

PGCertInfoTech Final Project

A Personal Blogging System

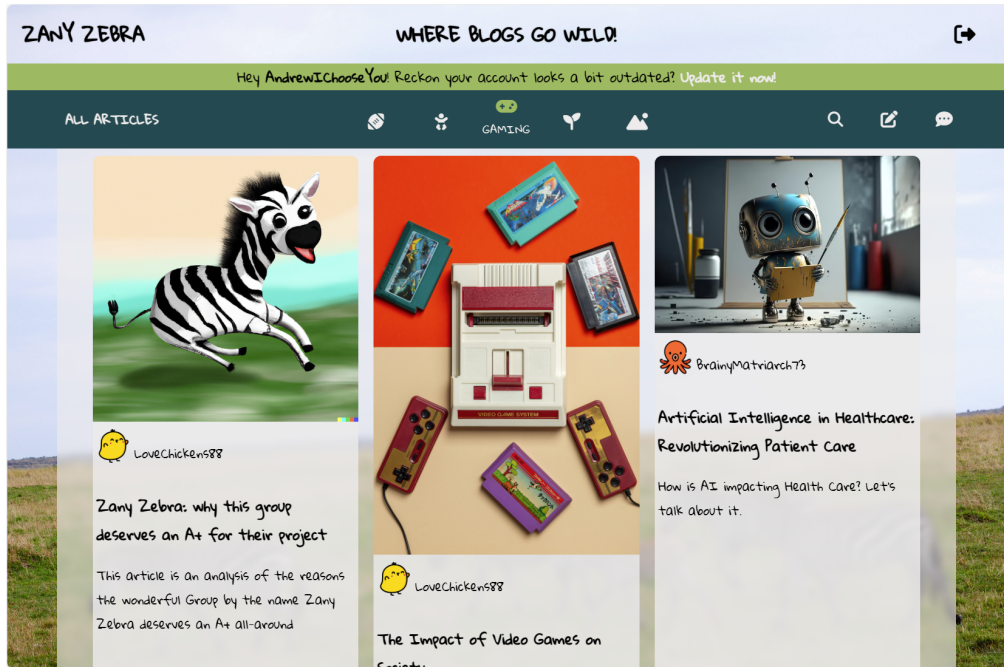


Fig 1: "Zany Zebra - Where Blogs Go Wild!" (© Ana Nozdria, Abdul Muhammed, Julian Nieves & Je Yeon Han, 2023)

Key Dates

February 2nd	Project introduction + project planning techniques
February 5th	Check-in with the teaching team. Project design feedback. Risk plan due.
February 13th	Project source code due.
February 16th	Group presentation day.
February 17th	Individual report due.

Table of Contents

Key Dates	1
Table of Contents.....	2
Introduction	4
Project Overview.....	4
Team components: Implementation, presentation, project management, git workflow	4
Feature implementation	5
Code quality & readability	5
Git workflow	5
Check-ins / project management.....	5
Presentation & demonstration	5
Individual component: Report	6
Implementation requirements	7
Website requirements	7
User accounts.....	7
Articles	8
Comments	8
Usability & Interface Design.....	9
Add your own feature(s)!	9
Administrative API requirements.....	9
Java Swing administrator interface requirements	10
Project Management.....	12
Teamwork.....	12
Communication	12
Code Sharing	12
Demonstrating Progress.....	13
Deliverables & Submission Instructions.....	14
Planning & initial design	14
Progress check-ins.....	Error! Bookmark not defined.
Group project source code	14
Individual reports	14
Presentation day	14

Introduction

In this project you will develop a blogging website using the skills and technologies you have learnt through the *Programming for Industry* and *Programming with Web Technologies* courses. The project will also give you the opportunity to show how you can use online resources to discover and apply content *not* taught within the course. The snapshot shown above is taken from a previous PGCert IT student group's submission.

Through the website, users can register for an account, which is needed to be able to post articles and to leave comments on articles. When logged in, users have full control of the content they have authored: creating, updating and deleting their content and comments.

In addition to the website, administrators will be able to use a Java Swing frontend to perform certain administrative tasks.

You are given a list of requirements for the blogging system. Your group can also customise aspects of the project, within the broader requirements.

The project will give you the opportunity to work as a team on a larger-scale project than you've had experience with previously in the course. It will also allow you to showcase your individual software development skills.

Project Overview

The final project is worth **30%** of your total grade, and consists of the following components:

	Indicative weighting
Feature implementation, code quality & readability	2 / 3
Git workflow	<i>Compulsory</i>
Check-ins / project management	<i>Compulsory</i>
Presentation & demonstration	<i>Compulsory</i>
Individual report	1 / 3

Those components listed as “Compulsory” are not directly worth marks, but you will be penalized in your overall grade if you do not satisfactorily meet those requirements.

Team components: Implementation, presentation, project management, git workflow

The team portion of this project is worth roughly **two thirds** of the project, and consists of the implementation itself, as well as your presentation, project management, and git workflow.

While it is a group project, individual group member's marks are dependent on participation in the group, contribution to the project and attendance at progress meetings. We do put careful consideration into marking to make sure groups are not disadvantaged in the case that a member is not able to participate for any reason. We do not expect everybody to have the same technical ability, but participation and collaboration are essential. Planning goals and the contributions of each group member is essential. It is also important to get to know the members of your group so that you can collaboratively plan tasks that each member can contribute to.

Feature implementation

There are several requirements for each team's blogging system. These are given in the [Implementation Requirements](#) section below. All requirements must be implemented to a high standard to receive full marks for feature implementation.

Code quality & readability

Your team's code must be easily understandable by third parties and conform to best practices. This includes the use of appropriate variable and identifier names, sufficient commenting, and breaking your code up into appropriate modules, amongst other considerations. It should be written in a way that would make it easy for other people to understand and modify.

You should use code organisation and quality techniques covered in the course content where possible and appropriate, such as modules, routes, design patterns, middleware, etc. Good code practices not only make your code easier for the markers to understand, but your teammates as well – leading to better collaboration!

In general, you should *not* use third-party libraries / frameworks outside those taught in class, unless explicitly mentioned in this handout. If you would like to use any third-party libraries not taught in the course or mentioned in this handout, you *must* get permission from Andrew (by [sending him an email](#)).

Git workflow

Your group must follow good Git collaboration workflows, each team member must commit using their own GitHub account. If pair programming is practised, the other team members should be credited in the commit message. **No direct commits to the main branch are allowed.**

Check-in / project management

During the project, your team will be required to report progress to a member of the teaching team, on the date given in ["Key Dates" above](#). Each team member must be present at the meeting. The meetings will be approximately 10-15 minutes per team. Evidence of good teamwork is required at these meetings - the instructor should be able to see and agree upon a fair workload allocation for each team member. Evidence of good project planning should also be visible, via a provided GitHub Projects board.

For more information regarding important project management aspects to consider, see the [Project Management](#) section of this document.

Presentation & demonstration

On the presentation date (see [Key Dates](#) on the front page of this document), each team will get the opportunity to present their project to the rest of the class. This is your chance to show off all your hard work and it should be a fun session.

Each team will be allowed 15 minutes to speak, followed by some questions. The time limit is strict, and you will be forced to stop your presentation if you go overtime. Presentations should be professional, be slideshow-based or video-based (i.e. attempting to do a live demo is *not recommended* unless informed otherwise), and focus on the overall system architecture and features of your web app, rather than delving into low-level implementation details. Each member of the team is expected to speak during the presentation, as well as answer questions during Q&A.

Individual component: Report

In addition to the group and individual implementation, each **individual** must submit a written report on or before the report due date (see [Key Dates](#) above). This is worth roughly **one third** of your project grade.

It is suggested that you work on the report throughout the group project and use it as an opportunity to do some extra reading related to the principles and technologies you are using. The report must cover the following topics:

- In your own words, explain how the system has been designed; the overall system architecture and how various components fit together and communicate to form the whole.
 - Brief overview of the system and architecture design. You must draw an ER diagram and we strongly recommend that you draw other diagrams to supplement your report (such as component diagram showing how each Svelte component interacts with each other)
 - Highlight targeted problems and solutions.
- Detail your particular contributions to the team and project; you are expected to have worked on the front-end and back-end so outline what you did on both.
- Detail which topics, taught in class, have been used within the project, and where.
- Detail any topics or technologies, which were not taught in class, that you have used, and where.
- How has your team used Generative AI during this project? What are the benefits and drawbacks of using AI to assist with development?
- Describe the lessons you have learned from working in a team. What are the benefits, and are there any drawbacks or difficulties? How have those been overcome in your team?

The length of the report should be *approximately six pages* in **IEEE two-column format** (though there is no minimum or maximum limit). Word and LaTeX templates for the report are available [here](#).

The report should be written as a structured piece of academic writing; you should follow the formatting and conventions of the IEEE format as linked above.

Implementation requirements

Your team must implement the following requirements. These are organised into **website requirements**, **API requirements** and **Java Swing administrator interface requirements**.

Website requirements

The following requirements form the core functionality of your team's blogging system. You will need to design API routes and response codes for these features, your API design should sync up with the [Administrative API requirements](#) section.

User accounts

1. Users must be able to create new accounts. Each new user should be able to choose a username (which must be unique) and a password. At minimum, a user's real name and date of birth should also be recorded, along with a brief description about themselves.
2. When selecting a username while creating an account, users should be *immediately* informed if the given username is already taken. Users should *not* have to submit a form to discover whether their chosen username is taken.
3. When selecting a password while creating an account, users should be presented with *two* password textboxes (e.g. "Choose password", and "re-enter password"). They must type the same password in each box to proceed. If the user didn't enter the same password in both textboxes, they should not be allowed to submit the form. Ideally, a visual notification message, such as ("passwords do not match"), should also be displayed. For this requirement, and others which require visual notifications of errors, please do *not* use the standard `alert()` box – these will not be awarded with any marks as they are considered poor practice for production websites.
4. Users' passwords should not be stored in plaintext - they should be appropriately hashed and salted. You will need to research hashing and salting; there are NPM packages that can be used for hashing and salting passwords which you can use.
5. When creating an account, users must be able to choose from amongst a set of predefined "avatar" icons to represent themselves or choose to upload their own custom avatar image; you can choose exactly where in the interface the user avatars appear, but you should make sure that users can see other users' avatars in relevant parts of the page. For example, you may wish to display a user's avatar and username beside comments they have made.
 - **Note about image uploads:** check out the [file upload example in the cs719-examples repository](#) to see how we can handle this, using [FormData](#) on the frontend and [multer](#) on the backend.
6. Once a user has created an account, they must be able to log in and log out; make sure to use an authentication system that allows users to stay logged and choose to log out when they wish to. Remember the lab activities dealing with authentication - these will serve you well as a starter point!
7. Users must be able to edit any of their account information (*including* their username) and be able to delete their account. If a user deletes their account, all their articles and comments (see below) should also be deleted.

Articles

8. Users must be able to browse a list of all articles, regardless of whether they are logged in or not. If logged in, they should additionally be able to browse a list of their own articles.
9. When viewing the lists of articles identified above, users should be able to search and sort article lists by *article title*, *username*, and *date* (but only one at a time). Aim to follow UI/UX conventions for user friendly sorting functionality. The search option should perform a case-insensitive match on all article titles and contents. It is expected that the usability of your search and sorting options is intuitive and shows good interface design.
10. When logged in, users must be able to add new articles and edit or delete existing articles which they have authored.
11. When creating or editing articles, users should be presented with a WYSIWYG (what you see is what you get) editor. The WYSIWYG editor should allow users to edit the formatting of an article without having to edit the HTML markup. A WYSIWYG editor will allow a user to create content that may break your site or display incorrectly, you should configure your editor to disallow any unsupported operations and best fit with the requirements. The editor should (*at minimum*) allow users to:
 - Add headings (or titles and subtitles)
 - Make text bold, italic and underline
 - Add bulleted and numbered lists

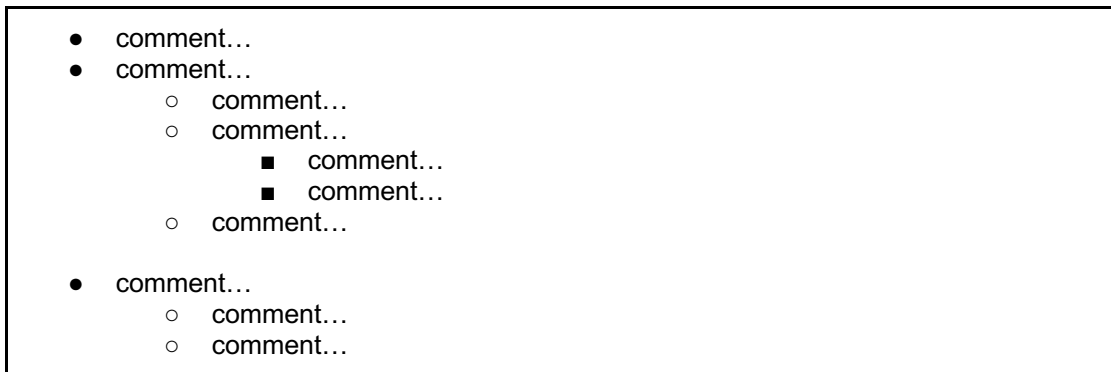
There are several editors available which you may use. We recommend TinyMCE as it has good Svelte integration, and you can sign up for a free API key which should be more than sufficient for the duration of this project. Whichever you choose, you should thoroughly read the documentation and explore how to integrate it into your project.

- TinyMCE Svelte integration (ignore the part about creating a new project - that part's already done!) [Svelte integration | Docs | TinyMCE](#)
 - TinyMCE basic setup & configuration guide: [Basic setup | Docs | TinyMCE](#)
 - **Hint:** The required WYSIWYG functionality above is achievable with just the basic TinyMCE Svelte component plus the “lists” plugin.
12. When creating new articles, users must be able to add one or more images to that article (if they choose - whether a user adds an image is up to them). When editing articles, users must be able to change, add, or remove any images. For full marks, users must be able to add a “header” image to an article, as well as embed any number of images in the article itself using the WYSIWYG editor. Instructions for how to configure this can be found in the documentation of your editor of choice.

Comments

13. When logged in, users must be able to comment on articles. When viewing articles, comments associated with that article should also be viewable.
14. Comments must show the username of the commenter, and the time & date the comment was made, in addition to the comment itself.
15. Users should be able to comment on comments to any depth of nesting. In other words, users can comment on articles and comment on other comments (including comments on comments). Comments should be listed chronologically below the article or comment they are replying to. Comments that are replies to other comments should be indented and directly below the comment they are in reply to.

An example of what two levels of nesting could look like is included below.



Hint: Check out the [svelte:self](#) special element in the Svelte docs for something which will make this requirement much easier to implement!

16. Commenters should be able to delete their own comments; article authors should be able to delete any comments on their own articles.

Usability & Interface Design

17. The website must have a consistent look and feel and must be responsive. The interface should respond well to the resizing of the screen and be usable throughout the range of common screen-widths.
18. The website must be user friendly and have good usability. When adding the features, consider how such features have been implemented in other websites you've used before. What did you like about those websites? What could use improvement?

It is suggested you investigate usability and review resources such as [Neilsen's heuristics](#) and / or [PACMAD](#), and test your interface often so you can put yourselves in the shoes of your potential users.

19. It is suggested that you have a structured approach to interface design and consider creating 'wireframe' outlines of your pages so your group can discuss and plan what elements need to be on each page and how they are positioned. You may wish to create hand-drawn wireframe designs or investigate using a tool like [Figma](#).

Add your own feature(s)!

20. In addition to the compulsory requirements above, you must extend the website with your own features. You are free to do whatever you want here, though you must seek approval from the teaching team during the progress check-in. The teaching team will give you feedback as to whether your scope is appropriate.

Administrative API requirements

The following requirements necessitate that your backend includes some routes (endpoints) that accept JSON and return JSON and / or various HTTP status codes and / or cookies. Your API must follow good REST principles. Remember that your API will be used & assessed by the both the Svelte frontend, and the Java Swing-based administrative interface ([see below](#)). If the web interface shares similar API routes such as login and logout, then such routes should be reused.

1. When a **POST** request is made to **/api/login**, a user should be authenticated using the username and password supplied as *JSON* in the request body
 - a. If authentication is successful, a **200** response should be returned along with information that can be used to identify the authenticated user in the future (presumably some kind of *authentication token*).
 - b. If unsuccessful, instead a **401** response should be returned.
 2. When a **POST or GET** request is made to **/api/logout**, a user should be logged out (presumably by deleting their authentication token that was created above). Then, a **204** response should be returned.
 3. When a **GET** request is made to **/api/users**
 - a. If the requestor is authenticated **as an admin**, an array of all users should be returned, as JSON. For each user, all their profile information should be included, along with the number of articles that user has authored.
 - b. If the requestor is unauthenticated, it should return **401**. If the user is authenticated but not an admin, instead a **403** response should be returned.
- Hint:** How will you determine if a particular user is an admin? Perhaps an extra column in the database?
4. When a **DELETE** request is made to **/api/users/:id** (where :id is a user id):
 - a. If the requestor is authenticated as an admin, the user with the given id (if any), along with all of their articles and comments, should be deleted. Then, a **204** response should be returned.
 - b. If the requestor is unauthenticated, it should return **401**. If the user is authenticated but not an admin, instead a **403** response should be returned (and nothing should be deleted).

Java Swing administrator interface requirements

You must implement the following requirements in Java Swing. These requirements form a basic administrative user interface, built using Java Swing. The requirements listed here involve interaction with the Node.js backend. To do this, make appropriate HTTP requests to your API endpoints, as created in the [API requirements](#) section above.

1. A Java Swing application must be created, consisting of a single window (i.e. JFrame).
2. The UI must include two text boxes - one each for allowing a user to enter their username and password.
3. The UI must include two buttons - one for logging in, and the other for logging out.
4. The UI must include a JTable to display user data.
5. The UI must include one further button, for deleting the currently selected user.
6. When a user clicks the login button, they should be authenticated using the username / password they entered.

7. If the user is authenticated as an admin above, then a list of users should be obtained from the backend and displayed in the JTable. If the user was not authenticated, or was authenticated but not as an admin, instead an appropriate error message should be displayed in a dialog box, and the user should be logged out again.
8. When the user clicks the logout button, the JTable should be cleared and the user should be logged out.
9. While authenticated, when the user selects a row in the JTable a **separate JPanel** should load the corresponding username and profile image from the server, an HTTP request to load the image must be sent upon selection, the Swing application must not freeze.
 - a. The design of this user profile display JPanel is up to you, however the selected user must be clearly visible. The size of the avatar should be a thumbnail size and placed in the centre of JPanel.
10. While authenticated, when the user selects a row in the JTable then clicks the “delete user” button, the selected user should be deleted from the backend (and the associated row should be removed from the JTable).
11. All buttons and other UI elements should only be enabled if it is currently possible to carry out their functionality. For example, the login button shouldn’t be enabled if the user is already logged in, and the “delete user” button shouldn’t be enabled if there is no row selected in the JTable.
12. Your program must use appropriate design patterns (e.g. observer / adapter / MVC for displaying items in the JTable) and apply the concepts taught in CS718.
13. The Java Swing implementation is designed to provide an opportunity to apply the technologies and concepts taught in CS718. It is expected that you will use the general approaches and principles taught in CS718 to approach the implementation. You should not use any premade templates or GUI builders. If you have doubts about what tools are appropriate to use, ask the teaching team.

Notes

The Swing interface will require you to investigate how to send HTTP requests, and to convert between JSON and standard Java objects, from within the Java language. A fully functional sample application is provided to you on Canvas that demonstrates how HTTP connectivity can be implemented in Java. You should look at the example application and do your own research to implement these features.

Project Management

There are several project management considerations to keep in mind while working on this project.

Teamwork

For the group portion of this project, you will be expected to work with members of your team to effectively come up with a list of requirements, prioritise that list, and divide the work up amongst team members. Make sure to maintain good communication with your group and lecturers if anything affects your attendance and/or ability to contribute to the project. While working on the project, also keep in mind that each team member will need to demonstrate that they have worked on both the frontend (HTML / CSS / JavaScript) and backend (Node.js / SQL), and that they have contributed equally and fairly to the team; teams should support group members to work on a variety of parts. Make sure to outline your contributions to the frontend and backend in the final report.

Those considerations notwithstanding, how you divide the work amongst your teammates is entirely up to you. However, you **must** use the provided GitHub Project board to assign tasks to team members, and you must keep this board up-to-date, ready to show to instructors at any time. Your team will lose marks if there is evidence of the group having poor teamwork, unfair task allocation or failing to keep the project board up to date; if there are issues with individuals this will be addressed on an individual basis.

It is important to communicate clearly throughout the development process. Do not be afraid to discuss ideas with your group if you are not sure what tasks you can be helping with. It is important that you communicate clearly to your group if you are unsure of things like who is doing what or how to approach a task you have been assigned. It can be helpful for groups to plan and assign tasks quite clearly so every group member understands who is editing different files at different times.

If a group member chooses to include code from external sources (including generative AI), they **must be able to understand the code and explain the code to other team members**. If teamwork is impacted by inclusion of poorly understood code, warnings will be given to the student responsible and individual marks can be impacted.

Communication

Remember to organise some easy way in which your group can communicate – especially if you're not working in the same location (though working in the provided lab space together is strongly encouraged). The tool must be able to run in a desktop browser, accept images and links, and be well known in the industry (for reliability). Feel free to use any other tool your group agrees upon.

When bugs occur, you are responsible for researching the relevant error messages online and building a history of things that you have tried to attempt to fix the bug. If you are still unable to solve the bug (given an acceptable amount of effort) you should then ask your team members. If your team is still unable to resolve this bug and it is impeding progress, you are welcome to ask your instructor for directions and tips; they will most likely suggest other diagnosis steps or alternative solutions to help you continue working on the project.

Code Sharing

It is vital to properly utilise version control to reduce problems when working on the same codebase. The first thing your team should do is familiarise yourself with your GitHub repository, which you will all use collaboratively. You should work out who will be responsible for approving pull requests and communicating to other people in the group to update their main branches. Remember only one person

can merge changes to the shared master at a time. You must use the Feature Branching workflow, as demonstrated in project introduction session. Resources are available on Canvas. Remember to work on small features and merge regularly as resolving merge conflicts can take up a lot of time if you do not do it regularly.

Demonstrating Progress

You will be required to always keep an up-to-date visualisation of your team's progress. Assuming you keep your GitHub Project board up to date, this will be a great way for you to demonstrate your progress to instructors. It is also the best way to make sure your whole team knows what they are doing at all times.

Deliverables & Submission Instructions

Planning & initial design

Planning is the key to success! To ensure each team is on the right track, we ask the team to prepare the following and submit them at the check-in with the teaching team (see [key dates](#)).

1. Risk mitigation strategy plan
2. Sprint backlog with initial task allocation
3. ER diagram & API design
4. Optional front end interface design (e.g. Figma)

Group project source code

Your team git repository serves as the submission for your project source code. Make sure that your team repo main branch is up-to-date on or before the due date above and make clear in your final commit message that it is the final commit (see [key dates](#)). Any commits after this deadline will be ignored by the markers.

In addition to your source code, your repo should also include the following:

1. An SQL script with your CREATE TABLE statements and any initialization data (db-init.sql). This script should be auto-run to initialize the database, the first time the marker runs your backend.
2. A README.md file containing the following information:
 - a. Team name
 - b. Are there any special setup instructions, beyond initialising the database and running your project?
 - c. At least two usernames / passwords for existing users in your system with some already-published articles & comments
 - d. API documentation
 - e. Description of all pages in your webapp
 - f. Description of how to use your Java Swing admin interface
 - g. Any other instructions / comments you wish to make to your markers

Individual reports

Your individual reports should be submitted separately to Canvas as a single PDF document, on or before the due date.

Presentation day

Your team must come to the presentation day with a functional version of your blogging system. Your team will be giving their presentation to the class.

Your team will have fifteen minutes to present, followed by a five-minute Q&A session. Each team member should speak during the presentation, as well as answer questions during Q&A.