Next.js: Backend-Frontend Communication and MUI

In this discussion we will build out communications between our backend and frontend. Additionally, we will build out the frontend using MUI.

Task 1: Create mock data

1. Create a file called **mock.json** at the root of the project
2. Fill out the file with some mock data following the following form. Feel free to copy and paste or make up your own data

```
{
  "mockPosts": [
    {
        "id": "id1",
        "title": "favorite class",
        "content": "cs391 is my favorite class. what's your favorite class?",
        "upvotes": 20,
        "downvotes": 1
    },
    {
        "id": "id2",
        "title": "restaurants in boston",
        "content": "what are some restaurants you recommend that are close to campus?",
        "upvotes": 12,
        "downvotes": 6
    },
    {
        "id": "id3",
        "title": "post-grad",
        "content": "i can't wait to graduate. what are your post-graduation plans?",
        "upvotes": 22,
        "downvotes": 28
    },
    {
        "id": "id4",
        "title": "cs class recs",
        "content": "what are some easy CS classes to fulfill group D?",
        "upvotes": 13,
        "downvotes": 9
    },
    {
        "id": "id5",
        "title": "study spots",
        "content": "what are some nice study spots at BU? i like cilse",
        "upvotes": 20,
        "downvotes": 3
    }
  ]
}
```

    a. If you choose to copy and paste, make sure all characters are copied correctly

Task 2: Write server-side functions to retrieve and create new mock posts

1. Create a directory with the path **/lib**
2. In this directory, create a file called **getAllPosts.ts**
3. Declare a function that will be used to fetch and return our mock data

```ts
import mockData from "@/mock.json";
import { PostProps } from "@/types";

export default async function getAllPosts(): Promise<PostProps[]> {
  return mockData.mockPosts;
}
```

4. In the same directory, create another file called **createNewPost.ts**
5. Declare a function that will be used to create a new post.
    a. Since this function will be called from a client-side component, we need to explicitly declare this as a server-side function
    b. the function **getAllPosts()** is also a server-side function. however, it will only be called from a server-side component

```ts
"use server";
import { PostProps } from "@/types";

export default async function createNewPost(
  title: string,
  content: string,
): Promise<PostProps | null> {
  const p = {
    title: title,
    content: content,
    upvotes: 0,
    downvotes: 0,
  };
  // insert into db (this will be done next week)

  return { ...p, id: "newId" };
}
```

Task 3: Use MUI to create a form for creating a new post

1. Install MUI using the command **npm install @mui/icons-material @mui/material @emotion/styled @emotion/react @mui/joy**
2. Create another component file called **new-post.tsx**
    a. To use the **useState** hook, this needs to be a client-side component. However, since it will be used within another client-side component, we do not need to explicitly declare it to be a client-side component

```tsx
import { Button, FormHelperText, TextField } from "@mui/material";
import { Textarea } from "@mui/joy";
import { useState } from "react";

export default function NewPost({
  createFunc,
}: {
  createFunc: (title: string, content: string) => Promise<boolean>;
}) {
  const [title, setTitle] = useState("");
  const [content, setContent] = useState("");

  async function submitNewPost() {
    if (await createFunc(title, content)) {
      setTitle("");
      setContent("");
    }
  }

  return (
    <form
      className="w-96 rounded-xl p-4 bg-sky-300"
      onSubmit={(e) => {e.preventDefault(); submitNewPost();}}
    >
      <TextField
        variant="filled"
        sx={{ backgroundColor: "white", width: "100%" }}
        label="Title"
        value={title}
        onChange={(e) => setTitle(e.target.value)}
      />
      <Textarea
        sx={{
          padding: "0.5rem",
          height: "100px",
          width: "100%",
          borderRadius: 0,
        }}
        variant="soft"
        placeholder="Content"
        value={content}
        onChange={(e) => setContent(e.target.value)}
      />
      <FormHelperText>What&apos;s on your mind?</FormHelperText>
      <div className="w-full flex justify-center">
        <Button
          sx={{ width: "80px" }}
          variant="contained"
          type="submit"
          disabled={title.length === 0 || content.length === 0}
        >
          Create
        </Button>
      </div>
    </form>
  );
}
```

b. Here is some documentation for the specific MUI components we use
    i. https://mui.com/material-ui/react-button/
    ii. https://mui.com/material-ui/api/form-helper-text/
    iii. https://mui.com/material-ui/react-text-field/
    iv. https://mui.com/joy-ui/react-textarea/

Task 4: Create a component to display a preview of all posts

1. Create a component file called **display-all-posts.tsx**
2. In this file we will use the **useState** hook. Since this function can only be used client-side and this component will not be called within another client-side component, we declare this at the beginning of the file

```tsx
"use client";
import { PostProps } from "@/types";
import { useState } from "react";
import PostPreview from "./post-preview";
import NewPost from "./new-post";
import createNewPost from "@/lib/createNewPost";

export default function DisplayAllPosts({
  inputPosts,
}: {
  inputPosts: PostProps[];
}) {
  const [posts, setPosts] = useState(inputPosts);

  async function addNewPost(title: string, content: string) {
    const p = await createNewPost(title, content);
    if (p === null) {
      return false;
    }

    setPosts([p, ...posts]);
    return true;
  }

  return (
    <div className="flex flex-col items-center">
      <NewPost createFunc={addNewPost} />
      {posts.map((p, i) => (
        <PostPreview key={i + p.title} post={p} />
      ))}
    </div>
  );
}
```

    a. Here we create a function to create a new post and append it to the list of posts. In this function we await the previously declared server-side function used to “actually” create a new post

Task 5: Display all posts

1. We will now edit the home page (**app/page.tsx**) to display all post previews instead of a single preview

```tsx
import DisplayAllPosts from "@/components/display-all-posts";
import getAllPosts from "@/lib/getAllPosts";

export default async function Home() {
  const posts = await getAllPosts();

  return (
    <div className="flex flex-col items-center bg-blue-200 p-4">
      <DisplayAllPosts inputPosts={posts} />
    </div>
  );
}
```

   a. In order to await the **getAllPosts()** function, we need to change the **Home()** function to be asynchronous. This is done by using the **async** keyword.

2. You should now be able to see all the mock posts from the **mock.json** file. You should also be able to create new posts, although they won't persist after refresh. We'll do that next week :)