LOST & FOUND MANAGEMENT SYSTEM REPORT

By Team 05
Sakibur Rahman (Team Leader)
Arun Kumar Sah (Secretary)
Peter Nagy (Developer)
Muhammad Rayan (Developer)
Prantik Paul (Tester)

Table of contents

Introduction

Project Description:

This Lost and Found Management System is a C# console application that allows users to report lost items, view and claim found items, and manage item statuses. Users can submit details of lost items, edit or delete them, and claim found items they believe belong to them.

Administrators have additional control to create, update, or delete found items, as well as manage user claims by approving or rejecting them. The system automatically updates the status of items (e.g., from "Lost" to "Claimed") based on claim approvals.

Entity Framework Core is used for data management, ensuring seamless handling of relationships between users, lost items, and claims. The application offers a simple interface for users to interact with the system and provides efficient tracking and management of lost and found items.

Report Layout:

The first part of the document is a basic project description that outlines what we created and what its functions are. The second part focuses on the design, detailing how we began building the project. This section includes the database schema, pseudocode, and the data structures we used to implement our developed idea.

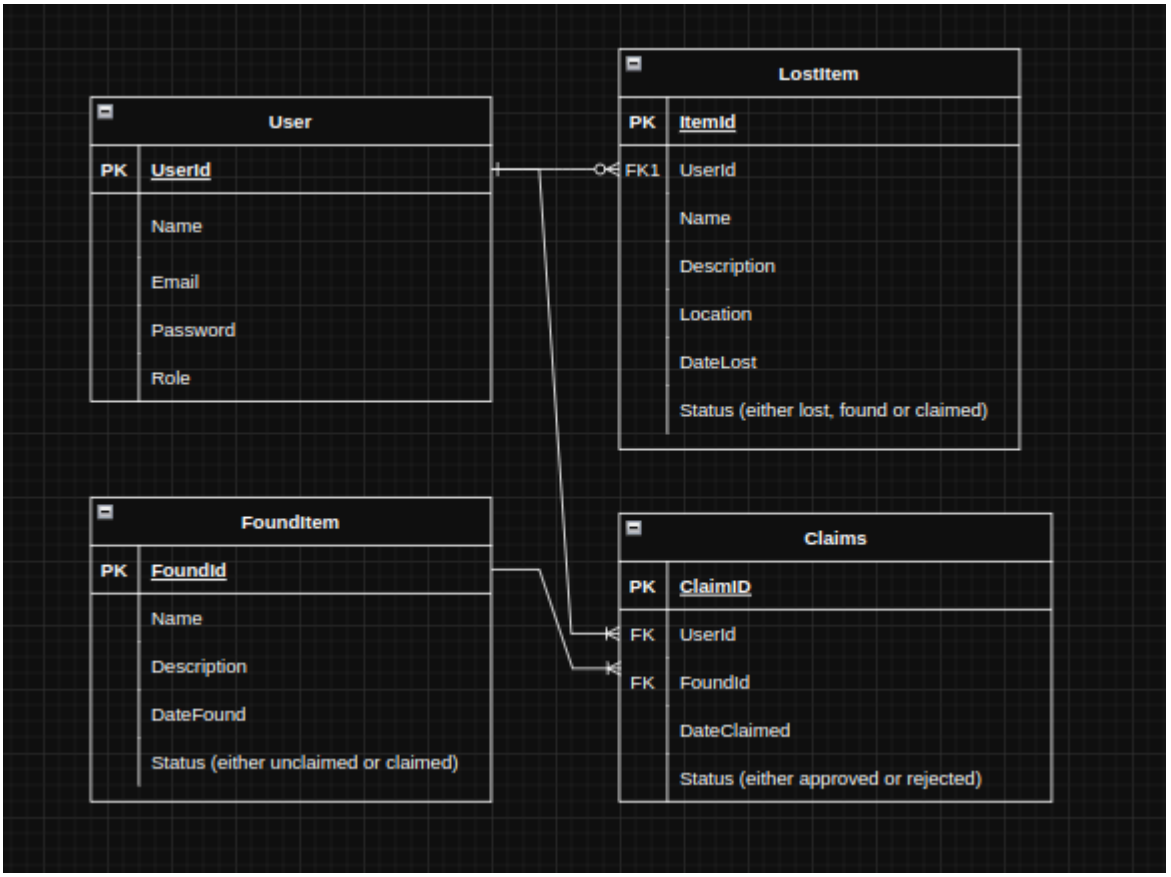
The third part covers testing. We conducted thorough manual testing of each method during development before integrating it into the main code.

The final two sections consist of the conclusion, which summarises our findings and discusses limitations, as well as a plan for future improvements as we continue to work on the project. Finally, I should mention that the very last part contains references included in the code comments as well.

Design

Data structures

For this project we decided with the following database schema:



Users

The users table is responsible for storing a user's contact information as well as their login info. This table is in a one-to-many relationship with the "lostItems" and "Claims" table.

Algorithm	Pseudocode	Time Complexity
Login algorithm	Email= input Password = input User = query users table from database WHERE Email and Password match	As there's no index on email the time complexity will be **O(n)** with n being the number of rows in the table.
Registration algorithm	Email = input Password = input Query = query database WHERE email match Query2 = insert new user into User table email and password	The email query's time complexity is **O(n)** with n being the number of rows in the database. The insert operation has a time complexity of **O(1)**

LostItems

The lost items table stores a lost items ticket of a user. It is in a many-to-one relationship with the "Users" table as one user may have many lost items tickets.

Algorithm	Pseudocode	Time Complexity
View My Lost Items	Query ALL from LostItems table WHERE userId == logged in user	As it has to query all from the database the complexity **O(n**) with **n** being the number of rows in the table
Update my Lost Item	Update from LostItems table WHERE ID matches input	**O(log n)** as we are querying by the indexed primary key

Delete my Lost Item	Delete from LostItems table WHERE ID matches input	**O(log n)** as we are querying by the indexed primary key
Add Lost Item	INSERT to LostItems table	**O(1)** as it is an insert operation
Manage lost Items	Query all lost items from database Ask admin if they want to update the status If so Ask for id of lost item Update WHERE lostitem ID is user input Else exit	**Querying: O(n)** with n being the number of rows in the table **Updating: O(log n)** as we are querying by an indexed primary key

Claim items:

The Claims table tracks user requests to claim found items. It has a many-to-one relationship with both the Users and FoundItems tables, allowing multiple claims per item and user.

Algorithm	Pseudocode	Time Complexity
Managing Claims (Approving/Rejecting)	Get all claims with user and item info. Display all claims. Prompt for claim ID. Validate and update claim status. If approved, reject others for the same item. Save changes.	**Fetching claims:** O(n), where n is the total number of claims. **Finding related claims for rejection:** O(n) (worst-case all claims belong to the same item). **Overall Time Complexity: O(n).**
Create Claim	if the item has already been claimed by a user. return "Duplicate claim" create new Claim(userId, foundItemId, "Pending") Save to the database	**O(1)** for lookup and insert. **O(n)** if duplicate check scans the user claims.

Found item:

The FoundItems table stores information about items that have been found. It has a one-to-many relationship with the Claims table, allowing multiple claims per found item.

Algorithm	Pseudocode	Time Complexity
Deleting a Found Item	Prompt the user for the item ID. If the input is invalid, return. Retrieve the item with claims. If the item is not found, return. Remove item (and related claims). Save changes.	**Lookup by ID + include claims:** O(1) + O(n), where n is number of related claims **Delete:** O(n) **Overall Time Complexity: O(n).**

Algorithm	Pseudocode	Time Complexity
View My found Items	SELECT * FROM FoundItems WHERE userId == loggedInUserI	As it has to query all from the database the complexity **O(n**) with **n** being the total found items.
Update my found item	UPDATE FoundItems SET ... WHERE id == inputId	**O(log n)** looked up by the indexed primary key
mange my found Item	DELETE FROM FoundItems WHERE id == inputId	**O(log n)** – Same as update; relies on index.
Create found Item	INSERT INTO FoundItems (...) VALUES (...)	**O(1)** as it is an insert operation

Testing

To test the database system, just follow this link:

- https://docs.google.com/document/d/18Rc26MJzbVMUJIk4zoVUiUlpgQukwWlpirYxUlkTY4E/edit?tab=t.0

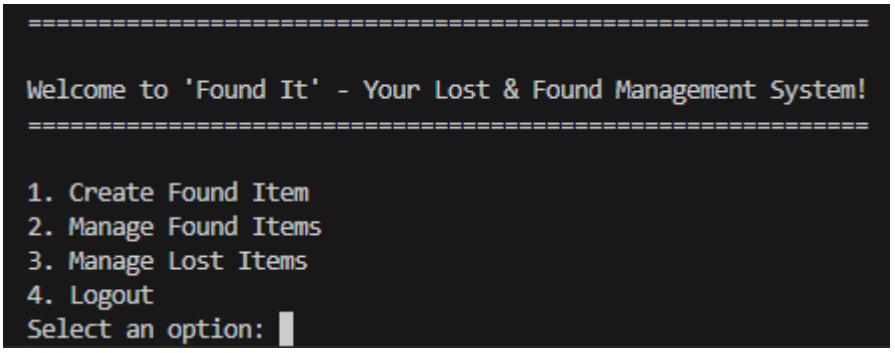
Changing what needs to be fixed:
for main menu:

-make it so the user can type "login, register or exit"

for register:

-username cant have any spaces
-make sure username and password have a max of 16 chars
-email can have max of 20 or 24 chars

ADMIN main menu:



- **Create Found Item works as intended**
- **Manage Found Items works as intended and displays correctly**
  - **Update works**
  - **Delete works**
  - **Manage claims works**
  - **Main menu works**

- **Manage lost item works as intended and displays correctly**

- **Logout works properly**

Conclusion:

The **Lost and Found Management System** is not just a software application—it represents a practical solution to a problem faced in everyday life. Misplaced belongings are a universal issue, whether it’s in a supermarket, school, office, or any public space. By designing a structured, role-based system, this project introduces order, clarity, and accountability into what is often a chaotic, manual process.

The choice of **C#** and Entity Framework Core allows for a scalable backend with clean database interactions, while the console interface ensures minimal overhead and ease of testing. Despite being a console-based app, the program is thoughtfully structured with a strong emphasis on modularity, readability, and user interaction**.**

One of the standout aspects of this system is its user-focused approach:

- Regular users can manage their own reports easily, with tools to edit, delete, and search through lost items.
- Administrators can maintain a centralized overview, enabling them to take fast action and support users more efficiently.

In a real-world context, this type of system can be further expanded into a web or mobile platform—demonstrating how the project lays a solid foundation for future growth. It is both a technical achievement and a practical blueprint for scalable item recovery systems.

Whether it's recovering a misplaced phone or managing dozens of daily reports, this system proves one thing: even the smallest lost item deserves a chance to be found.