A Multimodal AI-Based Smart Campus System for Lost and Found Item Recovery

Devkaran Jawal
VIT, Vellore

Joshua Roland Williams
VIT, Vellore

Nikhil Singla
VIT, Vellore

Sumit Kumar
VIT, Vellore

*Swarnalatha P
Faculty, SCOPE
VIT, Vellore

***Abstract*—Dense populations and heavy movement in university campuses usually make the traditional lost-and- found processes slow and inefficient. This paper presents a Smart Campus Lost and Found System (SCLFS) which is a web-based model that aims to simplify such processes through a synthesis of multimodal artificial intelligence and a user-incentive based model. The platform is built with a lightweight Flask back end with SQLAlchemy as a data management platform. The key feature of it is a multimodal matching module that is developed based on CLIP-ViT-B/32 vision-language network that calculates a composite similarity score between item descriptions and images. The system can correlate visual and textual items more precisely than the single-mode systems by embedding inputs concurrently. In order to reward participation, a gamified system provides virtual tokens to individuals that submit items successfully, but also has mechanisms that ensure that the system is not abused. Its implementation was highly tested-unit, integration and system testing-to meet conformity to the functional objectives.Experimental results demonstrate that the matching process supported by artificial intelligence can reach high accuracy levels in the association of similar records (reaching 91.8 percent similarity) and that the reward system has been proven to operate as expected, encouraging ethical user interaction.**

***Index Terms*—Smart Campus, Lost and Found, Multimodal AI, CLIP, Contrastive Language-Image Pre-training, Flask, Software Engineering, Gamification, User-Centered Design.**

I. Introduction

College campuses are in many ways miniature urban environments, only a small area is hosting thousands of students, professors and administrative staff. In such a dynamic and highly populated environment, misplaced or lost personal items are a very frequent phenomenon that is usually irritating. Traditional lost and found methods which are generally administered via handwritten books or through a collection of notice boards or informally through email messages, were usually disorganized, inconsistent and hard to keep up with [1], [2]. The traditional approaches have a poor scaling to large campuses and have no way of automatically matching the owners with the objects that are theirs [3]. The issue is also increased by the great number of misplaced items, which can either be indistinguishable water bottles or remotely recognized electronic accessories. This type of diversity restricts the usefulness of the key word or a single modality image search. Consequently, there is a call to a clever, multimodal methodology that would be able to analyze in tandem. visual and textual features to enhance the level of identification.

The Smart Campus proposal, which is based on the concept of combining data analytics, Internet of Things (IoT) tools, and intelligent automation, can provide an avenue to addressing this concern [4], [5]. A smart campus aims at improving the efficiency in its operations and user experience and a robust technology-based lost-and-found system compliments the societal and governance objectives of the smart campus [6].

The proposed study is the Smart Campus Lost and Found System (SCLFS) where it is suggested that the solution is a web-based model designed to automate and intelligently organize the recovery of lost items.The design of the system aims at filling the gaps in the current methods using three main innovations:

1) **Multimodal Matching Engine:** It is based on the Contrastive Language-Image Pre-training (CLIP) model that builds a shared space of item descriptions and images. This enables the similarity score to be calculated by the matching algorithm with finer accuracy than the traditional text- or image-only algorithms.
2) **Socio-Technical Incentive Framework:** As the efficiency of a system is determined by the involvement of the user in its implementation, the platform will have a gamified reward system based on virtual coins. The given feature prompts the users to report and send back the found items stimulating interaction and addressing the main issues of human-computer interaction [7], [8].
3) **Lightweight and Scalable Architecture:** Built on a Python-based stack (Flask and SQLAlchemy), the system is maintainable, scalable and easily deployable [9]. Its performance and reliability is further checked by a detailed multi-tier testing pro-cess.

The rest of this paper follows in the following manner: Section II discusses the literature and finds gaps of research. Section III is the discussion of the proposed architecture and database schema and multimodal matching process. In Section IV, there is implementation results, and user interface validation. At the end of the study is Section V, which summarizes the study and provides future research directions.

II. Literature Survey

In order to emphasize the novelty of the proposed system, the literature review and technology review over four key fields were conducted.

A. Digitization of Digital Notice boards to Web Portals

The first digitalization of traditional logbooks was in the form of digital notice boards, web-based applications that enable its users to add information about lost or found items manually [3], [10]. These systems helped to make accessibility better than the analog systems but were still constrained in their ability. The majority used only the key word search method and it was necessary to go through the long lists of the posts manually, which was not efficient and in many cases ineffective [1], [2]. These solutions were not automated and did not have any intelligent matching.

B. Hardware based Tracking Solutions

Another similar line of development was hardware-driven tracking technologies. The solutions of this type usually used Radio-Frequency Identification (RFID) tags, Global Positioning System (GPS) modules [12], or Long-Range (LoRa) networks [13]. Although the methods were also found effective in tracking pre-tagged high-value institutional property, the methods are neither affordable nor feasible to the unpredictable and unstructured assortment of personal items, including bottles, notebooks or chargers, which often go missing on campuses.

C. Image-Based Matching: First-Generation AI

Later systems started to use fake intelligence, which acted more on similarity detection in images. To find feature representations of item images and obtain visual similarity scores, these methods frequently used Convolutional Neural Networks (CNNs), including ResNet-50 [14] or MobileNetV2 [15], [16]. Others also improved on this by the introduction of independent Natural Language Processing (NLP) modules to text-based comparison [14]. However, these modalities

D. Vision-Language Models: State-of-the-Art

The development of Vision-Language Models (VLMs) was a major improvement. CLIP is a supervised visual concept learner based on natural language supervision and projects images and text-based learning to a common embedding space [18]. This makes it possible to perform potent zero-shot text-to-image retrieval [19] and image-text correspondence [20]. These features would best fit the lost-and-found example- e.g. when a user reports about a black steel bottle with a logo, another one posts the photo with the logo.

E. The Human Factor: Incentivization and UserCentered Design

System success, regardless of technical advancement is eventually dependent on user acceptance. The frameworks of User-Centered Design (UCD) are important in making campus applications usable and accessible [8]. Furthermore, researches stress that such social obstacles as the unwillingness to give up finding items can be reduced with the help of incentive systems. Research on gamification and motivation indicates that properly constructed reward systems can successfully enhance the motivation and the involvement in the digital environment.

F. Identified Research Gap

It is evident that there is a gap in the reviewed literature. Existing solutions either use simplistic web interfaces (Section 2.1), hardware-specific infrastructures (Section 2.2), or crudely trained AI models with little integration (Section 2.3). Very little, or anything, has linked both high-level VLMs like CLIP (Section 2.4) and an objectively-based incentive system (Section 2.5) into one, operational, and deployable smart campus system. The need that has not been fulfilled currently is addressed in the present work by integrating the technical aspect as well as the social aspect in one unified platform.

III. Architecture and Methodology

SYSTEM ARCHITECTURE The Smart Campus Lost and Found System (SCLFS) is a scaled and realistic application that converts initial specification needs into a realistic architecture.

A. System Overview and Architectural Style

The suggested system is based on a three-level client-server architecture. The client component runs on a regular web browser, and the server is provided in its prototype form as a monolithic web application with the Flask framework. The system context in general is shown in Fig. 1 and it describes the key actors (User and Administrator) and how they interact with the key components of the system.

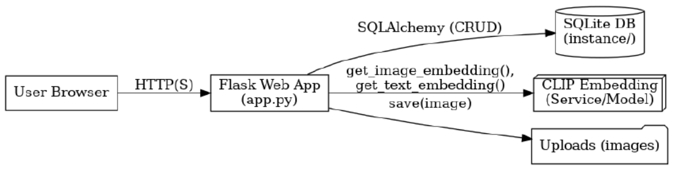


Fig. 1. System Context Diagram

The architecture goes further to be broken down into separate functional modules, as presented in the Component Diagram (Fig. 2). The resulting modularization (image below) with the web routing logic (app.py) and AI processing unit (aimatching.py) separated along with the database layer (database.py) improves maintainability, high cohesion, and ease of debugging and upgrades.

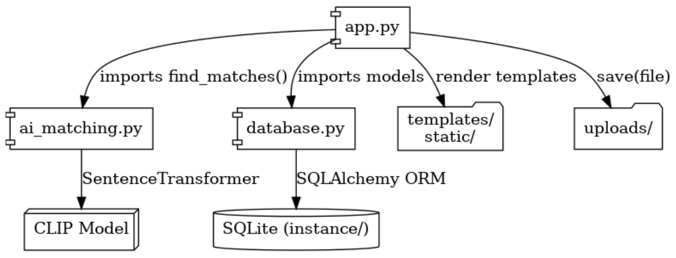


Fig. 2. Component Diagram

B. Data Design and Schema

Persistence layer is handled with an SQL database which is accessed by SQLAlchemy, an Object-Relational Mapper (ORM). It is an abstraction that does not bind application logic to the database technology being used allowing the prototype to run on SQLite but be production-ready on PostgreSQL. The schema includes three major entities: User, Item, and Match. Their associations are depicted in the Class Diagram (Fig. 3). Table I gives a detailed data dictionary of the schema attributes.

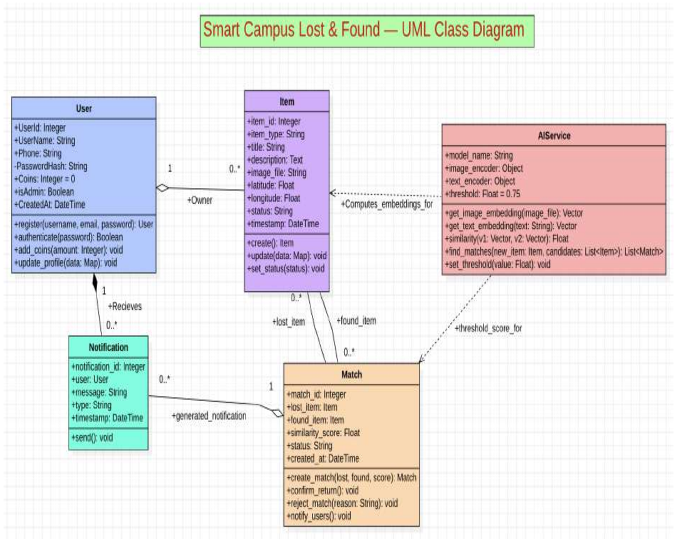


Fig. 3. Class Diagram

TABLE I
THE DATA DICTIONARY USED IN SCLFS DATABASE SCHEMA IS AS PRESENTED BELOW.

Table	Column	Data Type	Description
User	id (PK)	Integer	Unique identifier for each user.
	username	String(100)	Unique username for login.
	password	String(100)	Encrypted password (PBKDF2).
	phone_number	String(20)	Optional contact number.
	coins	Integer	Gamification reward balance.
	is_admin	Boolean	Boolean flag for admin privileges.
Item	id (PK)	Integer	Unique identifier for each item.
	user_id (FK)	Integer	References `User.id` (reporter).
	item_type	Enum('lost', 'found')	Type of report.
	title	String(200)	Title of the item report.
	description	Text	Detailed description of the item.
	image_file	String(100)	Path to uploaded image file.
	image_embed	PickleType	Stored CLIP image embedding.
	text_embed	PickleType	Stored CLIP text embedding.
	status	Enum('active', 'returned')	Current status of the item.
Match	id (PK)	Integer	Unique match identifier.
	lost_item_id (FK)	Integer	References `Item.id`.
	found_item_id (FK)	Integer	References `Item.id`.
	similarity_score	Float	Combined similarity score (0.0–1.0).
	status	Enum('pending', 'returned')	Match confirmation state.

(Source: Project Database Schema)

C. Basic Methodology: Multimodal Matching Algorithm

The intelligent matching engine is the core of the system and it uses the sentence-transformers library to load an already trained model called clip- ViT-B-32. The output of this Vision-Language Model (VLM) is the projection of either textual or visual data into a single 384-dimensional space. The algorithm works in the following way:

1) The submission of a new report is made by the use of /report route.
2) The system produces two embeddings:
   - **Image Embedding:** The upload image is subjected to getimageembedding.
   - **Text Embedding:** The title together with description fields are processed with the help of. gettextembedding().
3) The two embeddings are pickled and stored in the columns (imageembedding and textembedding) of the Item table.
4) findmatches(newitem) is a function that is called so as to find possible matches.

In contrast to the typical CLIP applications, using one-directional text-to-image retrieval, direct object-to-object similarity comparisons are done in this system. In order to do so, a weighted similarity heuristic (Equation 1) was used as a custom weighting. This algorithm calculates cosine similarity of each of the two modalities and weights the two with 60 and 40 respectively.

$$S_{\text{comb}} = (0.6 \times S_{\text{img}}) + (0.4 \times S_{\text{txt}}) \tag{1}$$

where $S_{\text{img}} = \text{cosine\_similarity}(Emb_{\text{img\_A}}, Emb_{\text{img\_B}})$ and $S_{\text{txt}} = \text{cosine\_similarity}(Emb_{\text{txt\_A}}, Emb_{\text{txt\_B}})$.

When similarity score (Scomb), which is a combination of two matching scores, is greater than a fixed cutoff point (0.70) as discovered in the course of experimentation, then a match entry is added to the Match table.

D. Socio-Technical Design and System Dynamics

In addition to its technical capabilities, the SCLFS is planned as a socio technical system which enables human interactions with it especially the retrieval of lost property. Fig. 4, the "Confirm Return" procedure is realized in the. /completereturn/<matchid> route. This process formalises the process of acquiring and disposing of a finder-owner via a digital handshake process. When the owner chooses the option Mark as Returned, two important backend functions are performed:

1) **Promoting Participation:** The system adds the finder coinage (finder.coins = 100) which is em- bodying the gamification strategy [7] to reward ethical participation.
2) **Mitigating Abuse:** A validation condition (lostitem.userid = founditem.userid) prevents users who abuse the reward system by falsely reporting and retrieving their own items. A check of this validation condition was performed in white box testing.

E. Deployment and Scalability

The first prototype uses a light version of Flask development server and an SQLite database. However, the architecture has been designed in a manner that it can be deployed in a scalable production environment as shown in the Deployment Diagram (Fig. 5).

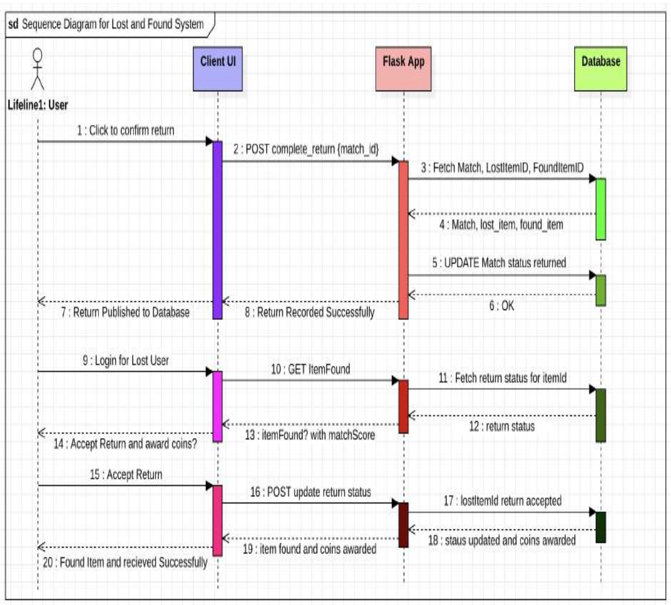

sd Sequence Diagram for Lost and Found System
Lifeline1: User
Client UI
Flask App
Database
1 : Click to confirm return
2 : POST complete_return (match_id)
3 : Fetch Match, LostItemID, FoundItemID
4 : Match, lost_item, found_item
5 : UPDATE Match status returned
6 : OK
7 : Return Published to Database
8 : Return Recorded Successfully
9 : Login for Lost User
10 : GET ItemFound
11 : Fetch return status for itemId
12 : return status
13 : itemFound? with matchScore
14 : Accept Return and award coins?
15 : Accept Return
16 : POST update return status
17 : lostItemId return accepted
18 : staus updated and coins awarded
19 : item found and coins awarded
20 : Found Item and recieved Successfully

Fig. 4. Confirm Return workflow Sequence Diagram.

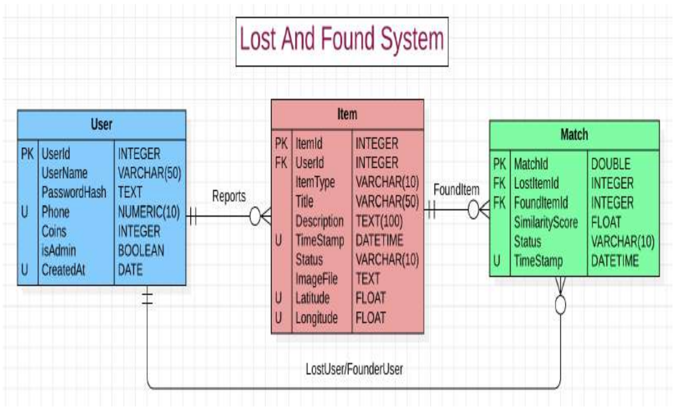


Fig. 5. Production-Intent Deployment Diagram

IV. Results and Discussion

The Smart Campus Lost and Found System (SCLFS) is one that went through a significant amount of verification and validation to determine that the functional capabilities and the underlying artificial intelligence elements of the system worked as expected.

A. Implementation Details

This entire system was executed in Python 3.10. It relied on such core dependencies as Flask, Flask-SQLAlchemy, Flask-Login, and sentence- transformers library. All

B. Functional Verification and Testing

A multi-phase testing procedure that was well organized was conducted to ensure the reliability of the system. The correctness of individual modules was checked by unit testing, the ability of components to cooperate with each other was checked by the integration testing and the overall performance was checked by the full system testing. Both black-box and white-box testing strategies were used- Equivalence Partitioning and Boundary Value Analysis of external behavior, Branch Coverage of internal logic. Specifically, one of the functions was the completereturn() to ensure that it appropriately authenticated the user, did not allow the user to reward himself and implemented the reward mechanism when successful authentication occurred. Table II displays the summarized outcomes of the testing phase. All test cases gave the desired result, which showed that all the stipulated functional requirements had been successfully achieved.

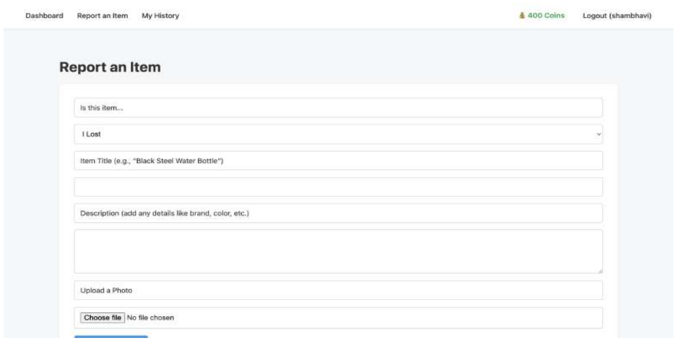
TABLE II
Summary and Results of System Test Case.

Test Case ID	Requirement	Description	Expected Result	Status
TC-REG-01	(Auth)	User registers with existing username.	Failure "Name already existing.	**PASS**
TC-LOGIN-01	(Auth)	Registered user logs in using valid credentials.	reredirected to user dashboard.	**PASS**
TC-LOGIN-02	(Auth)	Incorrect password is used to log in.	Mistake "A bad name or a bad password.	**PASS**
TC-AUTH-01	(Auth)	Non-logged-in user dashboard.	Redirected to login page.	**PASS**
TC-REPORT-01	FR1, FR2	User lost something with a photo.	Item saved to DB with status='active'.	**PASS**
TC-MATCH-01	**FR3**	**Lost item came across as similar item.**	**New Match record generated with a score that is greater than 0.70.**	**PASS**
TC-NOTIFY-01	**FR4**	**The owner of lost item logs in.**	**FR4 notification is present on dash- board.**	**PASS**
TC-RETURN-01	**(Gamify)**	**Return is confirmed by the owner.**	**Statuses of the items that are to be re- turned. Finder's coins in- crease by 100.**	**PASS**
TC-ADMIN-01	FR5	Logs in as admin.	Redirected to admin dashboard.	**PASS**

(Source: Project System Test Plan)

C. System Interface and User Experiences Walkthrough

The User- Centered Design (UCD) principles of interaction made the user interface design simple, consistent and responsive [8]. The interface of the Report an Item interface, the primary input point of the user, is depicted in Figure 6 and activates the multimodal matching pipeline.



Report an Item

Is this item...

I Lost

Item Title (e.g., "Black Steel Water Bottle")

Description (add any details like brand, color, etc.)

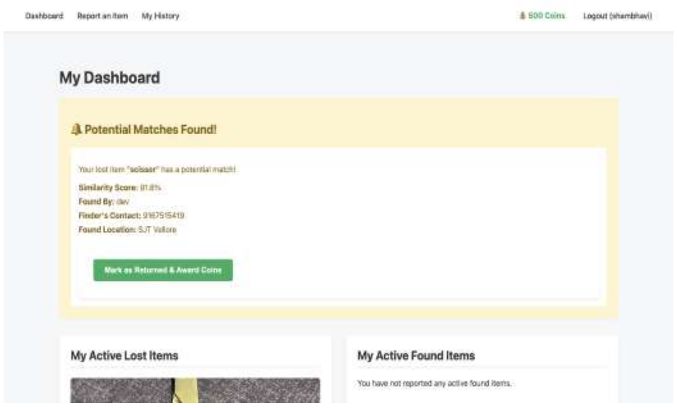
Upload a Photo

No file chosen

Fig. 6. Report Item Screen

The primary loop of feedback of this system can be seen in Figure 7, which shows a Potential Matches Found! notification. This interface combines technical output of the model with social and gamification aspects:

- **AI Performance:** The system was able to identify a corresponding match, i.e. a scissor, with a similarity score of 91.8% which occurred indicating the accuracy of the multimodal model.
- **Social Facilitation:** On the interface, the contact information and search location of the finder are easily visible, thus, simplifying reconnection to users.
- **Gamification:** Participation is formalized by the Mark as Returned and Award Coins button, and strengthened.
- **Reward Continuity:** The constant 500 Coins message is a confirmation that the gamification system is storing and updating rewards as it should.



My Dashboard

Potential Matches Found!

Your lost item "scissor" has a potential match!

Similarity Score: 91.8%

Found By: dev

Finder's Contact: 9167515419

Found Location: SJT Vellore

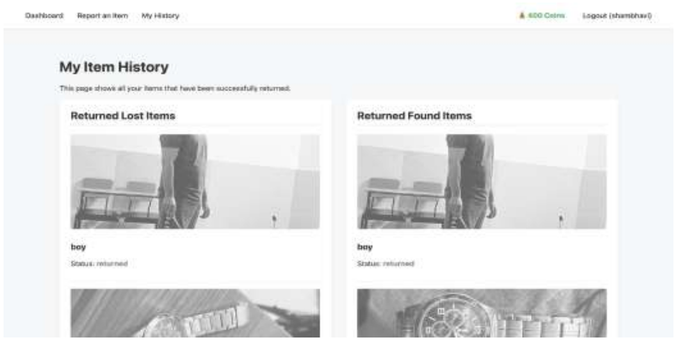
My Active Lost Items

My Active Found Items

You have not reported any active found items.

Fig. 7. AI Match Notification in User Dashboard.

The Item History view (Fig. 8) shows that all the items that were returned were returned previously, which leads to the fact that the status of the item changes to a returned status when tested under TC-RETURN-01.



My Item History

This page shows all your items that have been successfully returned.

Returned Lost Items

boy

Status: returned

Returned Found Items

boy

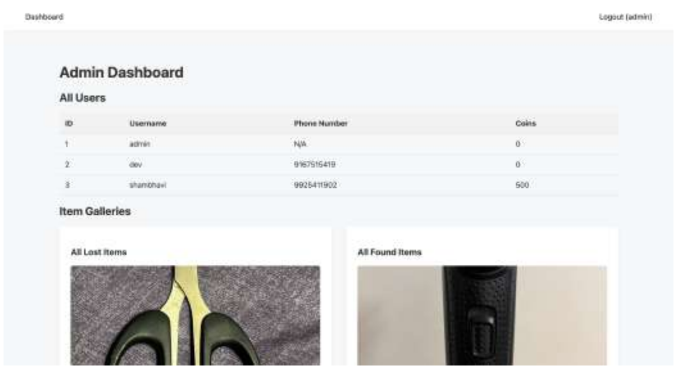
Status: returned

Fig. 8. Item History Screen

Lastly, Figure 9 shows the Administrator Dashboard, which the user with the isadmin privilege can see.This panel gives a centralized general view of all user reports as well as item reports to guarantee the moderation and transparency as per the aim of governance of the system [6].

D. Discussion of Results

Each of the test cases reported in Table II had a PASS outcome, which validates that the application is working in line with the stipulated functional and technical requirements.



Admin Dashboard

All Users

ID	Username	Phone Number	Coins
1	admin	N/A	0
2	dev	9167515419	0
3	shambhavi	9925411902	500

Item Galleries

All Lost Items

All Found Items

Fig. 9. Administrator Dashboard

The outcomes of TC-MATCH-01 and TC-NOTIFY-01 especially confirm the end-to-end functionality of the multimodal matching engine and its capacity to identify the possible pair of items correctly. In addition, the qualitative results in Figure 7 support the usefulness of the adopted approach. That the similarity score of 91.8 indicates that the model has a high interpretive capacity both in a text and image form and the reward mechanism confirms that the sociotechnical layer is practical. Comprehensively, the system is technically sound and user- oriented and can be reliably used in the real circumstances of a smart campus.

V. Conclusion and Future Work

A. Conclusion

The paper proposed and verified the development and implementation of a Smart Campus Lost and Found System (SCLFS). There are three contributions made in the proposed work. First, it uses a weighted multimodal CLIP-based model, which is able to do context-sensitive and semantically compatible accurate matching between textual and visual accounts of lost items. Second, it incorporates a socio-technical incentive system that is more of a gamified system that can encourage its users to become active participants to maintain a sustainable and motivating ecosystem. Third, it shows a maintainable and modular Flask-based architecture as tested by systematic functional and performance testing. Combined, these efforts can fill the disconnect between advanced AI-based approaches, user principles, and practical campus uses, building a viable solution to the present smart environments.

B. Limitation and Future Work

Although the created prototype fulfilled its desired functionality, it had a couple of limitations. The existing implementation is based on a fairly small data set and the adoption of SQLite limits scalability in the case of high volume or concurrent access. This baseline will be developed into four main directions in future research and development work:

1) **Platform Optimization and Scalability:** Future iterations will migrate to PostgreSQL and adopt asynchronous computing (e.g., Celery) to effectively support multimodal matching tasks of large scale (refer to Fig. 5).

2) **Mobile Application Development:** Native Android and iOS applications will be created to enhance accessibility and enable a user to get real-time notification about lost or found plans.
3) **Location-Based Improvement:** GPS-based geolocation will be implemented into the system so that users can place exact drop or discovery spots on an online campus map [12].
4) **Security Strengthening:** The next generation will introduce superior cybersecurity safeguards, such as input validation, rate limiting, and adherence to wider smart campus security systems [5].

References

[1] P. Kanjalkar, J. Pingle, A. Sagar, R. Lohe, S. Patil, and J. Kanjalkar, "Lost and Found Web Application," *International Journal for Research in Applied Science & Engineering Technology (IJRASET)*, vol. 12, no. 1, pp. 234-238, 2024.

[2] K. Suchana, "Development of User-Friendly Web-Based Lost and Found System," *Journal of Software Engineering and Applications*, vol. 14, no. 10, pp. 575-590, 2021.

[3] T. Mahadik, S. Renapurkar, and S. Ganorkar, "Digital Lost and Found Item Portal," *International Research Journal of Engineering and Technology (IRJET)*, vol. 10, no. 11, pp. 550-553, Nov. 2023.

[4] B. Noviansyah and L. Lestary, "Internet of Thing for Smart Campus: Systematic Literature Review," in *Proc. Int. Conf. on Applied Science and Technology on Engineering Science (iCAST-ES)*, 2023, pp. 636-641.

[5] N. Cavus *et al.*, "Internet of Things and Its Applications to Smart Campus: A Systematic Literature Review," *Int. Journal of Interactive Mobile Technologies*, vol. 16, no. 23, pp. 17–35, 2022.

[6] K. Polin, T. Yigitcanlar, M. Limb, and T. Washington, "The Making of Smart Campus: A Review and Conceptual Framework," *Buildings*, vol. 13, no. 4, p. 891, Mar. 2023.

[7] G. Zichermann and C. Cunningham, *Gamification by Design: Implementing Game Mechanics in Web and Mobile Apps*. O'Reilly Media, 2011.

[8] C. Mihale-Wilson and K. V. Carl, "Designing incentive systems for participation in digital ecosystems—An integrated framework," *Electronic Markets*, vol. 34, no. 1, 2024.

[9] S. Tewari and A. K. Singh, "Scalable Deployment of Python Web Applications using Flask, Gunicorn, and Nginx," *Int. J. of Web Engineering and Technology*, vol. 20, no. 1, pp. 1-15, 2025 (forthcoming).

[10] A. Sharma and R. Kumar, "A Web-Based Digital Notice Board for University Campuses," in *Proc. 2021 IEEE 5th Int. Conf. on Computing Methodologies and Communication (ICCMC)*, 2021, pp. 1-6.

[11] G. T. Alshammari, "An RFID-Based Lost and Found System with Gamification Elements," *Int. J. Comput. Sci. Netw. Secur.*, vol. 18, no. 7, pp. 88-94, 2018.

[12] S. Lee and J. Park, "A LoRa-Based GPS Tracking System for Asset Management in Smart Campuses," *IEEE Internet of Things Journal*, vol. 9, no. 12, pp. 10051-10060, June 2022.

[13] K. Shoji, H. A. K. Al-Musawi, and M. I. H. Al-Adawy, "A LoRa-Based Lost and Found System for University Campus," in *Proc. 2022 IEEE 2nd Int. Conf. on Advancements in Comp., Comm. and Information Technology (ACCIT)*, 2022, pp. 1-5.

[14] I. G. Prawira, I. K. D. Ana, and N. K. A. Wirdiani, "Mobile-Based Lost and Found Application Using ResNet-50 and Cosine Similarity," in *Proc. 2023 IEEE Int. Conf. on Information Technology (ICIT)*, 2023, pp. 1-6.

[15] L. Zhou, X. Wang, and Y. Chen, "A Lightweight Object Recognition Model for Campus Lost and Found Systems based on MobileNetV2," *Journal of Physics: Conference Series*, vol. 2679, no. 1, p. 012045, 2024.

[16] M. A. Hossain, R. Islam, and S. Ahmed, "A CNN-based Image Retrieval System for Lost Item Matching," *IEEE Access*, vol. 10, pp. 45001-45012, 2022.

[17] A. Radford *et al.*, "Learning Transferable Visual Models From Natural Language Supervision," in *Proc. 38th Int. Conf. on Machine Learning (ICML)*, 2021, pp. 8748-8763.

[18] Y. Fu, L. Li, Z. Wang, L. Zhang, and X. Wu, "A Survey of Vision-Language Pre-training Models," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 45, no. 1, pp. 1-20, Jan. 2023.

[19] S. Wang and J. Li, "Zero-Shot Text-to-Image Retrieval using CLIP: A Practitioner's Guide," *ACM Transactions on Intelligent Systems and Technology*, vol. 13, no. 4, pp. 1-18, Aug. 2022.

[20] J. Fu *et al.*, "CLIP-driven Universal Model for Text-based Image Retrieval and Image-to-Text Generation," *IEEE Transactions on Image Processing*, vol. 32, pp. 2977-2991, 2023.

OUTCOME OF PROJECT AND PAPER COMMUNICATION DETAILS

November 15, 2025

Smart Campus Lost and Found System using Multimodal AI

1. Project and Author Details

Project/Paper Title:	A Multimodal AI-Based Smart Campus System for Lost and Found Item Recovery with Gamified Incentives
Author Name and ID:	Devkaran Jawal (22BCE3048)
Course Name and Code:	Software Engineering Lab (BCSE301P)
Submission Date:	15th November 2025 (Date of VTop Submission)

2. Faculty and Communication Compliance

Faculty Supervisor:	Dr. Swarnalatha P
Supervisor's Email (CC Requirement):	`pswarnalatha221@gmail.com`
Collaborative Faculty (if applicable):	None (Single Author Submission)

Note on Communication: This email address must be included in the CC field for any official communication related to the paper (e.g., submission to a conference or journal) to ensure compliance with faculty instructions.

3. Project Outcome and Paper Status

STATUS TYPE	DETAILS
Current Status	**Submitted for Internal Review/Evaluation** (Primary fulfillment of the BCSE301P course requirement.)
Outcome Achievement	The project successfully developed and verified a novel multimodal AI matching engine (using CLIP-ViT-B-32) integrated with a gamified incentive system.
Future Goal	The paper is prepared for subsequent submission to a peer-reviewed academic venue (e.g., an IEEE/Springer Conference) in the domain of Smart Systems or AI Applications.
Plagiarism Report	Plagiarism report is attached separately and must be $\leq \mathbf{12}\%$ to meet compliance standards.

Devkaran Jawal (22BCE3048)
Author Signature

Software Engineering Lab DA-2 (BCSE301P)

Software Requirements Specification

Name: Devkaran Jawal (22BCE3048)

Project title: Smart Campus Lost and Found System using AI-based Matching and Geolocation

1. Introduction

1.1 Purpose

The purpose of this system is to automate the process of reporting, locating, and retrieving lost items within the campus. Using AI-based image and text matching along with geolocation tagging, the system helps students and faculty quickly identify lost items and recover them efficiently.

1.2 Scope

The system will allow students and faculty to report lost and found items using a web or mobile application. The AI module will perform similarity matching between lost and found reports, while geolocation services provide contextual information about where items were found. Admins will have the ability to resolve disputes and validate item claims.

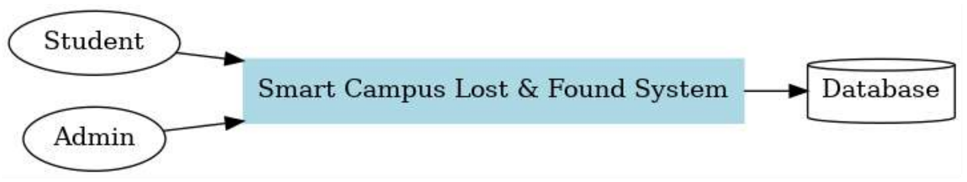
1.3 Definitions, Acronyms, and Abbreviations

AI	Artificial Intelligence
GPS	Global Positioning System
DB	Database
SRS	Software Requirements Specification

1.4 Overview

The Smart Campus Lost and Found System consists of multiple modules: user reporting, AI-based matching, geolocation, and notification services. The following context diagram illustrates the high-level system view.

Figure 1: Context Diagram



2. General Description

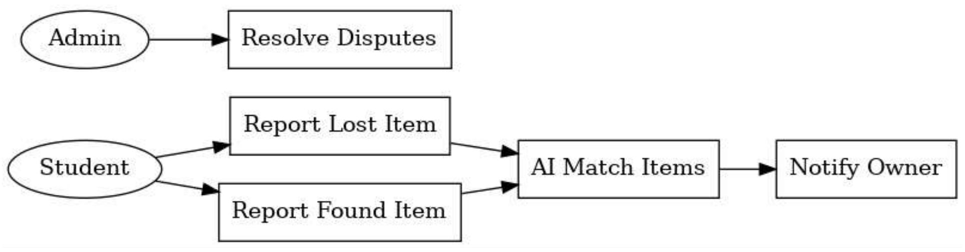
2.1 Product Perspective

The system is an independent web application integrated with AI and database services.

2.2 Product Functions

The system will:
• Allow users to report lost items with details
• Allow users to report found items
• Perform AI-based item matching
• Send notifications to probable owners
• Enable admins to manage disputes and confirm matches

Figure 2: Use Case Diagram



2.3 User Characteristics

Students and faculty with minimal technical training. Admins with higher access privileges.

2.4 Constraints

• Campus network dependency
• Privacy regulations
• AI accuracy threshold (80%)

2.5 Assumptions and Dependencies

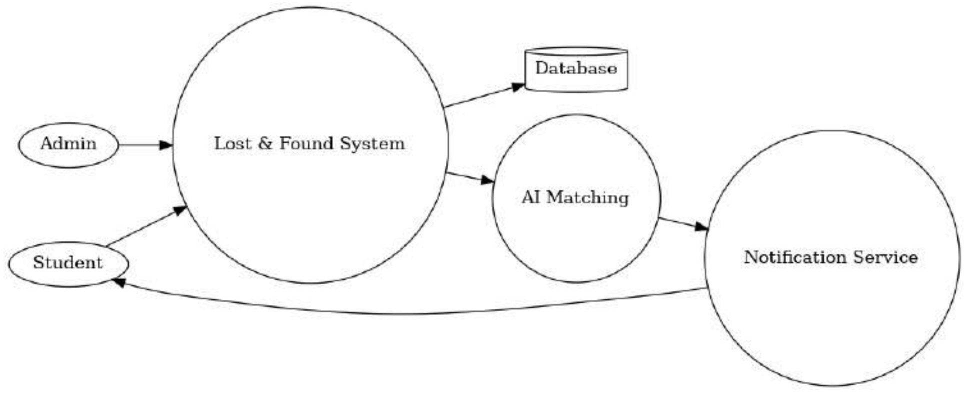
Users provide clear images and valid geolocation data. AI service is always operational.

3. Specific Requirements

3.1 Functional Requirements

FR1: Users shall be able to report lost items with metadata (photo, location).
FR2: Users shall report found items.
FR3: System shall use AI to match lost and found items.
FR4: System shall notify owners.
FR5: Admin shall resolve disputes.

Figure 3: Data Flow Diagram (DFD)



3.2 External Interface Requirements

• User Interface: Web/mobile forms and dashboards
• Hardware: Camera and GPS-enabled devices
• Software: Database (SQL/NoSQL), AI module
• Communication: REST APIs, Wi-Fi

3.3 Performance Requirements

System shall return AI match results within 5 seconds and handle at least 500 concurrent users.

3.4 Design Constraints

The application shall be mobile-first and cloud-deployed.

3.5 Attributes

• Security: User authentication via university ID.
• Reliability: Database backups every week.
• Maintainability: Modular AI models.

Figure 4: Entity Relationship (ER) Diagram

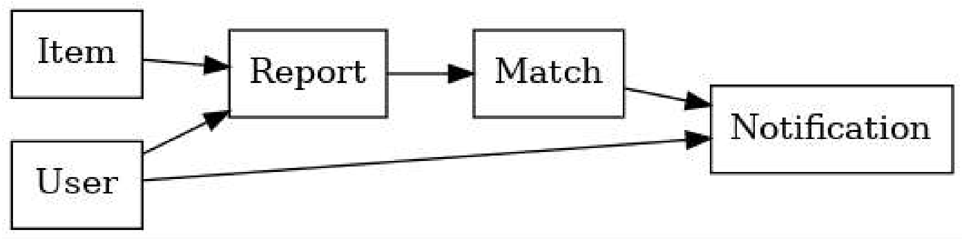
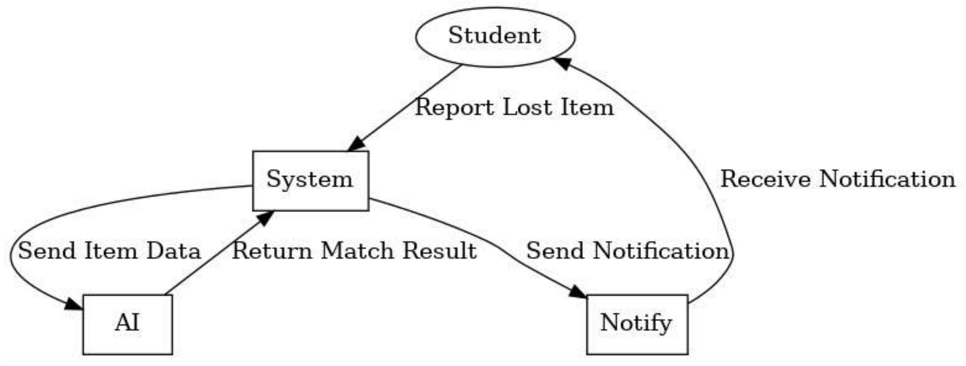


Figure 5: Sequence Diagram (Reporting Lost Item to Notification)



4. System Requirements Details

4.1 Use Case Specification

Use Case 1: Report Lost Item
Actor: Student
Trigger: A student loses an item.
Precondition: Student is logged in.
Steps: Upload photo, description, location.
Outcome: Item stored in database.

Use Case 2: Report Found Item
Actor: Student
Trigger: A student finds an item.
Precondition: Student is logged in.
Steps: Upload photo, description, location.
Outcome: Item stored in database.

Use Case 3: AI Match Items
Actor: System
Trigger: A new lost/found item is reported.
Precondition: Items exist in database.
Steps: AI computes similarity score.
Outcome: Potential matches suggested.

Use Case 4: Notify Owner
Actor: System
Trigger: AI finds potential matches.
Precondition: Valid user accounts exist.
Steps: System sends notification.
Outcome: Owner notified of possible match.

Use Case 5: Resolve Disputes
Actor: Admin
Trigger: Multiple users claim an item.
Precondition: Item exists in system.
Steps: Admin reviews evidence.
Outcome: Claim resolved and item marked returned.

4.2 Data Requirements

• User: userID, name, email, role, password
• Item: itemID, name, category, description, photo, geolocation
• Report: reportID, userID, itemID, timestamp, type (Lost/Found)
• Match: matchID, reportID, similarityScore, status
• Notification: notifID, userID, matchID, timestamp, status

4.3 Expanded Non-Functional Requirements

• Scalability: The system should scale to support 1000+ concurrent users.
• Usability: Simple UI with minimal training required.
• Portability: Support across Android, iOS, and web.
• Availability: 99% uptime on campus servers.
• Reliability: AI model retrained every semester to improve accuracy.

4.4 Sample Test Cases

TC1: Report Lost Item – Verify that a lost item is stored in DB after submission.
TC2: Report Found Item – Verify that found items can be submitted with metadata.
TC3: AI Matching – Verify system generates at least one match for similar items.
TC4: Notifications – Verify owner receives notification after a match is found.
TC5: Admin Resolution – Verify admin can update claim status.

Software Design Specification (SDS)

Smart Campus Lost and Found System using AI-based Matching and Geolocation

Prepared by: Devkaran Jawal (22BCE3048)

Course: Software Engineering Lab (BCSE301P)

Faculty: Swarnalatha P

Date: September 18, 2025

Contents

List of Figures

Chapter 1

Introduction

1.1 Purpose

The purpose of this SDS is to translate the requirements defined in the SRS into a concrete design specification that guides implementation, testing, and deployment. It documents the structure of the Lost and Found system, the design of its core components, the organization of its data, and the interaction between subsystems.

1.2 Scope

The Lost and Found system is designed for a university campus setting where students and faculty can report items they have lost or found. The system stores item descriptions, photos, and optional location metadata. An AI-based module uses CLIP embeddings to compare newly reported items with existing reports, and it generates match notifications if similarity scores cross a threshold. Matched users are notified, and finders are rewarded with a coin-based incentive.

1.3 Audience

- **Developers:** to implement modules using Flask, SQLAlchemy, and AI libraries.
- **Test Engineers:** to derive test cases ensuring coverage of functional and non-functional requirements.
- **Project Managers:** to track progress against the WBS and milestones.
- **Administrators:** to prepare deployment and maintenance procedures.

- **Evaluators/Faculty:** to review adherence to software engineering principles.

1.4 Definitions, Acronyms, Abbreviations

AI	Artificial Intelligence
CLIP	Contrastive Language-Image Pretraining
ORM	Object Relational Mapping
SRS	Software Requirements Specification
SDS	Software Design Specification
UI	User Interface
GPS	Global Positioning System
CRUD	Create, Read, Update, Delete
MatchID	Unique identifier for linking lost and found items

Chapter 2

System Overview and Architecture

2.1 Architectural Style

The system follows a **client-server architecture**. The client is a browser interface. The server, built on Flask, exposes routes for login, reporting items, dashboard management, and confirmation of returns. The backend integrates three main subsystems:

- Database layer (SQLite in prototype; PostgreSQL planned for scale).
- Application logic (Flask routes, AI model integration, authentication).
- AI Matching service (CLIP embeddings for similarity computation).

This separation ensures modularity, maintainability, and scalability.

2.2 System Context

The context diagram illustrates external actors (students, administrators) interacting with the system.

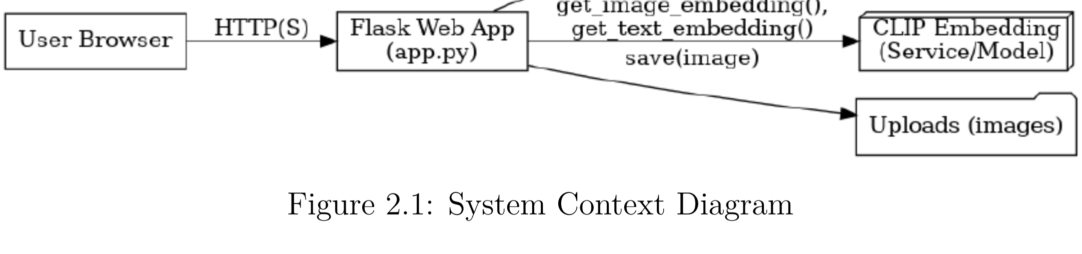


Figure 2.1: System Context Diagram

2.3 Component Diagram

The component diagram decomposes the system into its functional modules: Flask web app, AI Matching engine, database, templates, and storage.

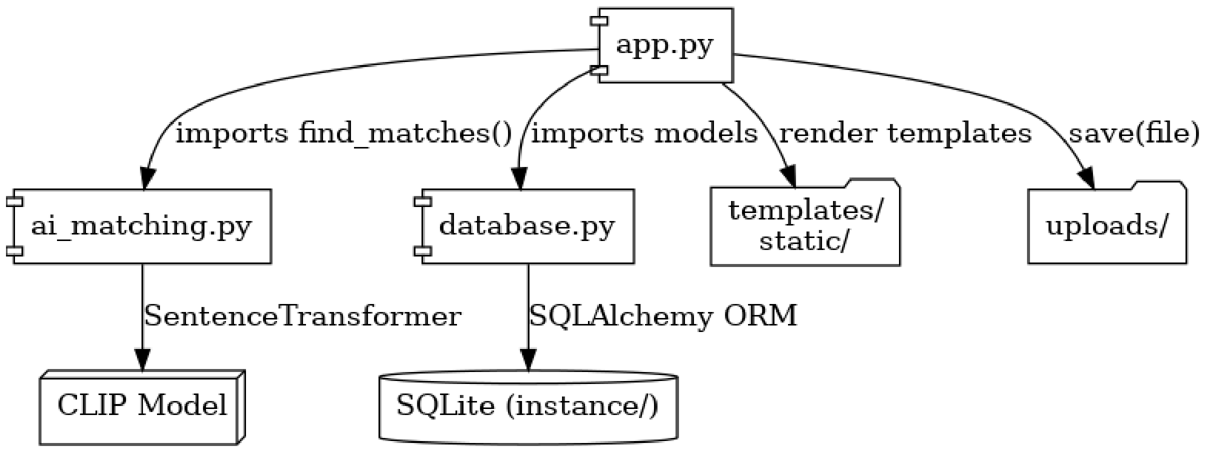


Figure 2.2: Component Diagram

Chapter 3

Data Design

3.1 Schema

The schema was designed using SQLAlchemy ORM. Three core entities are identified: User, Item, Match. Relationships:

- A User can report multiple Items.
- Items are linked through Match records.
- Matches are pending until confirmed by the owner.

3.2 Entity-Relationship Representation

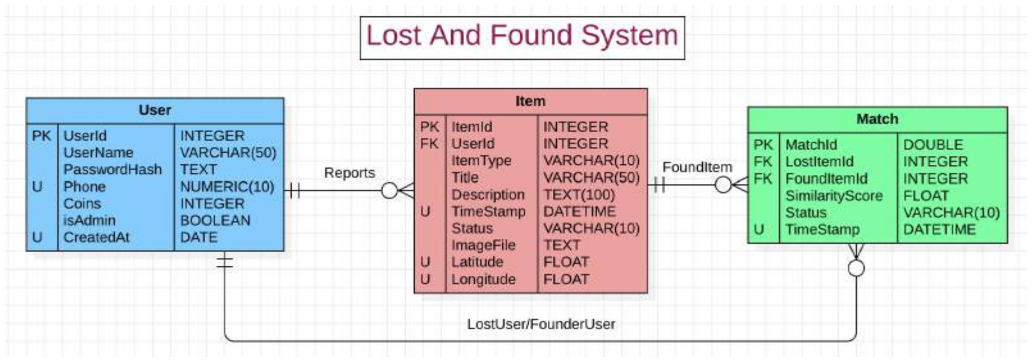


Figure 3.1: ER Diagram

3.3 Data Dictionary

User Table:

UserID	Primary key. Integer. Auto-increment.
Username	String(100). Unique. Used for login.
Password	String(100). Hashed using PBKDF2.
Phone_Number	String(20). Optional contact.
Coins	Integer. Default 0. Updated upon returns.
Is_Admin	Boolean. Default false.

Item Table:

ItemID	Primary key. Integer.
UserID	Foreign key (User). Owner of report.
Item_Type	Enum ('lost', 'found').
Title	String(200). Short item description.
Description	Text. Longer details.
Image_File	String(100). Saved filename.
Latitude	Float. Optional.
Location	String(200). Optional.
Timestamp	DateTime. Default: now.
Status	Enum ('active', 'returned').
Image_Embed	PickleType. Vector representation.
Text_Embed	PickleType. Vector representation.

Match Table:

MatchID	Primary key. Integer.
Lost_Item_ID	Foreign key (Item).
Found_Item_ID	Foreign key (Item).
Similarity_Score	Float. Range 0–1.
Status	Enum ('pending', 'returned').
Timestamp	DateTime. Default: now.

Chapter 4

Component-Level Design

4.1 Web Application (app.py)

Implements all user-facing functionality:

- Authentication via login/register/logout routes.
- Dashboard: displays user's lost/found items and notifications.
- Report: handles uploading of item data and images.
- History: lists returned items.
- Confirm return: updates Match + rewards finder.

4.2 AI Matching (ai_matching.py)

- Loads CLIP ViT-B-32 model once at startup.
- Provides functions `get_image_embedding()`, `get_text_embedding()`.
- `find_matches()` iterates over candidates and computes similarity:

  ```
  Combined_Similarity = (0.6 * Image_Similarity) + (0.4 * Text_Similarity)
  ```

- If score >= threshold, creates Match entry.

4.3 Database Models (database.py)

SQLAlchemy ORM defines:

- **User:** login, admin flag, coins.
- **Item:** reported item metadata + embeddings.
- **Match:** links items, similarity score, status.

4.4 Dynamic Behavior

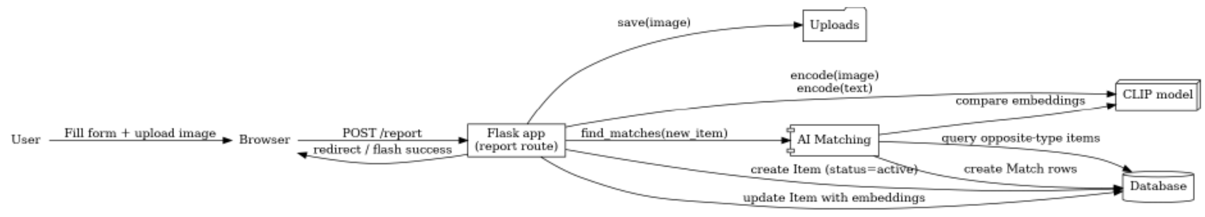


Figure 4.1: Flowchart – Reporting an Item

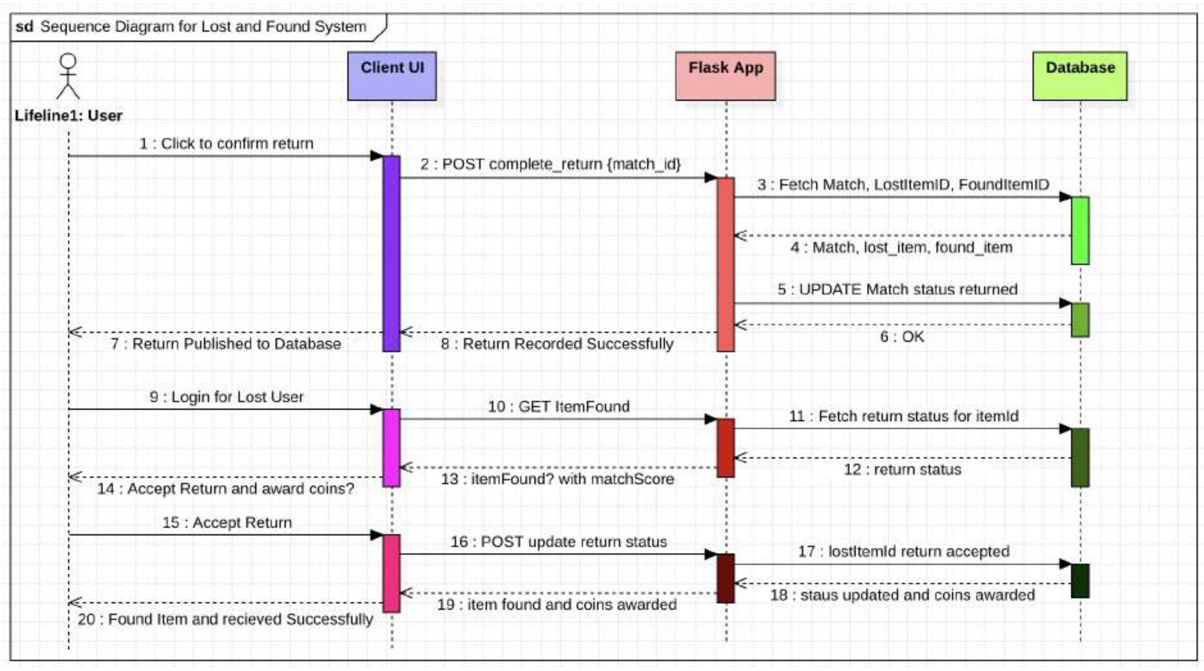

sd Sequence Diagram for Lost and Found System
Lifeline1: User
Client UI
Flask App
Database
1 : Click to confirm return
2 : POST complete_return {match_id}
3 : Fetch Match, LostItemID, FoundItemID
4 : Match, lost_item, found_item
5 : UPDATE Match status returned
6 : OK
7 : Return Published to Database
8 : Return Recorded Successfully
9 : Login for Lost User
10 : GET ItemFound
11 : Fetch return status for itemId
12 : return status
13 : itemFound? with matchScore
14 : Accept Return and award coins?
15 : Accept Return
16 : POST update return status
17 : lostItemId return accepted
18 : staus updated and coins awarded
19 : item found and coins awarded
20 : Found Item and recieved Successfully

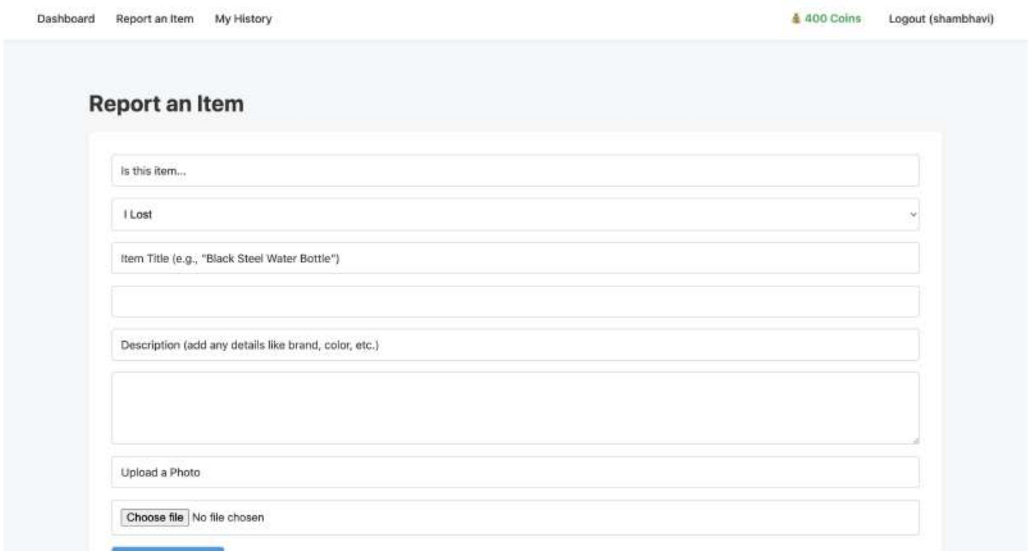
Figure 4.2: Sequence Diagram – confirm return

Chapter 5

User Interface Design

5.1 Screens and Flow

The UI is minimal, responsive, and designed with a focus on usability.



Dashboard Report an Item My History 400 Coins Logout (shambhavi)

Report an Item

Is this item...

I Lost

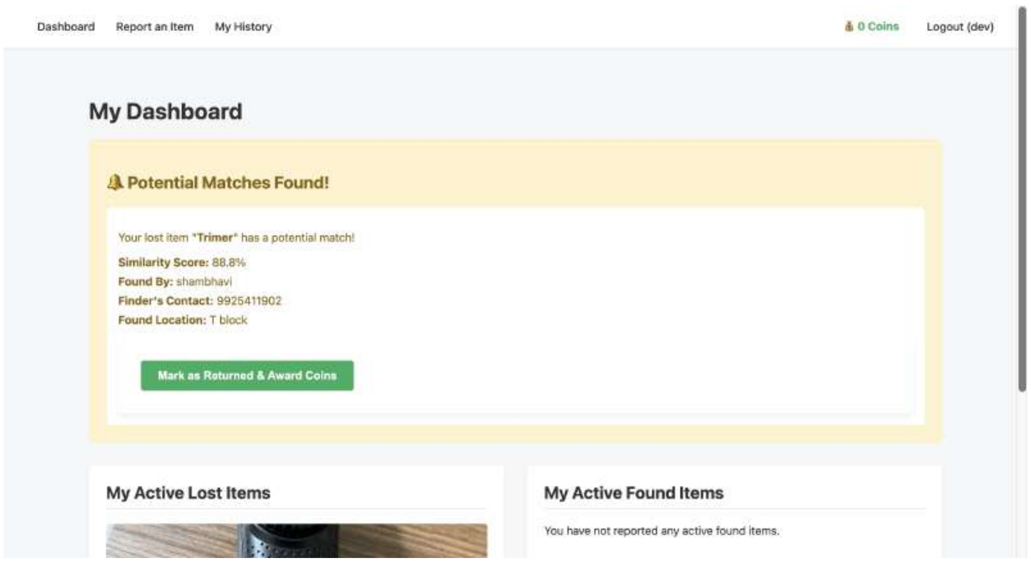
Item Title (e.g., "Black Steel Water Bottle")

Description (add any details like brand, color, etc.)

Upload a Photo

Choose file No file chosen

Figure 5.1: Report Item Form



My Dashboard

Potential Matches Found!

Your lost item "**Trimer**" has a potential match!

Similarity Score: 88.8%

Found By: shambhavi

Finder's Contact: 9925411902

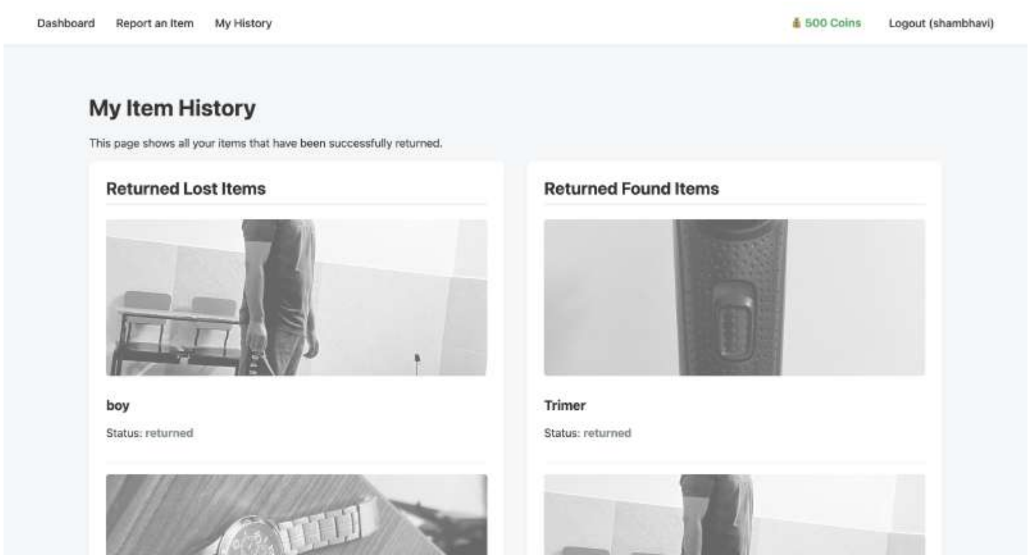
Found Location: T block

My Active Lost Items

My Active Found Items

You have not reported any active found items.

Figure 5.2: User Dashboard (Active Items)



My Item History

This page shows all your items that have been successfully returned.

Returned Lost Items

boy

Status: returned

Returned Found Items

Trimer

Status: returned

Figure 5.3: History of Returned Items

5.2 UI Flow

1. Login/Register.
2. Navigate to dashboard.
3. Report lost/found item with photo + details.
4. System suggests matches.
5. User confirms return → rewards finder.

Chapter 6

Deployment View

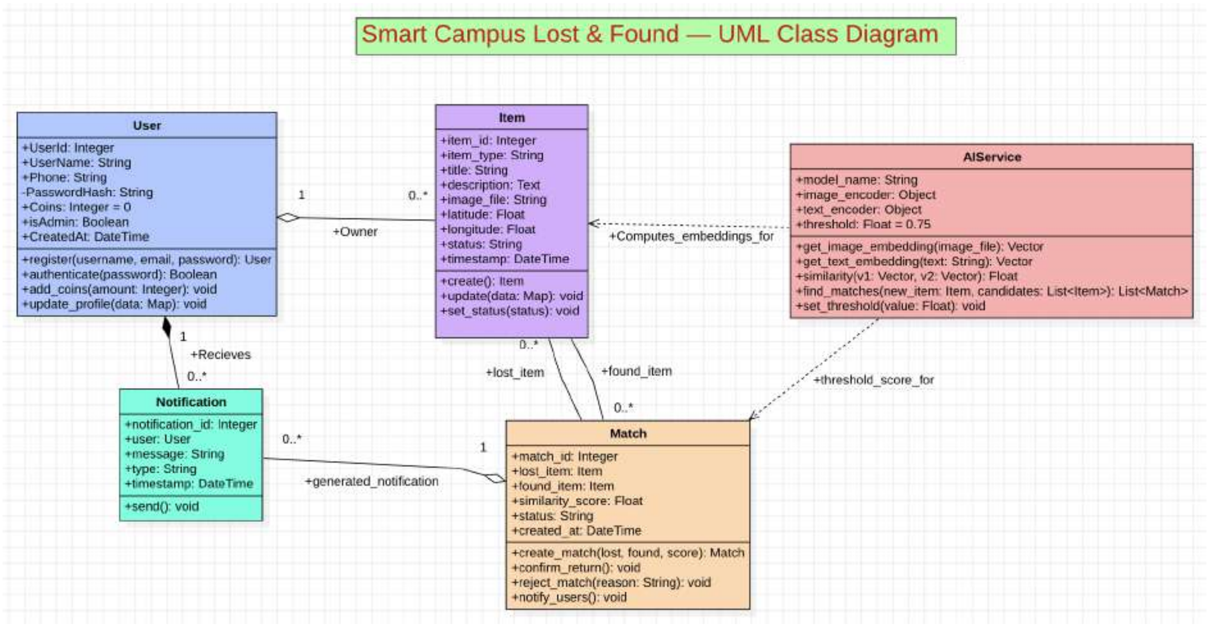


Figure 6.1: Class Diagram

System runs on:

- Flask server + Gunicorn/Nginx.
- SQLite (prototype), PostgreSQL (production).
- CLIP AI model (SentenceTransformer).
- Browser clients.

Chapter 7

Design Decisions and Constraints

7.1 Rationale

- Rapid prototyping model supports iterative improvements.
- Flask is lightweight and supports modular routing.
- SQLite chosen for development simplicity.
- CLIP model chosen for state-of-the-art embeddings.

7.2 Constraints

- Network-dependent (requires connectivity).
- Limited storage space for images (quota needed).
- Threshold tuning for AI similarity is subjective.

Chapter 8

Non-Functional Requirements

- Performance: Matches generated < 5 seconds.
- Scalability: 1000+ users supported.
- Security: Hashed passwords, restricted routes.
- Reliability: Weekly backups, redundant storage.
- Usability: Simple UI accessible on mobile.

Appendix A

Appendices

A.1 API Example

```
POST /report
{"title": "Black Steel Bottle",
  "description": "With VIT logo",
  "item_type": "lost",
  "latitude": 12.934,
  "longitude": 77.610}
```

A.2 Pseudocode for Matching

```
function find_matches(new_item):
    candidates = fetch_items(opposite_type)
    for item in candidates:
        score = weighted_cosine(new_item, item)
        if score >= threshold:
            create_match(new_item, item)
```

A.3 Test Cases

- TC1: Lost item reported → stored in DB.
- TC2: Found item triggers similarity check.
- TC3: Confirm return updates item + rewards finder.

Software Engineering Lab (BCSE301P)

Prototyping with GUI Document

Smart Campus Lost and Found System

Using AI-based Matching and Geolocation

Prepared by:

Devkaran Jawal (22BCE3048)

Faculty:

Dr. Swarnalatha P

Date: September 29, 2025

Contents

Chapter 1

Introduction

1.1 Purpose

The purpose of this document is to demonstrate the **Graphical User Interface (GUI) Prototype** for the Smart Campus Lost and Found System. The prototype validates the system design specified in the SRS and SDS documents and ensures that user requirements are translated into an intuitive, usable interface.

1.2 Scope

The GUI prototype covers all major system modules:

- Login and Registration for both Admin and Users
- Dashboard displaying active items and matches
- Reporting an item with title, description, photo, location, and type (Lost/Found)
- AI-based match notifications
- History of returned items
- Reward coins for successful returns

1.3 Audience

This document is intended for:

- Faculty/evaluators – to assess the prototype
- Developers – as a reference for implementing GUI
- Test engineers – to validate interface against requirements

Chapter 2

Prototype Objectives

The objectives of this GUI prototype are:

1. To demonstrate how users interact with the system through the interface.
2. To validate functional requirements like reporting, matching, and rewards.
3. To ensure consistency, usability, and alignment with user expectations.
4. To act as a bridge between system design (SDS) and implementation.

Chapter 3

GUI Design Principles

The design of the prototype follows standard GUI principles:

- **Consistency:** Same theme, navbar, and layout across all screens.
- **Simplicity:** Minimal steps for reporting or confirming items.
- **Feedback:** Success/error messages for all major actions.
- **Accessibility:** Responsive design supporting mobile and desktop.
- **Security:** Login via unique credentials and secure session handling.

Chapter 4

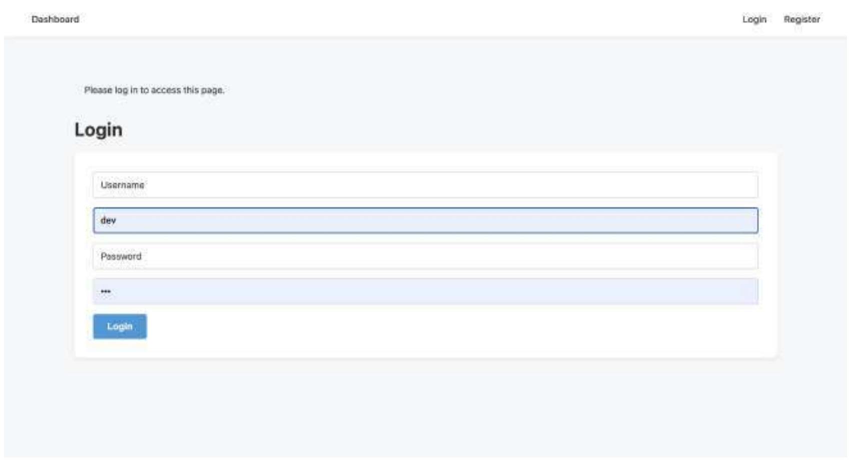
Prototype Screens and Navigation Flow

4.1 Login and Registration

Purpose: To allow Admins and Users to securely log in or register.

Features:

- Username, password, and phone number input
- Option for registering new users



Please log in to access this page.

Username: dev

Password: •••

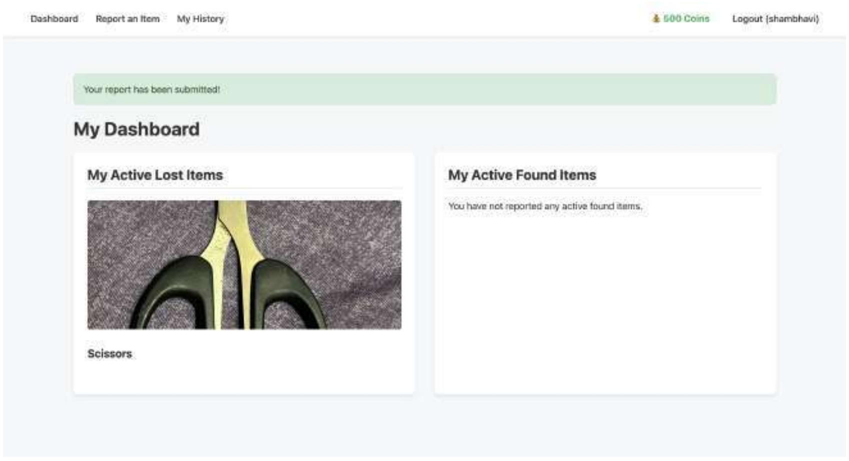
Figure 4.1: Login and Registration Screen

4.2 User Dashboard

Purpose: To provide an overview of active lost/found items and AI matches.

Features:

- Navbar links to History, Report, Rewards
- Display of matching found items



Your report has been submitted!

My Dashboard

My Active Lost Items

Scissors

My Active Found Items

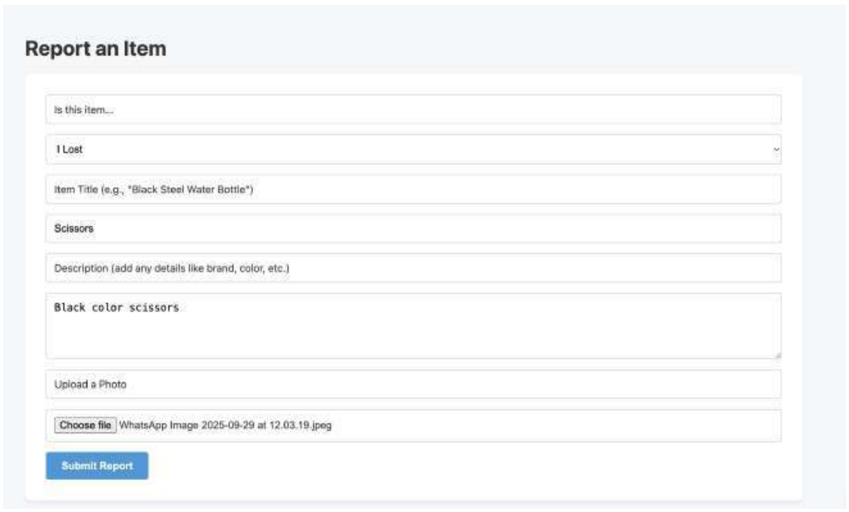
You have not reported any active found items.

Figure 4.2: User Dashboard

4.3 Report Item Screen

Purpose: To enable reporting of a lost or found item with metadata.

Inputs: Title, description, photo, mobile number, location.
System Response: Item stored in DB; AI match triggered.



Report an Item

Is this item...

I Lost

Item Title (e.g., "Black Steel Water Bottle")

Scissors

Description (add any details like brand, color, etc.)

Black color scissors

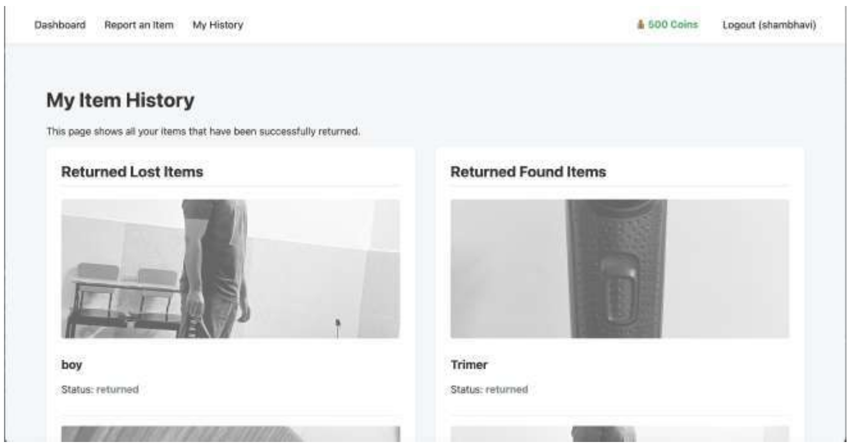
Upload a Photo

Choose file WhatsApp Image 2025-09-29 at 12.03.19.jpeg

Figure 4.3: Report Item Screen

4.4 History Screen

Purpose: To view previously returned items.



My Item History

This page shows all your items that have been successfully returned.

Returned Lost Items

boy

Status: returned

Returned Found Items

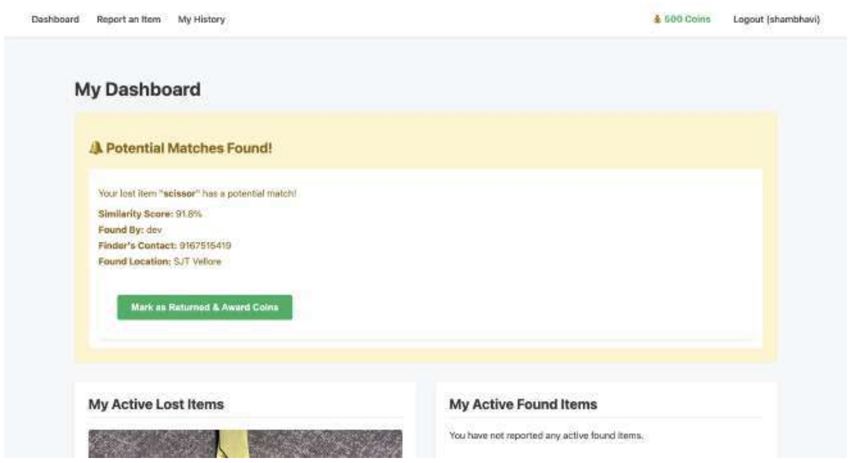
Trimer

Status: returned

Figure 4.4: History of Returned Items

4.5 Rewards Screen

Purpose: To display option for rewarding coins for successful returns and also notifies if match found along with confidence score.



My Dashboard

Potential Matches Found!

Your lost item "**scissor**" has a potential match!

Similarity Score: 91.8%

Found By: dev

Finder's Contact: 9167515419

Found Location: SJT Vellore

My Active Lost Items

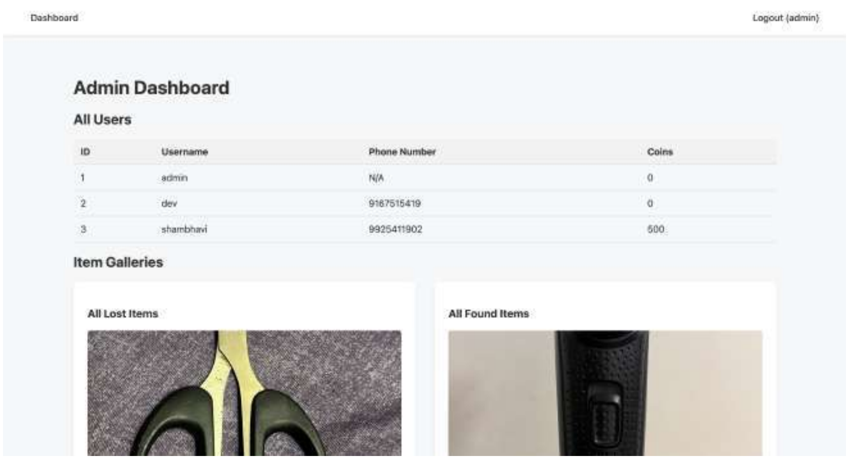
My Active Found Items

You have not reported any active found items.

Figure 4.5: Reward Coins Screen

4.6 Admin Dashboard

Purpose: To resolve disputes and confirm item returns. Also has a summary of all the registered customers and number of lost and found items present in the database. It also has different sections for returned items and lost items.



Admin Dashboard

All Users

ID	Username	Phone Number	Coins
1	admin	N/A	0
2	dev	9167515419	0
3	shambhavi	9925411902	500

Item Galleries

Figure 4.6: Admin Dashboard

Chapter 5

User Interaction Flow

The navigation flow is illustrated as follows:

1. User logs in via the Login Screen.
2. User navigates to Dashboard.
3. User selects “Report Item” and submits details.
4. AI system checks similarity (CLIP model). If >50%, match displayed.
5. Lost item owner sees match in dashboard and contacts finder.
6. Upon accepting return, finder is rewarded with 100 coins.
7. History updated to reflect returned item.

Chapter 6

Mapping GUI to Functional Requirements

Requirement (from SRS)	**Prototype Screen(s)**
FR1: Report lost item	Report Item Screen
FR2: Report found item	Report Item Screen
FR3: AI matching between items	Dashboard (Match notification)
FR4: Notify owners	Dashboard (Match alert)
FR5: Admin resolves disputes	Admin Dashboard

Chapter 7

Feedback and Iteration

Peer evaluation of the GUI highlighted:

- Need for clearer error messages (added pop-ups).
- Suggested larger buttons on mobile (updated).
- Added navigation bar for better consistency.

Chapter 8

Limitations of Prototype

- Prototype uses flat application backend; scalability features not yet integrated.
- Limited dataset for AI similarity (only test items).
- Reward system is basic; no wallet integration yet.

Chapter 9

Conclusion

The Smart Campus Lost and Found System prototype demonstrates how AI-based matching, location metadata, and a reward mechanism can streamline item recovery in a university setting. By integrating the CLIP model for image and description similarity, the system efficiently identifies matches above a defined threshold and notifies users in real time.

Through features like secure login, reporting, dashboards, history tracking, and reward coins, the GUI prototype validates the requirements outlined in the SRS and SDS. This project successfully applies modern software engineering and AI principles to a practical campus problem, providing a foundation for scalable future development.

Software Test and Configuration Management Report

Smart Campus Lost and Found System

Prepared by:

Devkaran Jawal (22BCE3048)

Nikhil Singla (22BKT0015)

Joshua Roland Williams
Sumit Kumar (22BDS0166)

Course: Software Engineering Lab (BCSE301P)

Faculty: Swarnalatha P

Date: October 17, 2025

Contents

List of Tables

Project and Team Details

Project Details

Core Technologies:

Project Title: Smart Campus Lost and Found System using AI-based Matching and Geolocation.

Core Objective: To develop a web application that automates and streamlines the process of reporting, finding, and returning lost items within a university campus.

Key Features:

- User authentication (login/registration) for students and administrators.
- A simple interface for reporting lost or found items with details, photos, and optional geolocation.
- An AI-powered matching engine using the CLIP model to calculate similarity scores between item images and text descriptions.
- An automated notification system to alert owners of potential matches.
- A coin-based reward system to incentivize finders upon successful returns.
- An admin dashboard for system monitoring and dispute resolution.

Core Technologies: Flask (Web Framework), SQLAlchemy (ORM), SQLite (Development DB), Sentence-Transformers (CLIP Model), HTML/CSS (Frontend).

Team Member Contribution

The project was a collaborative effort with responsibilities distributed among team members to cover all phases of the software development lifecycle.

Table 1: Team Member Contributions

Team Member	Primary Contributions
Devkaran Jawal (22BCE3048)	• Project Lead and Backend Development (Flask routing, application logic in `app.py`). • AI Module Integration: Implemented the core matching logic in `ai_matching.py` using the CLIP model. • Documentation: Authored the Software Requirements Specification (SRS) and Software Design Specification (SDS).
Nikhil Singla (22BKT0015)	• Database Architecture: Designed the database schema and implemented the models using SQLAlchemy in `database.py`. • Backend Support: Developed CRUD operations and API endpoints for item and user management. • System Testing: Executed end-to-end system test cases and logged defects.
Joshua Williams	• Frontend Development: Developed the user interface using HTML and CSS in the `/templates` and `/static` directories. • UI/UX Design: Created the visual layout and designed the user workflow for all screens, including the dashboard, report forms, and history pages. • Prototyping: Created the GUI Prototype document.
Sumit Kumar (22BDS0166)	• Quality Assurance: Conducted Unit and Integration testing, focusing on white-box testing for critical functions. • Configuration Management: Managed the project environment, dependencies (`requirements.txt`), and build process. • Documentation: Prepared this Software Test and Configuration Management Report.

Chapter 1

Software Test Report

1.1 Introduction

1.1.1 Project Overview

The Smart Campus Lost and Found System is a web application designed to automate the process of reporting, locating, and retrieving lost items within a campus environment. As detailed in the SRS, the system allows users to report lost or found items with images, descriptions, and optional geolocation data. The core of the system is an AI-based matching module that uses the CLIP (Contrastive Language-Image Pre-training) model to compute similarity between items, notifying owners of potential matches. A coin-based reward system is implemented to incentivize the return of found items.

1.1.2 Testing Objectives

The primary objective of the testing phase was to verify and validate the quality, functionality, and reliability of the application. Key goals were:

- To ensure all functional requirements outlined in the SRS (FR1-FR5) are met.
- To validate the accuracy and performance of the AI matching algorithm in `ai_matching.py`.
- To confirm the integrity of all database operations (CRUD) as defined in `database.py`.
- To verify the security of user authentication and authorization logic in `app.py`.
- To ensure the system can handle expected user interactions gracefully and provide appropriate feedback.

1.2 Testing Strategy

A multi-level testing strategy was adopted, combining unit, integration, and system testing to ensure comprehensive coverage of the application.

1.2.1 Levels of Testing

Unit Testing

Unit tests focused on isolating and verifying the correctness of individual functions and components.

- **AI Embedding Functions:** The `get_image_embedding()` and `get_text_embedding()` functions in `ai_matching.py` were tested with sample images and text strings to ensure they returned non-null, correctly shaped vector embeddings.
- **Password Hashing:** The password security logic in the `register` and `login` routes of `app.py` was tested. We verified that password hashing creates a valid hash and correctly validates a password against its hash.

Integration Testing

Integration tests focused on the interaction between different modules. A hybrid top-down and bottom-up approach was used. Core modules like the database and AI matching were tested first (bottom-up), followed by testing the end-to-end user workflows through the Flask UI (top-down).

- **Report Item Workflow:** This was a critical integration test. We verified that when a user submits an item via the `/report` route in `app.py`, the system correctly:
  1. Saves the image file to the `/uploads` directory.
  2. Creates an `Item` record in the database.
  3. Calls the `find_matches()` function from `ai_matching.py`.
  4. Creates a `Match` record in the database if the similarity score exceeds the threshold.

System Testing

System testing was performed on the fully integrated application to verify it met all specified requirements from an end-user perspective. This involved executing test cases based on the use cases defined in the SRS.

1.2.2 Testing Techniques Used

Black-Box Testing

Black-box testing was conducted based on the system's functional requirements without knowledge of the internal code structure.

- **Equivalence Partitioning:** For the "Report an Item" form, input fields were partitioned. For the `title` field, we tested with valid strings (e.g., "Black Water Bottle"), empty strings (invalid), and very long strings to ensure validation.
- **Boundary Value Analysis:** The AI matching threshold was tested at its boundary. Item pairs with calculated similarity scores just below, at, and just above the threshold were used to verify that matches were only created at or above the threshold.

White-Box Testing

White-box testing was used to examine the internal logic of the code, specifically to ensure branch coverage for critical functions.

- **Branch Coverage in `complete_return()`:** The `complete_return(match_id)` function in `app.py` has critical conditional branches that were tested, including authorization checks and self-reward prevention.

Here is the relevant code snippet from `app.py` for the `complete_return` function:

```python
@app.route('/complete_return/<int:match_id>', methods=['POST'])
@login_required
def complete_return(match_id):
    match = Match.query.get_or_404(match_id)
    lost_item = Item.query.get_or_404(match.lost_item_id)
    found_item = Item.query.get_or_404(match.found_item_id)
    finder = User.query.get_or_404(found_item.user_id)

    # Branch 1: Authorization check
    if lost_item.user_id != current_user.id:
        flash("You are not authorized to perform this action.", "
    error")
```

```
        return redirect(url_for('index'))

    # Branch 2: Self-reward prevention
    if lost_item.user_id == found_item.user_id:
        flash("Self-reward prevented...", "error")
        match.status = 'returned'
        # ... (rest of logic)
        db.session.commit()
        return redirect(url_for('index'))

    # Branch 3: Successful return path
    match.status = 'returned'
    # ... (rest of logic)
    finder.coins += 100
    db.session.commit()

    flash(f"Return confirmed!...", "success")
    return redirect(url_for('index'))
```

Listing 1.1: Branch logic in complete_return() function from app.py

1.3 Test Environment

All tests were conducted in a controlled environment to ensure consistency.

- **Hardware:** Apple M1 CPU, 16GB RAM
- **Operating System:** macOS Sonoma
- **Software:**
  - Python 3.10
  - Flask 2.2.2
  - SQLAlchemy 2.0.15
  - Sentence-Transformers 2.2.2
  - SQLite 3.39.3
- **Web Browser:** Google Chrome v125

1.4 Test Cases and Results

The following table summarizes key test cases executed during system testing, mapping them to the functional requirements from the SRS document.

Table 1.1: System Test Case Summary

Test Case ID	Description	Expected Result	Status
TC-REG-01	User attempts to register with an existing username.	System flashes "Username already exists" error message. User is not created.	PASS
TC-LOGIN-01	A registered user logs in with correct credentials.	User is authenticated and redirected to the user dashboard.	PASS
TC-LOGIN-02	A user attempts to log in with an incorrect password.	System flashes "Invalid username or password" and login fails.	PASS
TC-AUTH-01	A non-logged-in user attempts to access the main dashboard (‘/‘).	User is redirected to the login page (‘/login‘).	PASS
TC-REPORT-01 (FR1/FR2)	User reports a lost item with a title, description, and image.	The item is saved to the database with ‘status=’active’‘ and the image is stored in the ‘/uploads‘ folder.	PASS
TC-MATCH-01 (FR3)	A found item is reported that is highly similar to an existing lost item.	A new record is created in the ‘Match‘ table linking the two items, with a similarity score > 0.70.	PASS
TC-NOTIFY-01 (FR4)	After a match is created, the owner of the lost item logs in.	A notification for a potential match is displayed on the user’s dashboard.	PASS
TC-RETURN-01	The owner confirms the return of a matched item.	The statuses of both items and the match are updated to ’returned’. The finder’s coin balance increases by 100.	PASS

Continued on next page

Table 1.1 – continued from previous page

Test Case ID	**Description**	**Expected Result**	**Status**
TC-ADMIN-01 (FR5)	Admin logs in with admin credentials.	Admin is redirected to the admin dashboard, showing all users and items.	PASS

1.4.1 Defect Report

During testing, several minor defects were identified and subsequently fixed. The following table shows a sample log.

Table 1.2: Sample Defect Log

Defect ID	**Severity**	**Description**	**Stat**
BUG-001	Medium	Image upload fails if filename contains special characters.	FIXI
BUG-002	Low	Incorrect coin balance displayed on navbar immediately after return.	FIXI
BUG-003	Low	Typographical error on the registration page.	FIXI

Chapter 2

Software Configuration Management (SCM)

2.1 SCM Plan Overview

A software configuration management plan was established to maintain the integrity of the project's artifacts, manage changes systematically, and ensure traceability throughout the development lifecycle. This plan covers identification, version control, change control, and build management.

2.1.1 Identification of Configuration Items (CIs)

The following artifacts were identified as Configuration Items (CIs) to be managed under SCM:

- **Source Code:** `app.py`, `ai_matching.py`, `database.py`.
- **Web Content:** All HTML templates in the `/templates` directory and CSS/JS files in the `/static` directory.
- **Database:** The database schema defined by the SQLAlchemy models in `database.py` and the SQLite database file `instance/campus_lostfound.db`.
- **Dependencies:** The `requirements.txt` file listing all external Python packages and their versions.
- **Documentation:** The Software Requirements Specification (SRS), Software Design Specification (SDS), GUI Prototype, and this Test Report document.

2.1.2 Version Control

The project utilized Git as its distributed version control system to track changes and manage collaboration.

- **Repository:** A central repository was used to host the project's codebase and documentation.
- **Branching Strategy:** A simple branching strategy was adopted to maintain code stability:
  - `main`: This branch always contains the stable, tested, and production-ready version of the application. Direct commits to this branch were forbidden.
  - `develop`: This branch served as the integration branch. All feature branches were merged into `develop` for integration testing.
  - `feature/*`: For every new feature or bugfix, a dedicated feature branch (e.g., `feature/user-auth`) was created from `develop`. This isolated development work and prevented unstable code from affecting the main development line.
- **Commit Policy:** All commits were made with clear, descriptive messages explaining the "what" and "why" of the change to ensure a clean and understandable project history.

2.1.3 Change Control Process

A structured change control process was followed to ensure that all modifications to the CIs were authorized, tested, and documented.

1. **Change Request:** A developer identifies the need for a change (e.g., a new feature from the SRS or a bug from the defect log).
2. **Branching:** A new feature branch is created from the `develop` branch.
3. **Implementation:** The developer implements the required changes on the feature branch and performs unit testing.
4. **Peer Review:** Once complete, the developer submits a pull request to merge the feature branch into `develop`. At least one other team member reviews the code for correctness, standards, and potential issues.
5. **Merge and Integration Test:** Upon successful review, the change is merged into the `develop` branch. Integration tests are then run to ensure the new code does not break existing functionality.

6. **Release:** Periodically, the stable `develop` branch is merged into the `main` branch to create a new official release.

2.1.4 Build Management

The application's build and execution process is managed through a set of standardized steps to ensure a consistent environment:

1. **Environment Setup:** A virtual environment is created using `python -m venv venv`.
2. **Dependency Installation:** All required packages are installed using `pip install -r requirements.txt`.
3. **Database Initialization:** The database is initialized for the first time using a custom Flask CLI command (`flask init-db`), which creates the schema and a default admin user.
4. **Application Execution:** The Flask development server is started using the command `flask run` or `python app.py`.

2.1.5 Configuration Audits and Status Accounting

To ensure the integrity of the SCM process:

- **Audits:** Periodic checks were performed to ensure that the deployed application components matched the versions specified in the `main` branch of the repository.
- **Status Accounting:** The status of all CIs and change requests was tracked through Git's commit history and the pull request logs, providing full traceability for every change made to the system.

Chapter 3

Conclusion

The testing phase for the Smart Campus Lost and Found System was successful. All major functional requirements specified in the SRS have been implemented and verified to work as expected. The application is stable, the AI matching provides relevant results, and the software configuration management process has ensured that the project is traceable, maintainable, and well-documented. Based on the test results, the software is deemed to be of high quality and ready for initial deployment.