AI-BASED LOST & FOUND SYSTEM

REPORT OF PROJECT

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENT FOR

THE AWARD OF THE DEGREE OF

BACHELOR OF TECHNOLOGY

IN

COMPUTER SCIENCE & ARTIFICIAL INTELLIGENCE

Batch

(2022-2026)

SUBMITTED BY

Anushka Rathour

12200884

SUPERVISED BY

Dr. Ridhi Kapoor

DEPARTMENT OFCOMPUTER SCIENCE AND ENGINEERING

DAVUNIVERSITY JALANDHAR-PUNJAB144012

JALANDHAR -144001, PUNJAB

ACKNOWLEDGMENT

I would like to express my sincere gratitude to my project guide, **Dr. Ridhi Kapoor,** for their valuable guidance, constructive feedback, and continuous support throughout the duration of this project. I am also thankful to **Dr. Rahul Hans (Coordinator, CSE)** and the Department of Computer Science Engineering for providing the necessary resources and an encouraging academic environment.

My appreciation extends to all faculty members and classmates whose suggestions and cooperation helped strengthen the quality of this work. Lastly, I am deeply grateful to my family for their constant encouragement and support, which motivated me to complete this project successfully.

DECLARATION

I, **Anushka Rathour**, hereby declare that the work which is being presented in this project titled "**AI-BASED LOST & FOUND SYSTEM"** by me, in partial fulfilment of the requirements for the award of Bachelor of Technology (B. Tech) Degree in "Computer Science and Artificial Intelligence" is an authentic record of my own work carried out under the guidance of Dr. **Ridhi Kapoor.**
To the best of my knowledge, the matter embodied in this report has not been submitted to any other University/ Institute for the award of any degree or diploma.

ANUSHKA RATHOUR
12200884

ABSTRACT

The **AI-Powered Campus Lost and Found System** is a smart, web-based platform designed exclusively for university environments to streamline the process of reporting, managing, and locating lost and found items. Traditional manual methods—such as notice boards, WhatsApp groups, or physical registers—are often inefficient, unorganized, and time-consuming. To address this problem, the proposed system integrates **Artificial Intelligence using CLIP ViT-L/14 (Contrastive Language–Image Pretrained model)** for transfer learning–based image similarity matching, eliminating the need for custom model training. When a user uploads an image of a lost or found item, the system generates a high-dimensional embedding and compares it against stored embeddings to retrieve the most visually similar items, significantly improving search accuracy and speed.

The system is implemented using a **Fast API backend**, **SQLite database**, and a simple yet responsive **HTML, CSS, and JavaScript frontend**. Only authenticated university users—verified through a **Student ID and passcode**—can access the platform, ensuring controlled and secure usage. Additionally, an **admin dashboard** allows administrators to manage reports, update item statuses, delete inappropriate entries, and maintain database integrity.

This project demonstrates a practical and impactful application of AI within a campus setting, enhancing operational efficiency, security, and user convenience. By leveraging transfer learning instead of building a custom machine learning model, the system achieves high performance with minimal compute requirements, making it suitable for real-time deployment in resource-limited environments. The result is a reliable, intelligent, and user-friendly solution that significantly improves the lost and found management process in educational institutions.

TABLE OF CONTENT

1. INTRODUCTION

The rapid growth in university campus populations has increased the number of incidents involving misplaced belongings, such as ID cards, keys, documents, electronic devices, and personal accessories. Traditional notice-board-based lost-and-found systems are inefficient, slow, and heavily dependent on manual reporting. This creates difficulties for students and faculty members who are trying to locate their missing items in a timely manner.

To address this recurring challenge, the **AI-Powered Lost and Found System for Campus** has been developed. The system leverages modern artificial intelligence techniques—specifically **CLIP ViT-L/14 image-text embedding model through transfer learning**—to perform accurate similarity-based matching between uploaded images and existing lost/found records. The platform enables users to register lost items, report found belongings, and perform AI-based searches, resulting in significantly improved accuracy compared to traditional keyword search.

The system is designed exclusively for **university-registered users**, ensuring a secure and private environment where only authenticated students and campus staff can post, search, or manage records. With backend processing powered by **Fast API**, frontend interfaces built using **HTML, CSS, and JavaScript**, and persistent storage handled through **SQLite**, the system provides a lightweight, reliable, and easy-to-deploy solution tailored for campus requirements.

In addition to student access, the project includes a dedicated **admin dashboard**, allowing administrators to review, update, and delete suspicious or duplicate entries to maintain data integrity. This combination of AI-enabled similarity search, user authentication, and structured record management makes the system a modern, fast, and scalable approach to solving lost-and-found challenges on university campuses.

1.1 Overview of the Project

The **AI Lost and Found System for Campus** is a web-based application enhanced with an AI similarity-matching feature to help users quickly locate missing belongings. The system allows students and staff to upload details and images of lost or found items, after which the application generates **vector embeddings using CLIP ViT-L/14** and compares them with existing records to return the most relevant matches.

The platform is divided into three core modules:

1. User Module

- Students authenticate using their **Student ID and Passcode**.
- Users can create new lost/found item reports.
- Users can search for items using keywords or images.
- Users can view matches ranked by similarity score.

2. Admin Module

- Admin logs in using predefined credentials.
- Can view all reports submitted by students.
- Has authority to update, delete, or verify entries to maintain data consistency.
- Can mark items as "Found", "Returned", or "Resolved".

3. AI Similarity Search Module

- Uses **Sentence Transformers (CLIP ViT-L/14)** to generate embeddings for images and descriptions.
- Performs cosine similarity matching to return best possible results.
- Enables faster and more accurate searches than traditional text-only matching.

Overall, this system acts as an **intelligent bridge between lost item reporting and retrieval**, reducing time, effort, and manual dependency by integrating AI-driven matching within a secure campus-specific environment.

1.2 Problem Statement

Lost belongings are a frequent and inevitable issue within large university campuses, where thousands of students, faculty members, and staff move across classrooms, libraries, hostels, cafeterias, and other facilities every day. Traditional lost-and-found processes—such as notice boards, security offices, or verbal communication—are inefficient, time-consuming, and often unreliable. Students generally rely on chance encounters or manual inquiries to recover their lost items, which leads to:

- **Delayed retrieval** of important belongings such as ID cards, wallets, books, or gadgets.
- **Lack of centralized data**, making it difficult to track items accurately.
- **Limited visibility**, as only a small group of people may see physical notices.
- **Inability to verify authenticity**, leading to duplicate or false entries.
- **No intelligent search mechanism**, which means users must manually browse through lists.

Additionally, online lost-and-found systems without authentication can be misused by outsiders, leading to privacy concerns and data spamming.

Thus, there is a need for a **secure, centralized, and AI-enhanced platform** that can match images of lost items with reported found items more efficiently. The absence of such a system on campus forms the core problem this project aims to solve.

1.3 Scope of the Project

The **AI Lost and Found System for Campus** is specifically designed to function within the boundaries of a single university campus. The scope includes all features necessary for easy reporting and retrieval of lost items using AI-based similarity search.

In-Scope Features

- A complete **web-based platform** accessible to students and staff of the campus.
- **Secure login system** using Student ID and Passcode to restrict access to university-registered users only.
- Ability for users to **report lost or found items** by providing title, description, category, location, phone number, and an image.
- **AI-powered search** using CLIP ViT-L/14 to match uploaded images with existing database records.
- **Cosine similarity-based ranking** of matched results.
- Ability to view, edit, or mark items as resolved.
- Dedicated **Admin Dashboard** to monitor, approve, update, or delete entries.
- Database management using **SQLite**, including storage of item records, embeddings, and user credentials.
- Simple and responsive UI created using **HTML, CSS, and JavaScript**.

1.4 Motivation

The motivation behind developing the AI-based Lost and Found System arises from the recurring challenges faced within university campuses regarding misplaced belongings. Students frequently lose essential items such as ID cards, spectacles, calculators, or electronic devices, and existing manual processes are often slow and ineffective. Traditional notice boards and physical registers lack real-time accessibility and do not support efficient searching.

With the increasing size of student populations and the need for faster digital solutions,

there is a clear requirement for a centralized platform that simplifies the reporting and recovery process. Integrating artificial intelligence for similarity matching provides an added advantage by enabling users to locate items more accurately based on visual characteristics. The motivation is therefore to create a reliable, secure, and user-friendly system that enhances campus convenience and reduces the time and effort involved in retrieving lost belongings.

1.5 Technology Used

The project is implemented using a combination of frontend, backend, artificial intelligence, and database technologies:

Frontend Technologies

- **HTML5** – For structuring the user interface.
- **CSS3** – For styling and layout design.
- **JavaScript** – For interactive client-side functionality and form operations.

Backend Technologies

- **Python FastAPI** – For building the backend API, handling routing, authentication, and communication between frontend and database.

AI and Embedding Model

- **Sentence Transformers (CLIP ViT-L/14)** – Utilized for generating vector embeddings from images to perform similarity-based searches for lost and found items.

Database

- **SQLite** – Used for storing user credentials, item details, embeddings, and image paths in a lightweight and efficient manner.

Additional Tools/Utilities

- **Python Libraries** such as NumPy, PIL, and sqlite3 for embedding processing, image handling, and database connectivity.

These technologies collectively enable the system to provide a secure, scalable, and AI-enhanced platform tailored for campus use.

1.6 System Users

The system is designed for use within a single university campus and includes the following user categories:

1.6.1 Students

Students can log in using their registered Student ID and passcode. They can report lost or found items, upload images, search for matches using the AI similarity engine, and track the status of their submissions.

1.6.2 Administrative Staff

Administrators have access to an admin dashboard where they can view, update, approve, or delete any item record. They are responsible for monitoring system activity and ensuring that entries follow campus policies.

2. TITLE OF THE PROJECT

2.1 Project Title

The title of the project, "AI-Based Lost and Found Management System for University Campus," has been purposefully selected to represent the core technological and functional essence of the system being developed. A project title in a technical report serves not merely as a label but as a concise summary of the scope, intent, and methodology of the work. In this context, the title encapsulates three key elements—Artificial Intelligence, Lost and Found Management, and University Campus Context—each of which plays a significant role in defining the problem domain and the proposed solution.

The term "AI-Based" highlights the use of contemporary artificial intelligence techniques, specifically text and image embedding models, machine learning-driven similarity search, and automated information retrieval. By incorporating AI, the system is capable of performing tasks that traditional manual lost-and-found processes struggle with—such as intelligently identifying similarities between reported lost items and newly added found items. This AI-driven architecture adds efficiency, reduces human error, and significantly increases the probability of item recovery.

The phrase "Lost and Found Management System" reflects the operational purpose of the project. Unlike conventional lost and found procedures, which rely on handwritten logs, physical inventories, or scattered communication channels, this project introduces a centralized digital platform. The system enables users to report lost items, upload images, provide descriptions, and monitor status updates. Similarly, students and campus staff who find items can submit them through the interface. The combined data is processed through the backend AI model to generate possible matches. Thus, the "management system" aspect

of the title indicates the structured, end-to-end workflow: reporting, storing, processing, and retrieving.

The final component, “for University Campus,” clarifies the operational context. University campuses often span large areas and serve thousands of students, faculty members, and staff. High-traffic academic zones such as libraries, computer labs, cafeterias, parking areas, and auditoriums frequently experience cases of misplaced belongings. These include ID cards, mobile phones, books, accessories, and personal items, many of which have significant academic or personal value. A university campus is, therefore, an ideal setting for implementing an organized, technology-assisted solution. Limiting the scope to a specific environment also ensures that the system can be optimized in terms of usability, scale, workflow, and security based on institutional protocols.

Overall, the project title provides a precise yet comprehensive representation of the technological foundation, operational process, and environmental focus of the system. It aligns with the project's objectives, problem statement, methodology, and anticipated outcomes, making it suitable for academic and institutional documentation.

2.2 Rationale Behind the Title

The rationale behind selecting the title “AI-Based Lost and Found Management System for University Campus” originates from the need to accurately convey the system’s motivation, relevance, and technological innovation. A well-defined title should provide insight into *why the project exists*, *what it aims to accomplish*, and *how it intends to achieve its goals*. The selected title fulfils these roles by articulating a clear relationship between contemporary technological advancements and a real-world institutional problem.

Firstly, the inclusion of “AI-Based” in the title reflects the central role of artificial intelligence in the functioning of the system. Traditional lost and found systems depend heavily on manual data review, subjective judgment, and delayed communications. These processes are not only time-consuming but also unreliable in large environments where hundreds of items may be misplaced each semester. The adoption of AI, specifically sentence-transformer embeddings, allows the system to interpret text descriptions, extract semantic meaning from sentences, and compare them with vectors generated for newly added items. This AI-driven similarity detection considerably improves the accuracy and speed of identifying potential matches. It also demonstrates the integration of modern computational techniques with real-world administrative applications.

Secondly, the term “Lost and Found Management System” was chosen to emphasize that the project is not merely a tool or a database but a comprehensive management solution. The system supports all stages of the lost-and-found lifecycle—including item reporting, categorization, storage of associated metadata, AI-based matching, user notifications, and status tracking. The word “management” indicates that the system provides structured oversight, standardized procedures, and organized handling of information, which differentiates it from informal or uncoordinated lost-and-found practices. Additionally, the system ensures that once items are recorded, they remain easily accessible and traceable until they are returned to their owners.

Thirdly, specifying “for University Campus” provides a contextually appropriate scope for the project. University campuses generate a high frequency of lost item incidents due to densely populated academic spaces, student mobility, and frequent shifting of personal belongings. Unlike public spaces such as shopping malls or transportation hubs, a campus setting offers a controlled environment where a dedicated digital system can be implemented institutionally. The users—primarily students—are also tech-friendly and accustomed to mobile and web-based systems, ensuring usability and adoption. Furthermore, campus administration can benefit from centralized monitoring and reduced manual workload, making the system mutually beneficial for both students and staff.

Thus, the rationale behind the project title is grounded in its ability to concisely represent the technical approach, purpose, and operational scope in a manner suitable for academic documentation. The title is not only descriptive but also indicative of the project’s innovation and applicability, ensuring clarity for evaluators, administrators, and potential users.

3. Objectives

The objectives of the project outline the intended outcomes, the scope of development, and the structured direction taken during the creation of the AI-Based Lost and Found Management System for University Campus. Clearly defining these objectives is essential to ensure that the project remains aligned with the identified problem, technological feasibility, and institutional context. The objectives have been divided into primary and secondary categories for structured representation, followed by the problem-solving approach adopted to accomplish them.

3.1 Primary Objectives

The primary objectives refer to the core targets that the project aims to achieve through its design, development, and implementation. These objectives directly address the main problem identified on the university campus: the inefficient and inconsistent management of lost and found items. The primary objectives of the system are:

1. To develop a centralized digital platform for reporting and retrieving lost and found items.

This objective focuses on replacing scattered, manual, and outdated reporting mechanisms with a unified system that stores all relevant information in a structured database. A centralized platform ensures accessibility, reliability, and accountability in handling lost and found cases.

2. To integrate AI-based similarity search for identifying potential matches.

The use of CLIP ViT-L/14 (via Sentence Transformer) enables the system to map textual descriptions and images into a shared vector space. The primary goal is to enable accurate and automated identification of similarities between newly reported lost items and existing found items, thus improving recovery rates.

3. To implement secure user authentication for university students.

Since the system is campus-restricted, only registered university students must have access. This objective ensures security and prevents unauthorized submissions or misuse. Authentication is carried out using student ID and passcode, stored securely in the database.

4. To offer an admin dashboard for monitoring and controlling system operations.

Administrators must have the ability to view all reports, update the status of items, and manage entries. This objective ensures institutional oversight and smooth functioning of the platform.

5. To ensure a reliable backend architecture for seamless communication between frontend, AI model, and database.

Using FastAPI, the system aims to deliver fast API responses, efficient routing, secure endpoints, and proper integration of AI inference modules and SQLite database operations.

These primary objectives collectively ensure that the system fulfils its core purpose: improving campus lost-and-found operations through digitalization and AI-driven intelligence.

3.2 Secondary Objectives

Secondary objectives complement the primary objectives and enhance the usability, performance, and long-term viability of the system. Although not essential to the minimum functional requirements, these goals significantly improve user experience and administrative efficiency.

1. To design a user-friendly and visually consistent frontend interface.

The system aims to provide intuitive navigation, clear layout, and responsive design using HTML, CSS, and JavaScript, ensuring accessibility for all users.

2. To create a structured categorization of lost and found items.

Categorizing items such as electronics, documents, accessories, and clothing ensures quicker searches and helps the AI model refine similarity suggestions.

3. To ensure data persistence, safety, and structured storage using SQLite.

Implementing proper schema design, foreign key constraints, and item metadata storage supports consistent and long-term data management.

4. To enable real-time status tracking of lost items.

Users should be able to view the progress of their query—from submission to match detection and final return—thus increasing transparency and user trust.

5. To maintain logs and records for administrative review and reference.

This includes timestamps, item updates, user submissions, and admin actions, offering traceability for institutional audits.

6. To optimize search accuracy by storing AI embeddings in the database.

Embedding storage ensures faster similarity matching without requiring repeated computation.

7. To support future scalability and feature expansion.

Although the system is campus-specific, the architecture is designed to accommodate additional features such as notifications, QR-code tagging, or multi-campus integration.

These secondary objectives help refine the system beyond its core functionality and prepare it for long-term institutional use.

3.3 Problem-Solving Approach

The problem-solving approach describes the methodology adopted to conceptualize, design, and implement the AI-based lost and found system. It outlines the systematic workflow followed to ensure that the final product meets the objectives effectively.

1. Identification of the Problem Context

The first step involved observing existing campus challenges related to misplaced items. Students frequently lose ID cards, mobile phones, books, and accessories, and retrieval depends on manual notices, word-of-mouth communication, or limited physical registers. The need for an intelligent, automated system was clearly established.

2. Requirement Analysis and Scope Definition

A detailed analysis was conducted to understand user needs, administrative requirements, AI suitability, and technological constraints. Functional and non-functional requirements were documented to define the scope.

3. Selection of Technologies and Tools

Based on performance and compatibility:

- FastAPI was chosen for the backend due to its speed and ease of deploying AI models.
- Sentence Transformer (CLIP ViT-L/14) was selected for text–image embedding.
- SQLite was preferred for lightweight, local data storage.
- HTML, CSS, JavaScript were used to build a simple, responsive frontend.

4. System Architecture and Database Design

A modular architecture was designed, connecting:

- The frontend interface
- FastAPI backend routes
- Embedding generation model
- SQLite database

  Database tables for users and items were structured to store metadata, embeddings, and status attributes.

5. Implementation of AI Similarity Search

The CLIP model generates embeddings for:

- Uploaded item images
- User-provided textual descriptions

  These embeddings are compared through cosine similarity to determine the closest matches.

6. Development of Secure User Authentication

A login system was implemented where only university-registered users with valid credentials can access lost-and-found features.

7. Admin Dashboard and Monitoring Tools

Custom admin pages were developed to manage the system, update statuses, and verify submissions.

8. Testing, Evaluation, and Optimization

The system was tested for:

- Accuracy of AI similarity matching
- Database reliability
- API response time
- User interface functionality
- Bug fixes and optimizations were performed to ensure smooth operation.

4. Steps to Achieve Objectives

This chapter outlines the structured steps taken to accomplish the objectives defined in the earlier section. The development of the **AI-Based Lost and Found Management System for University Campus** follows a systematic and iterative approach to ensure accuracy, reliability, and alignment with user requirements. The steps include system design planning, workflow development, architectural structuring, and phased implementation.

4.1 System Design Approach

The system design approach defines the foundational planning undertaken before implementation. This involves conceptualizing the system's components, identifying functional relationships, selecting appropriate technologies, and designing interfaces for users and administrators.

The design approach used in this project is based on the following components:

1. Requirement-Oriented Design

The system design originated from the functional requirements gathered through observation of current lost-and-found processes on campus. Requirements were translated into modules such as:

- User authentication
- Item submission
- AI-based similarity search
- Administrative control panel
- Database management

Each module was evaluated individually to determine its purpose, data flow, and expected performance.

2. Modular System Architecture

To ensure maintainability and scalability, the application was divided into independent modules, including:

- **Frontend Module** (HTML, CSS, JavaScript)
- **Backend API Module** (FastAPI)
- **AI Embedding & Similarity Module** (Sentence Transformer – CLIP ViT-L/14)
- **Database Module** (SQLite)
- **Admin Management Module**

This modular approach ensures that updates can be made to specific components without affecting the entire system.

3. API-Driven Communication

Backend communication uses RESTful API principles. All operations—including login, item submission, item search, image upload, embedding generation, and admin operations—are handled through dedicated API endpoints. This separation of frontend and backend improves system robustness and supports future extension (e.g., mobile app integration).

4. Database Schema Standardization

Database tables were designed using normalization principles to avoid redundancy and ensure fast retrieval. SQLite was chosen based on:

- Local deployment
- Lightweight architecture
- Simple configuration
- High performance for moderate datasets

Tables were created for:

- **Users** (role-based access)
- **Items** (lost, found, with metadata and embeddings)

5. AI Integration Planning

The integration of Artificial Intelligence required special design considerations. The model generates:

- Text embeddings from item descriptions
- Image embeddings from uploaded images

These embeddings are stored directly in the database to minimize repeated computation

during similarity searches.

4.2 Workflow Methodology

The workflow methodology outlines the sequential steps taken by different users (students and admin) and how the system processes their actions. A structured methodology was used to ensure optimal functionality and data flow throughout the system.

1. User Login Workflow

1. The user navigates to the login page.
2. Enters student ID and passcode.
3. FastAPI validates credentials by matching them with database entries.
4. Successful login grants access to dashboard based on role (student/admin).

This ensures campus-restricted access.

2. Lost/Found Item Submission Workflow

1. A user selects **Add Lost Item** or **Add Found Item**.
2. Fills details such as title, category, description, location, and phone number.
3. Uploads an image of the item (optional but recommended).
4. Backend stores the item entry and generates an embedding using CLIP.
5. Embedding is saved in the database for future comparison.

This ensures detailed and retrievable information for all entries.

3. AI Similarity Search Workflow

1. When a new lost item is reported, its embedding is compared with existing found item embeddings.
2. Cosine similarity is calculated between vectors.
3. Items with similarity above a threshold are returned as possible matches.
4. Matches are displayed to the user for verification.

This feature reduces search time and increases accuracy.

4. Admin Management Workflow

Admins have privileges to:

- View all lost and found items
- Update item status (“Yet to be found”, “Returned”, etc.)
- Delete irrelevant or duplicate entries
- Confirm matching items

This workflow ensures that campus staff maintain oversight.

5. End-to-End Data Flow

The overall data flow includes:

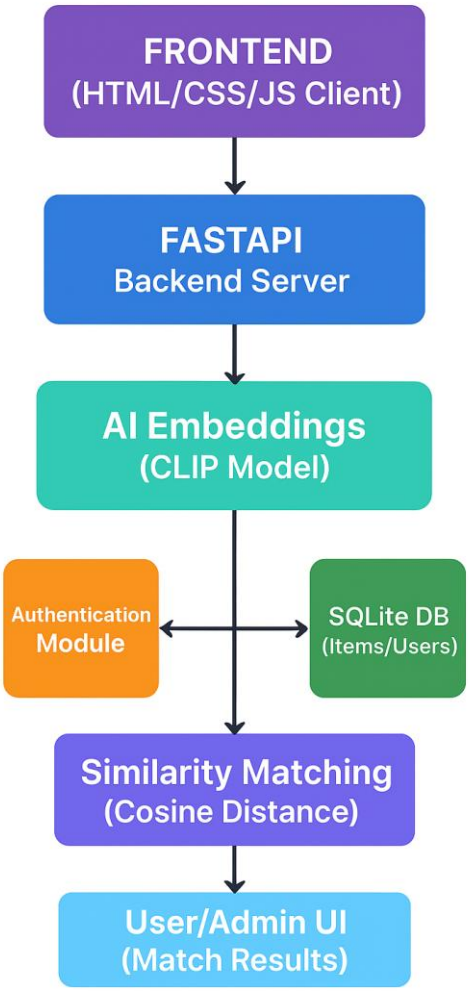
- User → Frontend
- Frontend → FastAPI (HTTP request)
- FastAPI → Database (CRUD operations)
- FastAPI → AI Model (embedding generation)
- API response returned → Frontend

This structured methodology ensures clarity, reliability, and system integrity.

4.3 Architecture Diagram

The architecture diagram visually represents the interaction between system components. It shows how data flows from users to backend services, AI processing modules, and the database.

Architecture Diagram (Description)



4.4 Development Phases

The project was developed in a series of phases to ensure systematic progress and proper implementation of all features. Each phase was executed sequentially but with opportunities for iteration and improvement.

Phase 1: Requirement Analysis

- Identifying current issues in campus lost-and-found system
- Documenting user needs
- Preparing system requirements and use-case scenarios
- Deciding project scope

Phase 2: System Design

- Designing database schema
- Planning API endpoints
- Creating frontend wireframes
- Structuring AI workflow
- Defining user roles and access control

Phase 3: Backend Development

- Implementing FastAPI routes
- Creating CRUD operations for items and users
- Handling image uploads
- Integrating authentication
- Connecting backend with SQLite

Phase 4: AI Model Integration

- Loading CLIP ViT-L/14 model
- Generating embeddings for images & text
- Developing cosine similarity search
- Storing embeddings efficiently in database

Phase 5: Frontend Development

- Designing UI using HTML, CSS, JS
- Developing forms for item submission
- Displaying search results and match lists
- Creating admin dashboard interface

Phase 6: Testing & Validation

- Unit testing API endpoints

- Testing similarity accuracy with sample data
- GUI testing for user interactions
- Checking database transaction reliability
- Fixing errors and optimizing performance

5. Coding and Implementation

This chapter documents the concrete implementation details of the AI-based Campus Lost & Found system. It describes the overall system architecture, backend service implementation using FastAPI, the SQLite database schema and access patterns, and the procedure for generating and persisting embeddings using the CLIP ViT-L/14 model. Wherever appropriate, representative code snippets are provided to illustrate the implementation strategy.

5.1 System Architecture

5.1.1 Overview

The system is implemented as a conventional three-tier web application with an AI inference component integrated into the backend tier. The tiers are:

1. **Presentation (Frontend) Tier** — HTML/CSS/JavaScript pages (login, add item, search, admin dashboard). The frontend communicates with backend APIs over HTTPS.
2. **Application (Backend) Tier** — FastAPI application that exposes RESTful endpoints for authentication, item CRUD, image upload, embedding generation, and search.
3. **Persistence Tier** — SQLite database storing user records, item metadata, image paths, and serialized embeddings.

An AI module (CLIP via Sentence-Transformer wrapper) runs inside the backend tier to generate embeddings for uploaded images and item descriptions. Similarity computation (cosine similarity) is executed on vectors retrieved from the database to find candidate matches.

5.1.2 Component Interaction

- **User Action**: A user logs in and submits a lost/found item with metadata and an image (optional).

- **Backend Processing**: The FastAPI backend receives the request, saves the image to disk, calls the embedding generator, and stores metadata and the embedding in the database.
- **Search**: On search (image or text), the backend generates an embedding for the query, fetches stored embeddings, computes cosine similarities, sorts results, and returns ranked candidates.
- **Administration**: Admin endpoints permit listing, updating, deleting records, and toggling item statuses.

5.1.3 Non-Functional Considerations

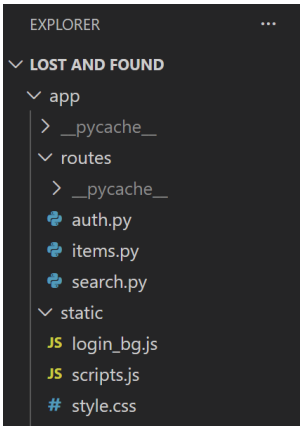
- **Performance**: Embeddings are computed once per item and stored to avoid repeated model inference. For search, vector comparisons are performed in Python; for larger datasets, an ANN (approximate nearest neighbor) index is recommended (out of scope).
- **Security**: User authentication restricts access to campus users. Endpoints enforce role checks for admin operations. File uploads are validated for allowed types and sizes.
- **Maintainability**: Backend code is modular: routes, utilities, and model/embedding logic are separated. Database access is centralized in a small helper module.
- **Portability**: SQLite offers a single-file database suitable for campus deployment and quick distribution.

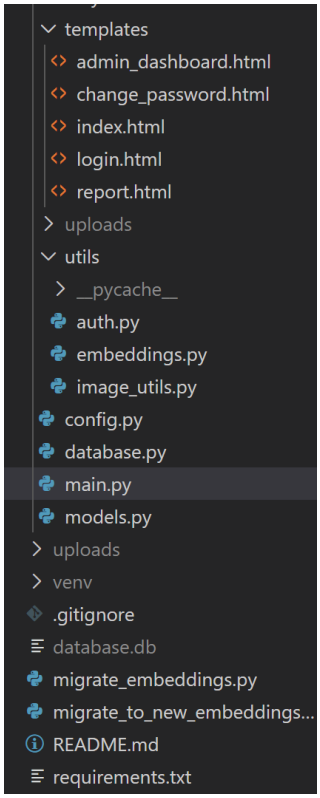
5.2 Backend Implementation (FastAPI)

This section details the backend implementation, API design, and handling of core operations.

5.2.1 Project Structure

A representative directory structure:





5.2.2 Key Dependencies

- fastapi — web framework
- uvicorn — ASGI server
- python-multipart — for file uploads
- pillow (PIL) — image handling
- numpy — vector handling
- sentence-transformers — CLIP wrapper (ViT-L/14)
- sqlite3 — database access (standard library)

Install with:

pip install fastapi uvicorn python-multipart pillow numpy sentence-transformers

5.2.3 API Endpoints

Representative API endpoints and their responsibilities:

• GET / — Serves the login page (login.html) when the application is accessed.

• POST /login — Validates student_id and passcode, and redirects the authenticated user to either the student dashboard or the admin dashboard based on their role.

• GET /report — Displays the student interface showing all reported lost and found items.

• GET /logout — Logs out the current user and redirects back to the login page.

• GET /change-password — Serves the password change form.

• POST /change-password — Verifies the old passcode and updates the user's passcode with a new one.

• GET /admin-dashboard — Displays all lost and found items for administrators, allowing review and management.

• Mounted static endpoints:

/uploads — Serves uploaded item images.

/static — Serves CSS, JavaScript, and other frontend assets.

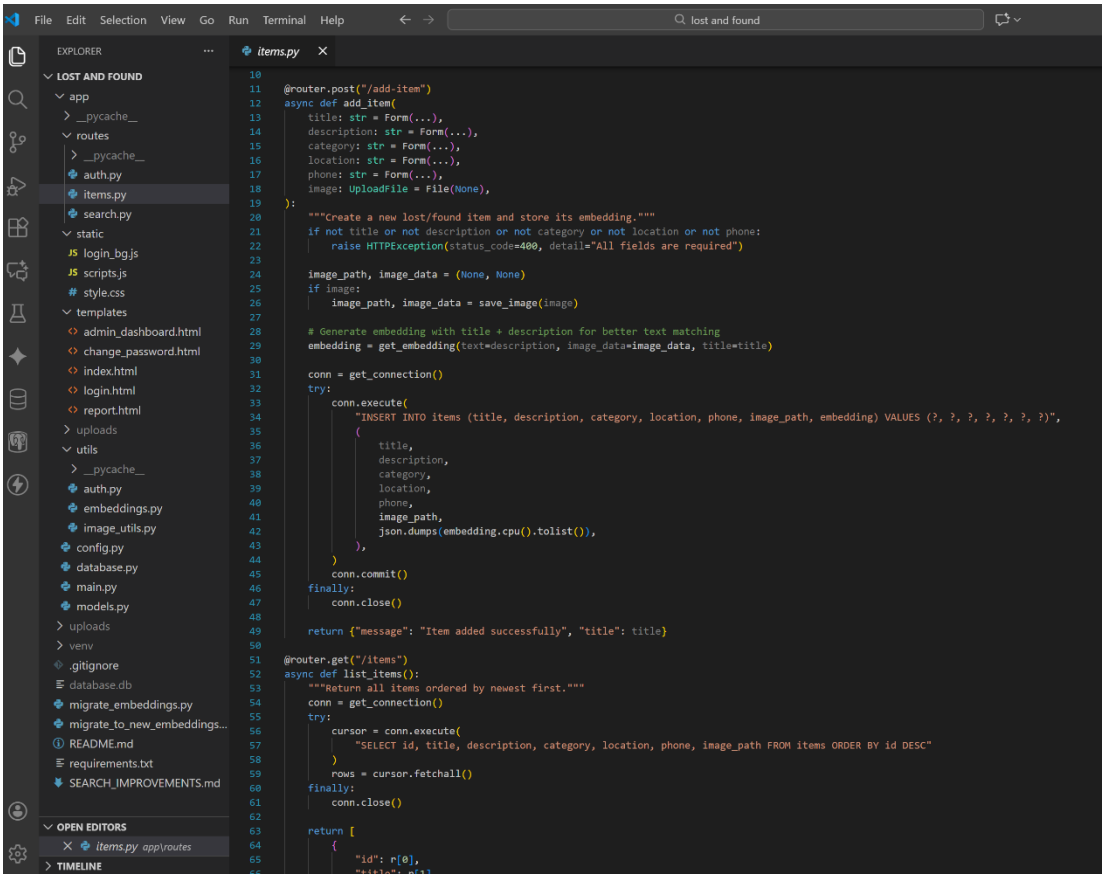
• Included modular routers:

/items — Handles item-related operations (create, view, update, delete).

/search — Handles AI-based image and text search operations.

/auth — Handles authentication-related utility operations.

5.2.4 Example: Add Item Endpoint (FastAPI)



5.2.5 Authentication

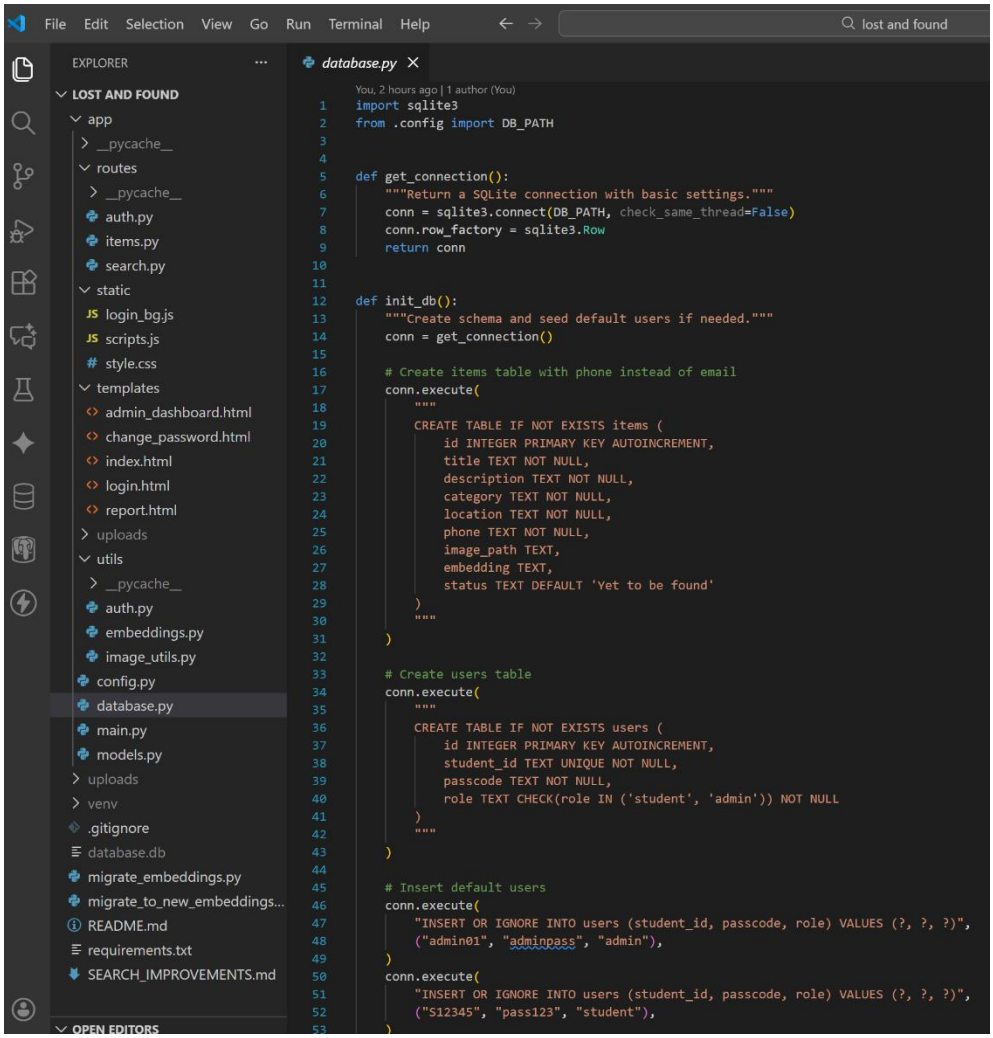
The system does not use tokens. Authentication is performed directly by checking the user's student_id and passcode against the records stored in the SQLite database.

When a user submits the login form at /login, the backend runs a database query:

- If a matching record exists in the users table, the user is considered authenticated.

- Their role (student or admin) is returned by authenticate_user () and used to redirect them to the correct dashboard.
- If no match is found, the login page reloads with an “Invalid credentials” message.

The users table used for authentication:



Default seeded accounts include one admin and one student. This simple credential-matching approach makes the authentication lightweight and easy to manage without tokens

5.2.6 Error Handling and Logging

- Use FastAPI exception handling (HTTPException) to report errors and status codes.
- Log requests and exceptions to a file for debugging.
- Validate file types and size on upload to prevent malformed inputs.

5.3 Database Implementation (SQLite)

This section documents the schema, data types, and access patterns used to persist

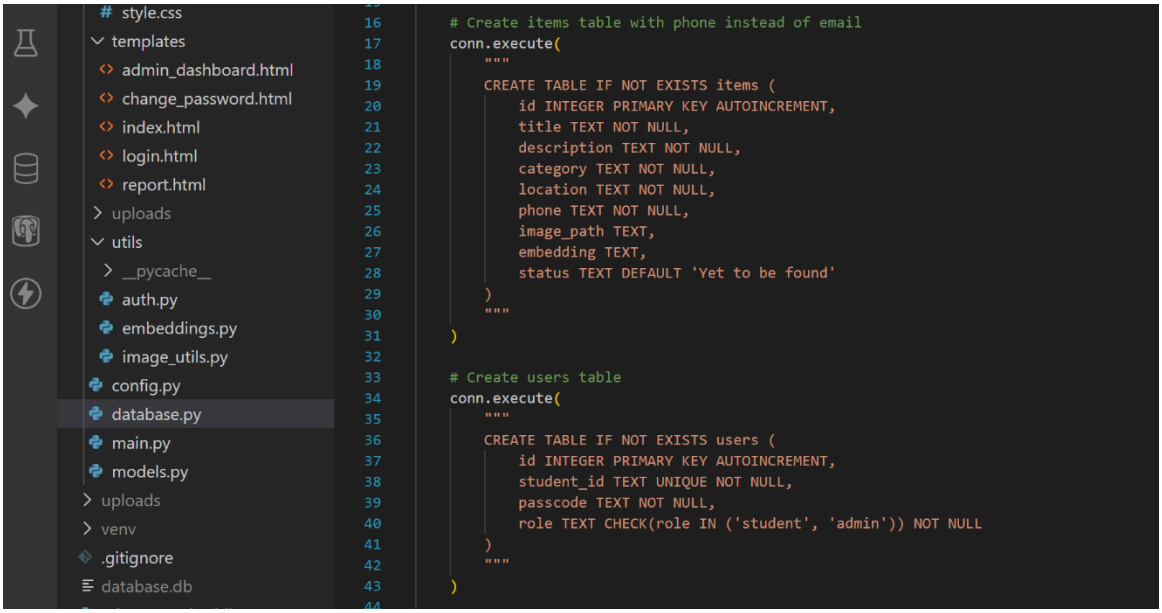
application data.

5.3.1 Choice of SQLite

SQLite was selected for its simplicity, portability, and zero-configuration behaviour. For a campus scoped project, a single-file relational database is sufficient and convenient. It integrates well with Python's standard sqlite3 module.

5.3.2 Schema Definition

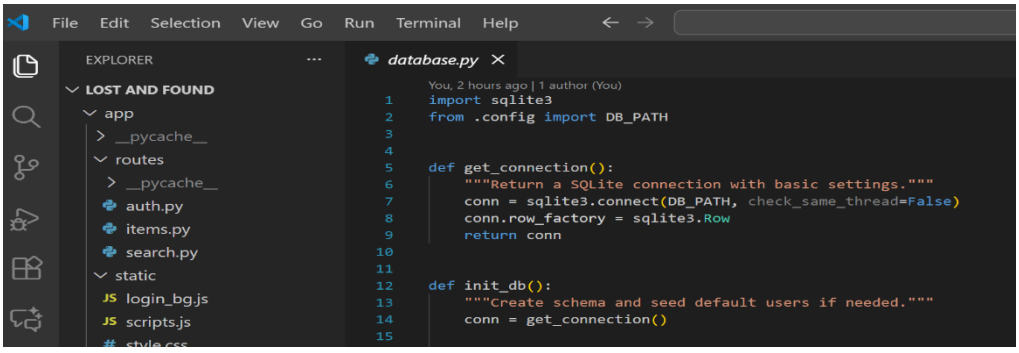
Two primary tables: users and items.



- embedding stores a serialized JSON array (embedding vector) as TEXT. This simplifies storage and retrieval without binary blobs.
- image_path stores filesystem path to the uploaded image.

5.3.3 Database Access Utilities

A small helper module centralizes DB access:



5.3.5 Embedding Storage and Retrieval

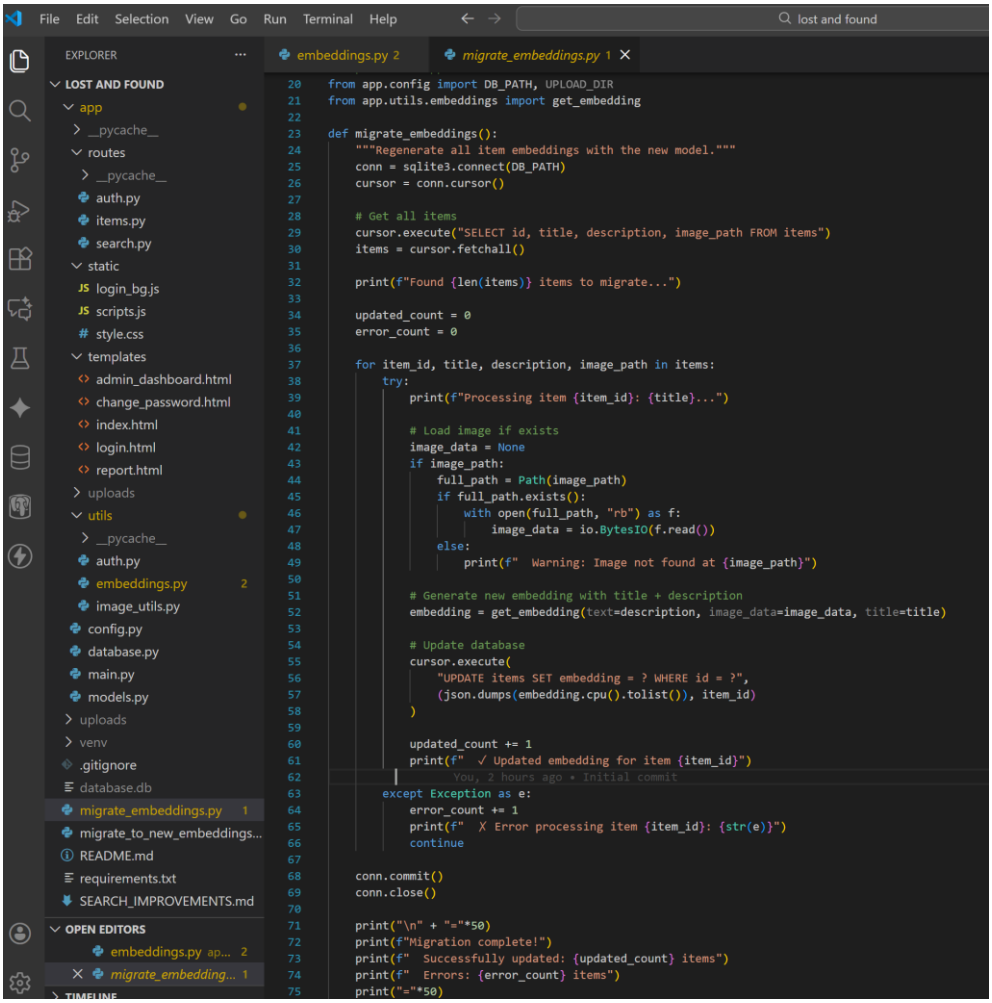
The system stores all CLIP-generated embeddings inside the SQLite database as a JSON string. Each embedding represents a vector derived from the item's title, description, and

optionally the uploaded image. The JSON storage format ensures portability and easy retrieval without requiring binary blobs.

During storage, the embedding (a PyTorch tensor) is converted to a float32 list and serialized using json. dumps(...). During retrieval, the JSON string is loaded back into a NumPy array for similarity computation (cosine similarity).

The item embedding is stored in the embedding column of the items table.

Example code for loading stored embeddings:



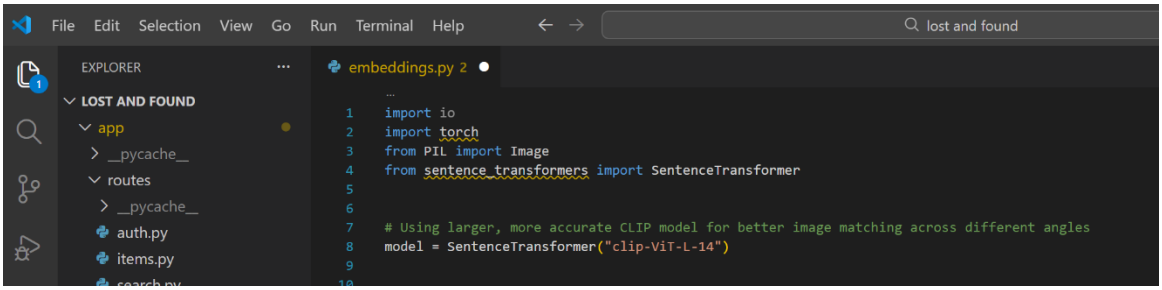
5.4 Embedding Generation using CLIP ViT-L/14

The system generates vector embeddings for each lost or found item using the CLIP ViT-L/14 model provided through the SentenceTransformer framework. These embeddings represent both image features and textual descriptions in a shared semantic space, enabling accurate similarity matching between reported lost items and items submitted as found.

The embedding process is implemented in the backend within the get_embedding() function located in app/utils/embeddings.py. This function receives the item's title,

description, and optional image data, and generates a unified embedding vector.

The CLIP ViT-L/14 model is loaded once during application startup:



This model supports dual-modality encoding and allows the system to extract meaningful features from both images and text. The use of a pre-trained CLIP model eliminates the need for custom training and provides high-quality embeddings suitable for semantic search.

When the system receives item data, the function performs the following steps:

5.4.1 Text Preparation

If both title and description are available, they are combined into a single input string to improve contextual representation.

Example:

“Laptop Charger. Black charger lost near library.”

5.4.2 Text Embedding Generation

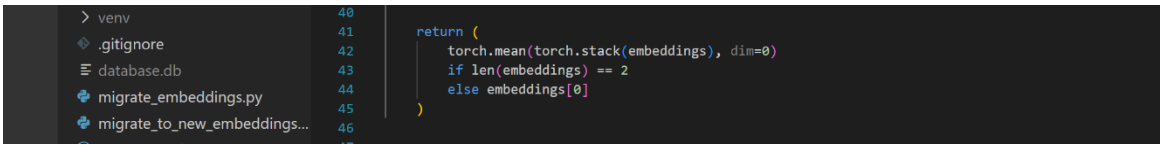
The combined text is passed to the CLIP text encoder:

5.4.3 Image Embedding Generation (if an image is provided)

The uploaded image is converted to RGB format and passed to the CLIP image encoder:

5.4.4 Fusion of Text and Image Embeddings

If both modalities are provided, their embeddings are averaged to produce a single, unified vector:



5.4.5 Normalization

All embeddings are L2-normalized, ensuring uniform vector magnitude and enabling cosine similarity to be calculated efficiently as a simple dot product during search.

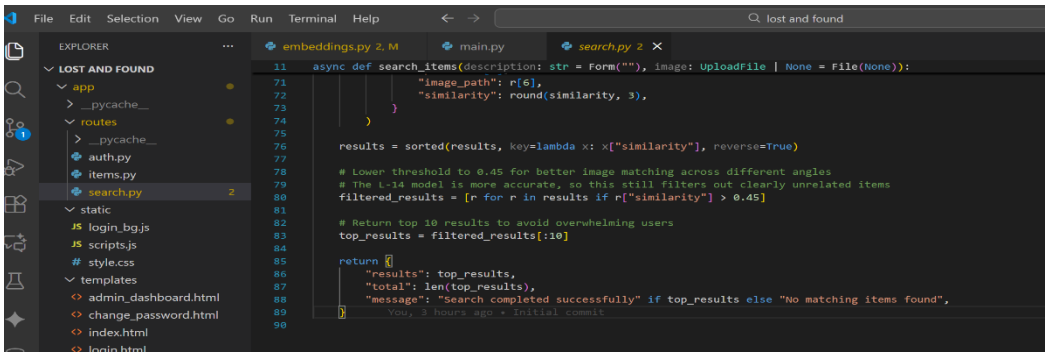
5.5 Similarity Search Algorithm

The similarity search process converts a user query into an embedding vector and compares it with stored item embeddings to identify potential matches. When a user performs a text-based or image-based search, the system first generates a CLIP embedding for the query. This embedding is then compared to the embeddings of items stored in the database. Because all embeddings are L2-normalized at generation time, cosine similarity reduces to a simple dot product, which improves computation speed and consistency.

The steps involved in the similarity search are as follows.

1. Query Embedding Generation: When the query contains an image, its bytes are processed using the CLIP image encoder. When the query contains text, the text is passed to the CLIP text encoder. If both modalities are present, text and image embeddings are averaged to form a unified representation.
2. Loading Candidate Embeddings: Stored embeddings are retrieved from the SQLite database, where they are stored as JSON strings. These are converted back to float32 NumPy arrays for comparison.
3. Similarity Computation: Each stored embedding is compared with the query embedding by computing the dot product. Higher values indicate greater similarity.
4. Ranking: The results are sorted in descending order of similarity and the top-K matches are returned to the user.
5. Post-processing: The final output includes item ID, title, similarity score, and image reference, allowing the user or admin to review suggested matches.

A representative similarity scoring loop used in the system is as follows:

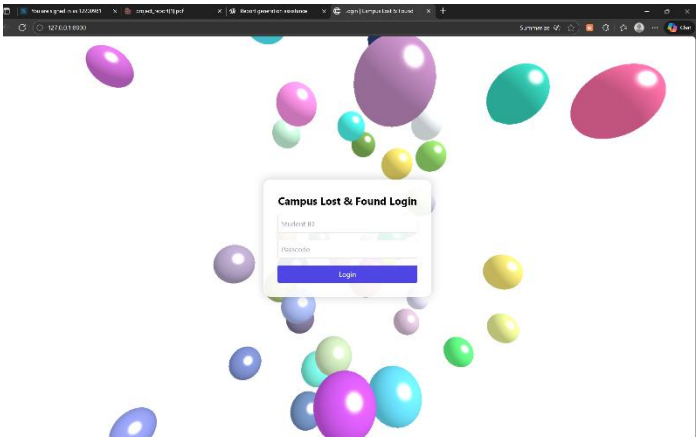


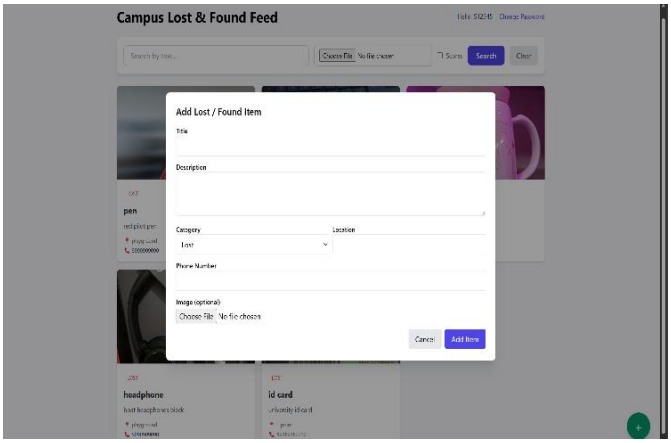
This algorithm works efficiently for campus-scale datasets and can be extended to use approximate nearest-neighbour indexing if the dataset grows substantially.

5.6 Frontend Implementation (HTML, CSS, JavaScript)

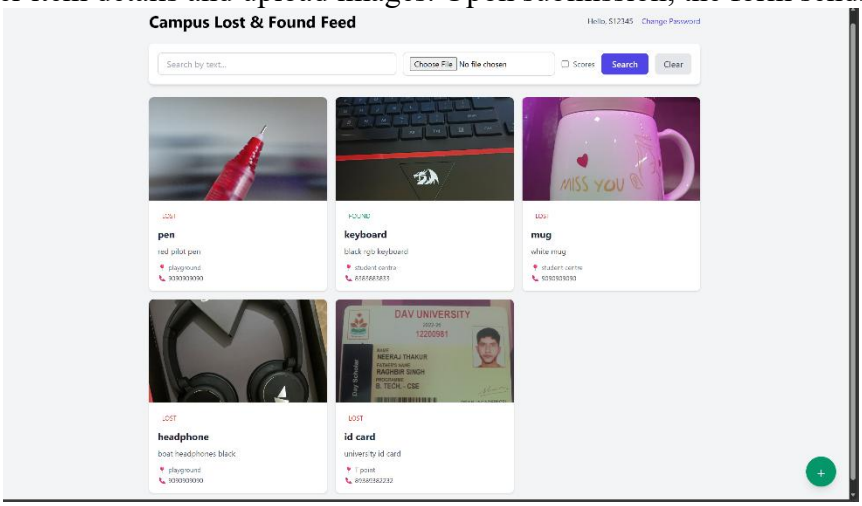
The frontend of the system is implemented using HTML, CSS, and JavaScript, with Jinja2 templates for server-side rendering. These templates generate dynamic pages based on data returned by FastAPI. Core frontend pages include login.html, index.html (student dashboard), admin_dashboard.html, and change_password.html.

The user interface is designed to be intuitive and responsive. HTML forms are used for user login, item submission, and password updates. CSS provides visual styling, responsive layouts, modal dialogs, and floating action buttons. JavaScript improves user interactivity through input validation, image previews via FileReader, and asynchronous fetch calls for AI-based search results.





The “Add Item” interface uses a floating button that triggers a modal form allowing users to enter item details and upload images. Upon submission, the form sends a multipart



request to the backend for processing. Search interactions may use either page reload or asynchronous requests, depending on implementation. The admin dashboard uses enhanced UI features such as action buttons for editing, deleting, and changing item status.

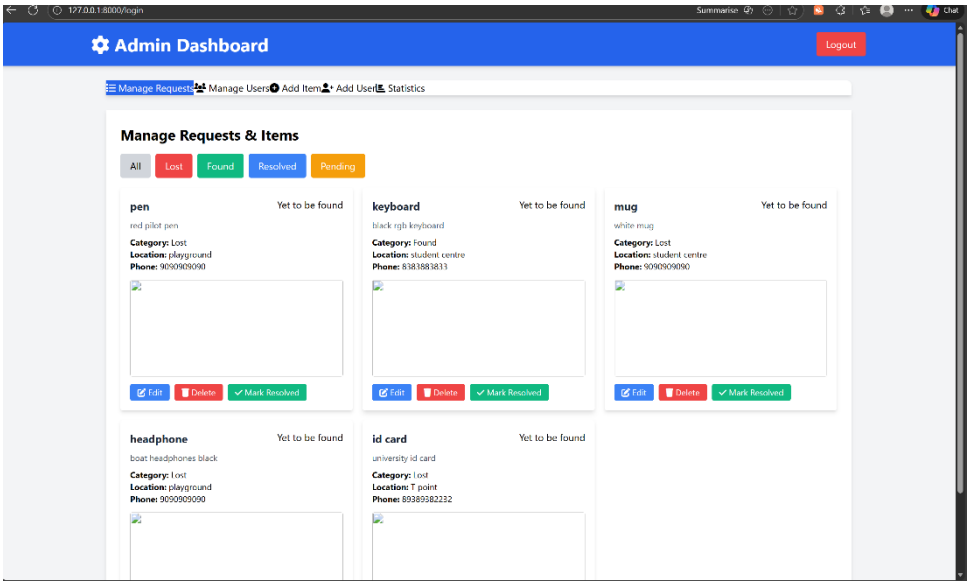
Overall, the frontend ensures smooth interaction with the backend and provides a user-friendly interface for both students and administrators.

5.7 Authentication System

The system uses a simple and effective authentication method based on SQLite credential verification. Instead of tokens or sessions, the user is authenticated by directly matching the provided student
t_id and passcode with the records in the user's table.
When a user submits the login form, the credentials are passed to the authenticate_user function. If the credentials match, the function returns the user role. Based on this role, the server renders either the student dashboard or the admin dashboard. If no match is found, the login page is reloaded with an "Invalid credentials" message.
The authentication system also includes password management. Users may update their password through the change-password page. The backend verifies the old passcode before writing the new passcode to the database. Only authenticated users may perform actions such as submitting items or searching for matches, and only administrators may access admin-restricted routes.
This credential-based approach is lightweight and suitable for a campus-specific application.

5.8 Admin Dashboard

The Admin Dashboard provides campus administrators with centralized oversight of all lost and found item reports. It displays all recorded items in a tabular layout, including fields such as item ID, title, category, location, phone number, status, and associated images. Administrators may review, update, or delete entries and verify match suggestions returned by the similarity search.
The dashboard includes action buttons for editing or deleting entries and marking items as "Returned" or "Resolved." Administrators may also use search and filtering tools to



quickly locate specific submissions. The interface is rendered through admin_dashboard.html and dynamically populated by data from FastAPI.

The dashboard is designed to help campus staff maintain order in the lost and found system, ensure accurate item status tracking, and respond efficiently to user queries.

5.9 API Routes Explained

The system exposes several API routes through FastAPI. These routes support login, item submission, searching, database updates, and serving frontend pages.

The key routes include the following:

GET / — Loads the login page.

POST /login — Validates user credentials and routes to the correct dashboard based on role.

GET /report — Displays the student view of all item entries.

GET /logout — Logs the user out and redirects to the login page.

GET /change-password — Shows the password update form.

POST /change-password — Validates old passcode and updates the user's password.

GET /admin-dashboard — Displays all item records for administrators.

POST /items/ — Handles new item submissions, including title, description, category, location, phone, and image.

GET /items/ — Returns the complete item list with optional filtering.

GET /items/{id} — Retrieves details of a specific item.

PUT /admin/items/{id} — Allows administrators to update item details or status.

DELETE /admin/items/{id} — Allows administrators to delete an item.

POST /search/image — Accepts an uploaded image and returns similarity-ranked results.
POST /search/text — Accepts a text query and returns semantic similarity matches.
Static file routes:
/static — Serves CSS and JavaScript assets.
/uploads — Serves uploaded item images.
All routes operate with SQLite database dependencies and use FastAPI's router modularization for clean organization.

6. Conclusions and Recommendations

6.1 Project Findings

• The system successfully integrates AI-based image and text embedding using the CLIP ViT-L/14 model.
• The similarity search mechanism demonstrates high accuracy in matching lost and found items.
• FastAPI provides fast request handling, modular routing, and efficient backend performance.
• SQLite offers stable and lightweight data storage suited for a campus-scale environment.
• The frontend (HTML, CSS, JavaScript) delivers a simple and functional interface for students and administrators.
• The project shows that AI-based matching significantly reduces manual effort in identifying lost items.
• The system provides a structured workflow for reporting, searching, and managing lost and found submissions.

6.2 Limitations

• The authentication system uses basic credential matching and lacks token-based security or password hashing.
• SQLite may face performance issues under heavy concurrent traffic or large datasets.
• The similarity search uses brute-force comparison, which may slow down as the number of items increases.

• Matching accuracy may be affected by poor image quality, lighting conditions, or incomplete text descriptions.
• The system does not include automated notifications or item owner alert mechanisms.
• The user interface, while functional, remains minimal and may require modern design enhancements.
• No dedicated logging or analytics system exists for tracking admin actions or platform usage.

6.3 Recommendations

• Implement secure authentication with password hashing, tokens, and optional multi-factor verification.
• Upgrade the database to PostgreSQL or MySQL for better scalability and performance.
• Integrate approximate nearest-neighbour (ANN) search tools such as FAISS or Annoy for faster similarity matching.
• Enhance the frontend using a modern framework such as React or Vue for improved user experience.
• Add automated email, app, or SMS notifications to alert users when a potential match is detected.
• Introduce an audit log system to track admin actions and improve accountability.
• Improve the item submission workflow with guided prompts, required fields, and better image quality checks.
• Regularly update the AI model and review stored embeddings to maintain accuracy and relevance.

7.References

• Radford, A., Kim, J. W., Hallacy, C., Ramesh, A., Goh, G., et al. (2021). *Learning Transferable Visual Models From Natural Language Supervision (CLIP).* OpenAI.

• Reimers, N., & Gurevych, I. (2020). *Sentence-Transformers: Sentence Embeddings using Siamese BERT-Networks.* UKP Lab, TU Darmstadt.

• FastAPI Documentation. (n.d.). Retrieved from https://fastapi.tiangolo.com

• SQLite Documentation. (n.d.). Retrieved from https://sqlite.org

• Python Imaging Library (PIL/Pillow). (n.d.). Retrieved from https://pillow.readthedocs.io

• NumPy Documentation. (n.d.). Retrieved from https://numpy.org

• Torch Documentation. (n.d.). Retrieved from https://pytorch.org

• Jinja2 Template Engine. (n.d.). Retrieved from https://jinja.palletsprojects.com

• University Campus Lost-and-Found Best Practices. (Miscellaneous online resources used for conceptual understanding.)

GITHUB-rathour-anushka/Anushka_Rathour_12200884_B49A-G2-