

Lecture#1: Integration

CSC105 Web Application Development (2/2022)

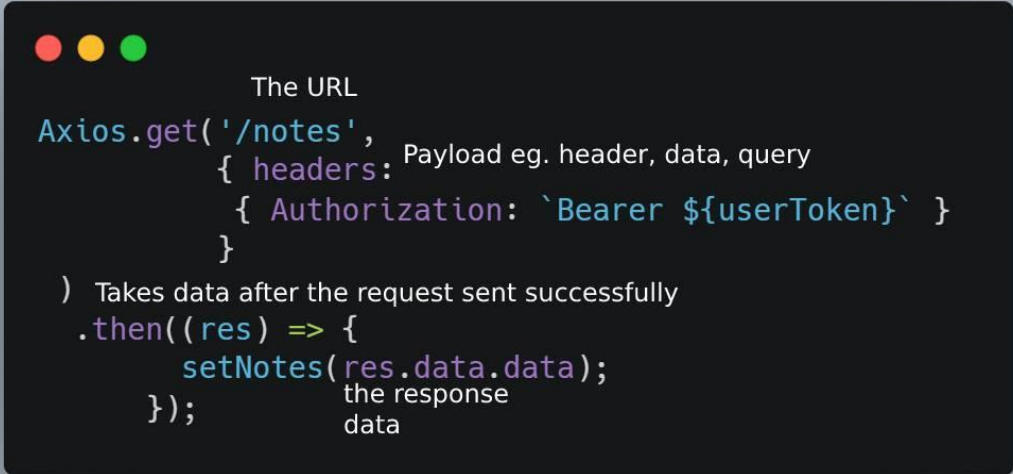
Thursday, 11 May 2023

1. Overview

Since you've learned frontend and backend from the previous module. As you can clearly see, the backend should provide information to the frontend to beautifully display to the user. The question that might pop up to you guys is "HOW ?". Integration module is missing from this equation. In this module, we will teach you to retrieve data from your backend and store them in your frontend app. We will give you all the best-practices that you might need in designing your routes and implementing your web application.

2. What is Axios

Axios is a promise-based HTTP Client for node.js and the browser. Simply means that we can use this as a replacement for Postman. Here is an example of Axios request.



```

    The URL
Axios.get('/notes',
  { headers: Payload eg. header, data, query
    { Authorization: `Bearer ${userToken}` }
  }
) Takes data after the request sent successfully
.then((res) => {
  setNotes(res.data.data);
});           the response
              data
```

You can see the document of Axios is here

[Axios API | Axios Docs \(axios-http.com\)](https://axios-http.com/docs/axios-api)

In Axios.get, you will see the URL only “/notes” and you might be wondering where the location or the destination of this url. Well in Axios, we can create an Axios instance somewhere and use that instance with provided baseUrl to call other paths. Here is an example of Axios instance

```
src > share >  AxiosInstance.js > ...  
1   import axios from 'axios';  
2  
3   const Axios = axios.create({ baseUrl: 'http://localhost:8000', withCredentials: true });  
4   export default Axios;  
5   |
```

Therefore the request in the first picture will be going to ‘http://localhost:8000/notes’ with the get request.

3. Example endpoint “View all note”

In this lab, we will be building a Note application but don’t worry because you’re going to be building only the integration part.

The App that we’re building is Note app that we can list, retrieve, update, delete notes

In this example we’re building a “view all note” route, you can download the boilerplate code then start running backend and frontend. Here is some boilerplate code you might need to download. [ThisTine/105_integration_lab \(github.com\)](https://github.com/ThisTine/105_integration_lab)

In this project, you will be given 2 main folders which are “backend” and “frontend”. We’re not going to talk much about the backend because our job is just implementing the integration between frontend and backend.

In order to run the project you may need to get your backend up and running first. By going to the backend folder, there are only 2 commands associated with the backend which are.

```
npm install
```

This will install needed packages

```
npm run dev
```

This will start the backend server

After you ran the backend, now you need to open the terminal and start running the frontend. By going to the frontend folder, You need to run the same 2 commands which are.

```
npm install
```

This will install needed packages

```
npm run dev
```

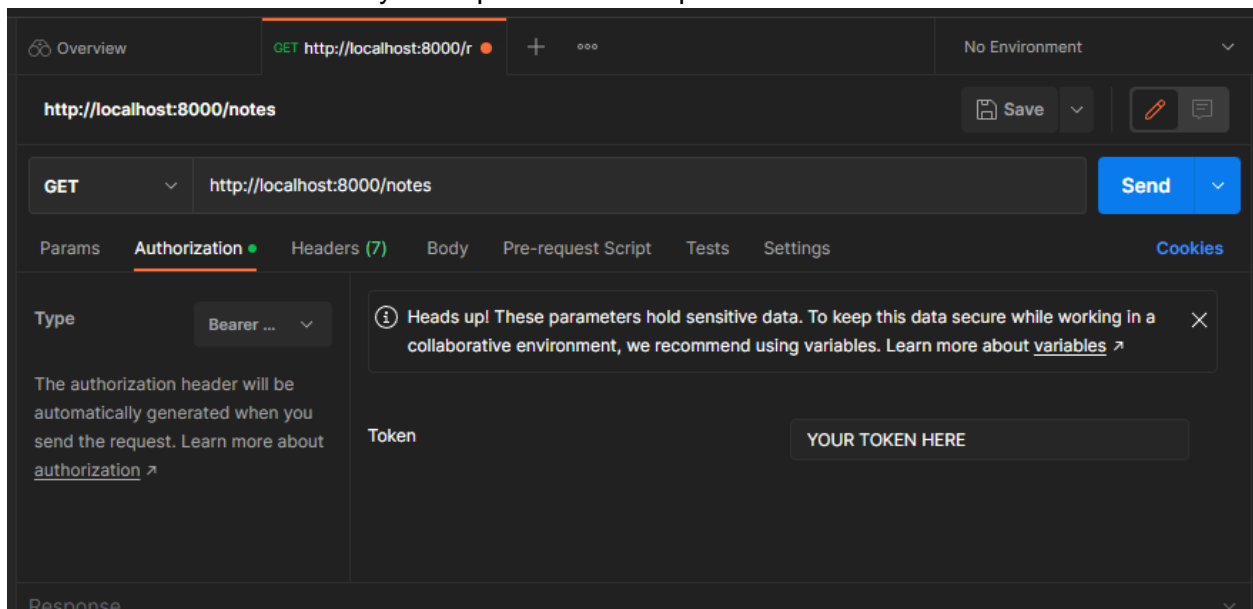
This will start the frontend dev server

Now your server is up and running. Let's see an example of our "View all notes".
In side the frontend folder there will be a file called Home.jsx inside
" src>pages>home>Home.jsx "

You will see the function that gets all notes inside the useEffect hook.

```
useEffect(() => {  
  // TODO: Implement get notes by user's token  
  // 1. check if user is logged in  
  const userToken = Cookies.get('UserToken');  
  if (userToken !== undefined && userToken !== 'undefined') {  
    // 2. call API to get notes  
    Axios.get('/notes', { headers: { Authorization: `Bearer ${userToken}` } }).then((res) => {  
      // 3. set notes to state  
      setNotes(res.data.data);  
    });  
  }  
}, [user]);
```

The function above is actually the equivalent of this postman

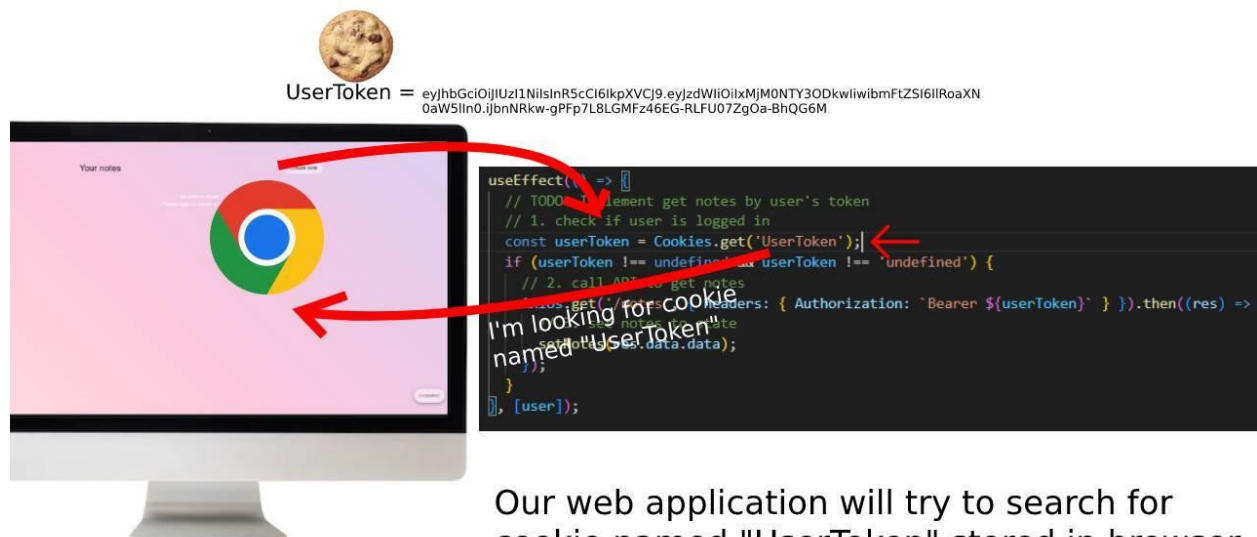


Here is the demonstration of how this function works.



The useEffect hook will get called
The content in the useEffect
will start working from top to bottom.

The web page is first loaded.



Our web application will try to search for
cookie named "UserToken" stored in browser
and stored that cookie in variable called
"userToken"

The useEffect starts working.

```
header {
  Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWll....gOa-BhQG6M
},
method: get
Destination: http://localhost:8000/notes
```

```
useEffect(() => {
  // TODO: Implement get notes by user's token
  // 1. check if user is logged in
  const userToken = Cookies.get('UserToken');
  if (userToken !== undefined && userToken !== 'undefined') {
    // 2. call API to get notes
    Axios.get('/notes', { headers: { Authorization: `Bearer ${userToken}` } }).then((res) => {
      // 3. set notes to state
      setNotes(res.data.data);
    });
  }
}, [user]);
```



Our web application will send the request to the backend server with the header Authorization: Bearer <Token>, method: get

The request is being sent to the backend

```
data:[
  {
    id: 1,
    title: Today note
    createdAt: 23/1/23
    ....
  }
]
```

```
useEffect(() => {
  // TODO: Implement get notes by user's token
  // 1. check if user is logged in
  const userToken = Cookies.get('UserToken');
  if (userToken !== undefined && userToken !== 'undefined') {
    // 2. call API to get notes
    Axios.get('/notes', { headers: { Authorization: `Bearer ${userToken}` } }).then((res) => {
      // 3. set notes to state
      setNotes(res.data.data);
    });
  }
}, [user]);
```



After the backend server received your request, they will process and send back your data. The sent data will be handed inside .then callback in this case, we use the data to set the state and use the state later on in the project

The backend response back

After this will be the first lab in this module, please read the API specification provided here for more information [105_integration_lab/readme.md at main · ThisTine/105_integration_lab \(github.com\)](https://github.com/ThisTine/105_integration_lab/blob/main/105_integration_lab/readme.md)

4. Lab

4.1. Authentication

This module requires action on two distinct endpoints: *Register* and *Login*. To begin, let's focus on the first endpoint, which is the "Register" endpoint.

Register

Create a user record in the database based on the user request body, you can checkout the backend code logic located in the "backend/src/routes/register.js" file within the backend folder.

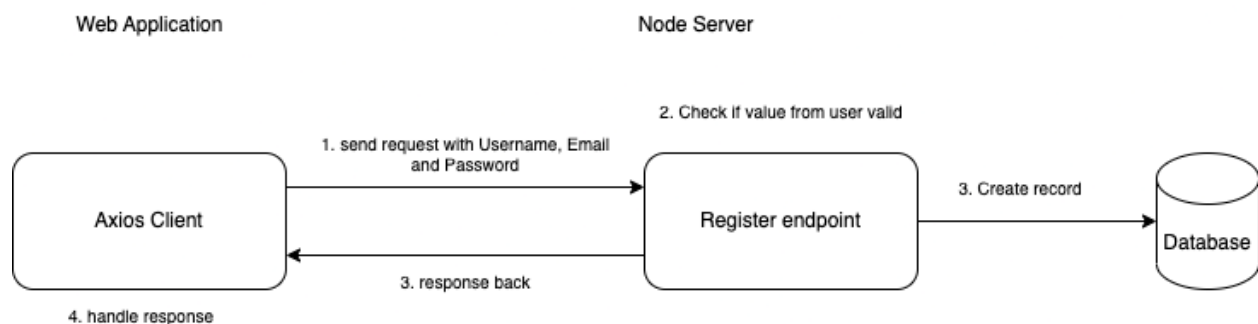


Figure 4.1: show overview of data flow

Overview :

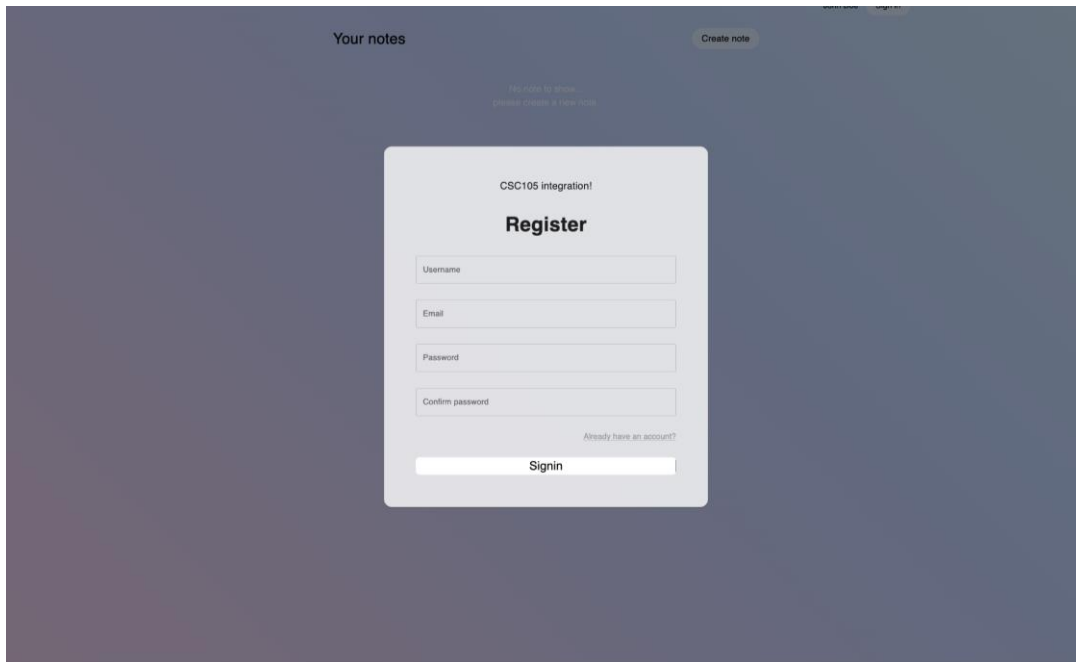
What you need to do is get data from react application form, and send **POST** request to backend server, and you can know the url through routing file in backend "backend/src/index.js"

```
app.post('/register', register);
```

Line 32th of index.js

Hands on :

After opening this modal by clicking on the top right button name “Sign in”, the modal will appear and to register you need to click “No account?” to switch mode from Login to Register. This is where you wanted to integrate.



The image shows a web application interface. At the top, there's a header with 'Your notes' on the left and a 'Create note' button on the right. Below the header, there's a message: 'No notes to show. Please create a new note.' In the center, a white modal box is displayed. The modal has a title 'CSC105 integration!' and a subtitle 'Register'. It contains four input fields: 'Username', 'Email', 'Password', and 'Confirm password'. Below the fields, there's a link 'Already have an account?' and a 'Signin' button.

Figure 4.2: show UI of the register form

Next, let go into our frontend code where we need to work in the path “frontend/src/share/modal/components/RegisterForm.jsx”. In the function `handleSubmit` you will see mentors’ comments, and steps required to finish integration.

Step 1: Validate the form first by checking if the required fields are filled, emails are in correct format, and password and confirm password are the same. First create function `validateForm` for cleaner code.

```
const validateForm = () => {
  let isValid = true;
  // check user
  if (!username) {
    setUsernameError('Username is required');
    isValid = false;
  }
  // check email
  if (!email) {
    setEmailError('Email is required');
    isValid = false;
  }
  if (!/^[\w-\.\.]+\@([\w-]+\.\.)+[\w-]{2,4}$/g.test(email)) {
    setEmailError('Invalid email format');
    isValid = false;
  }
  // check password
  if (!password) {
    setPasswordError('Password is required');
    isValid = false;
  }
  if (!rePassword) {
    setRePasswordError('Confirm password is required');
  }
  if (password !== rePassword) {
    setPasswordError('Password is not match');
    setRePassword('');
    setPassword('');
    isValid = false;
  }

  return isValid;
};
```


Then take this function to call in the `handleSubmit` method.

```
const handleSubmit = async () => {  
  // TODO: Implement login  
  // 1. validate form  
  if (!validateForm()) return;  
}
```

Step 2: Send a request to the backend server, so now you have to use axios library.

First you need to import axios client that mentors provided on “frontend/src/share/axios.js”

```
import Axios from '../..//axios';
```

Place this line of code on the top section of current file

And next we are going to make axios request /register route by writing this line of code in which the first parameter is url, and the second parameter is body that should contain `username`, `email`, and `password`.

****** make sure you put `async` in front of the function first because using `await` in the function required that function to be asynchronous.

```
const handleSubmit = async () => {  
  // TODO: Implement login  
  // 1. validate form  
  if (!validateForm()) return;  
  // 2. send request to server  
  const response = await Axios.post('/register', {  
    username,  
    email,  
    password,  
  });  
}
```

Step 3: Handle success response case by change modal to login mode

```
// 3. if success, change modal to login mode
if (response.data.success) {
  setIsLogin(true);
  setStatus({
    msg: response.data.msg,
    severity: 'success'
  });
}
```

Step 4: If fail, show error message alert, but as axios will throw error if response is error. Thus we need to handle errors in the frontend as well.

```
try {
  // 2. send request to server
  // 3. if successful, change modal to login mode
} catch (e) {
  // 4. if fail, show error message alert, and reset password fields
  setPassword('');
  setRePassword('');
  // check if e are AxiosError
  if (e instanceof AxiosError)
    if (e.response)
      // check if e.response exist
      return setStatus({
        msg: e.response.data.error,
        severity: 'error',
      });
  // if e is not AxiosError or response doesn't exist, return error
  message
  return setStatus({
    msg: e.message,
    severity: 'error',
  });
}
```

```
}
```

If done, try to test it if it works flawlessly by inspecting the network tab, and see if the axios did call a request or not.

Things you need to check :

1. Check if all validations we set are working.
2. Check if modal changes into login mode.
3. Check if error handling is working, try registering using the same username or email.

PS: If there is something wrong you might want to redo the above step again.

Login

Take user's username or email, and password from request body to authenticate in the backend server, you can checkout the backend code logic located in the "backend/src/routes/login.js" file within the backend folder.

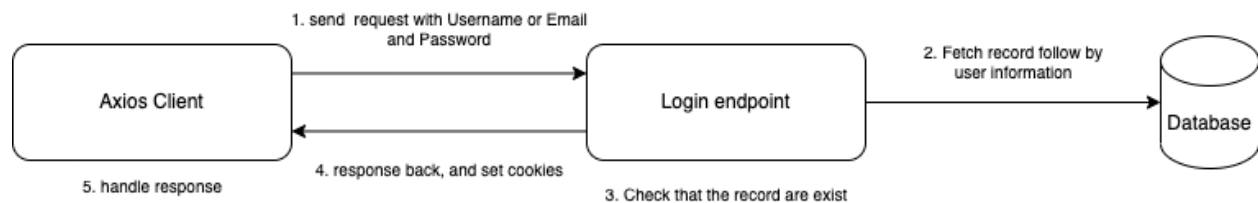


Figure 4.3: show overview of data flow

But before starting, integrate in this section. If you look carefully on step number 3, you will see that we need to set authorization JWT token in the cookie. And to allow web application to do that request sent need to specify option name withCredential to true

But again what is withCredential?

A boolean property that indicates whether or not cross-site Access-Control requests should be made using credentials such as cookies, authorization headers or TLS client certificates. If translating the above sentence to less technical, we can separate it into 2 cases :

1. In terms of **receiving a request**, If the value is true that means the response of that request can set cookies to the client.
2. In terms of **sending a request**, if the value is true that means the request sent will automatically send credentials with it (cookies, headers).

So in the axios client we provided, you can check that it is set withCredential to true by default.

PS: This is bad practice avoid doing this, thus only set withCredentials to true in required request only.

```
const Axios = axios.create({
  baseURL: 'http://localhost:8000',
  withCredentials: true
});
export default Axios;
```

"frontend/src/share/AxiosInstance.js"

Overview :

What you need to do is get data from react application form, and send **POST** request to backend server, and you can know the url through routing file in backend “backend/src/index.js”

```
app.post('/login', login);
```

Line 30th of index.js

Hands on :

After opening this modal by clicking on the top right button name “Sign in”, the modal will appear. This is where you wanted to integrate.

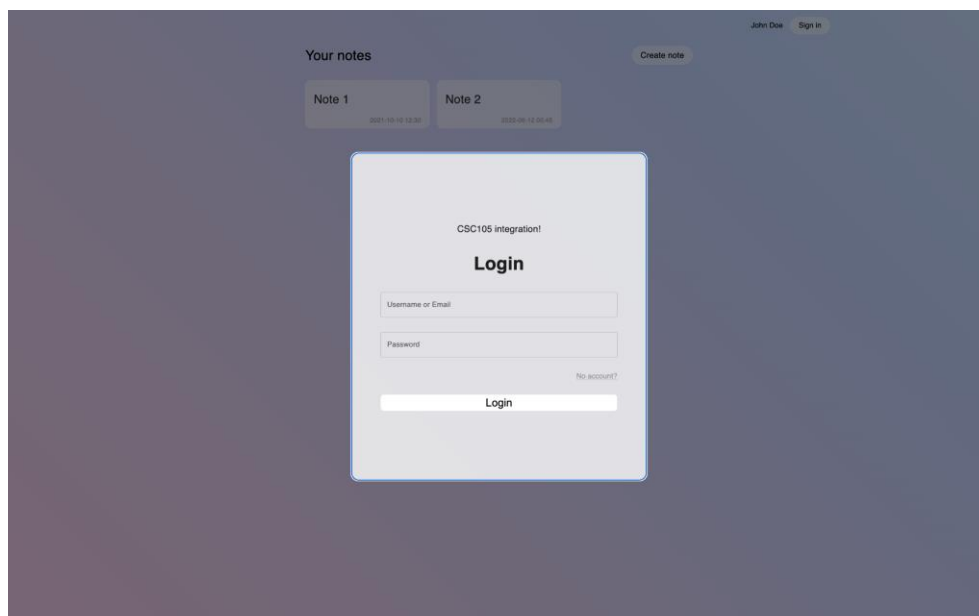


Figure 4.4: show UI of the login form

Next, let go into our frontend code where we need to work in the path “frontend/src/share/modal/components/LoginForm.jsx”. In the function **handleSubmit** you will see mentors’ comments, and steps required to finish integration.

Step 1: Validate the form by checking if all required fields are filled.

```
const validateForm = () => {  
  let isValid = true;  
  if (!usernameOrEmail) {  
    setUsernameOrEmailError('Username or email is required');  
    isValid = false;  
  }  
  if (!password) {  
    setPasswordError('Password is required');  
    isValid = false;  
  }  
  return isValid;  
};
```

Then take this function to call in the `handleSubmit` method.

```
const handleSubmit = () => {  
  // TODO: Implement login  
  // 1. validate form  
  if (!validateForm()) return;
```

Step 2: Send a request to the backend server, so now you have to use axios library as the same as register endpoint integration.

You need to import axios client that mentors provided on “frontend/src/share/AxiosInstance.js”

```
import Axios from '../../../AxiosInstance';
```

Place this line of code on the top section of current file

And next we are going to make a request /login route by writing lines of code in which the first parameter is url, and the second parameter is body that should contain `usernameOrEmail` and `password` as the backend requires this data to be sent with the request.

****** make sure you put `async` in front of function first because using `await` in the function required that function to be asynchronous

```
const handleSubmit = async () => {  
  // TODO: Implement login  
  // 1. validate form  
  if (!validateForm()) return;  
  // 2. call API to login  
  const response = await Axios.post('/login', {  
    usernameOrEmail,  
    password,  
  });  
};
```

Step 3: Handle success response case by closing the modal, and update user information.

```
// 3. if success, change modal to login mode  
if (response.data.success) {  
  setUser({  
    username: response.data.data.username,  
    email: response.data.data.email,  
  });  
  handleClose();  
  setStatus({  
    msg: response.data.msg,  
    severity: 'success'  
  });  
}
```

Step 4: If fail, show error message alert, but as axios will throw error if response is error. Thus we need to handle it using try catch.

```
try {  
  // 2. call API to login  
  // ...  
  // 3. if successful, close modal, and update user information.  
  // ...  
} catch (e) {  
  // 4. if fail, show error message, and reset text fields value  
  setUsernameOrEmail('');  
  setPassword('');  
  // check if e are axiosError  
  if (e instanceof axiosError) {  
    // check if e.response exist  
    if (e.response)  
      return setStatus({  
        msg: e.response.data.error,  
        severity: 'error',  
      });  
  }  
  // if e is not axiosError or response doesn't exist, return error  
  message  
  return setStatus({  
    msg: e.message,  
    severity: 'error',  
  });  
}
```

Although It looks like the code is working fine, it is not as if you try to send a request you will notice a long error with an error shout out about cross origin blah blah blah.

To simplify the error, it is the policy that backend servers need to enable **Cross-Origin Resource Sharing (CORS)** for the domain that is sending the request to it. So, to solve this problem you need to specify allowed domains before registering any request.


```
app.use(  
  cors({  
    origin: ['http://localhost:5174', 'http://localhost:5173'],  
    credentials: true,  
  })  
);
```

“backend/src/index.js” line 25th

If done, try to test it if it works flawlessly by inspecting the network tab, see if the axios did call a request or not,

Things you need to check :

1. Check if all validations we set are working.
2. Check if the modal did closed, after successful request
3. Check if the username on the top right, and “Login” button turn into “Logout”, and there is a username visible on the right.
4. Check if error handling is working correctly, try login with incorrect user information

PS: If there is something wrong you might want to redo the above step again.

After checking everything you could have noticed that everything works fine, unless you refresh the page. All the states you set are gone, although the cookie was set after login.

So, to update user information everytime the page is refreshed, the client must call a request to keep the data up to date.

Get User

Check if the user is logged in before by cookies, and if the value exists, you can checkout the backend code logic located in the "backend/src/routes/getUser.js" file within the backend folder.

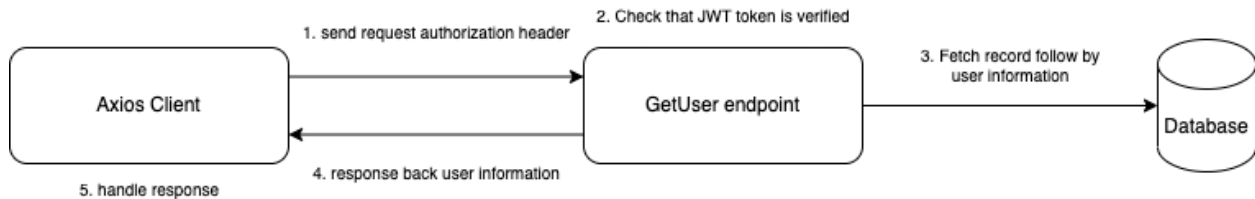


Figure 4.5: show overview of data flow

Overview :

What you need to do is check the cookie name “UserToken”, and if it exists make a request with an authorization header that contains “Bearer {your token}” to get user information. As always you can know the url through routing file in backend “backend/src/index.js”

```
app.get('/me', getUser);
```

Line 61th of index.js

Hands on :

In frontend code you need to work in the path “frontend/src/share/components/Navbar.jsx”. In the `useEffect` hook, you will see mentors’ comments, and steps required to finish integration.

Step 1: Check if cookie is set, or it is not undefined, so request is not call without correct data

```
useEffect(() => {
  // TODO: Implement get user
  const userToken = Cookies.get('UserToken');
  if (userToken == null || userToken == 'undefined') return;
```

Step 2: Send a request to the backend server, using the created axios client used in register and login.

So first import axios client that mentors provided on “frontend/src/share/AxiosInstance.js”

```
import Axios from '../../../AxiosInstance';
```

Place this line of code on the top section of current file

And next you need to make a request /me route, and in **GET** request, and also you need to set the authorization header to “Bearer {your token}”. And as you are going to call get request, the second parameter in the `Axios.get` function will not be the same as the `Axios.post` function, as GET request cannot send body in the request.

```
// 2. send a request to server
Axios.get('/me', { headers: { Authorization: `Bearer ${userToken}` } })
```

Step 3: handle response, if **success**, set user information.

```
// 2. send a request to server
Axios.get('/me', {
  headers: {
    Authorization: `Bearer ${userToken}`,
  },
}).then((res) => {
  // 3. if success, set user information
  setUser({
    username: res.data.data.username,
    email: res.data.data.email,
  });
});
```

If done, try to test it if it works flawlessly by inspecting the network tab, and see if the axios did call a request or not.

Things you need to check :

1. Check the first condition, cookie is set, and data is fetched.
2. Check if modal changes into login mode.
3. Check the second condition, cookie are not set, and request should not be made.
4. Check if the request are made and both username and logout button will be visible after the request is successful.

PS: if there is something wrong you might want to redo the above step again.

4.2. CRUD of a Note

This module is about CRUD of a note which consists of creating, reading, updating, and deleting a note. **However**, for this first blog you will implement only three APIs which is not include reading API since you already have a note data fetched from the API “view all note”.

Create Note

Create a new note record in the database based on the data in the request body. You can checkout the backend logic code located in the "backend/src/routes/createNote.js" file within the backend folder.

Overview :

You need to get data from the create note form and send **POST** request to the backend server. You can know the url through routing file in the backend

```
app.post('/note', createNote);
```

Path: backend/src/index.js

Hands on :

After opening the create form by clicking on the button name “Create Note”, the modal will appear and this is where you need to integrate.

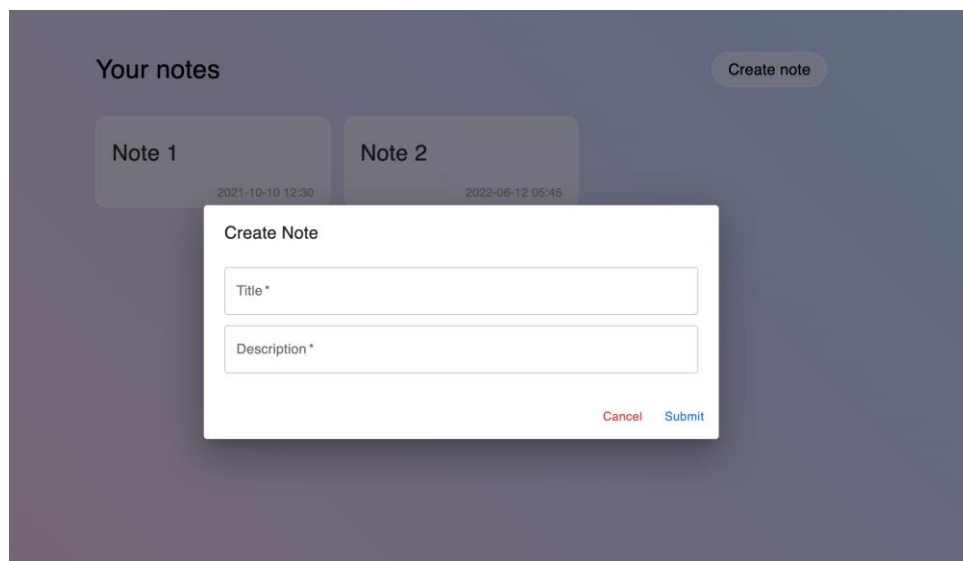


Figure 4.6: show UI of the create note form

Next, go to the frontend in the path

“frontend/src/pages/home/components/NoteCreateModal.jsx”. In the function `submit` you will see mentors’ comments and steps required to finish integration.

Step 1: Validate the form first by checking if all the required fields are filled. You have to create a function called `validateForm` for cleaner code.

```
const validateForm = () => {  
  const error = {};  
  if (!newNote.title) error.title = 'Title is required';  
  if (!newNote.description) error.description = 'Description is required';  
  setError(error);  
  
  if (Object.keys(error).length) return false;  
  return true;  
}
```

Then call this function in the `submit` method.

```
const submit = async () => {  
  // TODO: Implement create note  
  // 1. validate form  
  if (!validateForm()) return;  
};
```

Step 2: Send a request to the backend server.

You need to import axios client that mentors provided on “frontend/src/share/AxiosInstance.js”

```
import Axios from '../../../share/AxiosInstance';
```

Place this line of code on the top section of current file

Next, make an API request by taking three parameters. First is the url “/note”. Second is body that should contain `title` and `description`. And last parameter is object that contains a header object for attaching a cookie token of the user for authorization in the backend.

****** make sure you put `async` in front of the function first because using `await` in the function required that function to be asynchronous.

```
const submit = async () => {
  // TODO: Implement create note
  // 1. validate form
  if (!validateForm()) return;

  // 2. call API to create note
  const userToken = Cookies.get('UserToken');
  const response = await Axios.post('/note', newNote, {
    headers: { Authorization: `Bearer ${userToken}` },
  });
};
```

Step 3: Handle a `successful` response by adding new note to state and close the modal.

```
// 3. if successful, add new note to state and close modal
if (response.data.success) {
  setStatus({ severity: 'success', msg: 'Create note successfully' });
  setNotes((prev) => [...prev, response.data.data]);
  resetAndClose();
}
```

Step 4: If we get a **failure** response, then show an error message alert. However, axios will throw error if response has an error type. Thus we can to handle it using **try** and **catch**.

```
const submit = async () => {  
  // TODO: Implement create note  
  // 1. validate form  
  ...  
  try {  
    // 2. call API to create note  
    ...  
    // 3. if successful, add new note to state and close modal  
    ...  
  } catch (error) {  
    // 4. if create note failed, check if error is from calling API or not  
    if (error instanceof AxiosError && error.response) {  
      setStatus({ severity: 'error', msg: error.response.data.error });  
    } else {  
      setStatus({ severity: 'error', msg: error.message });  
    }  
  }  
}
```

Things you need to check :

1. Check if validation is working properly.
2. Check if a new note is added to the state and the modal is closed after receiving a successful response.
3. Check if error handling is working.

Update Note

Update a note record in the database based on the data in the request body. You can checkout the backend logic code located in the "backend/src/routes/editNote.js" file within the backend folder.

Overview :

You need to get data from the edit note form and send **PATCH** request to the backend server. You can know the url through routing file in the backend

```
app.patch('/note', editNote);
```

Path: backend/src/index.js

Hands on :

You can open the editing form by clicking on a note and you will see an *Edit* button on the bottom right corner, click it then you will see the form.

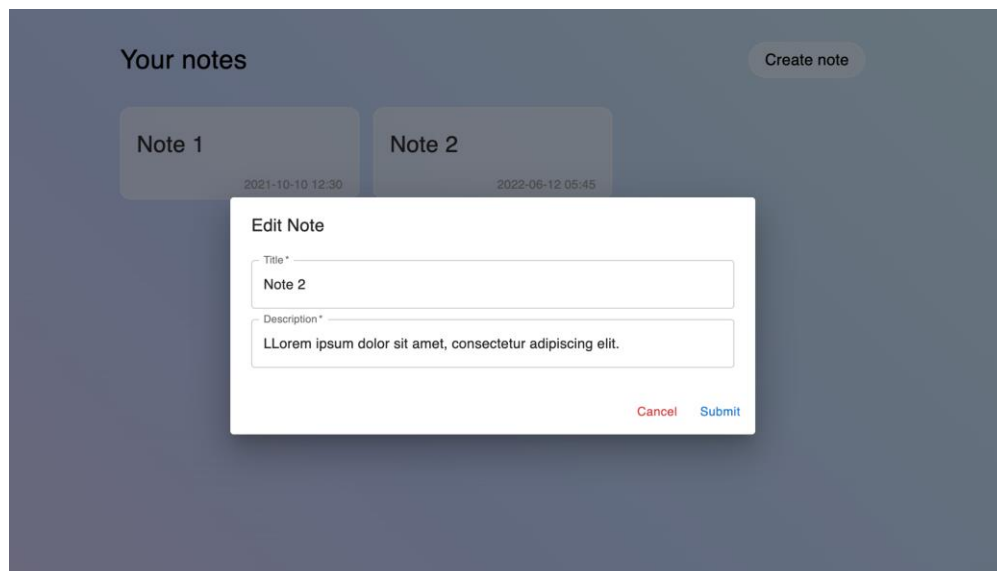


Figure 4.6: show UI of the edit note form

Next, go to the frontend in the path "frontend/src/pages/home/components/NoteEditModal.jsx". In the function **submit** you will see mentors' comments and steps required to finish integration.

Step 1: Validate the form by checking if all the required fields are filled as same as the creating form.

```
const validateForm = () => {
  const error = {};
  if (!newNote.title) error.title = 'Title is required';
  if (!newNote.description) error.description = 'Description is required';
  setError(error);

  if (Object.keys(error).length) return false;
  return true;
}
```

Then call this function in the `submit` method.

```
const submit = async () => {
  // TODO: Implement create note
  // 1. validate form
  if (!validateForm()) return;
};
```

Step 2: Send a request to the backend server.

You need to import axios client that mentors provided on “frontend/src/share/AxiosInstance.js”

```
import Axios from '../.../share/AxiosInstance';
```

Place this line of code on the top section of current file

Next, make an API request by taking three parameters. First is the url “/note”. Second is body that should contain `title`, `description`, and `noteId`. And last parameter is object that contains a header object for attaching a cookie token of the user for authorization in the backend.

****** make sure you put `async` in front of the function first because using `await` in the function required that function to be asynchronous.

```

const submit = async () => {
  // TODO: Implement create note
  // 1. validate form
  if (!validateForm()) return;

  // 2. call API to update note
  const userToken = Cookies.get('UserToken');
  const response = await Axios.patch(
    '/note',
    {
      title: newNote.title,
      description: newNote.description,
      noteId: newNote.id,
    },
    {
      headers: { Authorization: `Bearer ${userToken}` },
    }
  );
};

```

Step 3: Handle a **successful** response by updating the note in state and close the modal.

```

// 3. if successful, update note in state and close modal
if (response.data.success) {
  setStatus({ severity: 'success', msg: 'Update note successfully' });
  setNotes((prev) => prev.map(n => (n.id === newNote.id ?
response.data.data : n)));
  resetAndClose();
}

```

Step 4: If we get a **failure** response, then show an error message alert. However, axios will throw error if response has an error type. Thus we can to handle it using **try** and **catch**.

```
const submit = async () => {  
  // TODO: Implement create note  
  // 1. validate form  
  ...  
  try {  
    // 2. call API to update note  
    ...  
    // 3. if successful, update note in state and close modal  
    ...  
  } catch (error) {  
    // 4. if update note failed, check if error is from calling API or not  
    if (error instanceof AxiosError && error.response) {  
      setStatus({ severity: 'error', msg: error.response.data.error });  
    } else {  
      setStatus({ severity: 'error', msg: error.message });  
    }  
  }  
}
```

Things you need to check :

1. Check if validation is working properly.
2. Check if the note is updated to the state and the modal is closed after receiving a successful response.
3. Check if error handling is working.

Delete Note

Delete a note record in the database based on the id. You can checkout the backend logic code located in the "backend/src/routes/deleteNote.js" file within the backend folder.

Overview :

You need to get data from the edit note form and send **DELETE** request to the backend server. You can know the url through routing file in the backend

```
app.patch('/note/:noteId', deleteNote);
```

Path: backend/src/index.js

Hands on :

You will see the delete button by clicking on a note, it will be located on the bottom right corner.

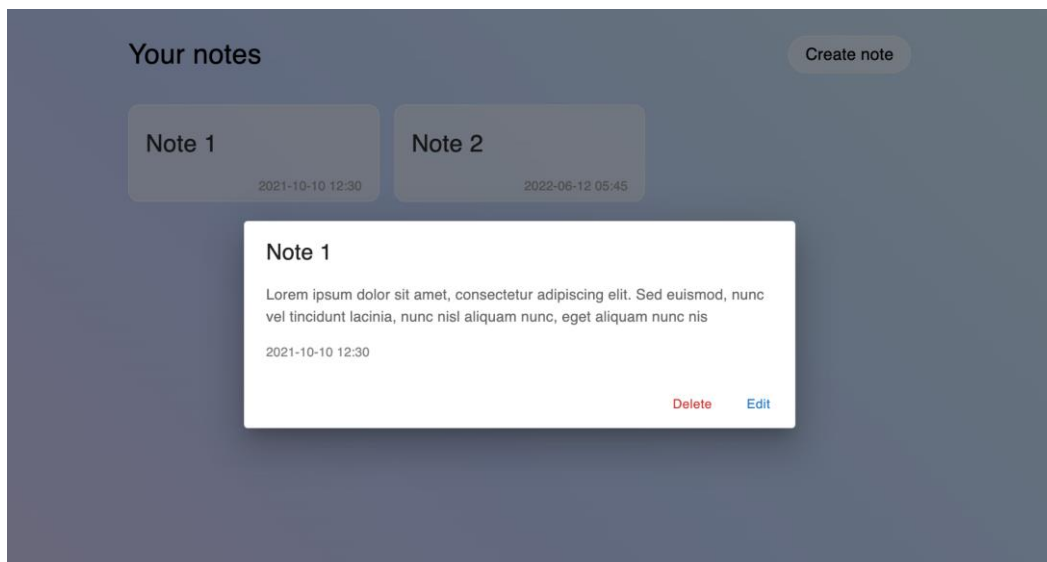


Figure 4.6: show UI where delete button is located

Next, go to the frontend in the path "frontend/src/pages/home/Home.js". In the function **handleDelete** you will see mentors' comments and steps required to finish integration.

Step 1: Send a request to the backend server.

You need to import axios client that mentors provided on “frontend/src/share/AxiosInstance.js”

```
import Axios from '../../../share/AxiosInstance';
```

Place this line of code on the top section of current file

Next, implementing the API by taking two parameters. First is the url “/note” and second is object that contains a header object for attaching a cookie token of the user for authorization in the backend.

****** make sure you put `async` in front of the function first because using `await` in the function requires that function to be asynchronous.

```
const handleDelete = async () => {  
  // TODO: Implement delete note  
  // 1. call API to delete note  
  const userToken = Cookies.get('UserToken');  
  const response = await Axios.delete(`/note/${targetNote.id}`, {  
    headers: { Authorization: `Bearer ${userToken}` },  
  });  
};
```

Step 2: Handle a `successful` response by removing the note from state and close the note modal.

```
// 2. if successful, set status and remove note from state  
if (response.data.success) {  
  setStatus({ severity: 'success', msg: 'Delete note successfully' });  
  setNotes(notes.filter((n) => n.id !== targetNote.id));  
  handleNoteDetailClose();  
}
```

Step 4: If we get a **failure** response, then show an error message alert. However, axios will throw error if response has an error type. Thus we can to handle it using **try** and **catch**.

```
const handleDelete = async () => {  
  // TODO: Implement delete note  
  try {  
    // 1. call API to delete note  
    ...  
    // 2. if successful, set status and remove note from state  
    ...  
  } catch (error) {  
    // 3. if delete note failed, check if error is from calling API or not  
    if (error instanceof AxiosError && error.response) {  
      setStatus({ severity: 'error', msg: error.response.data.error });  
    } else {  
      setStatus({ severity: 'error', msg: error.message });  
    }  
  }  
}
```

Things you need to check :

1. Check if the note is deleted from the state and the note modal is closed after receiving a successful response.
2. Check if error handling is working.