Development and Testing of a CRUD System for an Online School Using Flask and SQL

SECURE SOFTWARE DEVELOPMENT

Table of Contents

Development and Testing of a CRUD System for an Online School Using Flask and SQL

Introduction

"Online learning is the fastest growing market in the education industry." (Oxford College, 2023) It has grown substantially and is set to further increase. It is important for these online schools to be able to manage data efficiently and safely. The goal is to create a CRUD (Create, Read, Update and Delete) system for an online school. It aims to provide a simple web interface for students, teachers and admin users.

The system will make use of the Flask, Jinja2 and w3.css to provide the web interface, python will for the backend processing and SQL for storage and retrieval of user data over different sessions.

This document outlines the development and testing of the system, looking at the system architecture, testing and execution.

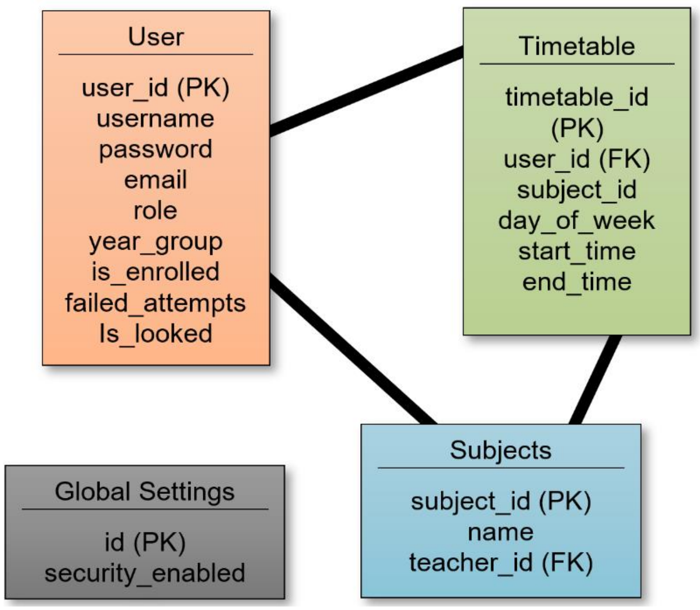
The code can be found at https://github.com/tbrays/School_CRUD_System.

Currently deployed at https://lovelace-school-of-software-development.replit.app/.

System Architecture and Design

Database

The database is built using the SQLite library, it consists of 3 main tables, users, subject and timetable. There is one separate table global Setting. This last table was implemented to store the security state across sessions.



Python Backend

The Python back end makes use of the Flask library as a web server framework. Flask was use over Django as it provided a lightweight, flexible framework that allows for greater customisation and control in web application development.

The *main.py* consists of the Flask routes that are responsible for handling the requests from the web interface. During a session the user details are managed through classes and are stored in a database for retrieval later.

HTML Frontend

The HTML frontend of the application is made using Jinja2 library. It allows for the webpages to be made from templates that extend a base. The pages are stored in the public folder so that the Flask server can access them. They consist of a base, home, login, register, dashboard, timetable, manage_timetable, manage_users and the settings page.

Code

Classes

A main user class is defined with class methods to handle general users' functionality. The student, teacher, parent and admin class inherit form the parent user class but has separate methods for role based behaviour. They also make use of polymorphism when instantiating a new instance of the class.

User Class

Attributes	Data Type	Description
user_id	Integer	Unique identifier for each user.
username	String	Username chosen by the user. Must be unique.
password	String	Hashed password for user authentication.
email	String	User's email address. Must be unique.
role	String	Defines the role of the user (e.g., student, teacher, admin).
year_group	Integer	Optional attribute to indicate the year group of a student.
is_enrolled	Boolean	Indicates whether the user is currently enrolled. Default is True.

Methods	Description
__init__(self, ...)	Initialises a new user object with the provided attributes.
create_user(cls, ...)	Creates a new user in the database with the given details and returns the user object.
validate_user(cls, ...)	Validates the provided username and password by checking against the stored credentials.
fetch_user(cls, ...)	Fetches and returns a user object by user ID from the database.
fetch_all_users(cls)	Retrieves all users from the database and returns them as a list of user objects.
update_user(cls, ...)	Updates the user's details such as username, email, and role in the database.
delete_user(cls, ...)	Deletes a user by user ID from the database.
reset_password(cls, ...)	Resets the user's password to a new value.

Timetable Class

Attributes	Data Type	Description
user_id	Integer	Foreign key that links to the user who the timetable belongs to.
subject_id	Integer	Foreign key that links to the subject being scheduled.
day_of_week	String	Indicates the day of the week for the timetable entry.
start_time	Time	The starting time of the scheduled class.
end_time	Time	The ending time of the scheduled class.

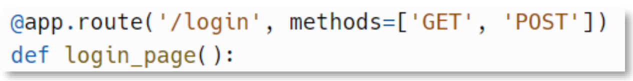
Methods	Description
__init__(self, ...)	Initialises a timetable entry with the provided attributes.
create_timetable_entry(cls, ...)	Creates a new timetable entry in the database for a user.
get_timetable(cls, ...)	Retrieves the timetable for a specific user and returns it.
delete_timetable_entry(cls, ...)	Deletes a specific timetable entry by its ID.

Subject Class

Attributes	**Data Type**	**Description**
subject_id	Integer	Unique identifier for each subject.
name	String	The name of the subject.
teacher_id	Integer	Foreign key linking to the teacher assigned to the subject.
Methods	**Description**	
__init__(self, ...)	Initialises a subject object with the provided attributes.	
create_subject(cls, ...)	Creates a new subject in the database and assigns a teacher if provided.	
delete_subject(cls, ...)	Deletes a subject by its ID from the database.	
fetch_all_subjects(cls)	Retrieves all subjects from the database and returns them as a list of subject objects.	

Key Functions

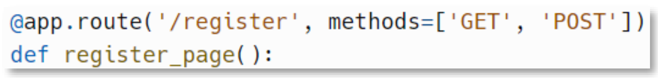
login_page()



```
@app.route('/login', methods=['GET', 'POST'])
def login_page():
```

This function processes the login requests. It accepts both GET, to display the page and POST to authenticate the user. It logs successful and failed login attempts. If the login is successful it redirects to the dashboard page and if it fails it displays an appropriate message.

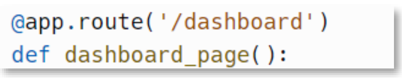
register_page()



```
@app.route('/register', methods=['GET', 'POST'])
def register_page():
```

The function allows a new user to register an account. It accepts GET request to display the page and POST request to handle the user data. The function asks for username, email, role and password, with further data for students. On successful registration it redirects to the login page. It also handles password creation and provides a suitable message if it is not a strong password.

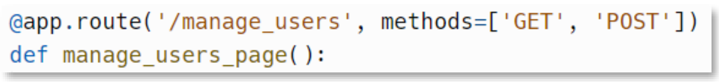
dashboard_page()



```
@app.route('/dashboard')
def dashboard_page():
```

The dashboard page is the main page for all users. The dashboard is created using Jinja2 allowing it to have different content depending on the role of the user.

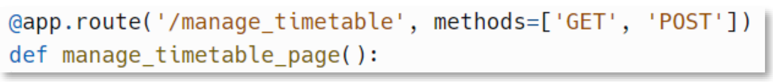
manage_users_page()



```
@app.route('/manage_users', methods=['GET', 'POST'])
def manage_users_page():
```

The manage user page is only accessible by the admin user. It allows for the creation, viewing, updating and deletion of users that are registered on the system.

manage_timetable_page()



```
@app.route('/manage_timetable', methods=['GET', 'POST'])
def manage_timetable_page():
```

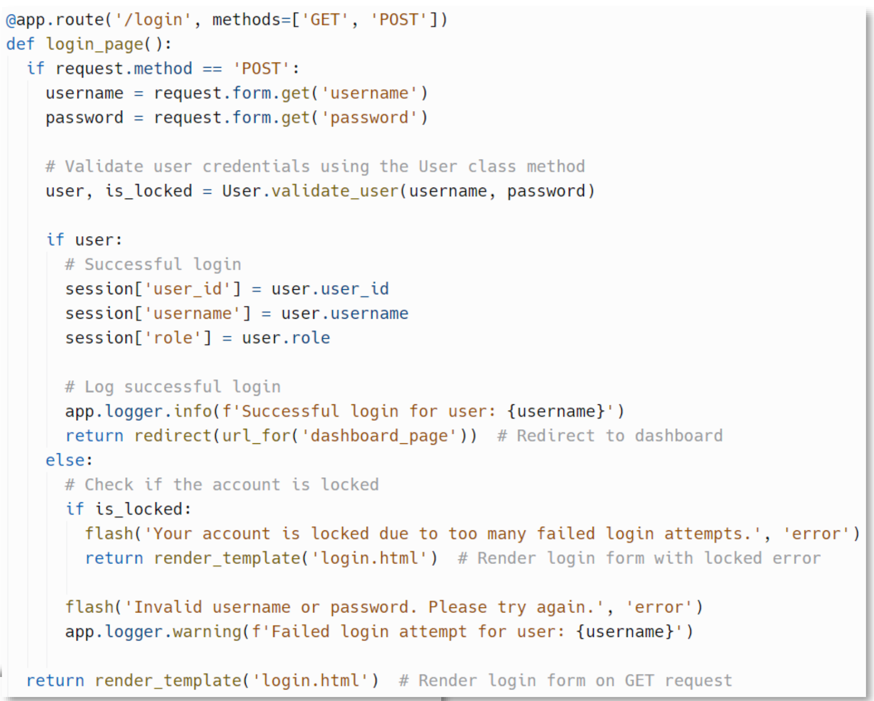
The manage timetable page is only accessible by admin users. It allows for the creation, viewing, update and deletion of timetable entries. This can be for either students or teacher's timetables.

Best Practices

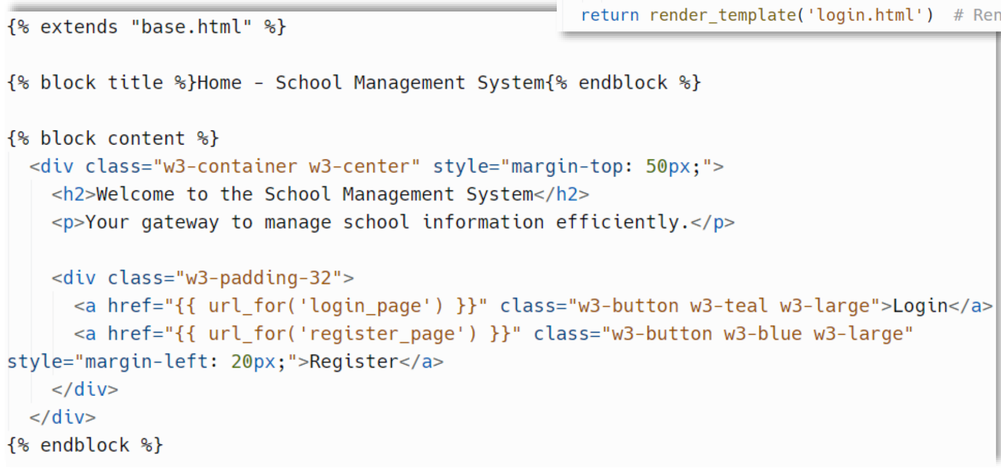
Throughout the code there is consistent formatting and meaningful variable names, which allows the code to be easily readable. The class structure allows for separation of concerns and each class is responsible for its own logic.

- Modular design to facilitate scalability and maintainability.
- Use of error handling to improve the user experience.
- The security enforces strong passwords and logs activities on the system.
- Meaningful variable and function names providing code clarity

The login page uses both GET and POST methods that are handled separately. On GET the login page is rendered, on POST the user's credentials are validated and if successful a session is created, redirecting to the dashboard.



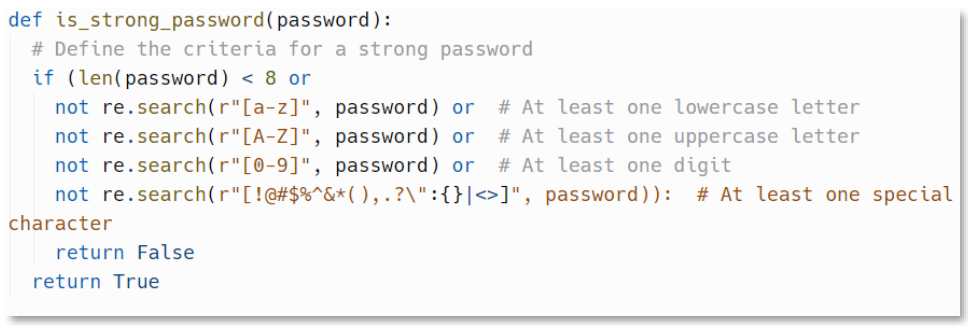
```python
@app.route('/login', methods=['GET', 'POST'])
def login_page():
  if request.method == 'POST':
    username = request.form.get('username')
    password = request.form.get('password')

    # Validate user credentials using the User class method
    user, is_locked = User.validate_user(username, password)

    if user:
      # Successful login
      session['user_id'] = user.user_id
      session['username'] = user.username
      session['role'] = user.role

      # Log successful login
      app.logger.info(f'Successful login for user: {username}')
      return redirect(url_for('dashboard_page'))  # Redirect to dashboard
    else:
      # Check if the account is locked
      if is_locked:
        flash('Your account is locked due to too many failed login attempts.', 'error')
        return render_template('login.html')  # Render login form with locked error

      flash('Invalid username or password. Please try again.', 'error')
      app.logger.warning(f'Failed login attempt for user: {username}')

  return render_template('login.html')  # Render login form on GET request
```



```html
{% extends "base.html" %}

{% block title %}Home - School Management System{% endblock %}

{% block content %}
  <div class="w3-container w3-center" style="margin-top: 50px;">
    <h2>Welcome to the School Management System</h2>
    <p>Your gateway to manage school information efficiently.</p>

    <div class="w3-padding-32">
      <a href="{{ url_for('login_page') }}" class="w3-button w3-teal w3-large">Login</a>
      <a href="{{ url_for('register_page') }}" class="w3-button w3-blue w3-large"
style="margin-left: 20px;">Register</a>
    </div>
  </div>
{% endblock %}
```

By making use of Jinja2 the web interface is modular allowing for the user interface to be scalable and easier to add to. It also makes use of the w3.css style sheet, making the presentation uniform across pages.

This function checks the entered password against a pattern using the re library. Along with the three failed login limit, this helps withstand brute force attacks. To further secure the system it makes use of a session with a secure private key.



```python
def is_strong_password(password):
  # Define the criteria for a strong password
  if (len(password) < 8 or
    not re.search(r"[a-z]", password) or  # At least one lowercase letter
    not re.search(r"[A-Z]", password) or  # At least one uppercase letter
    not re.search(r"[0-9]", password) or  # At least one digit
    not re.search(r"[!@#$%^&*(),.?\":{}|<>]", password)):  # At least one special
character
    return False
  return True
```

Testing

Unit Tests

Fowler, et al. (1999) argue that unit testing is crucial for early bug detection, which can significantly lower development costs and enhance overall code quality. Unittest.py is a built in framework with comprehensive features. It provides an automated testing function that can be applied to prewritten tests. Unit testing was complete with the use of unittest.py rather than pytest.py due to its ease of use and because it is a standard library. Each test was run then stored in the tests folder.

Unit	Description	Test Case	Status

User Management	Testing user creation and deletion	Create User
		Delete User
			`..`
			`----------------------`
			`Ran 2 tests in 0.647s`
			
			`OK`

Password Vlaidation	Ensure passwords meet the security standard	Strong password test
		Weak password test
			`..`
			`----------------------`
			`Ran 2 tests in 0.000s`
			
			`OK`

Timetable Management	Essure correct creation, update and deletion	Create entry
		View entry
		Delete entry
			`...`
			`----------------------`
			`Ran 3 tests in 0.061s`
			
			`OK`

Subject Management	Subject creation and deletion form database	Create subject Delete subject	Failed

Login Route	User authentication and correct redirect to dashboard	Valid login
		Invalid Login
			`..`
			`----------------------`
			`Ran 2 tests in 0.286s`
			
			`OK`

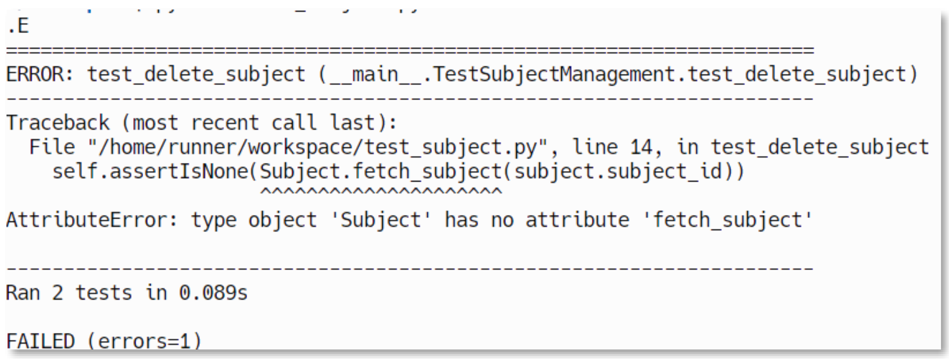
Manage Timetable Route	Admin only access to timetable management	Create entry Delete entry	Failed

Logging	Ensure successful and failed logins are logged	Successful login
		Failed login
			`..`
			`----------------------`
			`Ran 2 tests in 0.266s`

Subject Management Failure

When conducting the test for subject, it failed as there was no fetch to check if the test had passed.

- Added fetch and rerun for pass.

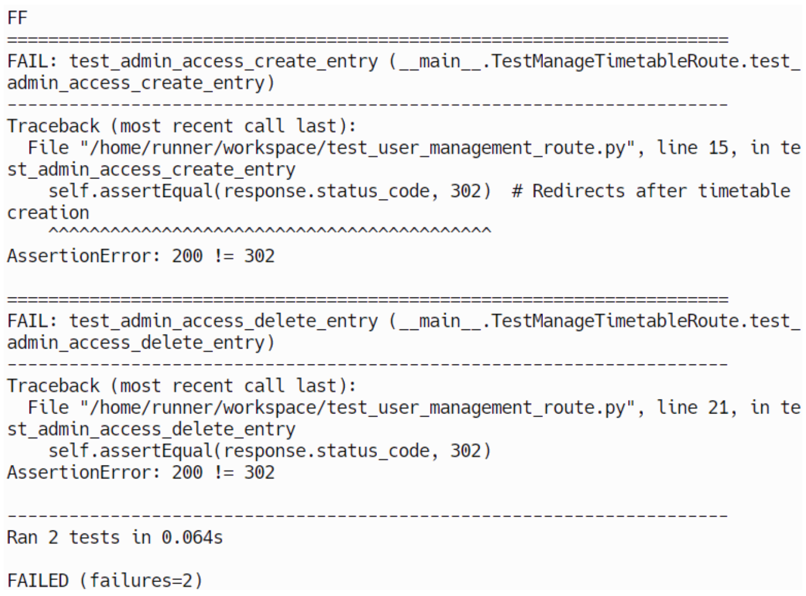


```
.E
======================================================================
ERROR: test_delete_subject (__main__.TestSubjectManagement.test_delete_subject)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "/home/runner/workspace/test_subject.py", line 14, in test_delete_subject
    self.assertIsNone(Subject.fetch_subject(subject.subject_id))
                      ^^^^^^^^^^^^^^^^^^^^^
AttributeError: type object 'Subject' has no attribute 'fetch_subject'

----------------------------------------------------------------------
Ran 2 tests in 0.089s

FAILED (errors=1)
```

Unit	Description	Test Case	Status

Subject Management	Subject creation and deletion form database	Create subject
		Delete subject
			`..`
			`----------------------`
			`Ran 2 tests in 0.065s`
			
			`OK`

Manage Timetable Failure



```
FF
======================================================================
FAIL: test_admin_access_create_entry (__main__.TestManageTimetableRoute.test_
admin_access_create_entry)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "/home/runner/workspace/test_user_management_route.py", line 15, in te
st_admin_access_create_entry
    self.assertEqual(response.status_code, 302)  # Redirects after timetable
creation
    ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
AssertionError: 200 != 302

======================================================================
FAIL: test_admin_access_delete_entry (__main__.TestManageTimetableRoute.test_
admin_access_delete_entry)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "/home/runner/workspace/test_user_management_route.py", line 21, in te
st_admin_access_delete_entry
    self.assertEqual(response.status_code, 302)
AssertionError: 200 != 302

----------------------------------------------------------------------
Ran 2 tests in 0.064s

FAILED (failures=2)
```

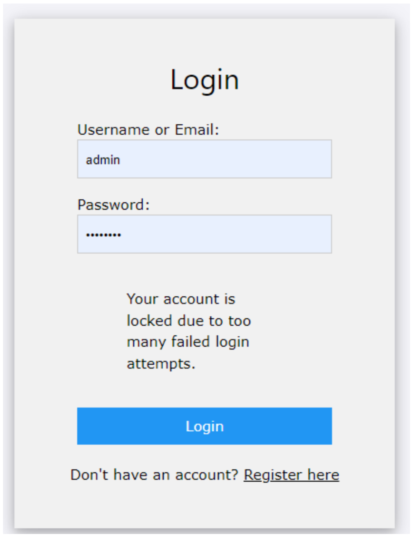
When running the manage timetable test two errors were received. This is due to missing redirects after the entries are complete.

- Added the redirects and rerun for pass.

Unit	Description	Test Case	Status
Manage Timetable Route	Admin only access to timetable management	Create entry Delete entry	.. ------------------------ Ran 2 tests in 0.039s OK

Limited Login Attempts

When a user fails to log in more than three times the account is locked making brute force attack difficult.



Login

Username or Email:

admin

Password:

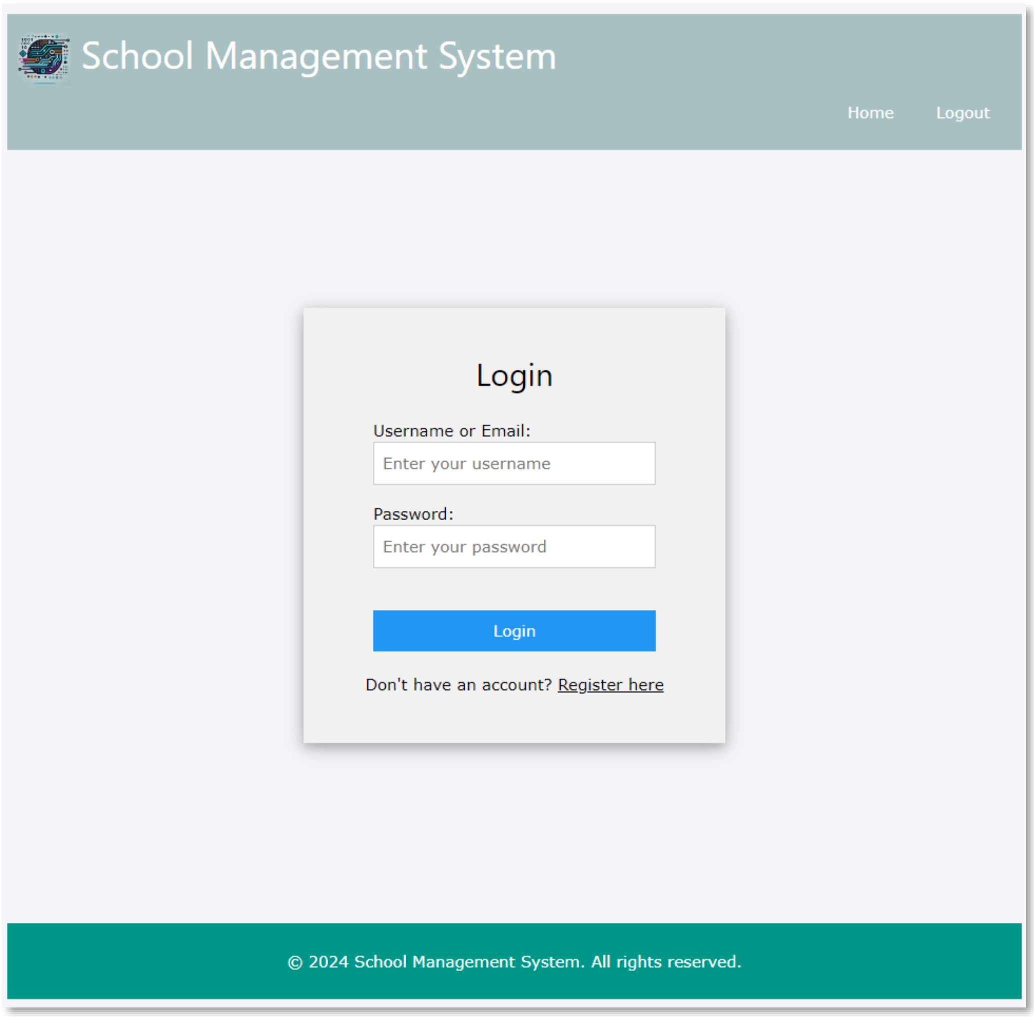
....

Your account is locked due to too many failed login attempts.

Execution

The database is initialised with three users for testing and demonstration purposes, admin, john_doe and jane_doe.

Login Attempt


School Management System

Login

Username or Email:

Enter your username

Password:

Enter your password

Don't have an account? Register here

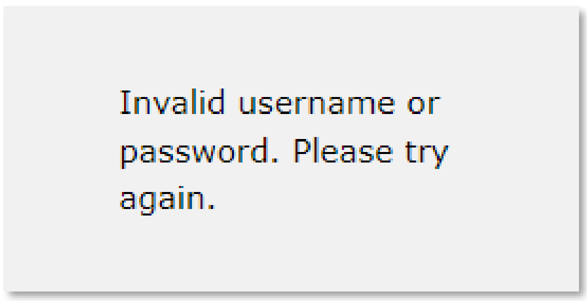
© 2024 School Management System. All rights reserved.

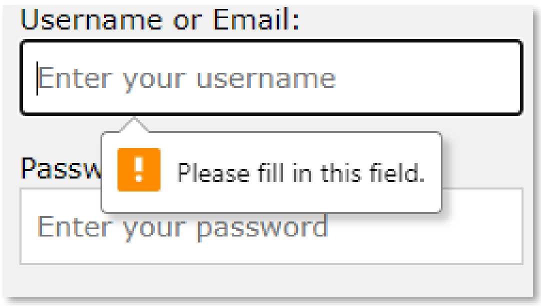
Starting the application the login screen is presented.

Entering an invalid user an error message is displayed.

Entering a valid user the system redirects to the appropriate dashboard depending on role.

If no username or password is entered a prompt is displayed.


Invalid username or password. Please try again.


Username or Email:

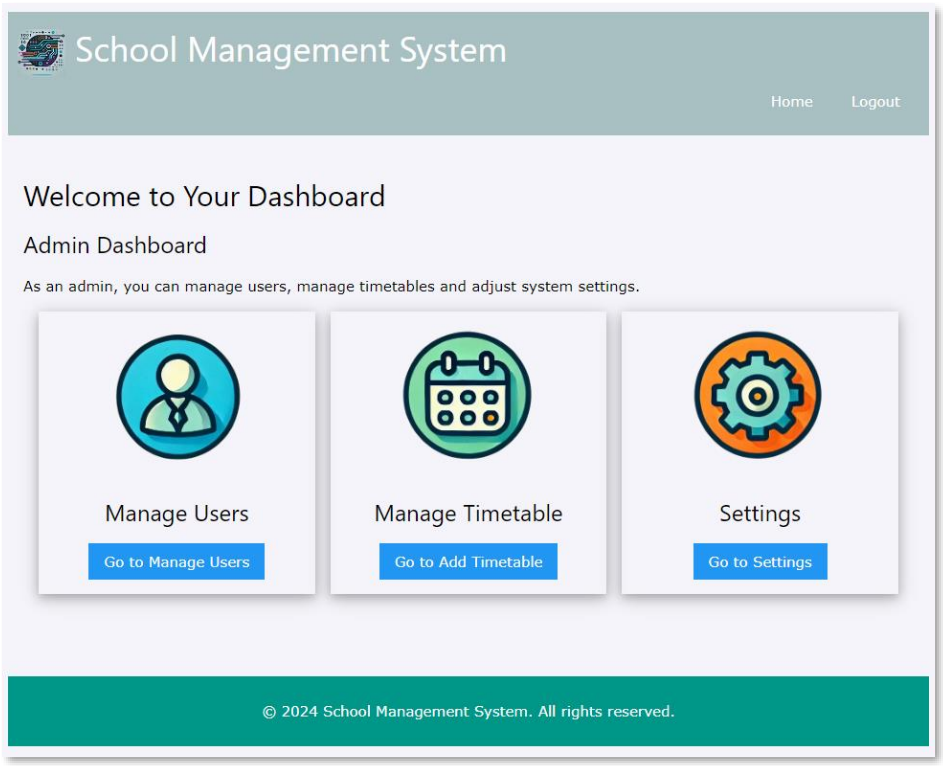
Enter your username

Please fill in this field.

Enter your password

Admin Dashboard

When logging in as an admin user the system loads the admin dashboard. This gives access to the user management, timetable management and settings pages.



Welcome to Your Dashboard

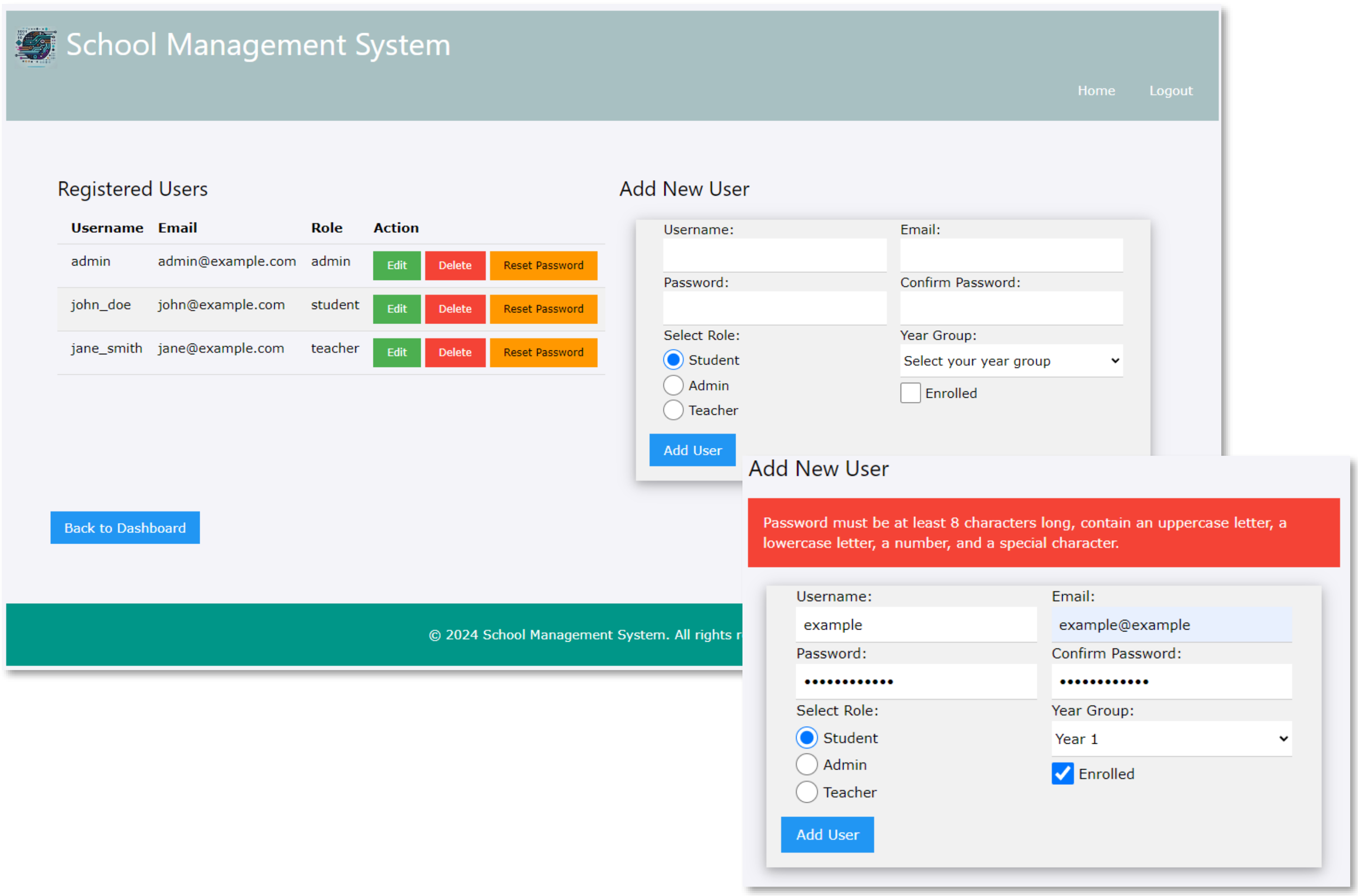
Admin Dashboard

As an admin, you can manage users, manage timetables and adjust system settings.

© 2024 School Management System. All rights reserved.

User Management

The user management page allows the admin user to view all the users registered in the system. It gives them the ability to edit and delete users as well as reset passwords. The right hand side of the screen allows for new users to be added and follow the same rules as the registration page. If a weak password is used, then an error message is displayed.



Registered Users

Username	Email	Role	Action
admin	admin@example.com	admin	Edit Delete Reset Password
john_doe	john@example.com	student	Edit Delete Reset Password
jane_smith	jane@example.com	teacher	Edit Delete Reset Password

Add New User

Username:	Email:
Password:	Confirm Password:
Select Role: [✓ Student] Admin Teacher	Year Group: Select your year group; Enrolled

© 2024 School Management System. All rights r

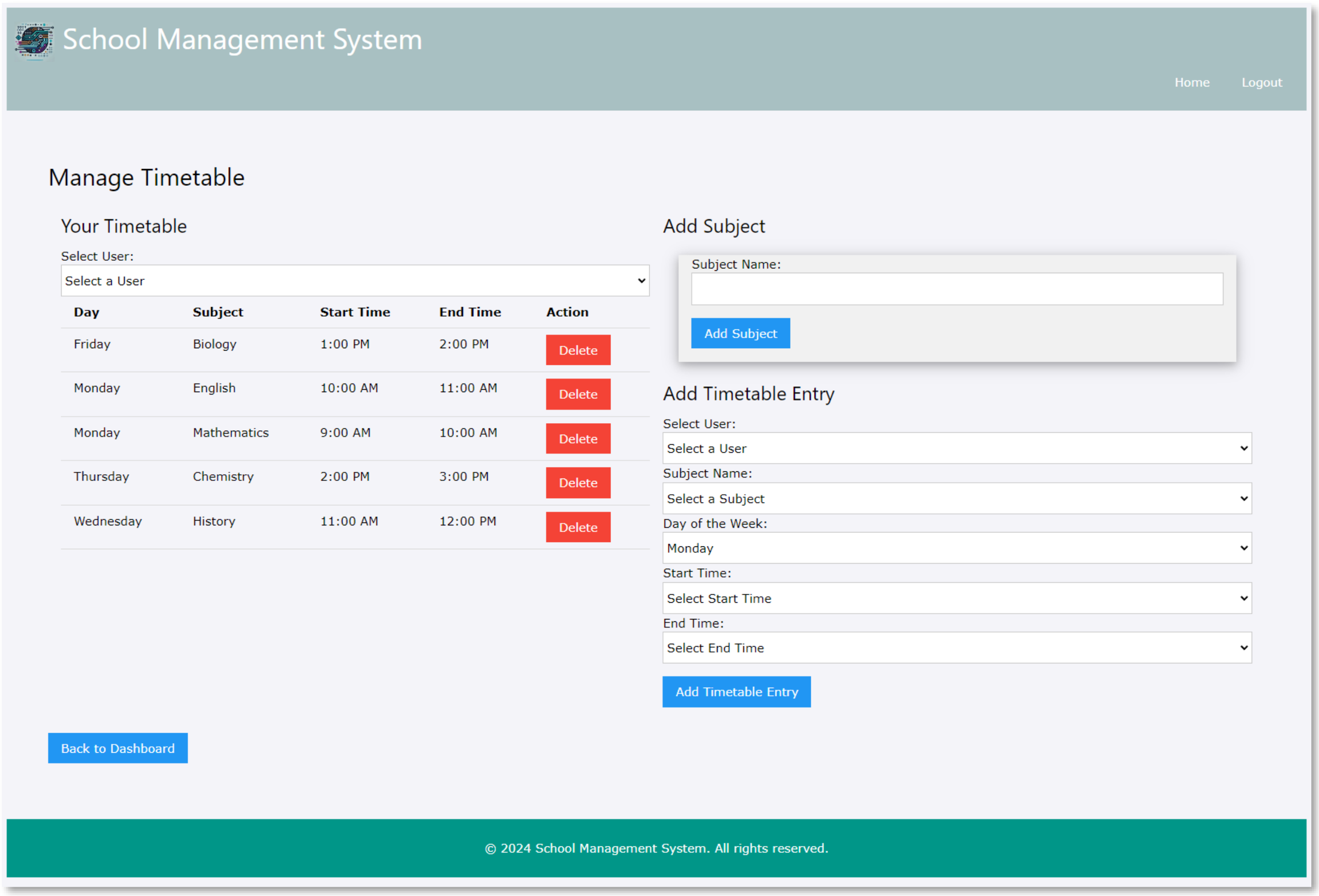
Add New User

Password must be at least 8 characters long, contain an uppercase letter, a lowercase letter, a number, and a special character.

Username: example	Email: example@example
Password: ••••••••••••	Confirm Password: ••••••••••••
Select Role: [✓ Student] Admin Teacher	Year Group: Year 1; [✓ Enrolled]

Timetable Management

The timetable management page allows admin users to view, add or delete entries form a user's timetable, either student or teacher. It also allows for the entry of new subjects.


School Management System

Home Logout

Manage Timetable

Your Timetable

Select User:

Select a User

Day	Subject	Start Time	End Time	Action
Friday	Biology	1:00 PM	2:00 PM	Delete
Monday	English	10:00 AM	11:00 AM	Delete
Monday	Mathematics	9:00 AM	10:00 AM	Delete
Thursday	Chemistry	2:00 PM	3:00 PM	Delete
Wednesday	History	11:00 AM	12:00 PM	Delete

Add Subject

Subject Name:

Add Subject

Add Timetable Entry

Select User: Select a User

Subject Name: Select a Subject

Day of the Week: Monday

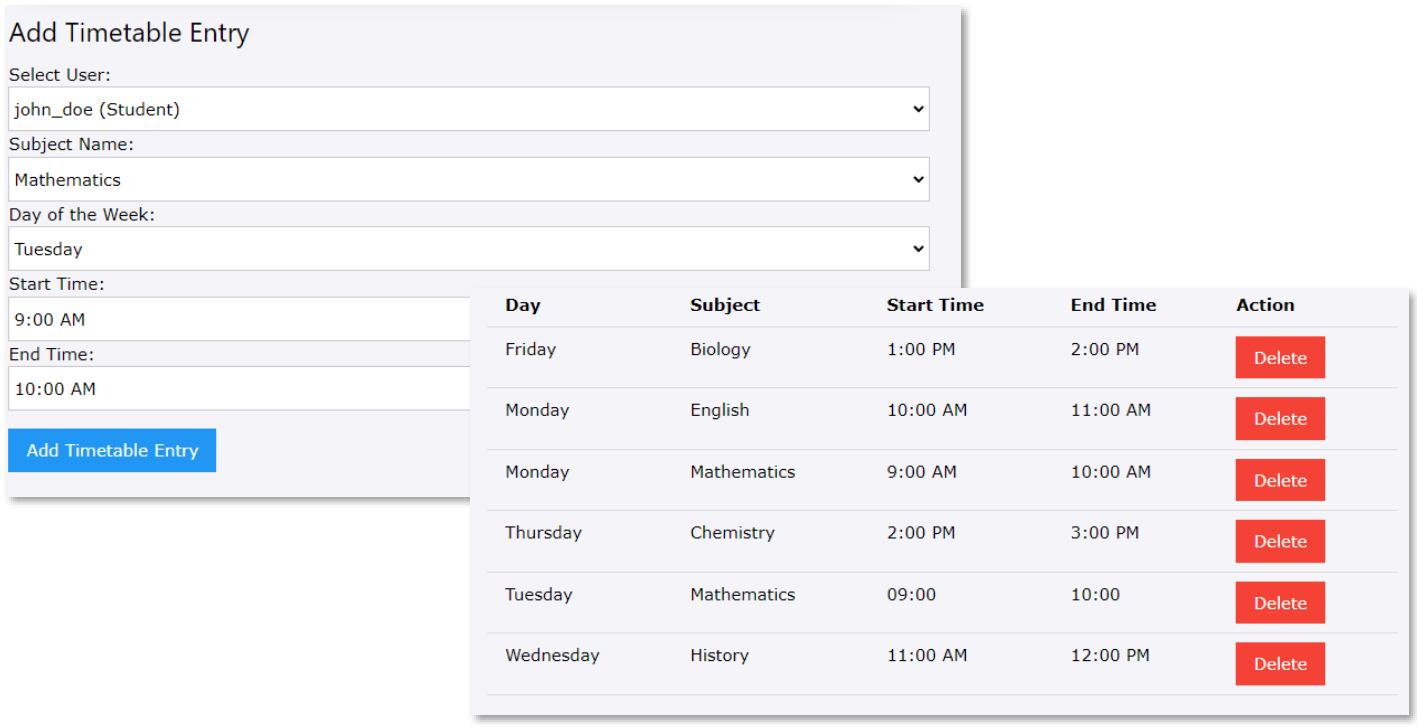
Start Time: Select Start Time

End Time: Select End Time

Add Timetable Entry

Back to Dashboard

© 2024 School Management System. All rights reserved.


Add Timetable Entry

Select User: john_doe (Student)

Subject Name: Mathematics

Day of the Week: Tuesday

Start Time: 9:00 AM

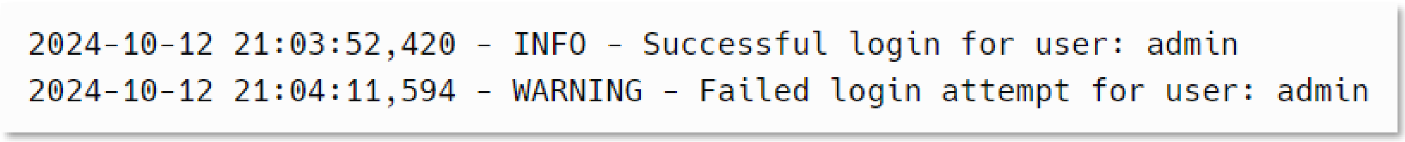
End Time: 10:00 AM

Add Timetable Entry

Day	Subject	Start Time	End Time	Action
Friday	Biology	1:00 PM	2:00 PM	Delete
Monday	English	10:00 AM	11:00 AM	Delete
Monday	Mathematics	9:00 AM	10:00 AM	Delete
Thursday	Chemistry	2:00 PM	3:00 PM	Delete
Tuesday	Mathematics	09:00	10:00	Delete
Wednesday	History	11:00 AM	12:00 PM	Delete

Logging Demonstration

When successfully logging in as the admin user the system will log this with time and date. It will also log the failed attempts.



```
2024-10-12 21:03:52,420 - INFO - Successful login for user: admin
2024-10-12 21:04:11,594 - WARNING - Failed login attempt for user: admin
```

Evaluation

The web interface of the system is responsive and intuitive. This application maintains its colour scheme and style throughout and has a well placed menu system on every screen for ease of use and better navigation.

Password management protocols have been implemented, criteria for password creation and limitations on the number of login attempts to prevent unauthorised access. Passwords are hashed using bcrypt before storage in the database to add protection for the credentials of the users. System events are logged into the system through Flask logging.

Changes applied to the database during its development phase increased its complexity and broke its normalisation. A well structured schema would make the changes and maintenance issues easier to handle in the future.

Parental access features and a staff directory were planned but the original database schema needed to be changed and would have required major revisions in much of the application. A drill down feature for the timetable was considered, but the display of the timetable was sufficient for the communicating of the information.

In future development, incorporating Google Authenticator for two factor authentication (2FA) should be considered to strengthen the login security.

This application gives a very strong backbone to the system in terms of user experience and security measures, but there are needs for further improvement in the future.

References

Fowler, M. et al. (1999) *Refactoring: Improving the Design of Existing Code.* Addison-Wesley Professional. Available from: https://learning.oreilly.com/library/view/refactoring-improving-the/0201485672/title.html [Accessed 10 October 2024].

Oxford College. (2023) *Online Education Statistics.* Available from: https://www.oxfordcollege.ac/news/online-education-statistics/ [Accessed 10 October 2024].

Bibliography

Google Cloud. (2024) *Multi-Factor Authentication (MFA) for Identity Platform.* Available from: https://cloud.google.com/identity-platform/docs/web/mfa [Accessed 10 October 2024].

Harvard University. (2022) *CS50's Introduction to Programming with Python (CS50P).* Available from: https://www.youtube.com/playlist?list=PLhQjrBD2T3817j24-GogXmWqO5Q5vYy0V [Accessed 20 September 2024].

Pallets Projects. (2024) *Flask Documentation (Version 3.0.x).* Available from: https://flask.palletsprojects.com/en/3.0.x/ [Accessed 30 September 2024].

Pallets Projects. (2024) *Jinja Documentation (Version 3.1.x).* Available from: https://jinja.palletsprojects.com/en/3.1.x/ [Accessed 30 September 2024].

Programming with Mosh. (2020) *Python Django Tutorial for Beginners.* Available from: https://www.youtube.com/watch?v=rHux0gMZ3Eg [Accessed 29 September 2024].

PyPI. (2016) *bcrypt 3.1.0.* Available from: https://pypi.org/project/bcrypt/3.1.0/ [Accessed 25 September 2024].

Python Software Foundation. (2024) *sqlite3 — DB-API 2.0 interface for SQLite databases.* Available from: https://docs.python.org/3/library/sqlite3.html [Accessed 25 September 2024].

Tech With Tim. (2020) *Flask Tutorials.* Available from: https://www.youtube.com/playlist?list=PLzMcBGfZo4-n4vJJybUVV3Un_NFS5EOgX [Accessed 30 September 2024].

W3Schools. (2024) *W3.CSS Tutorial.* Available from: https://www.w3schools.com/w3css/default.asp [Accessed 30 September 2024].

W3Schools. (2024) *Python Tutorial.* Available from: https://www.w3schools.com/python/default.asp [Accessed 25 September 2024].

W3Schools. (2024) *SQL Tutorial.* Available from: https://www.w3schools.com/sql/ [Accessed 25 September 2024].