CADT IDT

W5 PRACTICE

React + Axios Integration

Name: Chea Ilong
ID: IDTB100022
G1 SE Gen10

At the end of this practice, you can

Apply APIs integrations between your existing APIs with ReactJS **Understanding** APIs integration with using different request methods

Get ready before this practice!

Read the following documents to understand the nature of Express.js:
https://expressjs.com/

Read the following documents to know more about Axios:
https://axios-http.com/docs/intro

Read the following documents to know more about Axios with React:
https://www.digitalocean.com/community/tutorials/react-axios-react

How to submit this practice?

Once finished, push your **code to GITHUB**
Join the **URL of your GITHUB** repository on LMS

EXERCISE 1 – APIs integration for Articles

*For this exercise you will start with a **START CODE (EX-1)***

Goals

Q1 – Display All Articles

Task: Build a component that fetches and displays all articles.

API Used: `GET /articles`

Axios REQUEST in ReactJS

`useEffect(() => {`
`  axios.get('http://localhost:5000/articles')`
`  .then(res => setArticles(res.data))`
`  .catch(err => console.error(err));`
`}, []);`

Q2 – View Article Details

Task: Show full content of an article using dynamic route `/articles/:id`.

API Used: `GET /articles/:id` Use `useParams()` from React Router.

Axios REQUEST in ReactJS
`axios.get(`http://localhost:5000/articles/${id}`)`

Q3 – Add New Article

Task: Create a form to add a new article.

API Used: `POST /articles`

Fields: `title`, `content`, `journalistId`, `categoryId`

Axios POST REQUEST to create an article
`axios.post('http://localhost:5000/articles', articleData)`

Q4 – Update Existing Article

Task: Prefill a form and update an existing article.

API Used: `PUT /articles/:id`

Axios POST REQUEST to update an article
`axios.put(`http://localhost:5000/articles/${id}`, updatedData)`

Reflective Questions

1. How did using `useEffect()` and `axios` help separate logic from UI?

useEffect() and axios separate logic from UI by handling side effects like data fetching outside the render cycle, keeping the component's display logic clean and focused.

2. What state challenges did you face while managing form input and API response?

Managing form input and API responses was new and challenging, so I had to read the docs a lot.

3. How does REST structure help React developers write cleaner frontend code?

REST provides a consistent structure for API endpoints, making it easier to organize frontend logic around clear resource-based calls, which keeps React components simpler and more predictable.

EXERCISE 2 – API Integration: Filter Articles by Journalist & Category

*For this exercise you will start with a **START CODE (EX-2)***

Goals

Use dropdown selections (<select>) to trigger filtered API requests
Understand sub-resource routes in REST (/journalists/:id/articles, etc.)
Dynamically render filtered content
Manage dependent state and conditional rendering in React
Practice clean UI/UX with form inputs

Context

Your frontend application should now allow users to filter articles based on:

- **A selected journalist**
- **A selected category**

Each filter option will call a different API route and update the displayed list of articles accordingly. You will not filter client-side, but instead make API requests based on selection.

Q1 – Fetch All Journalists & Categories for Select Inputs

Task: On component mount, fetch journalists and categories to populate two dropdown menus.

APIs Used:

- `GET /journalists`
- `GET /categories`

Axios GET REQUEST to fetch an journalists and categories

```
useEffect(() => {
  axios.get('http://localhost:5000/journalists').then(res =>
setJournalists(res.data));
  axios.get('http://localhost:5000/categories').then(res =>
setCategories(res.data));
}, []);
```

Q2 – Filter Articles by Selected Journalist

Task: When a journalist is selected from the dropdown, fetch articles written by that journalist.

API Used:

- `GET /journalists/:id/articles`

Axios GET REQUEST to fetch articles

```
axios.get(`http://localhost:5000/journalists/${selectedJournalistId}/article
s`)
  .then(res => setArticles(res.data));
```

Q3 – Filter Articles by Selected Category

Task: When a category is selected from the dropdown, fetch articles belonging to that category.

API Used:

Axios GET REQUEST to fetch articles
`axios.get(`http://localhost:5000/journalists/${selectedJournalistId}/articles`)` `  .then(res => setArticles(res.data));`

Q4 – (Bonus) Combine Filters (Frontend Side)

Task: Apply **combined filtering** on the frontend (e.g., articles that match both selected journalist and category) after fetching results from one of the filters.

- Adjust the backend to support combined filtering through query parameters if needed:

```
// Example: GET /articles?journalistId=1&categoryId=2
```

REFLECTIVE QUESTIONS

For this part, submit it in separate PDF files

1. **How do sub-resource routes like `/journalists/:id/articles` help in designing a clear and organized API?**
   Explain the benefits of using nested routes for resource relationships in REST APIs.

   Sub-resource routes like /journalists/:id/articles help design clear and organized APIs by showing the relationship between resources directly in the URL. They make it obvious that the articles belong to a specific journalist.

   Using nested routes in REST APIs improves readability, keeps related data grouped, and simplifies backend logic for fetching related records.

2. **What challenges did you face when managing multiple filter states (journalist and category) in React?**
   Discuss how you handled state updates and conditional rendering when multiple inputs affect the same output.

   Managing multiple filter states like journalist and category in React was tricky because each input could change or could not change the same output list. The main challenges were keeping states in sync, avoiding unnecessary re-renders, and making sure filters didn't override each other. Since it was new to me, I had to read the instructions more carefully and go through docs and resources to better understand how to structure the state and update logic properly.

3. **What would be the advantages and disadvantages of handling the filtering entirely on the frontend versus using API-based filtering?**
   Compare client-side filtering (in-memory) vs. server-side filtering (via query or nested routes).

   Client-side filtering is fast and simple, ideal for small datasets since it avoids extra API calls. But it doesn't scale well—large data can slow down the browser and increase load times.

   Server-side filtering works better for large or changing data. It fetches only what's needed, reducing memory use and keeping data fresh. However, it adds complexity and may introduce delays from extra API calls.

4. **If you needed to allow filtering by both journalist and category at the same time on the backend, how would you modify the API structure?**
   Think about adding query parameters or designing a new combined route.

   To filter by journalist and category, use query parameters like /articles?journalistId=123&category=technology for flexibility and simplicity. Alternatively, a nested route with queries like /journalists/:id/articles?category=technology clearly shows the relationship while allowing combined filtering.

5. **How did this exercise help you understand the interaction between React state, form controls, and RESTful API data?**
   Reflect on how state changes triggered data fetching and influenced the UI rendering logic.

   I understand how React state, form controls, and RESTful API data work together. Changing form inputs updates state, which triggers API calls via useEffect and axios. The backend returns filtered data, updating state again and causing the UI to re-render. Error handling is key to showing clear messages when API calls fail. This exercise helped me see how frontend and backend connect through state and data fetching.