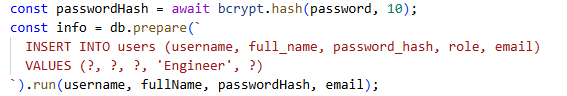
**System Functionalities – Narrative Explanation**

**🔐 User Registration and Authentication**

To fulfil the requirement of secure user registration, password hashing was implemented using the bcrypt library. This approach ensures that raw user passwords are never stored in the database, protecting user credentials even if the database is compromised. The hashed version is generated with a salt round of 10 before being persisted to the users' table. This improves overall security and aligns with standard authentication practices.



Code snippet: User sign-up with secure password hashing as part of the user registration endpoint

This ensures that no plaintext passwords are stored, which aligns with best practices in cybersecurity and regulatory expectations around data protection.

The rationale for building this functionality stems from the need to support accountability in inspection records and prevent unauthorised access. By ensuring each engineer has a unique identity and secure credentials, the system can log inspection activity accurately and prevent manipulation of records.

This functionality enables traceability and user-specific customisation in the future, such as role-based access control or engineer-specific dashboards. It also lays the groundwork for future scalability and potential compliance with industry security standards such as ISO/IEC 27001.

**🔑 User Login and Session Handling**

To support secure login functionality, the system retrieves the user record based on the submitted email and verifies the password using bcrypt. compare. The bcrypt library compares the hashed password stored in the database with the plain-text password entered by the user. This process ensures that password verification is done without ever exposing sensitive data.

This method was implemented to meet the authentication requirement while ensuring that user credentials remain secure, even in the event of a data breach. By only returning a successful login if a match is confirmed, the system defends against brute force and dictionary attacks.

The significance of this approach is that it upholds both security and usability. Users can log in easily while the system maintains high standards of data protection — an essential concern in any system that handles personal information.



**👷 Engineer Resolution for Inspections**

When submitting an inspection form, the system must associate the submission with a specific engineer. To achieve this, a helper method (resolveEngineerId) was implemented. This function first checks whether the engineer’s ID is already known. If not, it attempts to retrieve the user by their email. If no such user exists, it automatically creates one with the provided name and email, ensuring uniqueness.

This functionality was developed to bridge the potential gap between user registration and form submission, especially in scenarios where engineer data might come from pre-existing records or third-party forms. Automatically resolving or creating an engineer streamlines the user experience and prevents form submission errors due to missing user IDs.

The benefit of this approach is a reduction in user friction and database inconsistency. It ensures that every inspection is tied to a valid, unique engineer in the system, which is essential for reporting, auditing, and future analytics features.

**🏢 Site and Zone Management**

To support accurate categorisation of inspections, the system allows engineers to define **Sites** and **Zones**. A site represents a physical location (e.g., a building), while a zone represents a more specific sub-area (e.g., a floor or room). Functionality was implemented to create these entities via API endpoints, as well as to **“ensure”** them — meaning they are created only if they do not already exist.

This dual-purpose logic was implemented to reduce redundancy in the database and to enhance user experience on the frontend. Engineers interacting with the mobile app can input new site or zone names directly without needing to pre-register them in a separate interface.

The result is a more fluid and efficient workflow for users, with less manual data entry and fewer form submission errors. The backend logic also maintains data integrity and ensures normalised relationships between inspections, zones, and sites.

**📦 Item Management**

The application also supports **Items**, which represent the objects being inspected (e.g., emergency lights, fire extinguishers). Each item is linked to a **Zone** and has an associated **Item Type**. Items can be created through the API, and the system includes an ensure endpoint that allows frontend clients to dynamically check for existence and create items as needed.

This was necessary to align the digital representation of inspection forms with the physical layout and inventory of safety equipment in facilities. By allowing item-level granularity, the system enables a higher level of detail in inspection records.

This supports better maintenance tracking, historical reporting, and equipment lifecycle management. It also ensures that inspection data corresponds to real-world assets in a consistent, structured format.

**🧾 Item Types and Templates**

To standardise inspections, the system supports **Item Types**, which are high-level categories such as “Emergency Light” or “Fire Alarm.” These item types belong to categories like “Facility” or “Machine Safety.” Each item type can have associated **Subcheck Templates** — predefined checks that should be performed when inspecting that type of item.

The design and implementation of this model allow for the dynamic generation of inspection forms based on item type selection. Engineers select an item type, and the system retrieves the appropriate subcheck templates via the API.

This ensures that inspections are consistent, complete, and guided by predefined standards. It also reduces human error by automating form structure, making future reporting and compliance auditing more reliable.

**✅ Inspection Submission and Retrieval**

The core functionality of the app is to **create and retrieve inspection records**. Each inspection captures various fields, including date, item inspected, engineer performing the inspection, zone and site location, and a summary result. Once submitted, inspections are stored in the backend and can be retrieved individually or as a summary list.

The motivation behind this is to provide digital traceability for maintenance activities and safety compliance checks. Instead of relying on paper forms or isolated spreadsheets, organisations can maintain a centralised, queryable, and secure inspection database.

The result is a system that supports compliance tracking, defect resolution workflows, and historical analysis of equipment performance. Engineers and managers alike can benefit from increased transparency and faster information access.

**🔄 Dynamic Mobile Form Behaviour**

The mobile application integrates deeply with the backend to offer a responsive and personalised user interface. For instance, when an engineer selects the category “Facility,” the app dynamically fetches all related item types from the API and populates the form. Similarly, selecting an item type fetches the corresponding subcheck templates, which define the structure of the inspection form in real time.

This feature was implemented to accommodate evolving requirements in safety inspections, where templates may change or expand. By retrieving templates from the backend, the mobile app remains lightweight and adaptable.

This dynamic behaviour not only reduces maintenance overhead on the app but also empowers non-technical users (e.g., managers) to update inspection structures by modifying backend data without needing to deploy new app versions.

**🧪 Health Check and Infrastructure Readiness**

A /health endpoint was implemented to support deployment, monitoring, and testing activities. This endpoint simply returns { ok: true }, confirming that the API is running and reachable.

This may appear trivial, but it plays a critical role in infrastructure diagnostics, CI/CD pipelines, and frontend-app readiness checks. It supports resilience, faster debugging, and a smoother developer experience.

**🎯 Conclusion**

The system’s functionality is intentionally modular, scalable, and aligned with best practices in software engineering. Each feature was developed not in isolation, but as part of a cohesive architecture that supports usability, data integrity, and long-term maintainability.

The login and registration features recently implemented were especially crucial for grounding the entire application in secure user accountability, paving the way for all subsequent inspection records to be traceable and personalised.

By building both flexibility (dynamic forms, ensuring logic) and structure (normalised relationships, authentication), this solution stands as a robust prototype for a real-world maintenance inspection platform — with clear paths for future enhancements such as role-based access, reporting dashboards, and mobile offline capabilities.