# IMY 220 Project 2025

## Version control website

The project must be completed by Thursday **30 October**.

We recommend you constantly refer to **this spec as well as your individual deliverable specs** to make sure your project meets all of the requirements by the end.

Make sure you also **continuously test** your project as you add new functionality to make sure everything keeps working as expected.

# Contents

# Objective

Create a simple version control website focused on code that allows users to share and update files in shared projects, view other projects, add friends, collaborate, etc

**You must use the technologies and techniques specified in the project spec section to write your own code.**

Your website must have a unique visual design that fits with the theme of the website (i.e., a version control website focused on code). You are not allowed to simply use another website's design or create a "knockoff" of another website when designing the visual theme of your site. Note that you also don't have to use the terminology used in this document to describe functionality, such as "project", "check-ins", "discussion board", etc. You may name these however you want based on the theme of your website.

The purpose of the project is for you to demonstrate the skills that you have learned in IMY 220 - **not anything else**. Therefore, you will **NOT** get marks for proving your proficiency with any other skills OR for using code that is not your own (e.g., from other libraries or templates).

# A Note on AI Use

AI-generated content is **NOT** allowed as a substitute for your own code. **Do not use AI to generate code for your project**. The scope of the project is too large to generate code that is reliable / functional. You will also find it difficult to understand how your code works, which will be detrimental as you **will be required to explain** parts of your project during your deliverable demo sessions. If you are unable to explain how your code works you will get zero for that section.

# Submission Instructions

- ***Late projects will not be accepted.***
- ***Email project submissions / Google Drive links will not be accepted.***
- You are required to create a **GitHub repository** to store all of your project files and have some form of source control. The repository must be set to **public** so your code can be marked. The link for this repository should be submitted on **ClickUP** along with the relevant Docker files for each of the deliverables. You will still be required to submit all of your project files for the final deliverable as well as the GitHub link.
- Include a **readme.txt** that includes all commands that are required to build and run your project, as well as a basic explanation of your directory structure.
- You have to upload your files to **GitHub** as well as **ClickUP**. The markers should be able to download from either GitHub or ClickUP and it should work the same.
- ***IMPORTANT****: Please do not move your files around to different directories (in order to fix the directory structure) after you have finished writing your code. This can result in your code breaking.*

## ClickUP

The submission link on ClickUP will close at **exactly 23:59 on Thursday 30 October**.

- **Do not wait** until just before 23:59 to upload because your submission might take longer than usual to upload.
- Instructions:
    - Place: **all your project files**, along with your Docker files and your exported MongoDB database in a folder named with your Position_Surname. Your Position can be found on ClickUP.
    - Compress the **FOLDER** using a zip archive format.
    - Ensure that any **node_modules** folders are **NOT** included in your ZIP folder.
    - Upload the compressed folder (**Position_Surname.zip**) to ClickUP to the relevant project link.

***IMPORTANT: Do not expect the lecturer to immediately respond to communications around the submission time. It is your responsibility to make sure you submit all the relevant files well before the submission time.***

# Project Demo

You will have to demo your project during the demo sessions to get marks for it.

- You will have to demo using either the latest version of either Google Chrome or Mozilla Firefox.
- When you demo your project, you will be required to show how you have implemented all the instructions by demonstrating the functionality **on the website itself** (not just in the database).
    - You will receive marks for achieving working functionality according to the instructions. For example, if you are required to implement functionality that modifies and displays database data on the website itself, you will receive no marks for simply modifying the database.
- Demo sessions will be made available on ClickUP closer to the deadline.
- Demo sessions will take place on **Monday 3 November** and **Tuesday 4 November**.
- Project demos will take place in the **Immersive Technology Lab on the 4th floor of the IT building (IT 4-62)**.
- ***You will receive no marks if you do not demo your project.***

### *You will **lose marks** in the following conditions*

- If your system does not have all the required functionality.
- If you use any other technology or technique that is not part of the course.
- If you use code that is not your own, i.e., if you use code generators or templates, you will not get marks for it.

### *You will get **no marks** in the following conditions*

- If you submit work that is not your own.
- If you do not upload a Dockerfile / Docker Compose file.
- If the Dockerfile / Docker Compose file does not run the application.
- If you do not upload to ClickUP.

# Main pages

All pages must clearly display the name of the website creatively and have a unique visual design.

## Splash page

- When an unregistered user or a user that is **not logged in** visits the website, they must only be allowed to see a "splash page".
    - *A splash page is the webpage that the user sees first before being given the option to continue to the main content of the site.*
- Splash pages are used to inform new users about the services provided by the website and to promote it. You must thus include a tagline or other information that explains the contents and purpose of the website.
- The splash page is the only page that does not have the same consistent layout as the rest of the webpages.
- The splash page must contain at least the name of the website as well as a log in and registration form.
    - Users should be able to register using their email address
- ***BONUS****: catch new users' attention and show off your design skills by creating a well-designed landing page that explains the goal of your website. A landing page is generally more detailed compared to a simple splash page and is meant to sell the service you are offering.*

  *Divide this page into well-designed sections and use the page scroll to trigger visual effects when the user scrolls down. How you make use of page scroll is up to you.*

  *(You can find examples of this here: https://www.awwwards.com/inspiration_search/?text=Scrolling )*

## Home page

- When a user is **logged in**, the home page must contain the **local activity feed** for that user in reverse chronological order.
- A user should then be able to switch between their local activity feed (that displays the activity for all that user's friends) and a global activity feed that contains the activity for all users.
- By default, all activity must be sorted by creation date in reverse chronological order. Users should also be able to sort their activity feed in some useful way, for example, by popularity based on number of file downloads.

## Profile page

- A user should be able to go to their profile page when they are logged in.
- The profile must have a profile image.
    - ***BONUS****: allow users to add an image using drag-and-drop.*
    - ***BONUS****: come up with a way to randomise the placeholder image so that each one is different in some way. This should go beyond simply having a few images to select from.*

- A profile must display appropriate personal information about a user, e.g., birthday, work, connections, contact info, relationship, etc. Look at other platforms/websites for ideas/guidance on what to include.
  *Note: Since all this information will be publicly visible, no private information must be shown.*
- All the user's activity and a list of the projects they are the owner/member of must appear below the personal information.
  - ***BONUS**: show the user's favourite programming languages in a tagcloud. A tagcloud shows all the tags in some stylised way and visually represents the number of occurrences for each tag in some way, for example using font size and/or colour.*
- A user should be able to edit appropriate information in a logical manner.
- When you visit the profile page of another user, it must clearly indicate if you are connected (if you are "friends" on the website) or there must be some way to connect with that user (to send them a "friend request").

# Activity and projects

The website should show two activity feeds: a local activity feed and a global one. Activity in this case refers to check-outs, check-ins, and the messages accompanying the check-ins (as well as which members performed these actions).

The local activity feed consists of all the activity for projects that the user is a member of, while the global activity feed consists of all the activity for all users on the website. The activity on the activity feed must contain a relevant image and appropriate information for the project for which activity has taken place. Activity should be displayed in an **informative and intuitive way**, e.g., "[user] checked in a project called [project name]" with the relevant information and project's associated image.

Activities must be displayed in an **aesthetically appealing way**. Simply listing the pieces of activity in Bootstrap cards underneath each other is **not** sufficient; find inspiration from other websites/applications that use news feeds, activity feeds, etc.

## Projects

A project consists of a collection of files which represents some programming application, framework, etc., i.e., anything that contains one/more code files and/or other files such as media files.

- Any user can create a project, which would then make them the owner of that project. When creating a project, the owner must be able to do the following:
  - Add any number of files to the project (this would technically constitute the first check-in).
  - Fill in the following required details: name, description.
    *Note: All form fields must have working labels.*
  - Add a list of programming languages involved in the form of hashtags. These are all added manually.
    *Note: A hashtag is an interactive piece of text that is created by prefixing a predefined symbol to it, e.g., #hashtag. When a user clicks on a hashtag, it must be the same as*

*performing a search for that hashtag, i.e., it must display all other projects that have the same hashtag (i.e., use the same programming language).*

- Add a **type**, such as desktop-, web-, or mobile-application, framework, library, etc. The owner must be able to select a type by selecting it from a selection of types (e.g., by using a dropdown list).
  *Note: A user must not be able to add their own types, only the admin is able to do this.*
- Set the project version. This will also be updated when checking in new project files. You must come up with a sensible way to do this.
- A project-creation date must also be added automatically from a timestamp, i.e., the user should not be able to add this field manually.
- Upload an image for the project. The image should be displayed as a smaller version, which is enlarged when clicked on. The maximum size for the enlarged image must be 5MB. The system must limit the user from uploading files larger than 5MB e.g. by allowing them to upload a smaller image (instead of just giving an error message).
  - ***BONUS**: allow users to add files and images using drag-and-drop.*

- A project also has a status and details which are updated when the project is being worked on. This includes a status (checked in/out) and a feed of who checked the project in and out as well as their check-in messages.
- A single project can only have one owner, but can have any number of members.
- The owner can do the following:
  - Change the following project details: name, description, image, type.
  - Remove members from a project.
  - Relinquish ownership, in other words, make another member the new (sole) owner of the project.
  - Delete the project. Deleting a project also removes everything associated with that project, such as activity, discussion board, etc.
- All project members (including the owner) can do the following:
  - Add/edit files by checking them in/out.
  - Add other members to a project. Members can only add their friends to be members on projects they are on.
  - View and edit project discussion.
- Any user (not just project members) can do the following:
  - View the following project details: name, description, image, type, and hashtags.
  - View the activity for the project, i.e., check-ins, check-outs, and check-in messages.
  - View and download the files for the project (including when the project is checked out). See downloading vs checking out below for more details.
  - ***BONUS**: graphically represent project activity and statistics, such as who checked the project in and out, frequency of project activity over time, etc.*
  - ***BONUS**: allow users to view the contents of code files in the browser. This should go beyond simply linking to the file and should display it in an easily-readable format that is integrated into the design of the website.*
- ***BONUS**: instead of manually adding hashtags for a project, auto-generate the list of programming languages for a project based on the files that are in the project. You may use file-extensions to accomplish this (you don't have to check the contents of files). However, you cannot simply list the file*

*extensions in hashtags, you must list the programming languages they correspond with, for example, JavaScript for .js, Python for .py, etc.*

## Checking out the project

Check-out is similar to checking out a book from a library: when the book is in the library, anyone can check it out, but once it's checked out, only the person who checked it out can access it. Checking out a project thus "locks" the project files from being checked out by other members until the project is checked back in.

- Any project member can check out a project. This is the only way to make changes to a project's files.
- They can only do this if the project is currently checked in.
- Checking in and out is done on a project scale, i.e., on all files in the project, not individual ones, since editing files that are dependent on other files that are being edited by someone else could cause issues.
- Checking out a project downloads or gives access to download all files for that project.

## Downloading vs. Checking out

Note that simply downloading a project is not the same as checking out: downloading the project does not "lock" the project from being checked out by other members, but gives access to the project files. Checking out thus implies downloading files, but downloading does not imply checking out.

## Checking in the project

Checking back in is again similar to checking a book back in. The book is returned and everyone is given access to the book again with the option to check it out.

- Only the member who checked out the project may check it back in.
- When checking in a project, the owner must be able to add any number of files to the project. (This can be any files; you don't need to keep track of missing/changed files). File uploads must be done via a web form. You must come up with a way to do this that is sensible, intuitive and easy to use.
- When checking in a project, the member also has to provide a message describing the changes enacted by their check-in. This goes into the activity for the project. The member checking in a project must also be able to update the version number of the project.
    - ***BONUS**: implement a version history for projects which allows them to roll back check-ins to previous versions. In other words, if a member made changes, another member should be able to easily revert the project to the state and version number it was in before the other person made changes.*

# Search

Add search functionality that allows a user to search for the following:

- Other users by name, username, or email address.
- Check-ins by check-in message, project types, or hashtags (i.e., programming languages)
- ***BONUS**: The search functionality must be able to find something **even if the search term is a little incomplete or spelled incorrectly**, i.e., fuzzy string searching: the technique of finding strings that match a pattern approximately (rather than exactly).*

  ***Note**: this does **not** refer to simply adding wildcards to your searches. A fuzzy search should be able to find words that have letters missing, swapped around, etc.*

- ***BONUS**: Add **autocomplete** functionality to suggest results as the user is typing. You would need to come up with an intuitive design for this.*

*Search results must be interactive, i.e., where appropriate they must return links, for example, to user profile pages, clickable hashtags, etc.*

# User Roles

## The Unregistered User

An unregistered user can do the following:

- Register as a new user on the splash page. The splash page does not offer any functionality other than allowing the user to register or login.
  *Note: An unregistered user cannot interact with the website in any other way.*

## The Registered User

A registered user can do the following:

- Log in and log out.
- See all activity from themselves and all their friends (local activity feed) as well as from all users (global activity feed).
- Search for projects or users and view other users' activity.
- Edit all appropriate fields on their own profile.
- Create new projects.
- Manage (edit/delete) their own projects (how projects should be handled is discussed under "Projects").
- Delete their own profile. Deleting their profile removes their user account from the database and deletes any projects they created.
  - ***BONUS**: Instead of deleting projects that the user created, come up with a sensible way for projects to change owners if the owner of a project deletes their profile or is deleted by the admin.*

## User Interaction

A registered user can do the following:

- Connect with other users by sending a friend request to connect.
- View the complete user profiles of users that they are friends.
  *Note: When a user is not friends with another user, they can only see that user's name and profile picture.*
- Unfriend other users.

## The Admin (user account)

Some users should be able to log in as admin. The administrator has all the powers of a registered user.

Additionally, they can do the following:

- Manage (edit/delete) all activity.
- Manage (edit/delete) all projects, including their uploaded images.
- Manage (edit/delete) all user accounts.
- Add new project types.

- ***BONUS**: add functionality for admins to **verify accounts**. A (non-admin) user should only be able to request that their account be verified if their account is more than one week old, if they have added at least one project, and checked out at least one project. Admins should have access to a list of verification requests that they can approve/deny on a case-by-case basis. Once an account is verified, it should be visually indicated on the user's profile somehow.*

# Usability

- *Take care to design a website that is **intuitive and easy to use**. It is your own responsibility to come up with usable and intuitive interactions for the website's functionality.*
- *For example:*
  - *A log out button must not appear if a user is not logged in, or a login button must not be visible if a user is already logged in.*
  - *When a user **creates** an account, they must **automatically be logged in** with that account.*
  - ***All form fields must have working labels**, i.e., when you click on the label, focus is given to its accompanying input.*
  - ***Well-designed error messages should appear when appropriate**, e.g., when a user enters incorrect login details.*
  - ***Navigation must stay consistent** throughout pages and must provide a **logical way to access the core functionality**.*
  - *Text that is interactive must **look interactive** (buttons, links, hashtags, etc.).*
  - ***Clicking on the website logo/name must take the user to the home page**.*

- ***Tip**: Test the usability of your site by asking other people (especially non-IT students) to interact with it to find out how easy it is for them to navigate.*

- ***You will be penalised for sloppy design that is detrimental to the user experience, such as debugging alerts (or any unnecessary alerts)***.
- ***BONUS****: allow the user to toggle between a standard (light mode) and a dark theme (night mode).*

# Visual Design

- Your website **must have a design that is up to professional standards**. Refer to well-designed websites for inspiration, e.g., https://www.awwwards.com, https://dribbble.com/shots/popular/web-design, etc.
- **Use colour schemes that match**, e.g. browse the colour lovers website (https://www.colourlovers.com/) or use Adobe Color (https://color.adobe.com/) or Coolors (https://coolors.co/) to find colour palettes and patterns or search online for colour scheme creators.
- You **must** redesign all form elements to fit with your website's visual theme, in other words, you are **not allowed to have any default HTML/Bootstrap** input boxes, buttons, etc.
- You must include **at least two and maximum three fonts** as part of your website's design. These may **not** be the default fonts for HTML, Tailwind, or Bootstrap. Use fonts that look good and fit the overall design of the site. **You must embed fonts for your website. Consider using something like Google Fonts for this.**
  ***Note****: Fonts must be legible. Do not use more than three fonts for the entire website. Do not use stylised fonts for body text. Do not use standard Windows fonts. Embed fonts with CSS / Tailwind CSS.*

# Technologies and Techniques

**You may only use the following technologies and techniques:**

Technologies:

- **Valid JavaScript code everywhere** (no exceptions). This includes any JSX and additional styling / class names that you may include in your application. Ensure that valid HTML is returned when the application is viewed in the browser (view source), i.e., ONE doctype, ONE head, ONE body.
- **CSS or TailwindCSS** for styling, embedding of fonts, etc. Do not use inline or embedded style sheets. Use CSS variables where appropriate. No component libraries (like Material UI / shadcn / ChakraUI) are allowed.
- **CSS cascading variables** for aspects relating to the style identity of your website, e.g., your theme colours and fonts.
- You should use **GitHub** throughout the semester and submit a link to your repository for each deliverable.
- You are permitted to use Bootstrap but you will need to customise the styling of the components. You will have to use either CSS or TailwindCSS to style these elements according to your website theme. **You will lose marks if any default Bootstrap is styling visible on your website**.
- **JSON** to transfer your data back and forth between client and server (for example, during fetch calls).
- At least one instance of a Promise for asynchronous behaviour (this can include the built-in async behaviour of using fetch). No third-party request plugins (Axios, ky) are allowed, and you should use the native Fetch API.

- ***BONUS**: demonstrate branching in Github by having different branches for logically separated website content, e.g., having a branch with a README file explaining your project, having a branch dedicated to adding certain features. You must have **at least one branch that has been merged back into the main branch** to get the bonus.*

Techniques:

- **Responsive web design with at least two breakpoints.**
  ***Note**: Your website must display correctly on any resolution.*
- **Error-free** React and NodeJS code. Sensible coding, you must only connect to your database in one place. Keep your code DRY (**D**on't **R**epeat **Y**ourself).
  - Separate your code into reusable components. For example, one component for an add / edit form, one component for viewing a playlist summary, one component for the context menu etc.
- **Session / cookie management**.

# Favicon

- **Embed** a favicon that fits the theme of your website in all your pages.

# Database

- You are permitted to use MongoDB Atlas, hosting your own instance of MongoDB is optional.
- There must be **LOGICAL** test data in the database for demo purposes.
- There must be *at least* the following:
  - 10 projects in the database, including images. Most of these projects should have multiple members.
  - 20 project check-ins from users (across different projects).
  - 8 registered users.

  **NB**: for general testing purposes, you must have one regular user account with the following login details:
  Email address: *test@test.com*
  Password: *test1234*
  This user must have at least 1 project, 1 project checkin (from a different project), and 2 friends.
- For the purposes of testing and marking, do **not** hash your passwords.
- **Remember to export your database and include it in your final submission.**

# Directory Structure

- Structure all your files into a logical directory structure.
- Make use of correct file naming conventions.
- Use **relative file paths** for references.
- Put images, fonts, CSS files in their respective folders.

# Completing the Project

You have until 30 October to complete the project.

You have to submit on ClickUP. **Remember to download and double-check your submissions. Late submissions / submissions in the incorrect place will not be accepted.**

To avoid running into problems on 3 & 4 November, it is crucial that you test your project by running the Docker image locally throughout the semester. Ideally, you should test that everything still works each time you add new functionality. **Avoid blind-coding at all costs.** Do not wait until just before the submission deadlines to start testing your project.

Make sure to retest the entire site after submitting to ensure that what you will be assessed on is fully functional.

# Docker - Things to Look out for

- Your **entire application** should be Dockerised, this includes the frontend and backend code.
- On marking day, the markers will download your submission and run the relevant commands to run your Docker image.
- Ensure that you test your application frequently and thoroughly using Docker. Some things can differ between running your application locally (outside of Docker) and running it using Docker.
- Do not wait until submitting to learn how to Dockerise your project as you will **not be marked** unless you use it.

# Mark breakdown

Deliverable 0 - Wireframe: 10%

Deliverable 1 - Front-end: 15%

Deliverable 2 - Back-end 20%

Final Project (Deliverable 3): 55%

---

Start Today!