UNIVERSITY of BRADFORD

Faculty of Engineering and Informatics
Department of Computer Science

Testing Document

UNIVERSITY of BRADFORD

Faculty of Engineering and Informatics
Department of Computer Science

Team 14

Project 2 : Web Interface to Interact with "Assets"

GITHUB LINK : GitHub

Members:

- Mustafa Kamran | UB: 23021169 | UB Email: mkamra11@bradford.ac.uk
- Walid Muhammad Aslam | UB: 22014646 | UB Email: wmuhamma@bradford.ac.uk
- Mohammad Subhaan Khurram | UB: 23018649 | UB Email: m.s.khurram@bradford.ac.uk
- Talhah Farooq | UB: 23014063 | UB Email: tfarooq3@bradford.ac.uk
- Luqkman Khan | UB: 23020531 | UB Email: Lkhan14@bradford.ac.uk
- Sahil Kayani | UB: 23012898 | UB Email: skayani3@bradford.ac.uk
- Hitesh Lad | UB: 23011542 | UB Email: h.lad2@bradford.ac.uk

Contents

Introduction

1.1 Project Overview

This project aims to develop a web-based platform – The Bradford Council Data Management and Mapping Website – that allows council staff and users to register, upload geospatial datasets in CSV format, visualize assets on a map, and manage data securely. Administrators will have additional control over user accounts, datasets, and system configurations.

1.2 Rationale for Topic Choice

The team chose this project to support Bradford Council's need for an efficient and secure geospatial asset management system. Many councils struggle with managing large datasets effectively, and a centralized web-based platform will streamline data upload, visualization, and decision-making. This solution enhances asset tracking, improves operational efficiency, and supports smart city initiatives by enabling better urban planning and resource allocation. It also ensures GDPR compliance and role-based access control for data security. The project contributes to sustainable development by digitizing asset data while optimizing public services and reducing administrative costs.

1.3 Collaborators & Team Expertise

A team of seven members is working collaboratively on this project, each contributing their expertise to different aspects of development and implementation. The team consists of Mustafa Kamran, Subhaan Khurram, Talhah Farooq, Sahil Kayani, Hitesh Lad, Walid Muhammad Aslam, and Luqkman Khan. With a diverse range of skills in web development, database management, software engineering, and UI/UX design, the team ensures the successful design, development, and deployment of the system. Specific areas of expertise include front-end and back-end development, geospatial data integration, security implementation, and project management, allowing for a well-structured and efficient approach to meeting the client's requirements.

1.4 City Of Bradford Metropolitan District Council

The City of Bradford Metropolitan Council is the local authority responsible for governing the metropolitan district of Bradford, located in West Yorkshire, England. It serves a diverse population and oversees various public services, including education, transport, housing, social care, environmental policies, and economic development. As one of the largest metropolitan councils in the UK, it plays a crucial role in shaping the city's infrastructure and community well-being. Bradford is known for its rich cultural heritage, thriving business sector, and commitment to urban regeneration, with the council actively working towards sustainable growth and digital transformation to improve public services.

1.5 Project Specification

The system is designed to provide users with a seamless platform for uploading, managing, and visualizing geospatial datasets through an interactive map. Users will be able to register, upload CSV files, and view mapped datasets, while administrators will have the authority to approve or reject user registrations, manage datasets, and assign departments. It will feature a secure authentication mechanism and administrative tools for managing user permissions and data integrity. Additionally, the system will adhere to web accessibility best practices and follow the council’s branding guidelines, including its colour scheme and logo.

The key stakeholders include council staff, who will upload and view datasets; administrators, responsible for managing users, datasets, and permissions; and the IT team, ensuring security and maintaining the system infrastructure for seamless operation and risk mitigation.

2 Literature Review

2.1 Introduction

The development of a web-based geospatial asset management system requires a thorough understanding of existing solutions, technologies, and best practices. This literature review explores relevant geospatial data management platforms, the technologies used for implementation, security and compliance considerations, unit testing for system reliability, and the economic and social impact of digital asset management. By analysing these aspects, we ensure that our proposed system aligns with industry standards while addressing Bradford Council's specific needs for secure and efficient asset tracking.

2.2 Existing Solutions in Geospatial Data Management

Several platforms such as ArcGIS, Google My Maps, and OpenStreetMap facilitate geospatial data visualization. ArcGIS provides advanced spatial analysis but requires costly licensing and expertise (Esri, n.d.). Google My Maps is user-friendly but lacks robust data management and administrative controls (Inc., n.d.). OpenStreetMap is open source but requires custom development for seamless integration (Wiki, n.d.). The new system addresses these limitations by providing a web-based, interactive mapping platform with administrative control, secure data handling, and role-based access for Bradford Council's asset management needs.

2.3 Technologies and Tools for Implementation

The system utilises Google Maps API, JavaScript, HTML, CSS, Java Spring Boot (MVC), MySQL, and the Gmail Server API to deliver a secure and efficient geospatial platform. Google Maps API provides real-time interactive mapping, while JavaScript ensures dynamic frontend functionality. The backend is developed in Java Spring Boot, following the Model-View-Controller (MVC) pattern, ensuring modularity, maintainability, and scalability. MySQL is implemented for structured data storage, supporting secure authentication, dataset management, and optimized queries. Additionally, the Gmail

Server API is integrated for email notifications, user verification, and automated alerts, improving system communication and security.

2.4 Security, Compliance, and Data Privacy

Handling geospatial data requires compliance with General Data Protection Regulation (GDPR) and industry security standards ((ico.), n.d.). Our system implements role-based access control (RBAC) to restrict unauthorized data modifications, alongside data encryption, secure API authentication, and audit logging for monitoring system activity. The inclusion of Gmail Server API strengthens email-based authentication, password reset functionalities, and administrative alerts, ensuring both security and user convenience.

2.5 Testing and System Reliability

To ensure high system reliability and performance, our project incorporates unit testing using JUnit for backend validation and Jest for frontend testing. These tests will cover API responses, authentication mechanisms, database operations, and UI functionalities, ensuring robustness and reducing the risk of deployment errors. Additionally, continuous testing and debugging will be carried out throughout the development lifecycle to enhance system stability.

2.6 Economic and Social Impact

Web-based asset management solutions help councils reduce operational costs, improve resource allocation, and enhance urban planning. Studies highlight that digital transformation in public services streamlines infrastructure maintenance, decision-making, and emergency response. By integrating real-time data updates, a secure user management system, and automated notifications, our platform supports Bradford Council's goal of modernizing asset tracking while ensuring efficient and sustainable city management.

3 Functional & Non-Functional Requirements

3.1 Functional Requirements

This section outlines the key functional requirements that the system must fulfil as part of this project. A more detailed breakdown of the functional requirements, along with their priorities, is provided in Table 1, while the key system interactions and relationships are visually represented in the system context diagram in Figure 1

ID	Description	Priority Level
FR1	The system shall allow the users to register with the following fields: **FR1.1:** Name, Email, Password, Confirm Password, and Department	High
FR2	The security features will include encryption for stored data, data logs for auditing all transactions and changes, and validation checks to reject invalid or duplicate entries. Additionally, the system will comply with internal security policies, and if required, additional security reviews may be conducted to ensure full compliance.	High
FR3	An admin system must be implemented which should allow: **FR3.1:** The admin to receive an email saying the user wants to get registered and the admin must be able to approve/reject user registrations. (User Permissions and Access Control)	High

	FR3.2: The ability to manage datasets (add, delete, & amend) and assign departments to other users (if required). **FR3.3:** The ability to view logs of user activity and data operations. **FR3.4:** The ability to receive alerts/notifications for pending user approvals ensuring timely management of user requests.	
FR4	The system will integrate Google Maps to display datasets with the ability to overlay multiple data layers. Category types will be assigned to datasets for improved organisation and management. This integration is confirmed, ensuring that data visualisation performance remains within acceptable limits.	High
FR5	The user will have the ability to reset their password. In such cases, a password reset email will be sent to the user's registered email address. For security purposes, the reset link will remain valid for 15 minutes.	High
FR6	The system shall support multiple layers of data (Data Layering) to be overlaid on the map, allowing users to apply filtering options to customise the displayed information.	High
FR7	The system shall display key information in a tooltip or pop-up when users hover over datasets on the map, providing a quick and interactive data preview.	High

FR8	The system shall allow users to manually enter data in addition to bulk CSV uploads, providing flexibility for data input and updates.	Medium
FR9	A search and filtering system should be included which will include: **FR9.1:** The ability to search datasets and assets based on location, asset type, and data uploaded. **FR9.2:** Search results that should be interactive (e.g. filter results dynamically). **FR9.3:** The ability to filter through displayed datasets. **FR9.4:** The system's real-time filtering performance will be rigorously tested under expected user loads before deployment to ensure seamless interaction, responsiveness, and efficiency.	Medium
FR10	The datasets will be in CSV format and will include longitude, latitude, and asset details. The system will validate CSV files for duplicates and format errors as part of its error-handling mechanism. Additionally, if a CSV file upload fails or there is a format error or any missing relevant columns from the CSV file, an automated error report will be generated to inform users of the issue.	Medium
FR11	The system may include multi-factor authentication (MFA) (Optional) / two-factor authentication (2FA) (Optional)	Low

FR12	Accessibility features include the users to be able to adjust font sizes and include colour contrast options for better readability.	Low

Table 1 Functional Requirements for the web interface for interacting with assets

3.2 Non-Functional Requirements

In addition to the functional requirements described in the previous section, there are non-functional requirements that need to be considered as part of the system design and development. These non-functional requirements not only imply the various system characteristics that are critical to the general use of the software but also identify the very important legislative and regulatory requirements that need to be kept in mind while developing enterprise-level software.

1) Performance Requirements

Description: The system must be capable of handling high volumes of data efficiently while ensuring fast processing speeds for large datasets. Specifically, CSV file uploads should be processed within 10 seconds per file (for typical data sizes), and data retrieval must be optimized for seamless user interaction. Performance benchmarking tests will be conducted to validate compliance with this requirement.

Motivation: A high-performance system ensures that users experience minimal delays when working with large datasets, thereby improving productivity and efficiency. Fast processing is particularly crucial for administrators managing large-scale geospatial data uploads.

Acceptance Criteria: The system must process and validate CSV uploads within 10 seconds for standard dataset sizes. Performance benchmarking reports must be generated during testing to ensure system compliance with processing targets. Data retrieval queries should be optimized using indexing to ensure fast access to datasets without performance bottlenecks.

2) Usability Requirements

Description: The system must provide an intuitive UI/UX that follows best design practices and complies with the Web Content Accessibility Guidelines (WCAG) 2.1-. It should include clear notifications and feedback messages to guide user actions and provide an optimal experience.

Motivation: A well-designed user interface ensures that both council staff and administrators can efficiently interact with the system without extensive training. Ensuring WCAG 2.1 compliance will make the system accessible to all users, including those with disabilities.

Acceptance Criteria: The system should implement consistent UI elements, intuitive navigation, and real-time feedback mechanisms for user actions such as form submissions, data uploads, and search queries. User testing must be conducted to verify compliance with WCAG 2.1 accessibility standards. The user interface must also support keyboard navigation and screen readers.

3) Compliance & Security

Description: The system must adhere to strict security policies to protect user data and prevent unauthorized access. It must implement encryption for database storage, enforce role-based access control (RBAC) to restrict user permissions, and be hosted on a locally managed or council-approved server. Additionally, the system must comply with the General Data Protection Regulation (GDPR) to safeguard personal data.

Motivation: With sensitive data such as user credentials, geospatial information, and administrative logs being stored, implementing robust security measures is crucial to prevent data breaches. Ensuring GDPR compliance guarantees that personal data is protected under regulatory standards. Formal documentation confirming GDPR compliance and RBAC implementation must be provided before deployment.

Acceptance Criteria: The system must enforce password encryption, restrict administrative permissions via role-based access control, and log all user interactions for security monitoring. Hosting should be on a secure, council-approved infrastructure,

ensuring compliance with local data protection regulations. A compliance review will be conducted, and documentation must be submitted to confirm adherence to GDPR and RBAC security policies before deployment.

4) Branding & Aesthetics

Description: The system must maintain a consistent visual identity by adhering to the council's branding guidelines, including official colour schemes, logos, and UI styling. The user interface should be clean, professional, and intuitive.

Motivation: A consistent brand identity enhances user trust and recognizability, ensuring that the system aligns with council-wide digital standards. A visually appealing and user-friendly interface improves engagement and usability. Before implementation, UI mock-ups must be reviewed and approved by the client to ensure adherence to branding expectations.

Acceptance Criteria: The system must implement the official council logo and colour palette across all UI components. Fonts, spacing, and iconography should align with established branding guidelines, ensuring a professional and cohesive design. UI mock-ups will be submitted to the client for final approval before the development phase begins.

5) Meeting Frequency & Project Planning

Description: To ensure clear communication and progress tracking, a structured meeting schedule will be established. The supplier will provide regular updates and key milestones in the project plan.

Motivation: Regular updates will help maintain alignment between the development team and the client, ensuring any issues are identified early and resolved efficiently.

Acceptance Criteria: Weekly progress updates will be shared with the client via email. A mid-review meeting will be scheduled to address any concerns and ensure project alignment. A high-level project roadmap will be created, detailing key milestones, major development phases, and expected completion dates.

4 Acceptance Testing

4.1 Functional Requirements

Test Case ID	Description	Input	Expected Output	Actual Output	Pass/Fail	Justification
FR1, FR3.1 + FR3.4	User Registration.	Valid name, email, password, confirm password, and department.	User account created, account creation confirmation shown on a new page, ,account approval email sent to admin.	Account created, confirmation shown, email approval sent to user.	Pass.	Matches Software Requirement Specification, details stored in database, email sent as a proof to admin.
FR2 + FR3.3	Encryption + Validation checks + data logs.	Invalid email, weak password, duplicate email. Password stored during registration. Logs stored for interactions. Admin has ability to view logs	Form rejected with error message. Password hashed and stored. User logs stored. User logs appear when admin clicks view Logs.	Proper error messages shown. Encryption verified. Logs stored. Logs shown on screen.	Pass	Client-side and backend validation applied with BCrypt password hashing. Logs stored using Log Model and displayed through the Admin Logs Page.

FR3.2	FR3.2: Manage datasets (add, delete, amend) and assign departments to users	Upload CSV Edit dataset metadata Delete dataset Change user department via access control	Dataset visible with correct metadata. Updated metadata shown. Dataset removed from DB/UI. User department updated and reflected in permissions	All actions performed correctly with success messages and expected changes in the database and UI	Pass	Full CRUD functionality confirmed for datasets; department assignment reflected in user profile and access
FR4	Google Maps API Integration	User uploads dataset with coordinates	Markers shown on map with correct lat/lng	Markers plotted correctly	Pass	Uses Google Maps API + parsed dataset
FR5	Password Reset	User clicks Reset Password, adds email address and resets	Email with reset otp sent, valid for 15 min.	Otp Email sent with 15-min expiration	Pass	OTP expires after 15 minutes. Backend checks expiration and denies expired tokens.

FR6	Data layering with filtering on map.	User can upload multiple datasets at once which are shown with different coloured markers on map, filters are added to dynamically switch to or out of datasets	Layered data with multiple dataset upload shown, filtering to distinguish datasets.	Layered data shown ,Dataset filtering and markers update instantly	Pass	Layering and filter logic implemented in JavaScript; datasets visualized with distinct marker colors and toggles for filtering.
FR7	Tooltip Data Preview	On hover over dataset marker on map Tooltip appears with dataset summary (e.g., name, lat/lng)	Hovering over a marker pops up a tooltip showing details.	Tooltips shown with expected information on hover	Pass	JavaScript handles tooltip rendering using dataset data; improves interactivity and accessibility
FR8	Manual Asset Entry and Bulk Uploads	User clicks “Add Asset” and inputs lat/lng/name and user can select multiple datasets to bulk upload on map.	New marker placed and stored in database and Multiple datasets uploaded one by one.	Asset added onto map and multiple datasets upload with different coloured markers	Pass	Form inputs, DB store, multiple file upload allowed with marker sync.

FR9.1 + 9.2 + 9.3 +9.4	Search & Filtering System	Search using keywords, dataset names, emails, departments, etc.	Matching datasets and assets dynamically filtered on screen	Search results update instantly on input, with no reload	Pass	JS event listeners, dataset categories, and filter backend logic; tested with multiple dataset types and sizes for responsiveness
FR10	CSV Validation	Upload malformed CSV / missing lat/lng	Error message shown, file rejected	CSVs with missing headers rejected messages shown	Pass	Uses JS and backend validation logic
FR11	Two-Factor Authentication (2FA)	Valid login credentials + correct OTP from email	Redirected to user homepage after successful OTP	OTP prompt shown after login, redirect occurs only after valid OTP	Pass	Email service integrated via Gmail API; OTP stored temporarily in DB and validated before session begins.

FR12	Accessibility Features	Toggle colourblind mode and increase/decrease font size	Text changes to high-contrast yellow-on-black; font resizes accordingly	Colour scheme updates immediately; font size changes based on toggle	Pass	JavaScript handles colourblind mode with local Storage; font resizing implemented using accessibility buttons and Tailwind utilities

4.2 Non-Functional Requirements

Test Case ID	**Description**	**Input**	**Expected Output**	**Actual Output**	**Pass/Fail**	**Justification**
NFR1	Performance: Upload large CSV datasets	CSV file (2000+ rows)	Upload < 10 seconds	Upload completes in ~6 seconds	Pass	Meets benchmark requirement from SRS
NFR2	Usability: WCAG + accessibility	Toggle colourblind mode and adjust font size	Dark mode with readable yellow-on-black with adjusted font size	Colourblind mode toggles properly with adjustable font size	Pass	Implemented with local Storage, user tested.
NFR3	Security: Role-based access & encryption	Admin-only page access attempt	Unauthorized user blocked, passwords encrypted	Role-based access enforced; passwords hashed	Pass	Spring Security + BCrypt used; admin routes protected

NFR4	Branding & Aesthetics	UI Inspection	Council logo, blue theme, consistent design	UI matches branding guidelines	Pass	Follows bradford.gov.uk style, client approved
NFR5	Compliance: GDPR + Logs + RBAC	Data interaction	Logging, encryption, limited access	All implemented	Pass	Logging of user actions (e.g., login, file uploads) and secure access based on roles verified through backend and testing.

5 Unit Testing

5.1 Introduction

JUnit is a widely used testing framework in the Java ecosystem designed to write and execute repeatable automated tests. It plays a critical role in unit testing, allowing developers to verify the correctness of individual components (usually methods or classes) in isolation. JUnit promotes the test-driven development (TDD) approach, where tests are written before the actual code, helping improve software reliability and maintainability.

With features like annotations (@Test, @BeforeEach, @AfterEach, etc.), assertions (assertEquals, assertTrue, assertThrows, etc.), and integration with build tools and IDEs, JUnit enables developers to quickly write expressive and efficient test cases. Combined with mocking frameworks like Mockito, JUnit allows developers to simulate complex dependencies and focus on testing the business logic independently from the data layer or external services.

In this project, JUnit was used to validate the functionality of the service layer, ensuring that each component behaves as expected under various conditions before integration with the rest of the application.

5.2 Brief Overview

We used JUnit for this project to test the core application logic in a reliable and structured way. Our focus was on testing the service layer because it contains the main business logic and acts as the bridge between the controllers and repositories. By testing services in isolation using Mockito to mock dependencies, we ensured accurate behavior without relying on database or email systems. We did not test controllers or repositories directly, as they either contained minimal logic or were already covered through integration testing. This approach allowed us to validate critical functionality efficiently while maintaining clean and maintainable test cases.

5.3 Junit Tests

5.3.1 EncryptorTest

Description:

Verifies that the encryptString() method correctly hashes a raw password using BCrypt.

The test asserts that the hashed password can be validated against the original password using BCryptPasswordEncoder.matches(), confirming the encryption is valid and secure.

5.3.2 DatasetAccessRequestServiceImplTest

Description

The DatasetAccessRequestServiceImplTest class thoroughly tests the service responsible for managing dataset access requests within the system. Using Mockito to mock dependencies (DatasetAccessRequestRepository and EmailService), this test suite verifies that the service correctly retrieves all access requests, updates request statuses (e.g., APPROVED), and checks whether a user has already submitted a request. It also ensures that appropriate email notifications are sent to both the user and the admin when a new request is saved. Furthermore, it validates the retrieval of individual requests by dataset name and email, checks if a request is approved, and confirms the proper functioning of filtering and searching logic. Each test contributes to confirming that the access request feature operates reliably and meets both functional and integration expectations in isolation from other layers.

5.3.3 The DatasetMetadataServiceTest

Description

The DatasetMetadataServiceTest class rigorously validates the behavior of the DatasetMetadataService, which manages dataset metadata operations. Using Mockito, the test mocks the underlying DatasetMetadataRepository to isolate the service layer and verify method interactions. The test cases ensure that dataset metadata is correctly stored, retrieved, searched, and updated. It checks whether the service properly detects duplicate datasets by name and uploader, supports keyword-based and role-filtered searching, and can delete or update metadata as expected. These tests confirm that the service reliably delegates operations to the repository and handles both positive and negative conditions, providing confidence in the correctness and robustness of the metadata-related functionality within the application.

5.3.4 EmailServiceTest

Description

The EmailServiceTest class focuses on verifying the functionality of the EmailService, which integrates Gmail API to send system-generated emails such as OTPs, signup alerts, and approval notifications. To isolate testing from actual email sending, the Gmail service and its related methods are fully mocked using Mockito. The tests ensure that core functionalities like generating and sending OTP emails, signup admin notifications, and user approval confirmations execute without throwing exceptions. Additionally, the createEmail() method is verified to construct MIME-compliant messages successfully. By mocking external dependencies, this test ensures reliability and stability of the email notification flow without relying on live Gmail services.

5.3.5 IndividualAssetServiceImplTest

Description

The IndividualAssetServiceImplTest class provides comprehensive unit tests for the IndividualAssetServiceImpl, which manages CRUD operations for manually added map assets. The tests mock the underlying IndividualAssetRepository to isolate service logic and ensure methods function correctly without hitting the database. The testAddAsset_Success test validates asset creation, while testGetAssetsByDataset_ReturnsCorrectList checks dataset-level filtering. Additionally, update and delete operations are tested in testUpdateAsset_Success and testDeleteAsset_Success, confirming correct delegation and return values. Field injection is used to inject the mock repository, ensuring seamless and focused service testing.

5.3.6 LoginServiceImplTest

Description

The LoginServiceImplTest class thoroughly tests the LoginServiceImpl service, which handles user authentication and retrieval. It uses a mocked UserRepositoryImpl to isolate and verify service behavior. The testAuthenticateUser_ValidCredentials and testAuthenticateUser_InvalidPassword tests validate correct and incorrect password authentication using BCrypt hashing. The testFindUserByEmail_UserExists and testFindUserByEmail_UserNotFound methods verify retrieval behavior based on email presence. Additionally, testGetAllUsers_ReturnsList confirms the service properly fetches all users, and testUpdateUserPassword_HashesCorrectly ensures the password is securely hashed before updating. These tests confirm the integrity, security, and functionality of the login service layer.

5.3.7 LogServiceImplTest

Description

The LogServiceImplTest class validates the functionality of the LogServiceImpl, which is responsible for recording user activity logs such as login attempts. It uses a mocked LogRepositoryImpl to verify that log entries are properly delegated to the repository. The testLogSuccess and testLogFailed tests confirm that both successful and failed login attempts are logged by asserting that the saveLog() method is invoked exactly once with any LogModel. These unit tests ensure that critical actions within the application are consistently tracked, supporting accountability, auditing, and security compliance.

5.3.8 LogServiceMySQLIntegrationTest

Description

The LogServiceMySQLIntegrationTest performs integration-level testing of the logging functionality by interacting directly with the live MySQL database. It verifies that logs created via the LogServiceImpl are successfully inserted and retrievable from the logs table. The testInsertLogIntoMySQL test triggers the logging operation, while testLogExistsInMySQL establishes a JDBC connection to query the actual database and validate that the expected log entry exists. This integration test ensures end-to-end functionality and confirms that database connectivity, SQL execution, and log persistence work as intended in a real deployment environment.

5.3.9 RegistrationServiceTest

Description

The RegistrationServiceTest class thoroughly tests the RegistrationService logic using mocked dependencies for the repository, encryption, and email services. It verifies that a new user is successfully registered when their email does not already exist, and ensures that a notification email is sent to the admin. The test also confirms that duplicate emails are properly rejected without saving or triggering email notifications. Additionally, the authentication functionality is validated for correct password matching, incorrect password handling, and scenarios where the user is not found. These tests ensure both the registration and login flows behave securely and as expected, covering key functional and edge cases.

5.3.10 PermissionRequestServiceImplTest

Description

The PermissionRequestServiceImplTest class validates the behavior of the PermissionRequestServiceImpl service layer by mocking interactions with its dependent PermissionRequestRepository and EmailService. It tests critical functionalities such as retrieving all permission requests, approving requests (including email notification on success), denying requests, and retrieving request-related emails. The test ensures that user creation and notifications occur only when valid data is retrieved from the repository and verifies the correct invocation of methods. These unit tests reinforce the integrity of the permission management process in the system, covering both success and failure paths.

5.3.11 ResetPasswordServiceImplTest

Description

The ResetPasswordServiceImplTest rigorously validates the OTP-based password reset workflow by mocking dependencies such as EmailService and LoginServiceImpl. It confirms that OTPs are correctly generated, stored, and sent via email, and tests OTP verification under both valid and invalid conditions. The test cases also ensure that the password reset only succeeds when a valid, unexpired OTP is provided, and that the appropriate method is called to update the user's password. Reflection is used to inspect and manipulate the private otpStorage field, ensuring full control over the internal state during tests. This suite effectively ensures the integrity, correctness, and security of the OTP reset logic.

5.3.12 UploadDatasetServiceImplTest

Description

The UploadDatasetServiceImplTest validates the core functionality of dataset uploading, streaming, retrieval, and deletion within the service layer. It uses Mockito to mock the UploadDatasetRepository, ensuring isolated testing of the service logic without dependency on actual database operations. The tests confirm that processDatasetUpload() and uploadDatasetStreamed() delegate correctly to the repository, returning the expected success status. The retrieval test (getDatasetContent()) verifies that dataset content is correctly fetched and structured, while deleteDatasetFile() ensures the deletion logic is properly invoked. This test suite ensures that dataset handling operations are reliable and interact appropriately with the persistence layer.

5.3.13 UserDetailsServiceImplTest

Description

The UserDetailsServiceImplTest ensures the correctness of the custom implementation of UserDetailsService, which is a key component in Spring Security authentication. This test validates that when a user exists in the system (mocked using UserRepository), the loadUserByUsername() method returns a UserDetails object containing the expected email, password, and authorities. Conversely, when the user is not found in the repository, the service correctly throws a UsernameNotFoundException. This confirms that both successful and unsuccessful login scenarios are handled as intended, making the service secure and compliant with Spring Security's authentication requirements.

6 Code Inspection Report

6.1 Code Inspection Overview

Code inspection is a manual process where source code is reviewed for potential issues in readability, structure, consistency, and adherence to best practices. In this project, each static webpage (HTML/CSS/JavaScript) and non-object-oriented code was inspected to ensure clean, maintainable, and accessible design. The inspection focused on aspects such as consistent indentation, semantic HTML usage, responsive layout with Tailwind CSS, accessibility features (e.g., color contrast and font resizing), and proper separation of concerns in JavaScript logic. This process helped identify and fix minor bugs, enhance code quality, and ensure compliance with web standards and project requirements.

6.2 Code Inspection

6.2.1 CSS File

The CSS code in this project was inspected to ensure maintainability, consistency, and accessibility across all static webpages. It employs a clear structure with semantic class naming and grouped sections (e.g., navigation, tables, home content, footer, map controls, search, and permissions styling), promoting modularity and reusability. Consistent font choices (Arial, Roboto Slab), adequate padding and margins, and defined hover effects improve user experience. The layout is responsive using percentage widths and flexbox-based designs. Accessibility considerations are visible in colour contrast, button focus states, and large font sizes. The code also avoids redundancy and adheres to best practices, with scoped styles and minimal use of global selectors.

6.2.2 Map.js File

The JavaScript code was inspected for quality, clarity, and maintainability. It is modular and well-structured, separating responsibilities such as initializing the map, handling dataset uploads, parsing CSV/JSON formats, managing dynamic marker placement, and supporting interactive UI features like filtering, uploading, and manual asset editing. The use of fetch, FormData, and event listeners enhances asynchronous behaviour and user experience. Defensive programming is demonstrated through validation checks (e.g., missing coordinates, invalid files), and constants such as marker colours and API keys are handled systematically. Moreover, best practices like avoiding global variable pollution, using let/const, and leveraging modern JavaScript functions (e.g., arrow functions, forEach) are followed. The code also promotes extensibility through its use of reusable methods (e.g., createFilters, addAssetToSidebar). Overall, the script adheres to functional and readability standards expected of client-side mapping interfaces.

6.2.3 Admin Files

6.2.3.1 Admin_Dashboard HTML 5 File

This static HTML page, enhanced with Tailwind CSS and Thymeleaf templating, follows a clean semantic structure for a responsive admin dashboard. The layout is organized into a header, main content area, and dual footer sections. Accessibility is considered with a togglable colourblind mode, and the interface is mobile-responsive using Tailwind's utility classes. Each navigation block is clearly labeled with visual hierarchy through headings and iconography. Footer links are informative and follow good accessibility practices. Overall, the code is readable, well-indented, and modular, promoting usability and visual consistency across devices.

6.2.3.2 Admin_Dataset_List HTML 5 File

This page provides an admin-facing interface for managing uploaded datasets using Thymeleaf for dynamic rendering and Tailwind CSS for modern, responsive styling. It features a functional dataset search and filter section with inputs for department, role, and keywords. Each dataset is displayed in a structured table with clearly labelled action buttons—View, Edit, and Delete—enhancing usability. Accessibility is thoughtfully addressed through a colourblind mode toggle, and the layout adapts well across devices. The code is semantically organized, making it easy to maintain and extend, while visual hierarchy and spacing support clarity and user experience.

6.2.3.3 Edit_dataset HTML 5 File

This static HTML/Thymeleaf page facilitates the editing of dataset metadata by admins. It maintains semantic structure and uses Tailwind CSS for clean, responsive styling. The form ensures required input validation for editable fields while maintaining read-only status for non-editable fields such as the uploader and role. Accessibility features are integrated via a colourblind toggle that persists using local storage. The layout ensures clarity and usability, offering a smooth user experience across devices. Overall, the code is structured, maintainable, and meets the functional and accessibility requirements for editing datasets.

6.2.3.4 Logs HTML 5 File

This HTML/Thymeleaf template displays the user logs and includes filtering functionality for log records by email, action, and status. It is styled using Tailwind CSS for responsive design and features accessibility support through a persistent colourblind mode toggle. The layout is logically structured with a consistent header, navigation bar, main log table, and dual footer. The use of dynamic data rendering via Thymeleaf ensures server-side integration, while accessibility and responsiveness improve the user experience. Overall, the code follows good practices in separation of concerns and maintainability.

6.2.3.5 Permissions HTML 5 File

This static HTML/Thymeleaf template handles the permissions management view for both dataset access requests and user role/department change requests. It leverages Tailwind CSS for a clean, responsive layout and incorporates accessibility support via a colourblind mode toggle. The code is well-structured with a header, dynamic navigation, clear form inputs for search and filters, and two logically separated tables displaying requests. The use of Thymeleaf ensures dynamic server-side rendering, while embedded scripts maintain a seamless user experience. The design is consistent and aligns with usability and accessibility best practices.

6.2.3.6 View_dataset HTML 5 File

This static HTML/Thymeleaf template displays the contents of a selected dataset in a dynamic, responsive table format. It uses Tailwind CSS for layout styling and incorporates a consistent navigation bar and accessible design, including a toggleable colourblind mode. The template is dynamic—headers and rows are generated from backend data using Thymeleaf, allowing any dataset structure to be displayed without hardcoding column names. The interface is clean and user-friendly, with clear navigation, fallbacks for empty datasets, and styling that supports both readability and accessibility.

6.2.4 User's Files

6.2.4.1 Dataset_list HTML 5 File

This static Thymeleaf page presents a responsive and accessible interface for users to browse all uploaded datasets. It integrates filtering capabilities (by keyword, department, and role) and dynamically displays each dataset's metadata. Based on dataset access status, users are shown context-aware actions such as "View", "Request Access", or "Download". Tailwind CSS ensures consistent styling, while accessibility features like colourblind mode enhance usability. The use of Thymeleaf enables dynamic rendering and conditional display of UI components depending on user permissions or dataset status.

6.2.4.2 Requests HTML 5 File

This page displays a logged-in user's dataset access request history using a clean, responsive layout built with Tailwind CSS and Thymeleaf. It dynamically populates dataset names, statuses, and timestamps via server-side variables. The page includes accessible navigation, colourblind mode support, and consistent UI patterns. Static HTML and dynamic content are cleanly separated, and semantic tags ensure structure and maintainability. Overall, the code adheres to modern accessibility, responsiveness, and usability standards.

6.2.4.3 User_dashboard HTML 5 File

The User Dashboard provides a clean and intuitive interface for users to navigate available datasets and track their access requests. Built with Thymeleaf and Tailwind CSS, the page uses semantic HTML for accessibility and responsive design for cross-device support. It includes reusable components such as a consistent header, navigation bar, and footer. The colourblind mode toggle improves accessibility, and hover effects enhance interactivity. Overall, the code follows best practices in structure, styling, and maintainability.

6.2.4.4 User_Home HTML 5 File

The Index Page serves as the main interface for dataset interaction, offering file upload, asset addition, map visualization, and filtering functionality. It is built using Thymeleaf and Tailwind CSS for a clean, responsive UI. Accessibility features include a colourblind mode toggle with persistent preference using localStorage. The JavaScript-driven dataset upload logic supports both small (in-memory) and large (streamed) files in CSV/JSON formats, with user feedback and validation. The modular layout and interactive map area enhance usability. Overall, the page reflects a well-structured, scalable, and user-friendly design.

6.2.4.5 View_datasets HTML 5 File

The Dataset View Page displays detailed tabular data for a selected dataset in a clean, responsive format using Thymeleaf and Tailwind CSS. The table dynamically renders

column headers and values from the dataset object, with a fallback message if the dataset is empty. Accessibility is thoughtfully integrated through a persistent colourblind mode toggle using localStorage. The page includes standard navigation, a structured header and footer with contact/social links, and consistent user experience design across the platform. The layout is mobile-friendly, semantic, and visually accessible.

6.2.5 Login HTML 5 File

The Login Page offers a clean, responsive interface for user authentication, using Tailwind CSS for styling and Thymeleaf for dynamic content binding. It includes email and password fields with real-time client-side validation and error feedback. Accessibility is a core feature, with a persistent colourblind mode toggle supported via localStorage. The page also provides easy navigation to password reset and sign-up, plus consistent branding through the logo, navigation, and dual footer with council details and social links. Overall, the page is user-friendly, accessible, and securely designed.

6.2.6 Reset HTML 5 File

The Reset Password Page provides a secure and accessible interface for initiating the password recovery process. Built with Tailwind CSS for responsive and clean design, the page features a single-field form for email input, enhanced with real-time client-side validation and user-friendly error messages. Accessibility is prioritized through a colourblind mode toggle, persisted via `localStorage`. The layout is centred and minimalistic, directing user focus to the task at hand. Consistent branding is maintained via the header, footer, and council information, reinforcing trust and ease of navigation.

6.2.7 Reset_verify HTML 5 File

The Reset Password Verification page facilitates secure password resets by verifying the user's OTP and allowing them to set a new password. It maintains a clean, accessible layout with a centred form and Tailwind CSS for responsive design. Users must provide a valid email, a 6-digit OTP, and matching passwords. The page includes client-side validation for real-time feedback and accessibility support through a toggleable colourblind mode, preserved with localStorage. Consistent branding and contact information are presented in the header and dual footers, ensuring trust and continuity across the site.

6.2.8 Signup HTML 5 File

The Sign-Up page provides a clean, structured interface for new users to register, utilizing Tailwind CSS for modern styling and responsive layout. It includes fields for name, department selection, email, and password confirmation. Client-side validation ensures real-time error feedback, verifying non-empty fields, valid email format, minimum password length, and matching passwords. Accessibility is prioritized through a toggleable colourblind mode that persists via localStorage. Consistent branding is maintained across headers and dual footers, with clear navigation to login and contact links. The form integrates seamlessly with Spring's Thymeleaf model binding using th:field attributes.

6.2.9 Signup_pending HTML 5 File

The Signup Request Sent page delivers a user-friendly confirmation interface styled with Tailwind CSS. It clearly communicates to users that their registration has been submitted for administrative approval. The design is minimal yet informative, maintaining consistency with the overall application layout through a responsive structure, accessible colourblind mode toggle, and dual-footer branding. Navigation back to the login page is straightforward via a prominent CTA button. Accessibility considerations are persistently respected across scripts and styles, ensuring a clean and inclusive user experience post-registration.

6.2.10 Application Properties File

This application.properties file configures a Spring Boot web application named webinterface. It sets up a MySQL database connection (project_two) with Hibernate for ORM and configures Thymeleaf for rendering HTML templates. It specifies static resource locations and enables file uploads up to 50MB, supporting large dataset submissions. The application is integrated with Gmail SMTP to send emails (e.g., OTPs, verification), and includes a placeholder for a Google Maps API key for map-based features. Accessibility features and template caching are also configured to enhance usability and development efficiency.

Peer Review

7 Peer Review – Overview

The team worked collaboratively to develop a high-quality Testing document while also contributing to different technical and documentation aspects of the project. Each team member played a vital role in research, development, and implementation, ensuring that the project met its functional and non-functional requirements. The peer review assessment evaluates each member's contributions on a scale of 1 to 10 marks, based on research, technical work, collaboration, and overall contribution.

8 Evaluation Criteria

Each team member is rated based on:

Technical Contributions (Backend, Frontend, Database, API Integration, UI/UX)

Research and Documentation (SRS, Functional/Non-Functional Requirements, Testing Plans, Security Considerations, Testing Document)

Collaboration and Communication (Meetings, Teamwork, Sprint Planning, Code Reviews, Issue Resolution)

9 Peer Review Table

Team Member	Role & Key Contributions	Marks / 10	Comments & Justification
Mustafa Kamran (Team Leader)	Backend Development (Spring Boot), Logging & Permissions, Security Integration	10/10	Led backend development using Spring Boot with MVC + JDBC structure, implemented dataset upload and access request systems. Managed role-based access, Gmail API integration for notifications, and added custom logging with filters. Coordinated team workflow and ensured code quality.
Subhaan Khurram (Speaker)	Frontend Development, UI/UX Design, Interface Implementation, Tailwind Integration	10/10	Designed responsive UI using Tailwind CSS, improved accessibility (colourblind mode, mobile responsiveness), integrated maps with upload/view features. Ensured frontend matched Bradford Council branding. Delivered clear project walkthroughs.

Sahil Kayani	Documentation, Requirements, Compliance & UI Testing	9/10	Drafted and refined requirement specs and user stories. Contributed to non-functional requirements and ensured accessibility compliance. Helped validate usability and interface responsiveness.
Talhah Farooq	Database Design, Geospatial Integration, Map Functionality	10/10	Structured relational DB schema for metadata and asset storage. Managed asset CRUD operations and integrated Google Maps API for visualization. Ensured performance optimization with large datasets.
Walid Muhammad Aslam	Authentication, Security Flows, Email Notifications	9/10	Implemented login/logout flows with Spring Security, contributed to RBAC and password reset workflows. Connected Gmail SMTP for OTPs and approval alerts. Supported logging and 2FA groundwork.

Luqkman Khan	Testing, Bug Fixing, Deployment Workflow	9/10	Performed integration and junit testing, suggested key refinements for frontend responsiveness and backend logic. Assisted in planning deployment procedures and organizing the codebase for readability.
Hitesh Lad	UI Prototyping, Code Assistance, Documentation	9/10	Helped build wireframes and visual layout plans. Supported Java backend structure with method-level documentation and maintained minutes of meetings to keep progress consistent.

9.1 Justification for Scores

Each team member contributed significantly to the project, whether through backend development, frontend implementation, database structuring, UI design, security, or documentation. The high marks reflect the commitment, problem-solving ability, and teamwork displayed throughout the development process. Those primarily involved in core technical implementation (Backend, Frontend, Database, API Integration) received a full 10 marks, while those who contributed more to documentation, UI refinements, and testing were awarded 9 marks, recognizing their valuable contributions.

Bibliography

(ico.), I. C. O., n.d. Guide to the General Data Protection Regulation (GDPR). *https://ico.org.uk/media/for-organisations/guide-to-the-general-data-protection-regulation-gdpr-1-0.pdf*.

010, B. W., 2025. Spring MVC with MySQL and Junit. *https://medium.com/%40bytewise010/spring-mvc-with-mysql-and-junit-80c23935fe79*.

Blog, M., 2023. What is Agile Methodology? A 10-Minute Guide. *https://www.usemotion.com/blog/agile-methodology*.

Esri, n.d. ArcGis Online. *https://www.esri.com/en-us/arcgis/products/arcgis-online/overview*.

Inc., G., n.d. Google My Maps. *https://www.google.co.uk/maps/about/mymaps/*.

kaalel, n.d. MVC Framework Introduction. *https://www.geeksforgeeks.org/mvc-framework-introduction/*.

Tanzu, S. b. V., n.d. Spring Boot. *https://spring.io/projects/spring-boot*.

Wiki, n.d. About OpenStreetMap. *https://wiki.openstreetmap.org/wiki/About_OpenStreetMap*.