Web Services Project

A Lost and Found Carthage

Written by

Farah Jaouadi

Professor	Academic year	University
Montassar Ben Messaoud	2024-2025	Tunis Business School

Contents

List of Figures

1 Introduction

1.1 Background

Tunis-Carthage Airport is one of the busiest airports in Tunisia, serving both domestic and international travelers. Despite its high traffic, the airport has not yet adopted a comprehensive digital solution for managing Lost and Found items. Currently, passengers who lose belongings are instructed to visit the litigation office near the baggage claim area to file reports. This manual process is inefficient and can be time-consuming, especially for travelers who may not be able to return to the airport immediately. There is an increasing demand for airports to provide digitized solutions to improve the efficiency and transparency of passenger services.

This chapter introduces the Lost and Found API as a potential solution to address the gap in Tunis-Carthage Airport's services and provides an overview of its benefits.

1.2 Motivation

The motivation for implementing the Lost and Found API stems from the inefficiencies and challenges that both passengers and airport personnel currently face when dealing with lost items. Many passengers express frustration at having to wait in long lines or make multiple trips to the airport to check on the status of their lost belongings. There is also a lack of transparency and real-time updates regarding the whereabouts of lost items.

The motivation for this project is to enhance the passenger experience, improve operational efficiency, and bring Tunis-Carthage Airport in line with international standards. A digital Lost and Found system would reduce physical paperwork, improve communication, and streamline the process of reporting, claiming, and searching for lost items.

1.3 Problem Statement

Currently, Tunis-Carthage Airport lacks a digitalized solution for managing lost items. Passengers who lose their belongings are forced to visit the litigation office in person, which is inconvenient, time-consuming, and often leads to uncertainty and dissatisfaction. The lack of a digital platform means there

is no real-time tracking of lost items or transparency regarding the status of items reported as lost.

This manual process contributes to long waiting times, potential loss of items, and frustration among travelers. As a result, there is a significant gap in the service quality at the airport, especially for those who expect efficient and convenient solutions when it comes to lost property.

1.4 Objectives

The primary objectives of this project are:

- Digitize the Lost and Found process at Tunis-Carthage Airport by integrating the Lost and Found API.
- Automate the process of reporting, searching, and claiming lost items to enhance passenger convenience.
- Improve the efficiency and transparency of the airport's Lost and Found services.
- Ensure that passengers have access to a secure, user-friendly platform to track their lost belongings.
- Create a scalable solution that can be expanded to other airports or services in the future.

1.5 Scope

The scope of this project includes:

- Integration of the Lost and Found API with the airport's existing systems, allowing passengers to report lost items, search for items, and claim them online.
- Development of a user-friendly interface (mobile app or web platform) for passengers to interact with the Lost and Found system.
- User authentication to ensure secure access to the platform and track lost items securely.
- Training airport staff to use the system and assist passengers when needed.

2 Database Integration with SQLite

In this project, SQLite was chosen as the database management system for storing and managing information related to lost and found items. SQLite is a lightweight, file-based database that is easy to integrate with Flask, making it ideal for small-scale applications like this one.

SQLAlchemy, a Python Object-Relational Mapping (ORM) tool, was used to simplify interactions with the database. This allowed the application to manage database records as Python objects, abstracting the complexity of writing raw SQL queries. The configuration for the SQLite database was set in a configuration class within the project, pointing to a local database file named `lost_and_found.db`.

The following key configuration settings were used:

- **Database URI:** The path to the SQLite database file, ensuring that the application can connect to and store data within the database.
- **SQLAlchemy Modifications:** This setting was disabled to optimize performance by preventing unnecessary tracking of modifications in the database.
- **Secret Key:** A random key used for securing user sessions and form submissions.

3 System Design and Architecture

3.1 UML Diagram

The UML diagram included in this report provides a visual representation of the system's architecture. It illustrates the relationships between different components of the application.

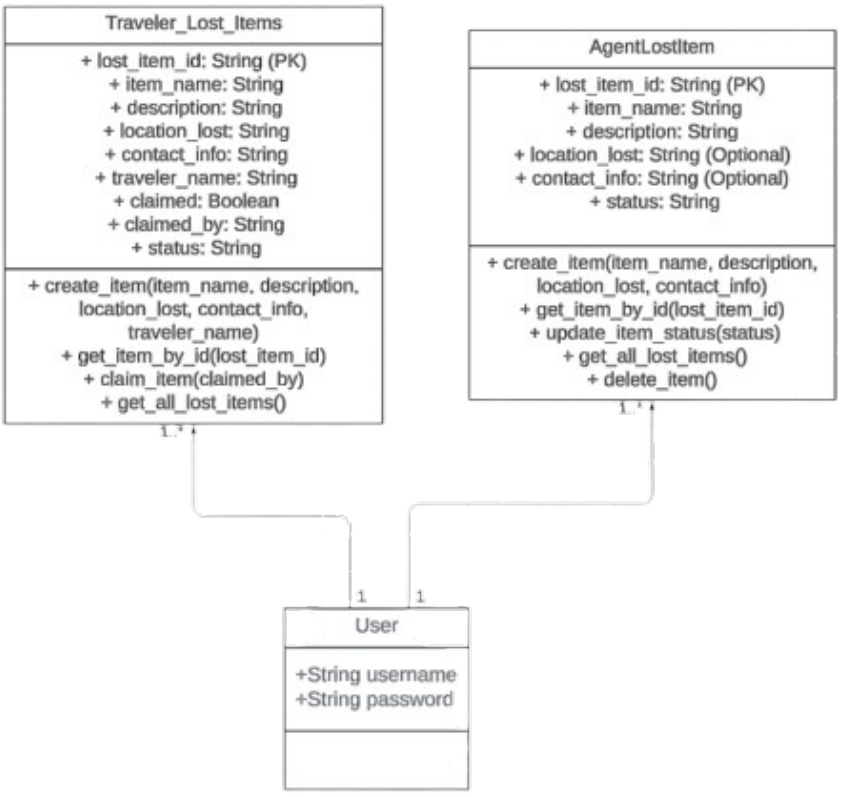


Figure 1: UML Diagram of the Lost and Found Application

This UML diagram provides an overview of how the application's structure is organized and how data flows between the different components, including interactions with the SQLite database.

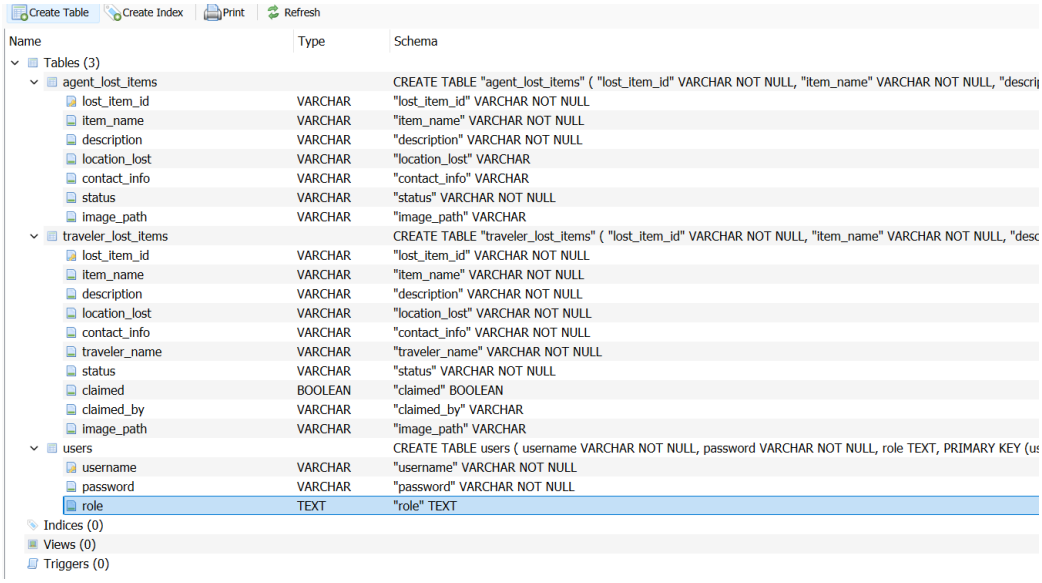
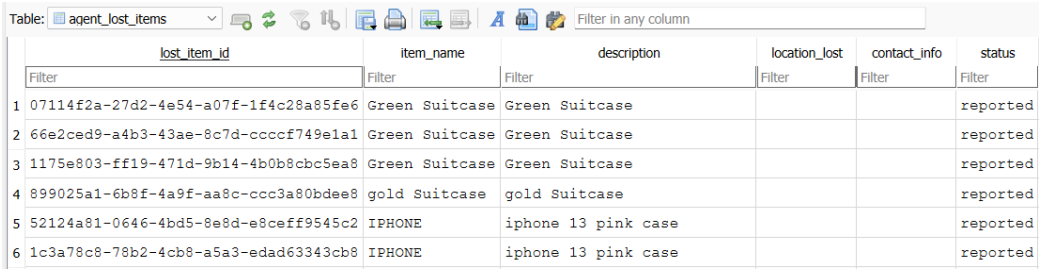


Figure 2: Database Schema in SQLite



	lost_item_id	item_name	description	location_lost	contact_info	status
	Filter	Filter	Filter	Filter	Filter	Filter
1	07114f2a-27d2-4e54-a07f-1f4c28a85fe6	Green Suitcase	Green Suitcase			reported
2	66e2ced9-a4b3-43ae-8c7d-ccccf749e1a1	Green Suitcase	Green Suitcase			reported
3	1175e803-ff19-471d-9b14-4b0b8cbc5ea8	Green Suitcase	Green Suitcase			reported
4	899025a1-6b8f-4a9f-aa8c-ccc3a80bdee8	gold Suitcase	gold Suitcase			reported
5	52124a81-0646-4bd5-8e8d-e8ceff9545c2	IPHONE	iphone 13 pink case			reported
6	1c3a78c8-78b2-4cb8-a5a3-edad63343cb8	IPHONE	iphone 13 pink case			reported

Figure 3: Sample data entries in the agentlostitems table

3.2 Use Case Diagram

The Use Case Diagram illustrates the interaction between the system and its users. This system supports two primary actors: Travelers and Agents, each with specific functionalities.

3.2.1 Actors

- **Traveler:**
  - Registers and logs into the system.
  - Reports lost items by providing detailed information about the lost object.
  - Searches the system for items that have been found or registered by agents.
  - Views item details for further clarification or to identify their lost belongings.
- **Agent:**
  - Registers and logs into the system to perform management functions.
  - Logs found items by adding detailed descriptions into the system.
  - Updates the status of items, marking them as "lost" or "found."
  - Deletes items from the database once resolved or deemed irrelevant.
  - Views and manages item details, ensuring accurate data maintenance.

3.2.2 Use Cases

The following are the key functionalities supported by the system:

- **Register:** Enables users to create accounts.
- **Login:** Authenticates users for secure access.
- **Report Lost Item:** Allows travelers to log reports for lost belongings.
- **Search Items:** Facilitates the search for found items in the system.
- **View Item Details:** Provides detailed information about logged items.
- **Change Item Status:** Lets agents update the status of items in the system.

- **Delete Item:** Grants agents the ability to remove irrelevant or resolved records.

This diagram simplifies understanding of the system's scope and interactions. It ensures both user roles and their respective functionalities are clearly defined, which is critical for system design and further development.

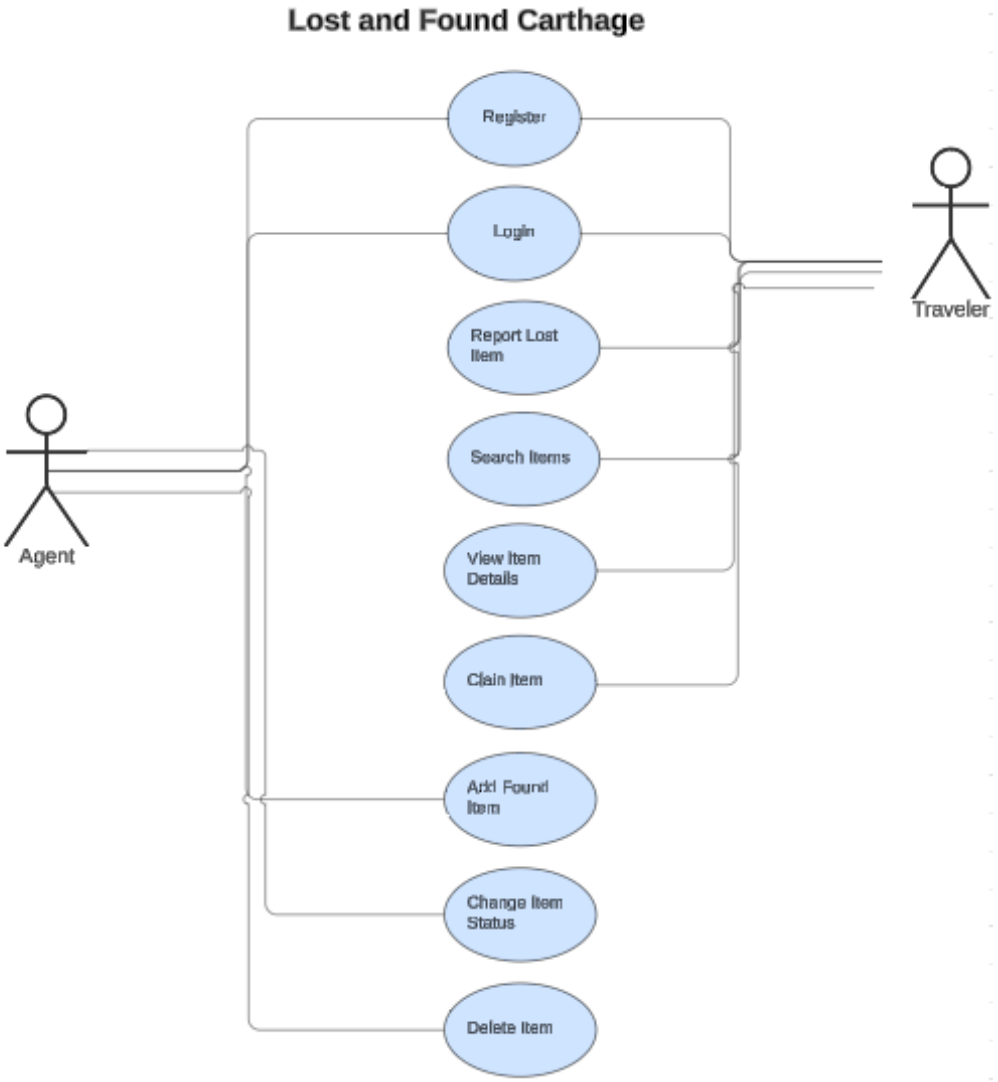


Figure 4: Use Case Diagram of the System

4 API Documentation

4.1 API Endpoints

For testing and interacting with the Lost and Found API, Insomnia, a popular tool for API development and testing, was used. It allowed for easy definition and execution of requests for the various endpoints of the API. Below are the default API endpoints implemented:

4.1.1 Default Endpoints

- **POST /lost-items/report** *Description:* Report a lost item by a traveler or agent.
- **GET /lost-items/{lost_item_id}** *Description:* Retrieve details of a lost item by its ID.
- **PUT /lost-items/{lost_item_id}** *Description:* Update a lost item's status (e.g., found, claimed).
- **DELETE /lost-items/{lost_item_id}** *Description:* Delete a lost item by its ID.
- **POST /lost-items/claim/{lost_item_id}** *Description:* Claim a lost item by the traveler who reported it.
- **GET /lost-items** *Description:* List all lost items reported by both travelers and agents.
- **POST /login** *Description:* User login to authenticate by providing a username and password.
- **POST /agent-login** *Description:* Agent login to authenticate by providing agent credentials.

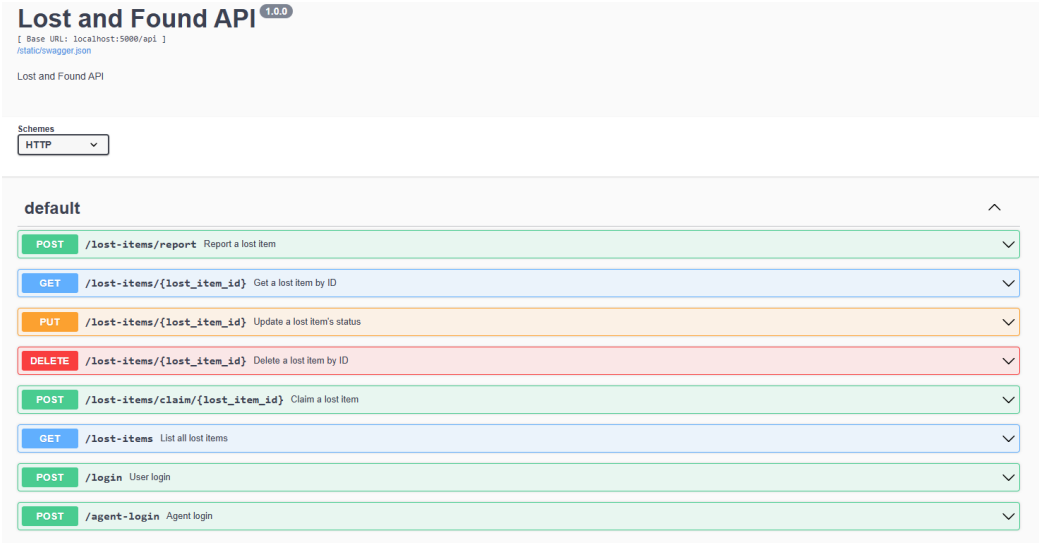


Figure 5: Swagger documentation for the Lost and Found API endpoints

4.2 JWT-Based Authentication Explanation

4.2.1 JWT Implementation

JSON Web Tokens (JWT) are used for securing authentication in web applications by ensuring stateless and secure communication between the user and the server. The following explains the implementation of JWT in this project:

- **Token Generation upon Login:** When a user logs in, the server authenticates the user's credentials (username and password) through the `/login` route. If the credentials are valid, the server generates a JWT that encodes the user's identity (e.g., the username) using `create_access_token`. Optionally, a refresh token is created using `create_refresh_token`.
- **Access Token:** The `access_token` is a short-lived token (valid for 1 hour, as defined in the `JWT_ACCESS_TOKEN_EXPIRES` setting) that contains the user's identity and other claims.
- **Refresh Token:** The `refresh_token` is longer-lived and can be used to obtain a new access token after the current one expires.

5 Frontend Development

5.1 Technologies Used

The frontend of the application was developed using a combination of HTML, CSS, and JavaScript. These technologies were chosen to create a dynamic, responsive user interface that ensures a smooth user experience across different devices.

5.2 Key Pages and Functionalities

5.2.1 Registration/Login Page

This page enables users to create accounts or log in to access the application's features. Key functionalities include:

- **Account Creation:** Users can register by filling out a form with necessary details.
- **Login:** Existing users can authenticate themselves using their credentials.
- **Validation Checks:** Input fields include validation to ensure correct and complete information.
- **User-Friendly Design:** Forms are designed for easy navigation and usability.

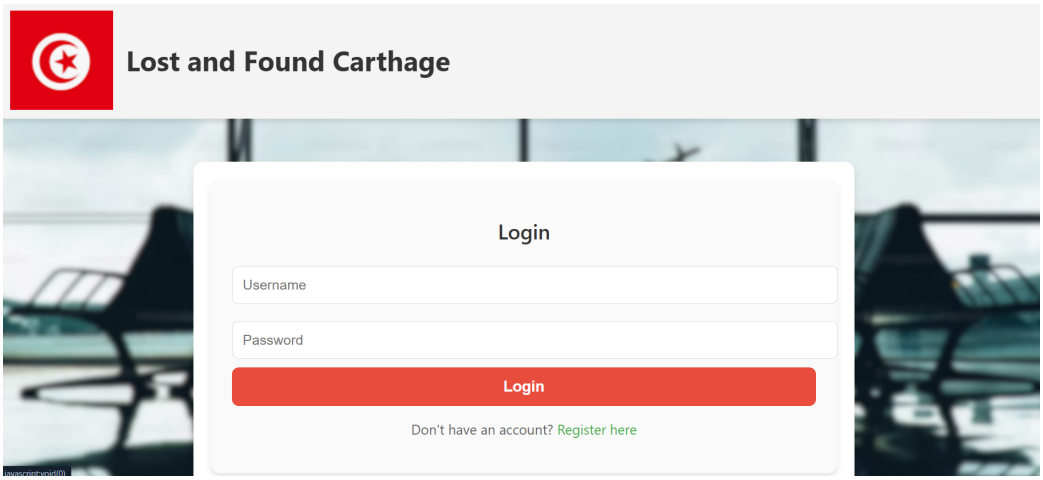


Figure 6: Login page

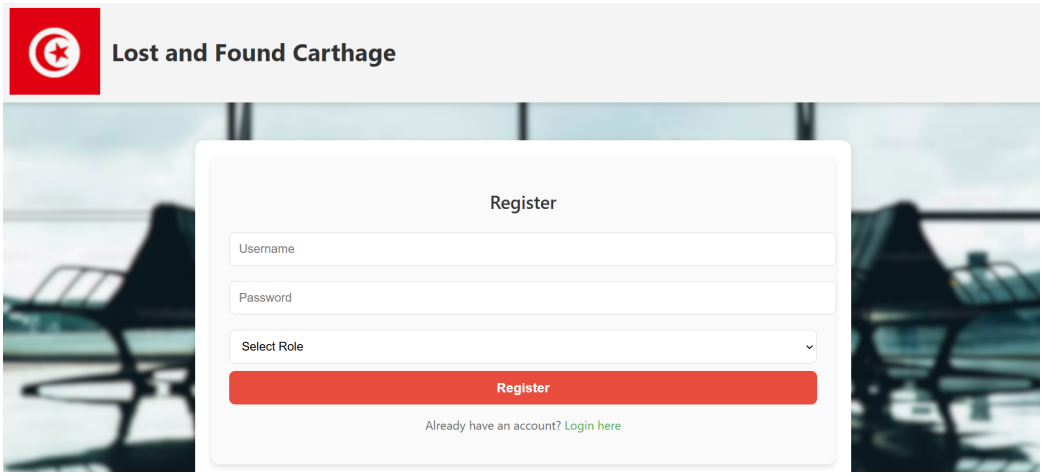


Figure 7: Register page

5.2.2 Dashboard for Travelers/Agents

The dashboard serves as a central hub for users, providing functionalities based on their roles:

- **For Travelers:**

- View their bookings.
- Check available trips and options.
- Access other travel-related features.

- **For Agents:**
  - Manage bookings for travelers.
  - Update itineraries to reflect changes.
  - Oversee transactions and ensure smooth operations.

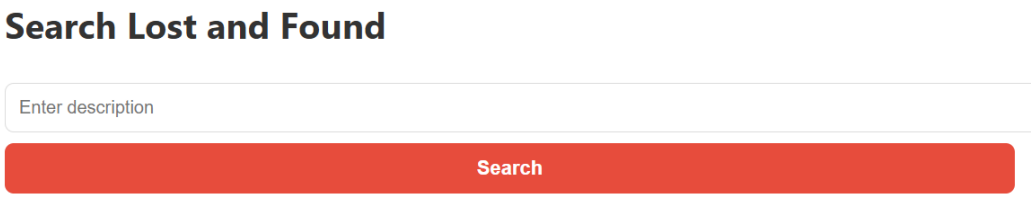


Figure 8: Search functionality for finding lost items

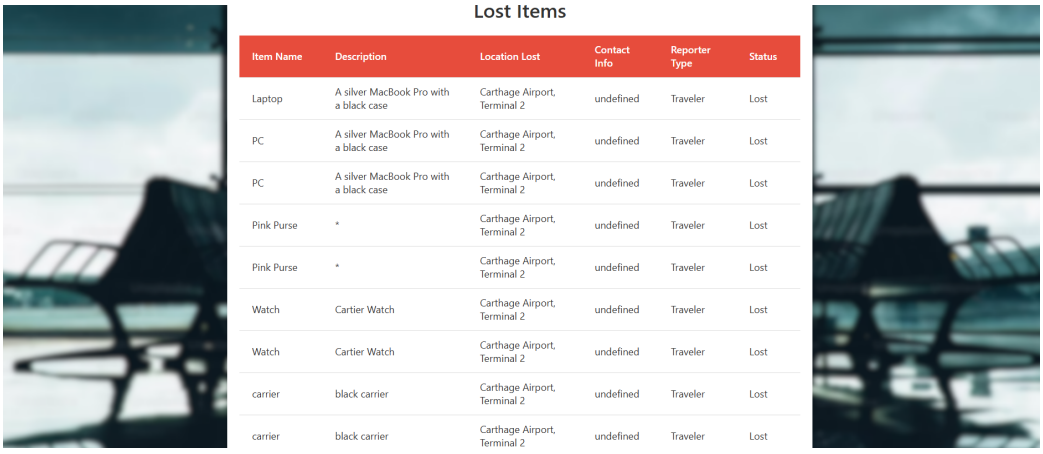


Figure 9: List of lost items functionality

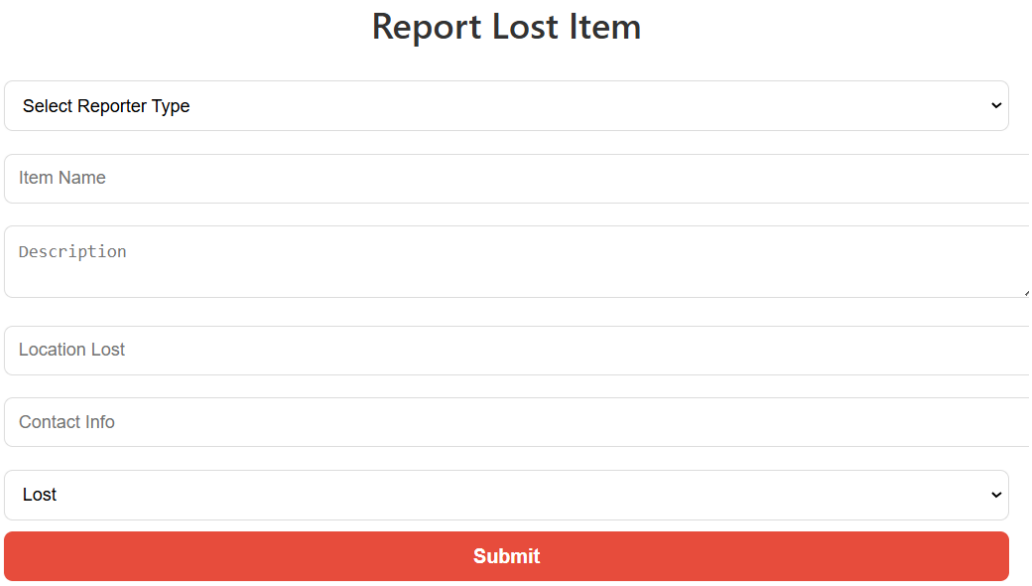


Figure 10: Report functionality for lost items

6 Overview of Docker Usage

Docker was utilized to containerize the application, providing a consistent environment across different platforms and simplifying the deployment process.

6.1 Dockerfile for Defining the Application's Environment

The Dockerfile contains the necessary instructions to build the application's container. It starts with the official Python base image and sets up the required environment for running the Flask app. This file ensures that the application's dependencies are installed and the app is configured to run in a containerized environment.

6.2 Benefits of Docker

6.2.1 Platform Independence

Docker allows the application to run consistently across various platforms and environments. From local development machines to production servers, Docker ensures that the application behaves the same way, minimizing platform-specific issues.

6.2.2 Scalability

With Docker, the application can be easily scaled by running multiple instances of containers. This ensures that the application can handle increased traffic or workloads efficiently without compromising performance.

6.2.3 Simplified Deployment

Docker simplifies the deployment process by guaranteeing that the application runs in the same environment, regardless of the infrastructure. This reduces the risk of deployment errors and configuration discrepancies, leading to smoother, more reliable deployments.

7 Future Enhancements

The following enhancements are planned for future versions of the application:

- **Enhance the Frontend:** The user interface will be improved to make it more visually appealing and user-friendly. Additional features like advanced filtering and better navigation will be incorporated for a smoother user experience.
- **Add Image Uploads:** The system will allow users to upload images of their lost items, making it easier to identify and track lost belongings. This feature will improve the accuracy of item reports and assist agents in managing lost items more efficiently.
- **Integrate an External API for Flight Arrivals:** To further enhance the functionality of the system, an external API will be integrated to allow travelers to add their flight details. This will help

streamline the reporting process, as passengers can automatically link their lost items to specific flights.

8 Conclusion

This report outlines the design and development of a Lost and Found API for Tunis-Carthage Airport, aimed at addressing the inefficiencies in the current manual process for managing lost items. The existing system is time-consuming and lacks transparency, leading to passenger frustration. By digitizing the process, this project streamlines reporting, searching, and claiming lost items, improving operational efficiency and enhancing the passenger experience.

The integration of SQLite with Flask and SQLAlchemy enables efficient storage and management of lost item data. The system includes a user-friendly interface and secure authentication, allowing passengers to easily track their belongings. Airport staff will be trained to manage the system, ensuring smooth operation.

This solution not only meets the needs of Tunis-Carthage Airport but also has the potential for scalability to other airports, improving overall efficiency and customer satisfaction. Through modern technology and an intuitive design, the Lost and Found API project is a significant step toward modernizing the airport's services and offering a more efficient, transparent approach to handling lost property.