FINAL REPORT

22i-0760 Talha Ahmad

22i-0873 Ibrahim Shahid

22i-1180 Omer Mustafa

Project Introduction

Our Lost and Found application is designed to help users report lost or found items. The application allows users to create accounts, report items with details, search for items, and claim items reported by others. The system also provides notifications for item claims and claim status updates.

Functional Requirements

1. User registration and login
2. Item reporting with details
3. Item searching and filtering
4. Item claiming and management
5. Notification system for claims

Non-Functional Requirements

1. Responsive user interface
2. Secure user authentication
3. Efficient search functionality
4. Scalable database storage

User Stories

1. As a user, I want to register and log in to use the application.
2. As a user, I want to report lost or found items with details.
3. As a user, I want to search for items using keywords.
4. As a user, I want to filter items by category.
5. As a user, I want to claim items reported by others.
6. As a user, I want to receive notifications when someone claims my item.
7. As a user, I want to receive notifications when my claim is accepted or rejected.
8. As a user, I want to edit items I've reported.
9. As a user, I want to delete items I've reported.
10. As a user, I want to view my notifications.
11. As a user, I want to mark notifications as read.
12. As a user, I want to see a list of my reported items.
13. As a user, I want to see a list of claimed items.
14. As a user, I want to navigate easily between different sections.
15. As a user, I want to log out of the application.

Product Backlog

High Priority

1. Secure User Authentication

- **Description**: As a user, I want to securely log in so that my account is protected.
- **Acceptance Criteria**:
    - Replace crypto.pbkdf2 with bcrypt for password hashing.
    - Enforce JWT secret via environment variables (remove hardcoded fallback).
    - Add rate limiting (5 attempts/minute) on auth endpoints.

2. Owner-Only Item Modifications

- **Description**: As a user, I want to ensure only I can edit/delete my reported items.
- **Acceptance Criteria**:
  - Fix MyItems filter to use reportedById (current code uses client-provided username).
  - Add middleware to validate user ownership on PUT/DELETE endpoints.

3. Input Validation

- **Description**: As a user, I want proper form validation to prevent invalid submissions.
- **Acceptance Criteria**:
  - Add regex validation for emails in signup/login (Frontend + Backend).
  - Sanitize item descriptions to block HTML/script tags.

Medium Priority

4. Advanced Search Filters

- **Description**: As a user, I want to filter items by date range and status (found/lost).
- **Acceptance Criteria**:
  - Add createdAt date picker in frontend.
  - Extend backend query to support date ranges and found status.

5. User Profile Management

- **Description**: As a user, I want to update my profile information.
- **Acceptance Criteria**:

    - Add PATCH /api/users endpoint.
    - Create profile page with editable username/email fields.

6. Pagination for Item Lists

- **Description**: As a user, I want paginated results to avoid long loading times.
- **Acceptance Criteria**:
    - Implement limit and offset parameters in GET /api/items.
    - Add "Load More" button in frontend.

7. Claim the Lost Item

- **Description**: As a user, I want to claim a lost item.
- **Acceptance Criteria**:
    - Add a claim option with every Lost Item.

Low Priority

8. Alerts

- **Description**: As a user, I want alerts when someone claims an item reported by me.
- **Acceptance Criteria**:
    - Claim the item.

Sprint 1 Backlog

Duration: 2 weeks | Focus: MVP & Security

User Story/Task	Priority	Acceptance Criteria	Story Points
1. Secure User Authentication	High	Implement bcrypt, enforce JWT secret via env, add rate limiting	8
2. Report Lost Items	High	Report an Item that you found	5
3. Input Validation	High	Email regex + sanitize item descriptions	3
4. Fix Static Salt in Password Hashing	High	Replace crypto with bcrypt dynamic salt	5
5. Remove Debug console.log	Medium	Clean logging in itemRoutes.js	1

Total Sprint 1 Points: 22

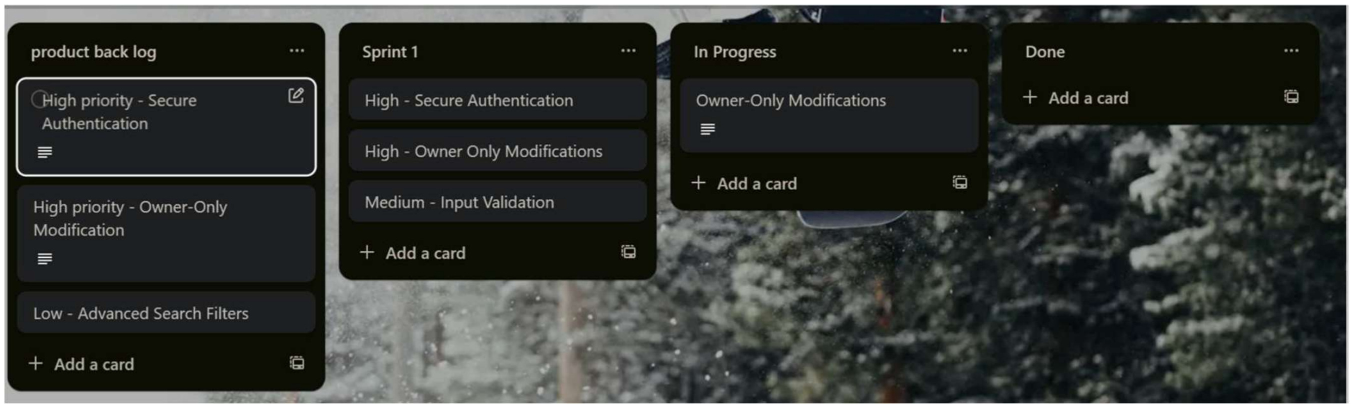
Sprint 2 Backlog

Duration: 2 weeks | Focus: Usability & Scalability

User Story/Task	Priority	Acceptance Criteria	Story Points
1. Advanced Search Filters	Medium	Add My Items filter	5
2. Owner-Only Item Modifications	High	Fix MyItems filter, add ownership middleware	5
3. Filtration for Item Lists	Medium	Implement filtering through the categories	3

Total Sprint 2 Points: 13 (~1/4 of total backlog size)

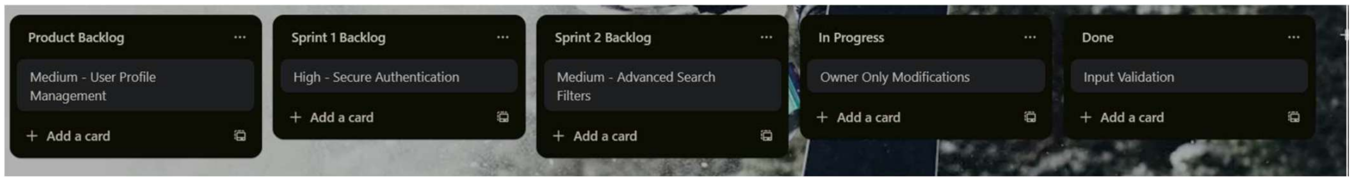
Trello Board


product back log
- High priority - Secure Authentication
- High priority - Owner-Only Modification
- Low - Advanced Search Filters

Sprint 1
- High - Secure Authentication
- High - Owner Only Modifications
- Medium - Input Validation

In Progress
- Owner-Only Modifications

Done

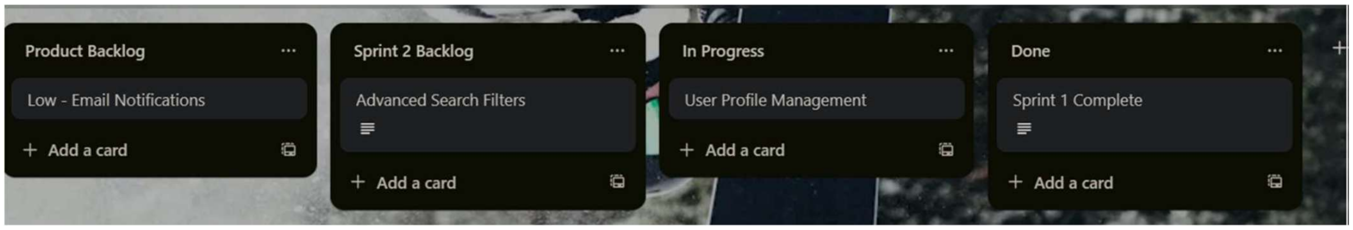

Product Backlog
- Medium - User Profile Management

Sprint 1 Backlog
- High - Secure Authentication

Sprint 2 Backlog
- Medium - Advanced Search Filters

In Progress
- Owner Only Modifications

Done
- Input Validation


Product Backlog
- Low - Email Notifications

Sprint 2 Backlog
- Advanced Search Filters

In Progress
- User Profile Management

Done
- Sprint 1 Complete

Project Plan

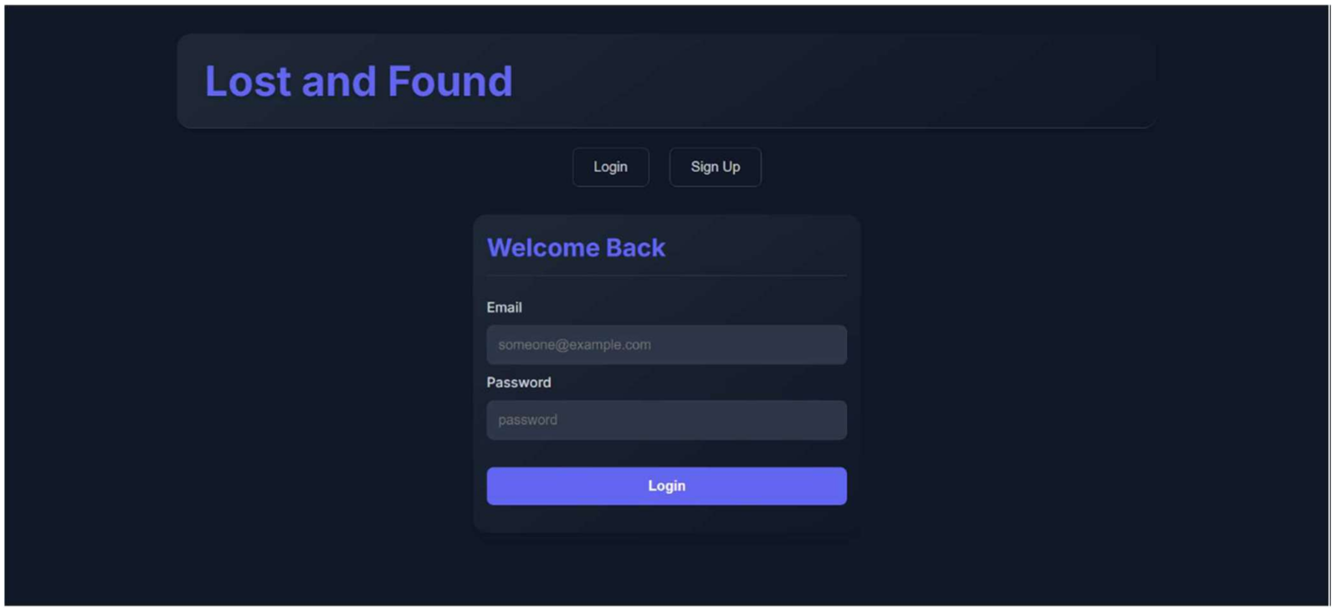
Our project follows an agile development methodology with two sprints. The first sprint focuses on core functionality, including user authentication, item reporting, and basic search. The second sprint implements input validation, advanced search filtering, and editing and deleting. The third sprint should implement claiming, and basic notification system.

Design

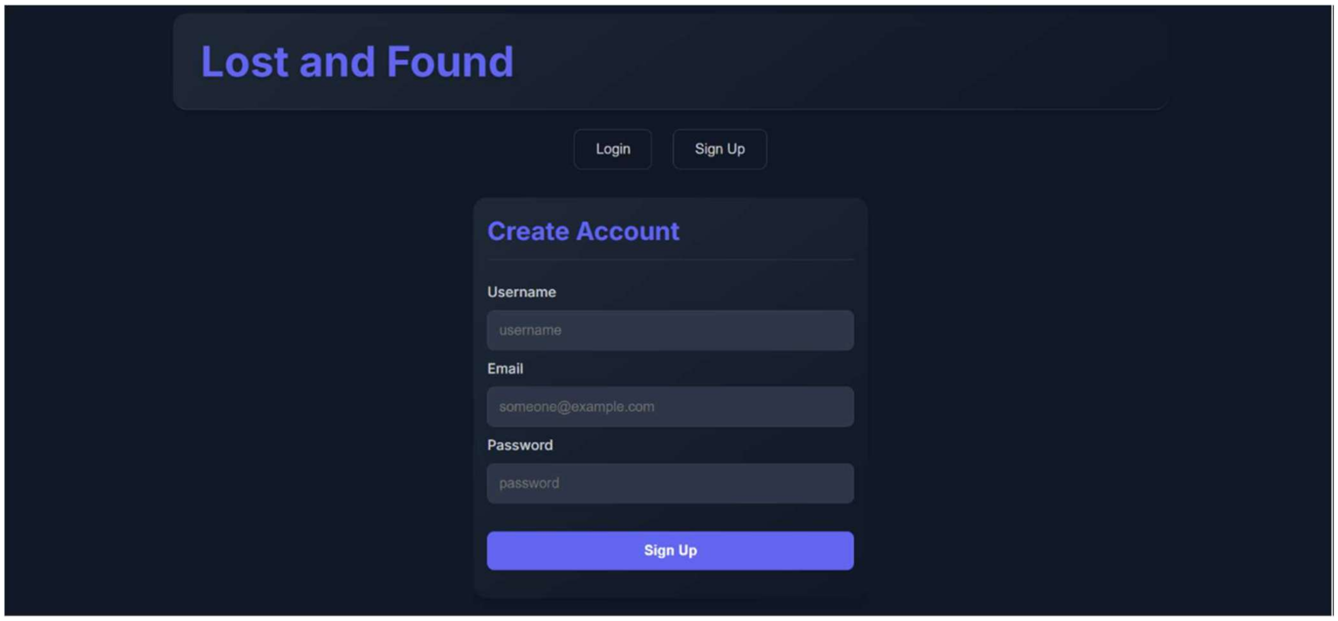
Our design follows a layered architecture with separate layers for presentation, business logic, and data access. The frontend uses React for a responsive user interface, while the backend uses Express.js and MongoDB for data storage

Screenshots

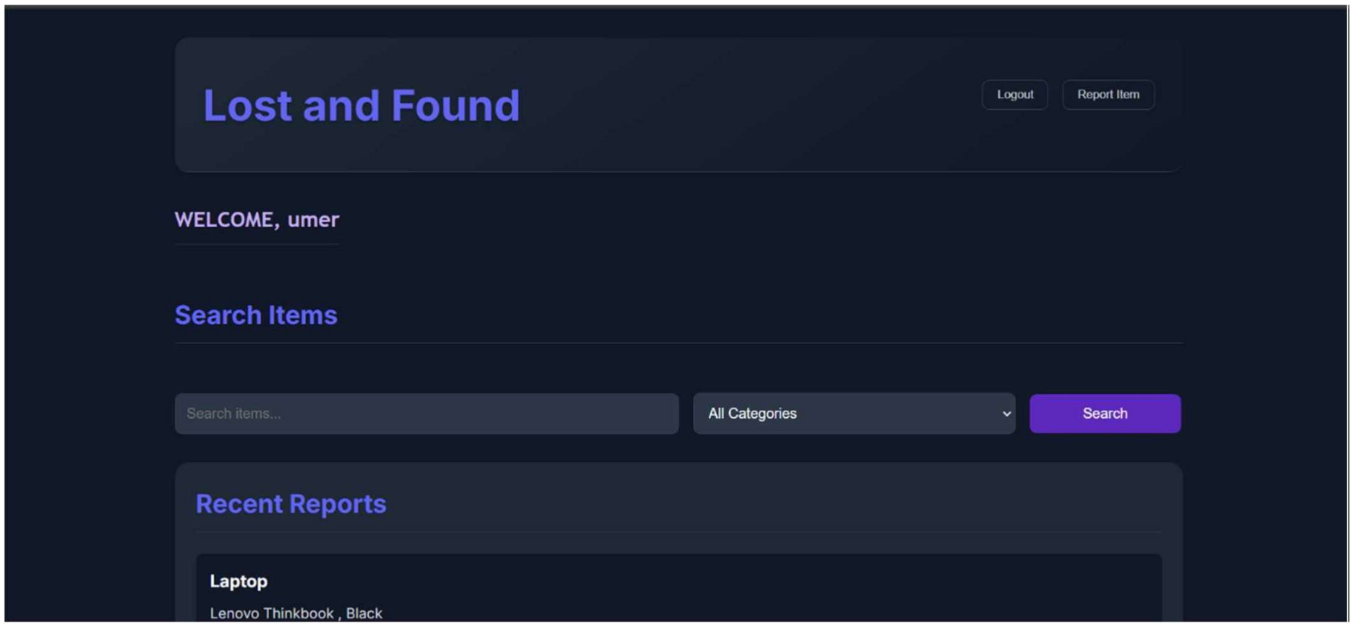
- **Log In**



- **Sign Up**



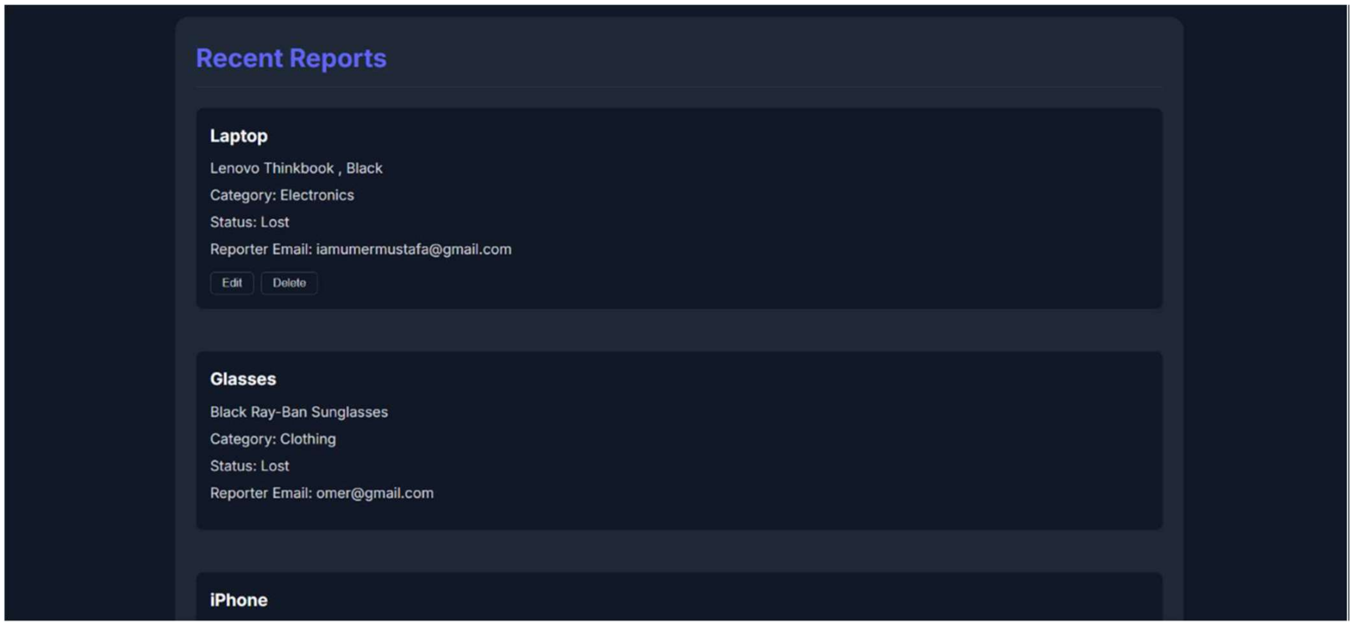
- **Home Page and Searching/Filtering**



Laptop

Lenovo Thinkbook , Black

- **Reports**



Laptop

Lenovo Thinkbook , Black

Category: Electronics

Status: Lost

Reporter Email: iamumermustafa@gmail.com

Glasses

Black Ray-Ban Sunglasses

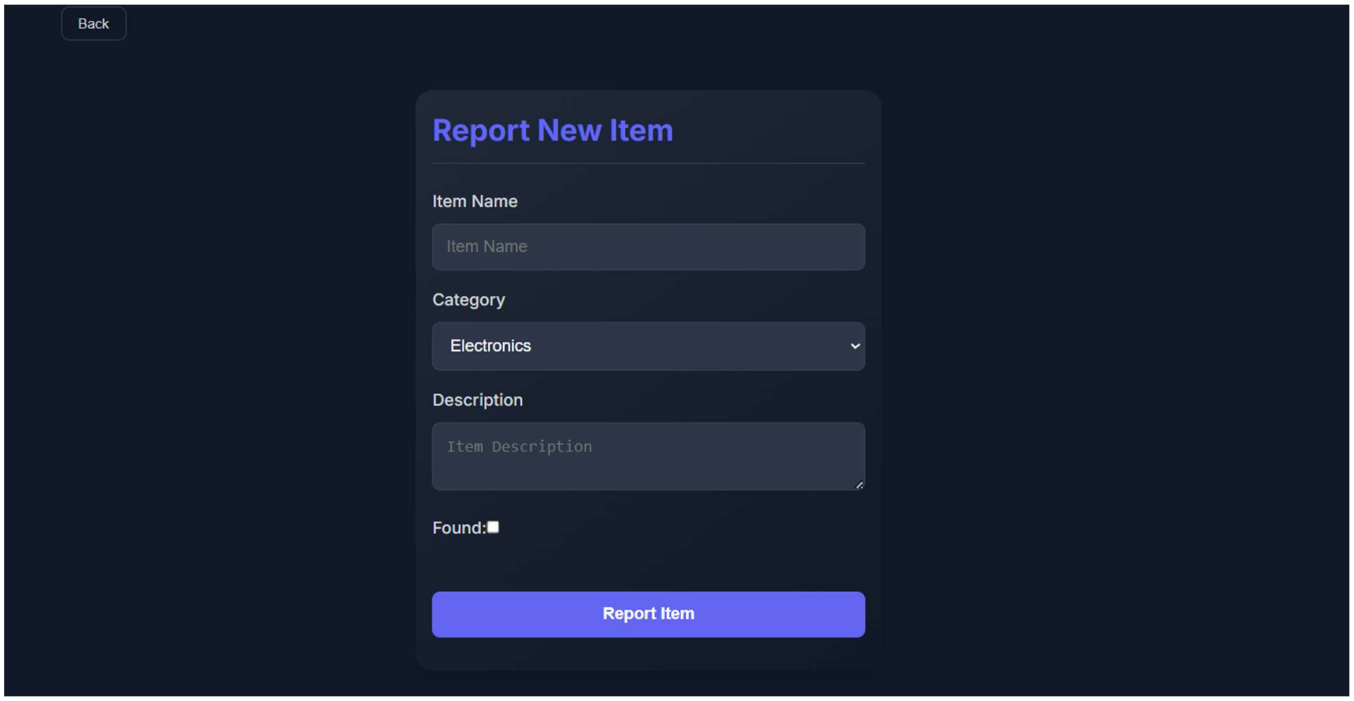
Category: Clothing

Status: Lost

Reporter Email: omer@gmail.com

iPhone

- **Reporting Item**



Back

Report New Item

Item Name

Item Name

Category

Electronics

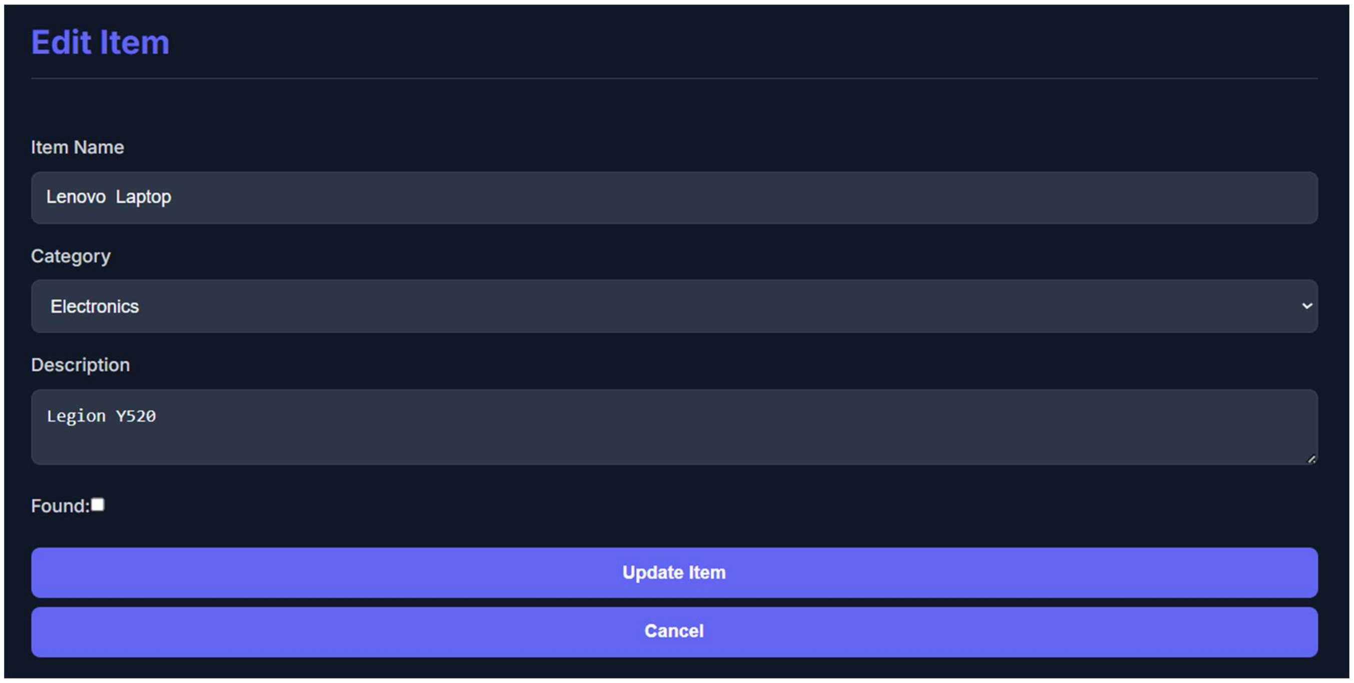
Description

Item Description

Found:

Report Item

- **Editing Item**



Edit Item

Item Name

Lenovo Laptop

Category

Electronics

Description

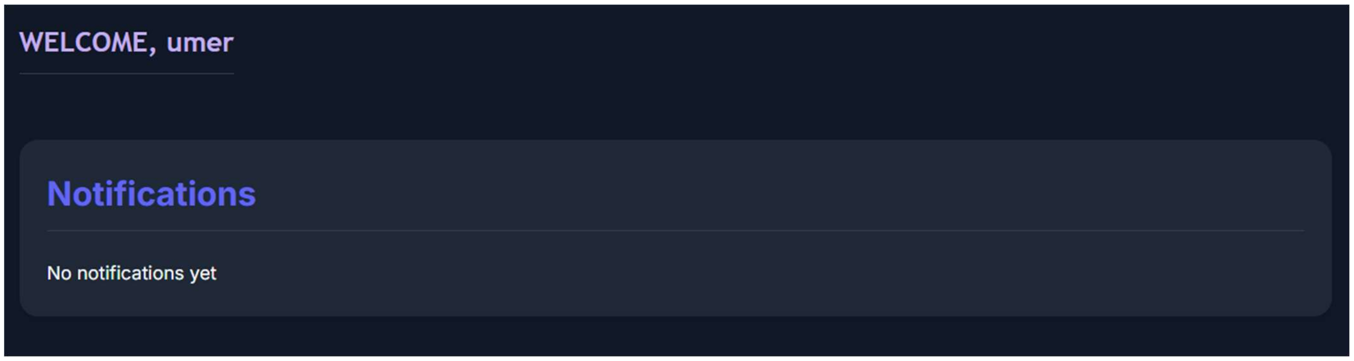
Legion Y520

Found:

Update Item

Cancel

- **Notifications**


WELCOME, umer

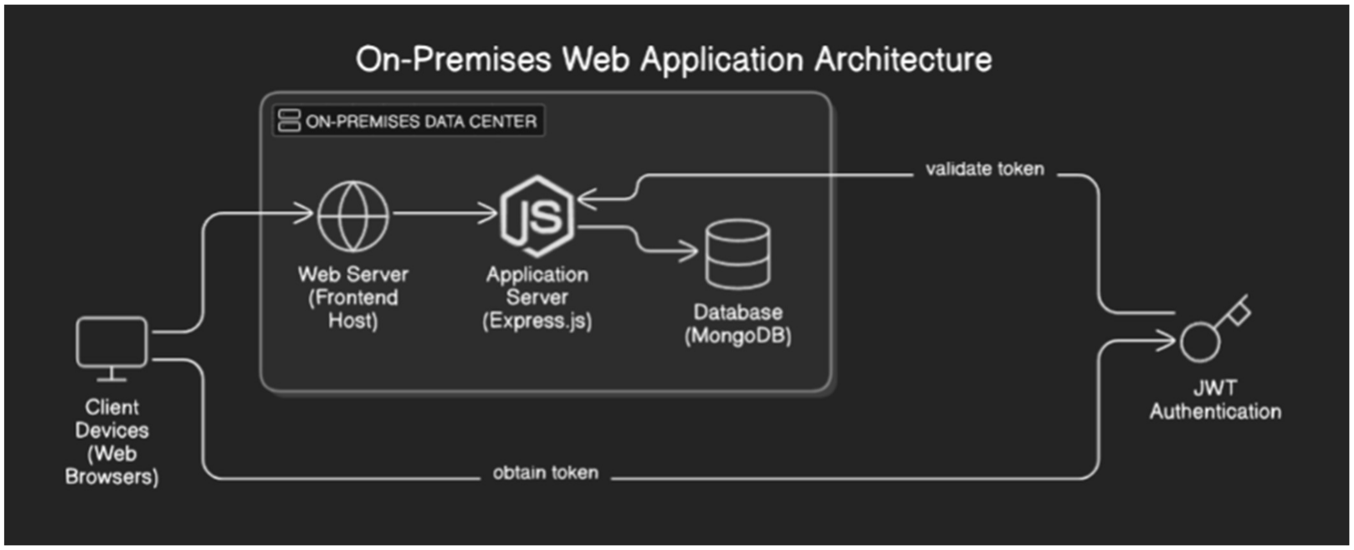
Notifications

No notifications yet

Project Burndown

Sprint	Total Tasks	Remaining Tasks
1	4	0
2	5	0

Architecture Diagram


On-Premises Web Application Architecture
ON-PREMISES DATA CENTER
Web Server (Frontend Host)
Application Server (Express.js)
Database (MongoDB)
validate token
JWT Authentication
Client Devices (Web Browsers)
obtain token

Test Cases

Black-Box Testing

Test Case 1: Valid User Login

- **Test Case ID**: TC-BB-001
- **Description**: Verify that a user can log in with valid credentials.
- **Preconditions**: User account exists in the system.
- **Test Steps**:
    1. Open the application and navigate to the login page.
    2. Enter a valid username and password.
    3. Click the "Login" button.
- **Expected Result**: User is redirected to the home page, and a welcome message is displayed.
- **Status**: Pass
- **Notes**: This test case verifies the basic login functionality with valid credentials.

Test Case 2: Invalid User Login

- **Test Case ID**: TC-BB-002
- **Description**: Verify that a user cannot log in with invalid credentials.
- **Preconditions**: No specific preconditions.
- **Test Steps**:
    1. Open the application and navigate to the login page.
    2. Enter invalid username and/or password.
    3. Click the "Login" button.
- **Expected Result**: An error message "Invalid credentials" is displayed.
- **Status**: Pass
- **Notes**: This test case verifies the system's response to invalid login attempts.

Test Case 3: Report a New Item

- **Test Case ID**: TC-BB-003
- **Description**: Verify that a user can report a new item.
- **Preconditions**: User is logged in.
- **Test Steps**:
    1. Navigate to the "Report Item" page.
    2. Fill in all required item details (name, description, category, etc.).
    3. Click the "Report Item" button.
- **Expected Result**: Item is reported successfully, and a confirmation message is displayed.
- **Status**: Pass
- **Notes**: This test case verifies the item reporting functionality.

Test Case 4: Search for Items

- **Test Case ID**: TC-BB-004
- **Description**: Verify that a user can search for items using keywords.
- **Preconditions**: Items exist in the system.
- **Test Steps**:
    1. Navigate to the search page.
    2. Enter a keyword in the search bar.
    3. Click the "Search" button.
- **Expected Result**: A list of items matching the keyword is displayed.
- **Status**: Pass
- **Notes**: This test case verifies the search functionality.

White-Box Testing

Test Case 1: User Authentication Middleware

- **Test Case ID**: TC-WB-001
- **Description**: Verify that the authentication middleware correctly validates tokens.
- **Preconditions**: Valid and invalid tokens are available.
- **Test Steps**:
    1. Call the authentication middleware with a valid token.
    2. Call the authentication middleware with an invalid token.
- **Expected Result**: Valid token is accepted, and invalid token is rejected with a 403 status.
- **Status**: Pass
- **Notes**: This test case verifies the authentication middleware's functionality.

Test Case 2: Item Reporting Functionality

- **Test Case ID**: TC-WB-002
- **Description**: Verify that a user can report a new item.
- **Preconditions**: User is logged in.
- **Test Steps**:
    1. Call the item reporting API endpoint with valid item data.
- **Expected Result**: Item is created in the database, and a 201 status is returned.
- **Status**: Pass
- **Notes**: This test case verifies the item reporting API endpoint.

Test Case 3: Item Claiming Functionality

- **Test Case ID**: TC-WB-003
- **Description**: Verify that a user can claim an item.
- **Preconditions**: User is logged in, and there is an item to claim.

- **Test Steps**:
    1. Call the item claiming API endpoint with a valid item ID.
- **Expected Result**: Claim is recorded in the database, and a 200 status is returned.
- **Status**: Fail
- **Notes**: This test case verifies the item claiming API endpoint.

Test Case 4: Notification Creation

- **Test Case ID**: TC-WB-004
- **Description**: Verify that a notification is created when an item is claimed.
- **Preconditions**: User is logged in, and there is an item to claim.
- **Test Steps**:
    1. Call the item claiming API endpoint with a valid item ID.
- **Expected Result**: A notification is created in the database.
- **Status**: Fail
- **Notes**: This test case verifies the notification creation functionality.

Work Division

Omer Mustafa

- **US-001**: Implement user registration and login functionality.
- **US-002**: Develop the feature to report lost or found items.
- **US-003**: Implement the search functionality using keywords.
- **US-004**: Develop the feature to filter items by category.
- **US-008**: Allow users to edit items they've reported.

Ibrahim Shahid

- **US-005**: Allow users to claim items reported by others.
- **US-006**: Notify users when someone claims their item.
- **US-009**: Enable users to delete items they've reported.
- **US-013**: Create a feature to view a list of claimed items.
- **US-014**: Improve user experience by enabling easy navigation between sections.

Talha Ahmad

- **US-007**: Notify users about the status of their claims.
- **US-010**: Enable users to view their notifications.
- **US-011**: Allow users to mark notifications as read.
- **US-012**: Create a feature to view a list of reported items.
- **US-015**: Implement the logout functionality.

Lessons Learnt

- **Code quality and testing**: The importance of writing clean, maintainable code was evident. Rigorous testing (both black-box and white-box) improved code quality and reduced the number of post-release issues.
- **Requirement elicitation**: Gathering comprehensive requirements upfront was invaluable. However, the team realized that requirements could evolve, necessitating flexible planning and documentation.
- **Time management**: Adhering to sprint timelines helped maintain project momentum. However, the team sometimes underestimated task complexities, leading to tight deadlines and the need for better time buffering in future planning.