COMP639 Studio Project – Semester 1 2025

Individual Assignment

Worth:	**25%**
Due:	**Sunday, 9th March at 11.59pm.**
Late Penalty:	Work not received by due time attracts an immediate penalty of up to 25% of the marks available. No work will be accepted after **Tuesday, 11th March 2025 11:59pm**.

ISSUE TRACKER – LINCOLN COMMUNITY CAMPGROUND

Develop a Python/Flask Web App to track and manage issues at a local community campground.

The focus of this assessment is on implementing a secure login system for a multi-user web app, and providing different levels of access for different user roles. The issue tracker will include three user types—visitors, helpers, and administrators—each with access to a different set of features.

IMPORTANT

This is an **individual** assessment. You must not collaborate or confer with others. You may help others by verbally explaining concepts and making suggestions in general terms, but without directly showing or sharing your own code. You must develop the logical structure, the detail of your code and the database on your own, even if you are working alongside others. Code that is copied or shares a similar logic to others will receive zero marks for both parties.

The use of Artificial Intelligence (AI) tools, such as ChatGPT or GitHub Copilot, to complete this assessment is **prohibited**. You **must not** use AI to generate source code, database scripts, or any other deliverable submitted for marking. Assessment answers will be analysed for evidence of the use of AI and penalties may be administered.

The University policy on Academic Integrity can be found here.

BACKGROUND

Lincoln Community Campground (LCC) is a privately owned campground on the outskirts of Lincoln. It is mostly used by a small community of regular visitors, who are allowed to camp for free. The campground is not permanently staffed, but operated by a group of local volunteers known as "helpers". These helpers maintain the on-site facilities, which include toilets, showers, Wi-Fi, fire pits, outdoor cooking equipment, and a small number of solar-powered charging stations for smartphones and tablets. Currently, visitors email helpers if anything goes wrong (for example, if a toilet is blocked or the Wi-Fi isn't working). However, this has become difficult to manage over the years as helpers come and go. Emails are often missed, or go to the wrong person. Your goal is to implement an "issue tracker" that allows helpers to easily view and manage issues reported by visitors. You can think of this as a very simple version of the online helpdesk systems you may have interacted with in the past.

SOFTWARE REQUIREMENTS

USER ROLES, SESSION HANDLING, AND ACCESS CONTROL

- **User Roles**:
  - Define three roles: **Visitor**, **Helper**, and **Admin** (administrator).
- **Session Handling and Access Control**:
  - Use Flask sessions to keep track of whether a user is logged in, and what their current role is.
  - Implement a system to limit access to certain pages or features based on the user's role.

WEB APP DESIGN

- **Responsive Design**:
  - The web app should adapt to different screen sizes. Most visitors will be using smartphones or tablets to report issues while they're at the campground. Helpers and admins use a mix of smartphones, tablets, and laptop or desktop PCs.
  - Use **Bootstrap** to create a responsive UI.
- **Theming**:
  - Style your web app to match the campground theme.
  - Name your app "LCC Issue Tracker" and incorporate this name into your design. Make it clear to users that your web app is the issue-tracking system for Lincoln Community Campground.

HOME PAGE

- **Requirements**:
  - Include the name of the app ("LCC Issue Tracker").
  - Provide links for login and registration.
  - Include your name and student ID (the footer is a good place).

REGISTRATION

- **New Visitor Registration**:
  - Allow new campground visitors to register and create their own account.
  - New users should always be registered as a **visitor**: people must not be able to register themselves as a helper or admin. Admins can later "promote" visitors to these other roles.
- **Required Details**:
  - Collect **username**, **email**, **password**, **first name**, **last name**, and the user's home **location** (users can choose how to represent their location: for example, they may enter "New Zealand", "Christchurch", "Lincoln, Canterbury", "UK", "Singapore", etc).
  - All fields should be required.
- **Username and Password**:
  - Ensure usernames are unique (i.e. no two accounts can have the same username).
  - Passwords should be at least 8 characters long with a mix of character types. Make these constraints clear to users during registration.
  - Do not store users' passwords directly in the database. Instead, hash passwords using the `flask_bcrypt` library and store those hashes.
- **Default Settings**:
  - Set each new visitor's role to "**visitor**" and their status to "**active**".

ROLE-SPECIFIC FUNCTIONS

Requirement	Visitor User Role	Helper User Role	Admin User Role
Login: Create one login form for all users (i.e. visitors, helpers, and admins should all log in using the same form, without having to specify their role). Store password hashes, not actual passwords.	✓	✓	✓
Logout	✓	✓	✓
Report a new issue: All issues must have a brief summary (e.g. "Fire pit damaged" and a longer description (e.g. "Some of the rocks around the fire pit in site 4 are missing, leaving it unsafe to use.") Any type of user can report an issue. Issues always begin in "new" status.	✓	✓	✓
View own reported issues: All users must be able to view all issues they've reported themselves, including any comments added to those issues with the date, username, role, and profile picture (if any) of whoever made the comment.	✓	✓	✓
Add comments to own reported issues: All users must be able to add comments to issues they reported themselves.	✓	✓	✓
View and edit own profile details (email, first name, last name, and location).	✓	✓	✓
View, replace, and remove own profile image	✓	✓	✓
Change own password: Make sure the same password constraints you apply to new passwords during registration are also applied here, and that the new password is not the same as the current password.	✓	✓	✓
View, add comments to, and change the status of issues created by any user. The status of any issue can be: • **new:** reported, but no work started • **open:** work has begun to investigate or address the issue • **stalled**: the issue cannot be progressed at the moment (e.g. waiting on parts for a repair) • **resolved:** the issue has been addressed When a Helper or Admin comments on a **new**, **stalled**, or **resolved** issue, it should automatically be set to **open**. However, the status should not change when a visitor makes a comment. **Resolved** issues should be listed in a separate page or view, so helpers and admins can focus on **new**, **open**, and **stalled** issues.		✓	✓
View and search the list of all users and **view user profile details.** Allow searching by username, first name, and last name.			✓
Change user status (active/inactive)			✓
Change user role: set any user to be a visitor, helper, or admin (e.g. promote an existing visitor to be a helper)			✓

DATA REQUIREMENTS

- **Database and Table Creation**
  - Create your own script to set up the database and tables based on the ERD (Entity Relationship Diagram) in Figure 1.
- **Database Population**
  - Create your own script to populate the database with initial test data.
  - Store password hashes in the database, not actual passwords.
  - Include at least **20 visitors**, **5 helpers**, and **2 administrators**.
  - Add at least **20 issues** and **20 comments** in total (i.e. a total of 20 comments divided between the 20 issues). Make sure you include **some issues with multiple comments**, and **some issues with no comments**.
    You may use generative AI to create issue and comment text. **This is the only allowed use of generative AI in this assignment.** You may not use generative AI to create the database script itself: only to create realistic example issues and comments to include in your script.
- **File Inclusion**
  - You must include both the **database creation script** and the **database population script** in your GitHub repository **and** on PythonAnywhere.

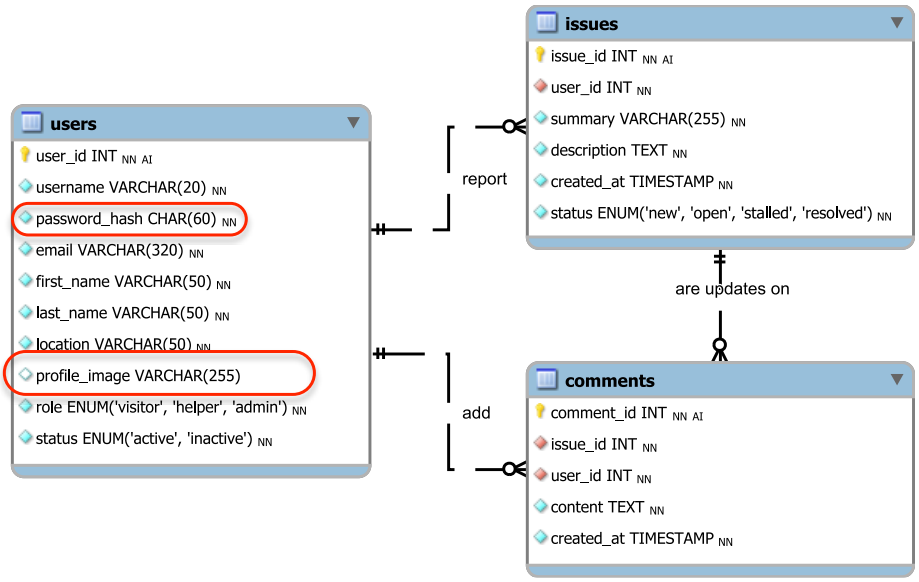


Figure 1: ERD illustrating the exact tables, fields, and relationships your database creation script must create.
Note: "NN" indicates a "not null" constraint. "AI" indicates an "auto increment" field.

Tip: To generate hashed passwords for your database population script, you may use the `password_hash_generator.py` script included with the Login Example project. You will need to install the `flask_bcrypt` library in your local development environment using `pip`. Make sure you follow all instructions given in the script, and ensure each user has a unique password.

The `password_hash` field in the `users` table (shown as CHAR(60) in Figure 1) is designed to store a `bcrypt` hash (60 bytes, which we store as characters). It is recommended that you create this column as "CHAR(60) BINARY" rather than just CHAR(60), to indicate this is a binary string (not just plain, human-readable text).

Note: This database design assumes that each user's profile image will be stored as a static file, and the `profile_image` column of the `users` table will contain the filename. Images themselves are not stored in the database. We chose this approach because it simplifies the deployment of your app to PythonAnywhere.

PROJECT REQUIREMENTS

1. TECHNOLOGY STACK

- Build your web app using **Python**, **Flask**, **Bootstrap CSS**, **JavaScript**, and **MySQL**.
- You may use CSS to override the default Bootstrap colours, but **do not** write full custom CSS layouts.
- **Do not** use SQLAlchemy, ReactJS, or other technologies.

2. GITHUB REPOSITORY

- Create a new private repository named "`LCC_Issue_Tracker`" (Lincoln Community Campground).
  - Add a README file with detailed instructions on how to deploy and use your web app, including any setup or configuration steps required.
  - Ensure your repository has a `.gitignore` file to exclude the virtual environment and your `connect.py` file, which stores your database connection details (different on your local machine and PythonAnywhere).
- Include the following in your repository:
  - All Python, HTML, image files, and any other necessary files for the web app.
  - A `requirements.txt` file listing all required `pip` packages.
  - Two MySQL scripts: one for database creation and one for record population.
  - Add GitHub user "lincolnmac" (computing@lincoln.ac.nz) as a collaborator to the repository.

3. HOSTING ON PYTHONANYWHERE

- Host your system, including the database, on PythonAnywhere.
- Add "lincolnmac" as your "teacher" via the site configuration.

4. SUBMISSION

- **Submit the COMP639 Web App Hand-In Sheet via the link on the COMP639 Akoraka | Learn page.**
- Include the following in your submission:
  - Your PythonAnywhere URL.
  - Your GitHub repository URL.
  - Usernames and passwords for different user roles for testing purposes.
  - Confirmation that certain files have been saved in your GitHub repository.

IMPORTANT: Do NOT make any changes to your app on GitHub or PythonAnywhere after the submission date until your marks have been released. Any changes will be considered the same as a late submission and may be penalised with a late penalty of up to 100% of your mark.

MARKING CRITERIA

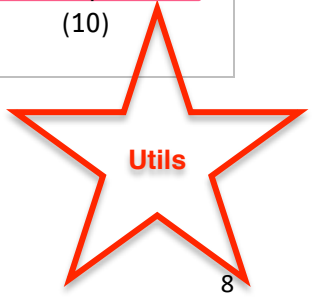
The functionality and design of your web app will **only** be assessed on PythonAnywhere. **If your app does not run on PythonAnywhere, it will not receive a mark in those areas.**

Your code, database scripts, and `README` file will be viewed on GitHub. Make sure you have shared your repository with the "lincolnmac" account.

Functionality (50)				
Registration and Login (10)				
Registration	New visitors cannot register an account. (0)	New visitors can register an account. (2)	All required details included (username, email, password, first name, last name, location). Password constraints enforced. New accounts are always visitors. (3.5)	Intuitive registration process with useful validation hints and error messages. Double-entry of password for confirmation. No notable bugs or usability issues. (5)
Login and Logout	Existing users cannot log in. (0)	Existing users can log in. (2)	Existing users (including newly registered users) can log in and log out. All users log in the same way, regardless of role. (3.5)	Intuitive login and logout functionality with no unnecessary steps. No notable bugs or usability issues. (5)
Reporting Issues (10)				
Viewing Issues	Users cannot see issues they have reported. (0)	Users can see issues they have reported, complete with date reported and current status. (2)	Users can see issues they have reported, and all comments made on those issues with the date each comment was made, and username and role of whoever added it. (3.5)	Users can see issues they have reported, and all comments made on those issues, in an easy-to-read format. Comments include profile pictures (if any). Visitors cannot see issues reported by other users. No notable bugs or usability issues. (5)
Reporting and Commenting on Issues	Users cannot report new issues. (0)	Users can report new issues. (2)	Users can report new issues, and add comments to issues they have previously reported. (3.5)	Users can easily report new issues and add comments to issues they have previously reported. No notable bugs or usability issues. (5)
Profile Management (10)				
View and Update Profile Details	Not recognisably implemented, or does not function as specified. (0)	Users can update their email address, first name, last name, and location. (2)	Users can update all details. Previous details are clearly visible during the update. (3.5)	Users can update all details. Previous details are directly editable. No notable bugs or usability issues. (5)

View and Update Profile Image	Not recognisably implemented, or does not function as specified. (0)	Users can add a profile image, and see that image on their profile. (1)	Users can add, replace, or remove their profile image. New users begin with a default image or placeholder. (2)	Users can add, replace, or remove their profile image. Images are displayed correctly regardless of size and shape. No notable bugs or usability issues. (3)
Change Password	Not recognisably implemented, or does not function as specified. (0)	Users can change their password. (1)	Users can change their password. The same constraints applied during registration are applied here. Password is not displayed on screen. No notable bugs or usability issues. (2)	
Working on Issues (10)				
Viewing Reported Issues	Helpers and Admins cannot view issues reported by other users. (0)	Helpers and Admins can view all issues reported by all users (including Visitors, Helpers, and Admins). (2)	Helpers and Admins can view all issues reported by all users, and view all comments added to those issues. (3.5)	Helpers and Admins can view all issues reported by all users, and all comments. Resolved issues are displayed in a separate page or view. No notable bugs or usability issues. (5)
Updating Reported Issues	Helpers and Admins cannot update issues reported by other users. (0)	Helpers and Admins can add comments to any reported issue. (2)	Helpers and Admins can add comments to, and change the status of, any reported issue. (3.5)	Helpers and Admins can comment on, and change the status of, any reported issue. Commenting on a new, stalled, or resolved issue automatically re-opens it. No notable bugs or usability issues. (5)
User Management (10)				
Search and View User Profiles	Not recognisably implemented, or does not function as specified. (0)	Admins can view a list of users and view the profile of any user. (2)	Admins can view a list of users, searchable by username, and view the profile of any user. (4)	Admins can view a list of users, searchable by username, first name, and last name, and view any user's profile. No notable bugs or usability issues. (6)
Change User Status	Not recognisably implemented, or does not function as specified. (0)	Admins can change the status of any user (visitor, helper, or admin) from "active" to "inactive" or vice-versa. (1)	Admins can change the status of any user via an appropriate control, which defaults to a user's current status. No notable bugs or usability issues. (2)	

Change User Role	Not recognisably implemented, or does not function as specified. (0)	Admins can change the role of any user to visitor, helper, or admin. (1)		Admins can change the role of any user via an appropriate control, which defaults to a user's current role. No notable bugs or usability issues. (2)
Design (15)				
Format and Layout	No visible formatting or layout (plain HTML). (0)	Some appropriate formatting and layout. (1.5)	Appropriate formatting and layout, consistent across most pages. (3)	Simple and attractive layout, consistent across all pages of the app. (4)
Colour Scheme and Style	No consistent colour scheme or style. (0)	Generally consistent across most pages. (1)	Consistent across most pages and matches the theme of the app. (2)	Consistent across all pages and matches the theme of the app. (3)
Responsive Design	Design does not respond to changes in screen size. (0)	Design is responsive to changes in screen size. (1)	Design adapts well to changes in screen size on desktop. (2)	Design adapts well to desktop and mobile screen sizes. (3)
Role-Based Access Control	Users can access functions for other roles. (0)	Users can only access functions appropriate to their role. (2)	Users can only see (in menus, etc) and access functions appropriate to their role. (3.5)	All users share the same UI where appropriate, but can only see and access functions appropriate to their role. (5)
Implementation (35)				
Source Code (20)				
Structure and Organisation	All code in one file, with no clear organisation. (0)	Code is organised into multiple files, with some logical structure. (4)	Code is well organised into multiple files, uses functions and constants to avoid unnecessary repetition, and has some consistency in naming conventions. (7)	Code is well organised, avoids unnecessary repetition, uses good naming conventions for variables and functions, and follows other best practices. (10)



Security	No password hashing, session management, or role-based access control. (0)	Passwords are hashed. Some form of session management and role-based access control. (2)	Passwords hashed. Good, generally consistent approach to session management and role-based access control. (3.5)	Passwords hashed. Good, highly consistent, and modular approach to session management and role-based access control, using functions and/or custom decorators to reduce repetition. (5)
Error Handling and Validation	Not recognisably implemented. (0)	Some error handling and/or validation. (2)	Appropriate error handling and some server-side validation. (3.5)	Appropriate error handling and consistent server-side validation. (5)
Documentation (10)				
Commenting	No useful comments included in source code. (0)	Some useful comments included in source code. (3)	Consistent and useful comments on modules, functions, and any sections of code that need explanation. (5.5)	Consistent and useful comments, with Python Docstrings for modules and functions. (8)
GitHub README	Not included. (0)	Explains how to set up and use the project. (1)	Provides a detailed guide to setting up and using the project. (2)	
Database (5)				
Table Structure	No table creation script provided. (0)	Implemented based on the ERD, but with one or more incorrect tables. (1)	Implemented based on the ERD, with all required tables. (2)	Implemented based on the ERD, with all required tables, fields, and relationships. (3)
SQL Script for Record Creation	Not provided. (0)	Script included to create visitors, helpers, administrators, and some issues/comments. (1)	Script correctly creates at least 20 visitors, 5 helpers, 2 administrators, 20 issues, and 20 comments. (2)	