Asset Management

Problem statement:

Hexaware currently uses a variety of systems to handle and keep track of various kinds of assets. However, we are developing our application to lessen the inconvenience and difficulties brought on by the lack of a single platform to track all the assets. All of the company's assets that an employee has borrowed will be tracked by the application on a single platform. It aids the business in enhancing the system for tracking and recording the assets connected to each employee, enhancing the current system's accuracy and accountability.

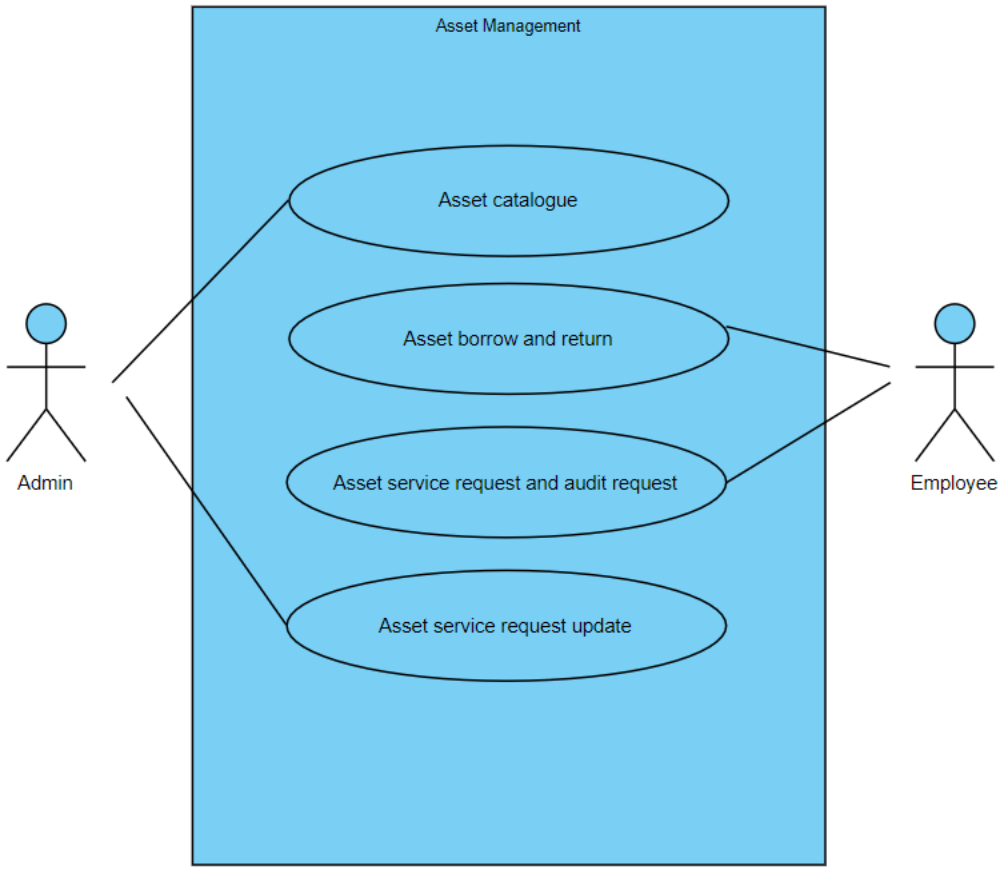
Scope

1. **Employee Registration and Authentication:** Allow users to register, log in, and securely manage their accounts.
2. **Asset catalogue:** Display a wide range of Asset with detailed information, including images, descriptions, and asset status.
3. **Asset Borrowing and Return:** Enable employee to request for new asset and return the tagged asset
4. **Asset Service:** Enable the employee to raise service request and admin can update the service request.
5. **Asset Audit and report:** Admin can send the audit request employee and employee can audit the asset.

Technologies:

- Frontend: React.js / Angular Js.
- Backend: Java, Spring Boot/C#, .Net / Python Djnago for API development.
- Database: MySql / Sql Server.
- Authentication: JSON Web Tokens (JWT) for secure user authentication.

Use case Diagram:



Use Cases:

Actor: User/Employee

- Use Case: User/Employee Registration and login.
- Use Case: View allocated asset.
- Use Case: Request for new asset.
- Use Case: Verify the asset from admin asset verification request.
- Use Case: Raise asset service/return request.

Actor: Administrator

- Use Case: Log In
- Use Case: Employee Management
- Use Case: Asset Catalog Management
- Use Case: IT Asset Audit and Reporting

System: Security and Authentication

- Use Case: Authenticate User

System: Database Management

- Use Case: Store IT Asset Information
- Description: Manages the storage of information related to IT assets, including details such as asset type, specifications, and current status.
- Use Case: Store IT Employee Data

- Description: Involves storing and managing IT employee data, including login credentials, personal information, and IT asset allocation history.

Development Process:

7. **Employee Registration and Admin Login:**
    a. Employee and Admin can create accounts, providing personal details (name, gender, contact number, address, etc.)
    b. The system validates the information and creates user profiles.
    c. Employee and Admin log in using their credentials (username/email and password).
8. **Employee Dashboard:**
    a. Users can browse Asset, view detailed descriptions, images, and prices.
    b. Users navigate through Asset categories (such as Laptop, Furniture, Car, Gadgets) search for specific items.
        i. Include filters for category.
        ii. Implement a search feature with auto-suggestions and predictive text.
        iii. Clicking on asset from listing asset to view more information about a asset. Asset details, images, status are displayed.
    c. Employee can raise the request for the new asset usage.
    d. Employee can raise the request for asset for the service by providing the following details (assetNo, description, issueType(malfunction, repair)).
9. **Admin Dashboard:**
    a. Admin can add new asset by providing following information assetNo, assetName, assetCategory, assetModel, manufacturingDate, expiryDate, assetValue.
    b. Admin can delete and update the asset and can delete the existing employee.
    c. Admin can send the asset audit request to all employee and can view the asset audit status (pending, verified, rejected)
    d. Admin can receive the asset service request and can change the status of request.
    e. Admin can view the total asset that assigned to employee.
    f. Admins can add, modify, or remove asset categories.
10. **Security and Compliance**
    a. User authentication and authorization are enforced to ensure data privacy.

1. JWT Authentication:

JWT authentication involves generating a token upon successful user login and sending it to the client. The client includes this token in subsequent requests to authenticate the user.

a. User Login: Upon successful login (using valid credentials), generate a JWT token on the server.
b. Token Payload: The token typically contains user-related information (e.g., user ID, roles, expiration time).
c. Token Signing: Sign the token using a secret key known only to the server. This ensures that the token hasn't been tampered with.
d. Token Transmission: Send the signed token back to the client as a response to the login request.
e. Client Storage: Store the token securely on the client side (e.g., in browser storage or cookies).

2. JWT Authorization:

JWT authorization involves checking the token on protected routes to ensure that the user has the required permissions.

a. Protected Routes: Define routes that require authentication and authorization.

b. Token Verification:
    i. Extract the token from the request header.
    ii. Verify the token's signature using the server's secret key.
c. Payload Verification:
    i. Decode the token and extract user information.
    ii. Check user roles or permissions to determine access rights.
d. Access Control: Grant or deny access based on the user's roles and permissions.

Logout:

- Logging out involves invalidating the JWT token on both the client and the server to prevent further unauthorized requests.

Project Development Guidelines

The project to be developed based on the below design considerations.

1	**Backend Development**	• Use Rest APIs (Springboot/ASP.Net Core WebAPI to develop the services • Use Java/C# latest features • Use ORM with database • perform backend data validation • Use Swagger to invoke APIs • Implement API Versioning • Implement security to allow/disallow CRUD operations • Message input/output format should be in JSON (Read the values from the property/input files, wherever applicable). Input/output format can be designed as per the discretion of the participant. • Any error message or exception should be logged and should be user-readable (not technical) • Database connections and web service URLs should be configurable • Implement Unit Test Project for testing the API • Implement JWT for Security • Implement Logging • Follow Coding Standards with proper project structure.

Frontend Constraints

1.	**Layout and Structure**	Create a clean and organized layout for your registration and login pages. You can use a responsive grid system (e.g., Bootstrap or Flexbox) to ensure your design looks good on various screen sizes.
2	**Visual Elements**	**Logo:** Place your application's logo at the top of the page to establish brand identity.
		Form Fields: Include input fields for email/username and password for both registration and login. For registration, include additional fields like name and possibly a password confirmation field.
		Buttons: Design attractive and easily distinguishable buttons for "Register," "Login," and "Forgot Password" (if applicable).

		Error Messages: Provide clear error messages for incorrect login attempts or registration errors.
		Background Image: Consider using a relevant background image to add visual appeal.
		Hover Effects: Change the appearance of buttons and links when users hover over them.
		Focus Styles: Apply focus styles to form fields when they are selected
3.	**Color Scheme and Typography**	Choose a color scheme that reflects your brand and creates a visually pleasing experience. Ensure good contrast between text and background colors for readability. Select a legible and consistent typography for headings and body text.
4.	**Registration Page, add product page by seller, add shipping address page by user**	**Form Fields:** Include fields for users to enter their name, email, password, and any other relevant information. Use placeholders and labels to guide users.
		Validation: Implement real-time validation for fields (e.g., check email format) and provide immediate feedback for any errors. **Form Validation:** Implement client-side form validation to ensure required fields are filled out correctly before submission.
	Registration Page	**Password Strength:** Provide real-time feedback on password strength using indicators or text. **Password Requirements**: Clearly indicate password requirements (e.g., minimum length, special characters) to help users create strong passwords.
		Registration Success: Upon successful registration, redirect users to the login page.
5.	**Login Page**	**Form Fields:** Provide fields for users to enter their email and password.
		Password Recovery: Include a "Forgot Password?" link that allows users to reset their password.

6.	**Common to React/Angular**	• Use Angular/React to develop the UI. • Implement Forms, databinding, validations, error message in required pages. • Implement Routing and navigations. • Use JavaScript to enhance functionalities. • Implement External and Custom JavaScript files. • Implement Typescript for Functions Operators. • Any error message or exception should be logged and should be user-readable (and not technical). • Follow coding standards. • Follow Standard project structure. • Design your pages to be responsive so they adapt well to different screen sizes, including mobile devices and tablets.

Good to have implementation features

1. Generate a SonarQube report and fix the required vulnerability.
2. Use the Moq framework as applicable.
3. Create a Docker image for the frontend and backend of the application .
4. Implement OAuth Security.
5. Implement design patterns.
6. Deploy the docker image in AWS EC2 or Azure VM.
7. Build the application using the AWS/Azure CI/CD pipeline. Trigger a CI/CD pipeline when code is checked-in to GIT. The check-in process should trigger unit tests with mocked dependencies.
8. Use AWS RDS or Azure SQL DB to store the data.