Project VAJ 2025 - Summer

Assignment repository:
When creating teams, please put your school IDs into the name of the team (both members). 2 people per team. Thank you
https://classroom.github.com/a/DrudJBJd

Write a React single page application that will connect to an API server. The app should allow CRUD (Create, Read, Update, Delete) operations on a single type of resource (For example Blog Posts) and limited CRUD operations on another connected resource (For example Comments).

Your project will consist of 2 JS apps:

- Back-end (folder be) which will be NodeJS Express Server
- Front-end (folder fe) which will be React SPA application
- Both of them will live in the same repository

Example project:

I have a blog post application with posts and comments.

- On the homepage there should be welcome text describing the project and your name and a few things about you.
- On url /posts
    - It should show simple list of blog posts with their title and author (not the full blog post)
    - Each blog post entry should contain a link to the blog post detail
    - There should be a link for page where you can create new post
- On the detail page of single post (on url /posts/<someID>)
    - It should show the full blog post (Title, Text, Author)
    - We should see all comments associated with the Post and there should be a way to post a new comment.
    - There should be a link to a form that allows editing the blog post
    - There should be a link (or button) to delete the blog post
- Throughout the whole app, I should see a navigation that allows me to navigate to the list of blog posts, or the home page

You can pick any combination of 2 resources. For example: (Movies & Actors, Trains & Train Wagons, Authors & Books) come up with your own! In a worse case scenario you can use Post/Comments but try to come up with something different.

Other info:

- We have to be able to run your project!
    - This means that if I do `git clone` there will be readme, with simple steps I need to do to run the project. Preferably `cd xx  && npm i && npm start` and the project runs.
    - Before final hand in, try your project in a new folder (preferably new environment, that it really runs - try cloning from repository)
    - Make sure that your app is initialized with some data or has some initialization scripts so we can see it in full glory.
- Other technologies for Database
    - You can pick any technology for the DB. We recommend using SQLite, but if you want to utilize MongoDB, Firestore, DynamoDB or anything like that, you can do so. However we need to be able to run the project on our machines and you need to fulfill the detailed requirements.
    - If you pick a “different” DB but we have to be able to run this on our machines in docker or utilize cloud DB. So for example if you want to use PostgreSQL there needs to be just a few commands how to run this using docker.
    - If you don’t know what docker is - stick with SQLite or cloud DB.

Total evaluation:

- 40 points
- 5 extra points (complexity, project idea, exceptional execution in any area)

Detailed evaluation:

- Back-end (NodeJS):
    - 5 Functional requirements
        - At least 2 resources (2 tables connected to each other via key)
        - All CRUD operations on one resource
        - At least 2 operations on second resource
    - 5 ORM Layer & DB schema
        - Connection to the DB via a ORM library
        - Error handling
    - 5 Express REST API layer
        - Valid API up to the REST specification for all operations on resources
        - Status handling & error messages
- Front-end (React):
    - 5 Functional requirements
        - App Layout with Navigation (Routing)
        - Main page with information about the app
        - All CRUD operations on one resource (Create Form, Edit Form, List/Table, Delete, Detail View)

        - 5 Component structure
            - Split your application into several components. Don’t make huge ones
                - *Each page should consist of several components*
                    - Top level component for the page
                    - “Smart” component with the business logic
                    - Presentational components for the UI
        - 5 Client-side routing
            - There should be proper routes for all pages
            - Best practices according to react router
                - Use loaders and actions
                - React router version 6+
        - 5 Data fetching
            - Custom API functions
            - Proper loading states
            - Proper error handling / states
                - *For example when you start up just FE without BE*
    - Overall:
        - 5 Code quality / project setup
            - README, Prettier, Able to run the whole project easily!

Must use:

- React
- NodeJS Express

Recommended (we worked with them on labs):

- Routing - React Router ⚠️Highly recommended. React router must be v6 ⚠️
- ORM - Prisma
- UI Component Library - MUI
- Code formatting - Prettier
- API requests - Axios or Fetch

Libraries which you can check out:

- Easier Forms - Formik
- UI Libraries - Tailwind, Mantine, ChakraUI, Antd, Semantic UI
- API requests - Native JS fetch
- Routing - React Location
- Data Fetching - TanStack Query
- Your project can be written in TypeScript!