Test Case Acceptance Criteria

User Story ID	User Story	Given	When	Then
1	As a user, I can securely log into the system and create an account for a specific user.	I have valid login credentials and multi-factor authentication is enabled.	I attempt to log in and complete the authentication process.	I am successfully authenticated and can create new user accounts with the correct permissions.
2	As a user, I can manage (create, view and edit) all data easily within the web app.	I have logged into the system, and the class data is ready for input.	I input data for tutorials, lectures, and other class types.	Information I enter is visible on data management pages and saved correctly to database.
3	As a user, I can generate an organised timetable based on data I've provided. Students should be able to be allocated to classes without clashes.	I have logged into the system, and the class data is already inputted	I am creating a timetable	The timetable is generated without any conflicting classes, and the data is accurately displayed.
4	As a user, I can move classes freely to different times and be assured that there are no clashes between students.	I have generated a valid timetable with movable events.	I drag and drop classes to different time slots.	The timetable updates in real-time, showing the changes, and ensures there are no conflicts between students' schedules.
5	As a user, I can export the created timetable to an Excel or .cal file for easy use.	A valid timetable has been generated with all the necessary class data.	I click the export button and choose my desired file format.	The timetable is successfully exported to Excel or .cal format, with all data intact and properly formatted.
6	As a user, I can use multi-factor authentication to access the website.	My user account has multi-factor authentication enabled.	I attempt to log in using my password and receive a prompt for additional authentication.	I can securely access the system after providing the correct multi-factor authentication code.

Frontend Testing

Timetable page

US3: As a user, I can generate an organised timetable based on data I've provided. Students should be able to be allocated to classes without clashes.

Test Type	Functional
Test ID	TR-F-1
Execution Type	Manual
Objective	Timetable is generated correctly and all classes are displayed accurately
Setup	Necessary data has been uploaded or manually inputted No timetable has been generated yet
Expected outcome	Class names and other information is correctly taken from the input data Classes are allocated so students can be allocated with no clashes
Steps	1. Navigate to the timetable view 2. Click the generate button to create a new timetable 3. Verify classes have been generated correctly
Time constraint	Min 2 Max 10

US4: As a user, I can move classes freely to different times and be assured that there are no clashes between students.

Test Type	Functional
Test ID	TR-F-2
Execution Type	Manual
Objective	Classes can be edited via editing form and changes are displayed correctly
Setup	Be on timetable view with an already created timetable
Expected outcome	Edited class should be displayed properly
Steps	1. click on a class event 2. change all data values in the pop up menu 3. verify class data is changed and displays correctly 4. click the square button at the top right of the class event to access context menu 5. select edit event option to access editing menu 6. change all data values in the pop up menu 7. verify class data is changed and displays correctly
Time constraint	Min 5 Max 15

Test Type	Functional
Test ID	TR-F-3
Execution Type	Manual
Objective	Classes can be moved and edited and changes are displayed correctly
Setup	Be on timetable view with an already created timetable
Expected outcome	Edited class should be displayed properly
Steps	1. drag and drop class to new location 2. verify class start and end times are changed and displayed correctly 3. click the square button at the top right of the class event to access context menu 4. change colour of class 5. verify class color is changed and displays correctly
Time constraint	Min 2 Max 10

Data management pages

US2: As a user, I can manage (view and edit) all data easily within the web app

Test Type	Functional
Test ID	TR-F-4
Execution Type	Manual
Objective	All uploaded data is displayed correctly
Setup	Be logged in
Expected outcome	all data is uploaded and organised correctly and intuitively for the user to access
Steps	1. click upload button 2. upload sample data 3. verify data is displayed correctly in data management pages
Time constraint	Min 2 Max 10

Test Type	Functional
Test ID	TR-F-5
Execution Type	Manual
Objective	All uploaded data is displayed correctly
Setup	Be logged in and have data uploaded

Expected outcome	all data is updated correctly and accessible for the user user is able to edit data easily
Steps	For each page in [Buildings, Personnel->Students, Personnel->Staff, Courses] 1. navigate to page 2. add item to database 3. edit random item in database 4. verify data is updated and displayed correctly 5. click save button to send edited data to database After all pages have been checked: 1. reload page 2. check edited data is still displayed correctly
Time constraint	Min 5 Max 10

US5: As a user, I can export the created timetable to an Excel or .cal file for easy use.

Test Type	Functional
Test ID	TR-F-6
Execution Type	Manual
Objective	Data can be exported
Setup	Be in timetable view and already have a valid timetable created
Expected outcome	Correct and readable data is downloaded
Steps	1. click download button 2. select location for download 3. open downloaded file and confirm information matches timetable
Time constraint	Min 2 Max 5

Security

US1: As a user, I can securely log into the system and create an account for a specific user.

Test Type	Functional
Test ID	TR-F-7
Execution Type	Manual
Objective	Create an account
Setup	be on log in menu
Expected outcome	A valid account is created

Steps	1. click new account button 2. register with a valid email 3. confirm registration 4. reload page and log in to make sure log in works
Time constraint	Min 2 Max 5

Test Type	Functional
Test ID	TR-F-8
Execution Type	Manual
Objective	Security authentication is functional, authorised user can access
Setup	be on log in menu already have a valid account
Expected outcome	Log in correctly
Steps	1. enter valid email and password 2. confirm log in successful and user is able to access all features
Time constraint	Min 1 Max 5

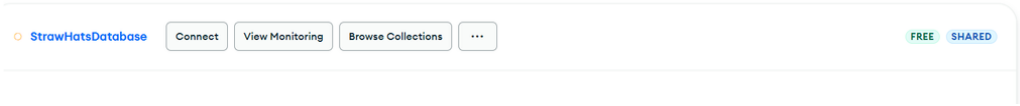
Test Type	Functional
Test ID	TR-F-9
Execution Type	Manual
Objective	Security authentication is functional, unauthorised users cannot access
Setup	be on log in menu
Expected outcome	Unauthorised individuals cannot log in
Steps	1. enter random invalid email and password 2. confirm log in is unsuccessful
Time constraint	Min 1 Max 5

Backend Testing

Test Setup:

- A separate database has been created in MongoDB for testing purposes.

Original Database:



Testing Database:

TESTING FOR BACKEND (FLASK ROUTES)

US1: As a user, I can securely log into the system and create an account for a specific user.

Test Type	Unit
Test ID	TR-B-1
Execution Type	Automatic
Objective	To rigorously verify that a new user can successfully register on the platform. The test ensures that the user's credentials (username and password) are stored correctly and securely in the database.
Setup	The test uses Flask's unit testing framework to simulate an HTTP POST request to the `/register` route in our Flask app.
Expected Outcome	• Response status code should be `200` (successful redirection). • The user should be present in the database (`users` collection). • No duplication or unauthorized registration should occur. • Proactive validation checks for password strength and uniqueness of the username.
Steps	The test submits a registration request with a sample data payload, e.g., `{'username' : 'Luffy', 'password': 'DevilFruiteater123'}`. It then verifies that the response status is `200` and that the user has been correctly added to the `users` collection using the `find_one` function.
Time constraint	

Test Type	Unit
Test ID	TR-B-2
Execution Type	Automatic

Objective	To ensure the application prevents the registration of duplicate usernames, safeguarding the integrity of user accounts. This test also checks whether the system returns appropriate error messages.
Setup	A pre-existing username is inserted into the database, followed by an HTTP POST request to the registration endpoint with the same username.
Expected Outcome	• The response should contain an error message: "Username already exists." • No new database entry should be created for the duplicate username. • The application should log the attempted duplication for security tracking. • Response status should remain `400` or an appropriate error code.
Steps	The test posts a request to the `/register` route using the duplicate username. The response is checked against the error message.
Time constraint	

US2: As a user, I can manage (create, view and edit) all data easily within the web app.

Test Type	Unit
Test ID	TR-B-3
Execution Type	Automatic
Objective	To rigorously ensure that a new document can be successfully added to the database.
Setup	The test simulates an HTTP POST request to the `/document` route with sample data, e.g., team names.
Expected Outcome	• The response status should be `201` (successfully created). • The response should include the ID of the new document. • The document should be visible in the `mycollection` collection, with consistent data formatting.
Steps	The test sends a JSON payload to the `/document` route. After the document is added, the test verifies the response status and checks that the new document appears in the correct database collection.
Time constraint	

Test Type	Unit
Test ID	TR-B-4
Execution Type	Automatic
Objective	To verify that a document can be successfully updated, ensuring data consistency and versioning in the database.
Setup	A sample document (e.g., "Old Name" and "Old Description") is inserted into the database. The test simulates an HTTP PUT request to the `/document/name/<name>` route to update the document.

Expected Outcome	• The response should indicate success, e.g., "Document successfully updated." • The status code should be `200` (OK). • The database should reflect the updated document, and prior versions should be archived where appropriate.
Steps	The test sends a PUT request to update the sample document with a new name or description.
Time constraint	

Test Type	Unit
Test ID	TR-B-6
Execution Type	Automatic
Objective	To ensure that a document can be reliably and successfully deleted from the database.
Setup	A sample document (e.g., "Delete me") is inserted into the database. The test simulates an HTTP DELETE request to the `/document/<doc_id>` route.
Expected Outcome	• The response should indicate success: "Document successfully deleted." • The status code should be `200`. • The document should be completely removed from the database with no residual references or data.
Steps	The test sends a DELETE request to remove the document using its valid ID.
Time constraint	

Test Type	Unit
Test ID	TR-B-7
Execution Type	Automatic
Objective	To verify that a document can be uploaded and stored in the root folder without errors.
Setup	The test simulates an HTTP POST request to the `/upload` route with a valid test file.
Expected Outcome	• The response should indicate success: "Document successfully uploaded." • The status code should be `200`. • The uploaded file should be accessible in the root folder, with appropriate checks for file integrity and naming conventions.
Steps	The test sends a POST request with a file and verifies that the file has been successfully uploaded to the root folder.
Time constraint	

US3: As a user, I can generate an organised timetable based on data I've provided. Students should be able to be allocated to classes without clashes.

Test Type	Unit

Test ID	TR-B-8
Execution Type	Automatic
Objective	To ensure that class data is correctly parsed into relevant components for processing.
Setup	The test simulates two CSV inputs that haven't been used for prior testing.
Expected Outcome	• The output file should match the expected output string, manually computed for accuracy. • Consistent parsing of all fields with error detection for missing or invalid data.
Steps	The CSV files are transformed into data frames, and the system iterates through the data, creating the correct data structure for the timetable.
Time constraint	

Comprehensive Test Coverage:

- **Security:** Proactively ensure robust user authentication, especially with multi-factor authentication for high-priority routes.
- **Data Integrity:** Ensure consistency across database operations (create, update, delete), with no duplication or data loss.
- **Error Handling:** Exhaustively test edge cases and validate error messages for accurate issue reporting.
- **Exporting:** Verify that data can be reliably exported in various formats (Excel, CSV), maintaining structure and content integrity.
- **Real-Time Updates:** Ensure immediate visual feedback for timetable adjustments to prevent conflicts.