** (!username || !firstName || !lastName) {<<the “!” means not or no username/firstname/last

res.status(400).send('Uh oh, it looks like you are missing some data.')

} **else** {

userDatabase.push(req.body) *// Adds the user that was sent from the front-end to our database.*

res.status(200).send('User successfully added.')

}

})

In this example, we first make sure that we have all the data we require, and if not, send back a status of 400 (bad request), along with a message.

**PUT**

**Updating Content with PUT**

**PUT** is used to update existing content.

PUT does not require all the properties of an object to be passed in. Ex:

**let** userUpdated = {

username: 'Dragon2'

}

axios.put('/api/users/Dragon', userUpdated).then(...)

In the above example, the front end is hitting the /api/users endpoint, giving a param of Dragon (the existing username), and passing in the userUpdated object, which only includes a username.

**PUT on the Back-End**

On the back-end, we will want to update the user with the username that matches the param passed in.

**let** users = [user1, user2, user3...]<<this code is for usernames that are unique

the params are:api/users/:username. From the slide before it was Dragon2 but now changing to just Dragon.

app.put('/api/users/:username', (req, res) => {

**let** existingUsername = req.params.username <<first pull original username

**let** newUsername = req.body.username <<then new username

**for** (**let** i = 0; i < users.length; i++) {

**if** (users[i].username === existingUsername) { <<if theres a match then update

user[i].username = newUsername

res.status(200).send("User updated.")

**return**

}

}

res.status(400).send("User not found.")

})

Research on how to block users from accessing these things

**DELETE**

**DELETE on the Front-End**

Like PUT, **DELETE** often requires a parameter to specify which object we want to delete. However, that is not required.

axios.**delete**('/api/users/Dragon').then(...)

In this example, we are letting our backend know to delete the user with the username ‘Dragon’.

**DELETE on the Back-End**

**let** users = [user1, user2, user3...]

app.**delete**('/api/users/:username', (req, res) => {

**let** existingUsername = req.params.username

**for** (**let** i = 0; i < users.length; i++) { <<check my users for dragons

**if** (users[i].username === existingUsername) { <<if it’s in there

users.splice(i, 1)

res.status(200).send("User deleted.") <<delete it

**return**

}

}

res.status(400).send("User not found.") <<if not found, alert them

})

Some places don’t delete and write a fake delete.

Things to make sure when you’re writing is: make sure they are not a bot, make sure they are an admin, make sure they are that user, make sure they have access to the right stuff and can’t get in to other things.

**Refactoring Code**

**Keep Your Server File Clean**

You may be realizing that any given site could have dozens of endpoints. Building out long algorithms in each endpoint would result in your server file being thousands of lines long, and quite complicated to navigate. This is where controller files come in handy.

**const** userController = require('./controllers/userController')

app.get('/api/users', userController.getAllUsers)

app.post('/api/users', userController.createUser)

app.put('/api/users/:username', userController.updateUser)

app.**delete**('/api/users/:username', userController.deleteUser)

We will showcase this in today’s Demo.

**Controller File**

All controller files, when using express, will have the following structure:

module.exports = {

someFunctionName: (req, res) => {

*// function body here*

},

someOtherFunctionName: (req, res) => {

*// function body here*

}

}

In our **user** example, our **userController** file would look something like this:

**let** users = [user1, user2, user3...]

module.exports = {

createUser: (req, res) => {

userDatabase.push(req.body)

res.status(200).send('User successfully added.')

},

updateUser: (req, res) => {

**let** existingUsername = req.params.username

**let** newUsername = req.body.username

**for** (**let** i = 0; i < users.length; i++) {

**if** (users[i].username === existingUsername) {

user[i].username = newUsername

res.status(200).send("User updated.")

**return**

}

}

res.status(400).send("User not found.")

}

}

© 2021 Devmountain