Lecture 4

Frontend SPA development, frontend deployment

Frontends

- So far our application doesn’t offer any good way for users to interact with it
  - End users will generally not use Postman.
  - End users will want a friendly user interface.
- We will add a frontend that runs in the browser.
- We will also deploy the frontend.
- The frontend will be written in Javascript or Typescript.

Options for frontends

- Many frontends are SPAs - Single Page Applications:
  - They are called this because the page never changes, only the displayed route and the contents change, but the browser only loads the page once.
- We have many options for writing a frontend, most with very good documentation that you can find online:
  - The **React** library: we'll discuss it today.
  - The **Angular** framework: we'll discuss it next course.
  - Vue, Svelte etc..

The frontend should be independent

- We use REST APIs so that our frontend is not tightly coupled with a specific backend implementation.
- You shouldn’t be looking at your backend code much when writing the frontend. You should implement Swagger and look at that.
- Many times you won’t be writing frontends for backends that you fully control.
- If something on the backend needs fixing, it should be fixed.

Swagger

- Swagger is a tool that documents a REST API automatically.
- You should set it up so that you have an easier time writing the frontend and so that other people that might be using your API know how it functions.
- Most frameworks have some form of Swagger support:
  - For Django, use https://github.com/tfranzel/drf-spectacular - others are no longer maintained.

Basic frontend principles

- Use the idiomatic way of doing things in your framework - don’t reinvent the wheel.
- Use different routes (we will also call these pages) for different things.
- Make things user friendly and easy to navigate.
- Align things properly.
- Think about the user experience.
- Have a menu for navigation.

Deployment of the frontend

- We will use netlify for automatic deployment: https://www.netlify.com/.
- Amazon has a similar service and there are probably others as well.
- Just link it to your github and it will deploy automatically once you push to your repository.
- Be advised that these services don't like the HTTP we're using for the backend. They want HTTPS. That's a bit trickier to setup, but there's usually a hack we can do.
    - Or you can look into configuring HTTPS with Nginx.

Some concrete frontend tips

- For show all:

#	Name	Faculty	Number of courses	Operations
1	Student 1	Computer science	4	
2	Student 2	Psychology	6	

Note:

- The # column does NOT contain the ID
- Objects are hyperlinked to their details page
- There is an aggregate count on the related entity (courses)
- There are quick links to operations: details, update, delete
- Delete should always ask for confirmation.

Some concrete frontend tips

- For details:
  - It should be on another route.
  - Show more data, such as the actual courses.
  - Put in the Update and Delete operations as well.
- For update:
  - Populate textboxes with the existing data.
  - Make sure you validate things again.
  - The PKs are never updated and the IDs are never updated directly.
- For delete:
  - Always ask for confirmation and only execute the delete if the user confirms.
- For add:
  - The user should never be asked for any sort of ID.
  - When adding a Student you might need their Faculty. This can be a dropdown from which you select it.

Some concrete frontend tips

- Make sure you show the user feedback for every action: whether there are errors, whether it was successful etc.
- Do **NOT commit your node_modules** folder!
- Your project structure should look similar this - one parent folder with two subfolders: one for the backend and one for the frontend. Commit the parent folder to git:
  - **AppName**
    - **Backend**
    - **Frontend**

About React

- We will be using React with Typescript, Functional Components and Vite: https://vitejs.dev/guide/why.html
- We will also be using the MUI components library: https://mui.com/material-ui/getting-started/overview/
- Why React?
  - It's functional in nature, so something new compared to all the OOP you've been doing
  - It's a lightweight library that's easy to get started with
  - Lots of jobs in it
  - You're free to use something else if you don't like it
- Some disadvantages:
  - Not very good official documentation, must rely on third party sources
  - Less out of the box stuff compared to Angular

React CRUD example

Let’s do an example together. Watch the recording if you didn’t attend the lecture and remember that you can find the code on github.