IMY 220 Project 2025

Version Control Website

Deliverable 2

Instructions

- This deliverable must be **completed by 07:30 on 3 October** as indicated on ClickUP.
- You must demo your deliverable submission in the demo sessions in order to be awarded marks.
- No late / email submissions will be accepted.

We recommend you constantly refer to the overall specification as well as your individual deliverable specs to make sure your project meets all of the requirements by the end of the project and for each deliverable.

It is crucial that you continuously test your project as you add new functionality to make sure everything keeps working as expected, especially with Docker**.**

The focus of this deliverable is to start working on the backend, interactivity of your project and integration of your backend and frontend to form a working website. This includes setting up the database schemas, collections and documents in MongoDB, creating your own API to interface with this database, as well as calling your API and loading data asynchronously into your project. Additionally, some of the basic styling for your project (fonts, colours) should be implemented.

Part 1: MongoDB Database & API

By now, as you have been building the skeleton structure for your functionality, you should have a good idea of the type and quantity of data that you will need to pass around your components, and what kinds of data are attached to a user, a project, a check-in message, etc.

Create a MongoDB database that has the required collections to represent your overall data structure for your project. How you structure this database is up to you. For example, you may want to create a “user” collection to store information about each of your users (profiles), a “projects” collection to store projects, and a “friends” collection to store a user's friends.

Think carefully about the relationships between your various collections and use them to structure your database so that it’s easy to connect them. For example, a user owns projects, which would have check-in messages within them; a user can create projects, add members to projects, and leave a check-in message. You’ll need to think about how you will store and pass around this data so that it is easy to

Create, Read, Update, Delete (CRUD) data in your frontend code. **Your database should have at least 2 user profiles, 2 projects, and 3 check-in with the corresponding messages.**

You should fully implement your backend before you think about integrating it into your frontend. You should make use of Express to create routes for each CRUD action for a specific piece of data, you should already have stubbed out your authentication endpoints. Tools like Postman (postman.com) or Hoppscotch (hoppscotch.io) can be used to test your backend code to ensure that it works as expected. Expand on these routes by adding error handling, and more complex functionality until you have a fully implemented API that you can call within your frontend React code to help you pass data from the frontend, to the backend database, and vice versa. You should be making use of JSON in your requests and responses.

> *You should ideally encapsulate this database communication either as exported functions or as an exported class. What we mean by this is that you should define CRUD actions for each collection separately and call upon them when you need to do something. This follows the DRY principle of coding and separation of concerns.*
>
> This API code (with all of your MongoDB communication) should live in your /backend folder.
>
> In this way, your backend (database) communication is handled by your API, and your React code focuses on rendering UI, client-side user interaction, and client-side validation and calls your API. You should never interact with your MongoDB database from your React code, all CRUD actions should take place in your API.

Make sure you include a working MongoDB connection string when submitting. If you include a non-working connection string and the markers cannot access your MongoDB database, you will lose marks. You can either use a MongoDB Atlas instance (MongoDB Atlas) - which is preferred as you don't have to worry about setting up and managing the database - or a self-hosted version of MongoDB using Docker (MongoDB & Containers) - if you decide to go this route, you will need to provide your data and a way to import the data into the self-hosted instance.

Part 2: Linking Frontend and Backend

Once your API is fully implemented and working as intended, start implementing the integration between your backend and your React frontend. You would achieve this by making use of the API routes that you defined in your backend and calling them by using the Fetch API (Fetch API on MDN) in the required components/pages (or any alternative method from the lectures that makes use of a promise).

By the end of this deliverable, you should have no dummy data anywhere in your components. You should make use of the relative API routes to receive data, render this data in your components, and be able to send data to your backend from your React components/pages and have it updated in the respective document. Essentially, you are performing all CRUD operations via the backend that you created.

You should start fully fleshing out your core functionality. You do not have to have all of the error handling implemented yet, but the basic functionality listed below should work.

The following needs to be implemented on your frontend, with the respective API routes available on your backend:

- **User Management:** Log in, sign up, and log out.
- **Profile Management:** View and edit your own profile.
- **User Interaction:** View other users' profiles, and send or accept friend requests to connect with them. You can also unfriend other users.
- **Project Creation:** Create a new project, adding files, a name, a description, hashtags, and an image.
- **Project Viewing:** Any user can view project details, including name, description, image, type, and hashtags. Any user can also view project activity and download project files.
- **Project Collaboration:** Check out a project to make changes, and check it back in with updated files and a message.
- **Project Administration:** Manage and delete your own projects.
- **Activity Feeds:** View a local activity feed of your friends' and your own activity, or switch to a global activity feed of all users.
- **Search:** Search for projects or users. You can search for users by name, username, or email and for check-ins by message, project type, or hashtags.

There is a specific functionality that each of these pages/components should consist of. You should refer back to the main project specifications to find out what these are and cater for them.

You do not need to implement any functionality related to admin users.

Part 3: Basic Tailwind Styling

For this deliverable, you are required to have started to implement the general theme of your website. This refers to the styling you decided upon in D0 in terms of the colour scheme, general layout (i.e., placement of elements), and fonts. You are not required to have it finalised, but there should be no default HTML or TailwindCSS fonts, no default Bootstrap components, and your colour scheme should be implemented. Your styling should be visible across all of your pages and components.

Everything should be created by you and no templates are allowed. This includes pre-built components you can find online or from component libraries.

Submission Instructions

Submit the following to ClickUP before the deadline:

Place the following into a ZIP archive called `Position_Surname_D2.zip`. Double-check that all required files are included before you submit.

- A text file containing a link to your GitHub repository.
- All files / code excluding node_modules.
- Your Dockerfile and all commands used to create the image and container. Include the commands in a file called README.txt for easy access.
- Your MongoDB atlas connection string to your complete database - or commands and data required to self-host the database.
