CAMPUS LOST AND FOUND WEB APPLICATION

A MICRO PROJECT REPORT

Submitted by

AHMED NAJAD PK (AWH24CS006)

AMANA KK (AWH24CS008)

ARSHAL KS (AWH24CS019)

HRIDYA DS (AWH24CS048)

MUHAMMAD DILSHAD M (AWH24CS062)

MUHAMMED FOUZAN PP (AWH24CS064)

NAHALA MP (AWH24CS073)

to

the APJ Abdul Kalam Technology University

in partial fulfilment of the requirements for the award of the

degree of

Bachelor of Technology In

Computer Science and Engineering

Departmentof ComputerScience and Engineering AWH College

of Engineering

Kuttikattoor,Kozhikode-673008 OCTOBER 2025

CAMPUS LOST AND FOUND WEB APPLICATION

A MICRO PROJECT REPORT

Submitted by

AHMED NAJAD PK (AWH24CS006)

AMANA KK (AWH24CS008)

ARSHAL KS (AWH24CS019)

HRIDYA DS (AWH24CS048)

MUHAMMAD DILSHAD M (AWH24CS062)

MUHAMMED FOUZAN PP (AWH24CS064)

NAHALA MP (AWH24CS073)

to

the APJ Abdul Kalam Technology University

in partial fulfilment of the requirements for the award of the

degree of

Bachelor of Technology In

Computer Science and Engineering

AWH ENGINEERING COLLEGE
CALICUT.KERALA.INDIA

Departmentof ComputerScience and Engineering AWH College

of Engineering

Kuttikattoor,Kozhikode-673008 OCTOBER 2025

AWH ENGINEERING COLLEGE

Kuttikattoor

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

This is to certify that the microproject work report entitled 'CAMPUS LOST & FOUND WEB APPLICATION' submitted AHMED NAJAD PK (AWH24CS006),AMANA KK (AWH24CS008) ARSHAL KS(AWH24CS019) HRIDYA DS (AWH24CS048) MUHAMMAD DILSHAD M (AWH24CS062) MUHAMMED FOUZAN PP (AWH24CS064) NAHALA MP (AWH24CS073), of Third Semester, B.Tech in Computer Science & Engineering has been successfully carried out under our supervision and guidance in partial fulfillment of the requirements of the microproject for the subject PBCST304 - Object Oriented Programming, prescribed by APJ Abdul Kalam Technological University, Kerala. It is further certified that this work is a bonafide record of the microproject carried out by the above students during the academic year 2025-2026.

Mrs Seena Binth KT	**Mrs,Ameetha Junaina**
(Project guide)	Professor and head
Dept.of Cse	Dept.of Cse

DECLARATION

We hereby declare that the microproject report 'CAMPUS **LOST & FOUND WEB APPLICATION**' submitted for the partial fulfilment of the requirements of the Third Semester Microproject for the subject PBCST304 Object Oriented Programming, prescribed by APJ Abdul Kalam Technological University, Kerala is a bonafide work done by us under the guidance of **Asst. Prof. SEENA BINTH KT** The submission represents our ideas in our own words and where ideas or words of others have been included, we have adequately and accurately cited and referred the original sources. We also declare that we have adhered to the ethics and integrity and have not misinterpreted or fabricated any data or idea or fact or source in our submission. We understand that any violation of the above will be a case for disciplinary action by the institution and/or the University and can also evoke penal actions from the sources which have thus not been properly cited or from whom proper permission has not been obtained. This report has not previously formed the basis for the award of any degree, diploma or similar title of any other University.

Place: Kozhikode

Date:24/10/2025

AHMED NAJAD PK

AMANA KK

ARSHAL KS

HRIDYA DS

MUHAMMAD DILSHAD M

MUHAMMED FOUZAN PP

NAHALA MP

AWH ENGINEERING COLLEGE

Kuttikattoor

INSTITUTION VISION

To be an Institute of repute recognised for excellence in education, innovation and social Contribution.

INSTITUTION MISSION

M1: Infrastructural Relevance

Develop, maintain and manage our campus for our stakeholders.

M2: Life Long Learning:

Encourage our stakeholders to participate in lifelong learning through industry and academic interactions.

M3: Social Connect:

Organize socially relevant outreach programs for the benefit of humanity.

DEPARTMENT VISION

To create industry ready and socially skilled Computer Science Engineers.

DEPARTMENT MISSION

M1: Provide a learning platform that encourages thinking and analytical ability in the area of Computer software and hardware.

M2: Inculcate and lifelong and professional skills through the interaction of academicians and Industrialists.

M2: Inculcate and lifelong and professional skills through the interaction of academicians and Industrialists.

M3: Engage with society through social programs in and out of campus.

AWH ENGINEERING COLLEGE

Kuttikattoor

PROGRAM EDUCATIONAL OBJECTIVES (PEO)

Graduates of Computer Science & Engineering will be able to:

PEO1: Professional Practice

Apply Engineering practices required for Software development, Hardware development and Embedded systems.

PEO2: Entrepreneurial Skills

Exhibit innovation, Self – confidence and team work skills in the organization and society.

PEO3: Technological Competence and Adaptability

Graduates shall be competent in the evolving field of computer science by adapting to new tools, technologies, and industry practices, ensuring continued professional and career growth

PROGRAM SPECIFIC OUTCOMES (PSOs)

Student of the Computer Science and Engineering program will:

PSO1: Professional Skills:

Ability to understand the architecture and working of computer hardware and software system.

PSO2: Design and Development Skills:

Ability to design and develop software for technology application to fulfill industrial and social Needs.

PROGRAM OUTCOMES (POs)

Engineering Graduates will be able to:

1.Engineering knowledge: Apply the knowledge of mathematics, science, engineering Fundamentals, and an engineering specialization to the solution of complex engineering problems.

2.Problem analysis: Identify, formulate, research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural Sciences, and engineering sciences.

3.Design/development of solutions: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate Consideration for the public health and safety, and the cultural, societal, and environmental Considerations.

4.Conduct investigations of complex problems: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the Information to provide valid conclusions.

5.Modern tool usage: Create, select, and apply appropriate techniques, resources, and modern Engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.

6.The engineer and society: Apply reasoning informed by the contextual knowledge to assess Societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the Professional engineering practice.

7.Environment and sustainability: Understand the impact of the professional engineering Solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for Sustainable development.

8.Ethics: Apply ethical principles and commit to professional ethics and responsibilities and Norms of the engineering practice.

9.Individual and team work: Function effectively as an individual, and as a member or leader In diverse teams, and in multidisciplinary settings.

10.Communication: Communicate effectively on complex engineering activities with the Engineering community and with society at large, such as, being able to comprehend and write Effective reports and design documentation, make effective presentations, and give and receive clear Instructions.

11. Project management and finance: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one’s own work, as a member and Leader in a team, to manage projects and in multidisciplinary environments.

12. Life-long learning: Recognize the need for, and have the preparation and ability to engage In independent and life-long learning in the broadest context of technological change.

COURSE OUTCOMES (Cos)

At the end of the course students should be able to :

	Course Outcome	**Bloom’s Knowledge Level (KL)**
CO1	Explain the process of writing, compiling, and executing basic Java programs, including their structure and components, to demonstrate proficiency.	K2
CO2	Utilize object-oriented programming principles in the design and implementation of Java applications.	K3
CO3	Develop and manage Java packages and interfaces, enhancing code modularity and reusability.	K3
CO4	Implement error handling using Java’s exception mechanisms and leverage interfaces for modular applications.	K3
CO5	Develop event-driven Java GUI applications with database connectivity using Swing and JDBC.	K3

ABSTRACT

Losing personal belongings on a college campus is a frequent and stressful issue for students and staff. The traditional methods for managing lost and found items, such as physical logbooks and notice boards, are inefficient, lack transparency, and result in a low recovery rate. This project addresses these challenges by developing a centralized, secure, and user-friendly web application designed to streamline the entire lost and found process on a college campus.

The application is built using Java Servlets for the backend logic, JavaServerPages (JSP) for the dynamic frontend, and MySQL for the database management. It features distinct modules for users and administrators. The user module allows individuals to register, report lost or found items, search the database with filters, and submit claims. A key innovation is the secure verification system, where the person reporting a lost item sets a secret question that only the true owner can answer, enabling administrators to verify claims with confidence. The admin module provides a comprehensive dashboard to oversee all reported items, manage claims, and maintain the system's integrity. By digitizing and automating the process, this project aims to significantly improve the efficiency of item recovery, reduce administrative workload, and foster a more secure and organized campus environment.

List of figures

CHAPTER 1

INTRODUCTION

1.1 Background

The daily life of a college campus is a vibrant and dynamic ecosystem, characterized by constant movement, interaction, and activity. Within this bustling environment, the misplacement of personal belongings is not just a minor inconvenience but a frequent and often stressful occurrence. Students, faculty, and administrative staff navigate busy schedules, moving between classrooms, libraries, laboratories, dormitories, and recreational facilities. In this transit, essential items such as smartphones, laptops, chargers, ID cards, wallets containing cash and cards, keys, textbooks, notebooks, and even personal items like jackets or water bottles are commonly left behind or lost.

The conventional methods employed by most educational institutions to manage these lost items have remained largely unchanged for decades. Typically, found items are handed over to a designated physical location – perhaps a central administrative office, the campus security desk, or the main library. Here, the items are recorded, often haphazardly, in paper-based logbooks or rudimentary spreadsheets. Details captured might include a brief description, the location where the item was found, and the date. The items themselves are then stored, sometimes unsystematically, in cupboards, boxes, or back rooms, awaiting reclaim.

This traditional approach is fraught with inherent inefficiencies that significantly hinder the recovery process. Firstly, it is highly decentralized; multiple potential drop-off points might exist across a large campus, requiring someone who has lost an item to embark on a time-consuming and often fruitless physical search, visiting each location. Secondly, the reliance on manual record-keeping lacks any real-time transparency or effective search capability. Sifting through pages of a logbook or relying on the memory of administrative staff is inefficient and prone to errors. Items might be poorly described, miscategorized, or simply overlooked.

Furthermore, these physical lost-and-found offices operate within standard working hours, rendering them inaccessible during evenings or weekends when students might urgently need to recover lost keys or ID cards. The lack of a robust tracking system also means that items can remain unclaimed indefinitely, leading to a significant accumulation of property. This creates a substantial logistical burden for administrators, involving not only storage space management but also the complex task of cataloging, attempting owner identification (if possible), and eventually deciding on the disposal or donation of unclaimed goods. Beyond the logistical challenges, the emotional and academic impact on students who lose crucial items like laptops before an exam or wallets containing essential identification cannot be understated. The current system often exacerbates this stress rather than alleviating it. Security is another concern, as verifying the true owner often relies solely on a verbal description, creating potential for mistaken identity or even fraudulent claims for valuable items.

1.2 Relevance

The need for a modern, digital solution is evident. A web-based platform can automate and streamline the entire process of reporting and retrieving lost items. By creating a centralized digital database, the system provides a single, queryable, 24/7 point of access for the entire campus community. This digital transformation not only improves the raw efficiency of the lost and found process but also fosters a sense of community and trust by encouraging users to assist one another.

This application is designed specifically to address these challenges within a college environment. It replaces the outdated, insecure, and inefficient manual system with an intuitive, secure, and robust web platform. By leveraging modern web technologies, it provides a solution that reduces stress for students, minimizes the workload for administrators, and significantly increases the likelihood of items being returned to their rightful owners. This makes it a highly relevant and valuable tool for any modern college community.

CHAPTER 2

AIM AND OBJECTIVES

2.1 Aim

The primary aim of this project is to architect, design, develop, and implement a secure, centralized, and highly efficient web-based Lost and Found management system. This system is specifically conceived for the dynamic environment of a college campus, with the overarching goal of fundamentally simplifying and significantly accelerating the typically cumbersome processes associated with recovering lost personal items and reporting found items. The ultimate vision is to establish a trusted, reliable digital platform that becomes the go-to resource for the entire campus community when dealing with lost or found property, thereby reducing stress, saving time, and increasing the rate of successful item recovery. This involves moving beyond mere digitization to create an integrated system that addresses the core inefficiencies and security vulnerabilities inherent in traditional, manual methods.

2.2 Objectives

To realize the comprehensive aim stated above, the project is broken down into the following key, measurable objectives. Each objective represents a critical component required for the successful development and deployment of the proposed system:

The key objectives required to achieve this aim are as follows:

1. **To Create a Centralized Platform:** This foundational objective involves developing a single, universally accessible online portal using Java web technologies (Servlets and JSP). This portal must serve as the definitive, singular hub for all lost and found activities within the college. Key aspects include ensuring 24/7 availability, allowing access from any internet-connected device (desktops, laptops, tablets, smartphones), and providing distinct, intuitive interfaces for reporting lost items, reporting found items, searching the database of found items with effective filtering mechanisms, and initiating the claim process.

2. **To Implement Secure User Authentication:** This objective focuses on engineering a robust and secure system for user registration and login. Security and accountability are paramount in a system handling personal belongings. The system must ensure that only verified members of the college community (students, faculty, staff) can post item reports or make claims. This involves creating a secure registration process (potentially using college email domains for validation) and a reliable login mechanism (using secure password handling). By linking every reported item and every claim to a specific, authenticated user account, this objective aims to create a trusted environment and deter misuse or fraudulent activity.
3. **To Develop a Robust Item Management System:** Beyond simple reporting, this objective requires building a dynamic system capable of managing the entire lifecycle of a lost or found item within the platform. Users must be able to provide comprehensive details when reporting items, including a clear item name, selection from predefined categories (e.g., Electronics, Apparel, Books, ID Cards), a detailed text description, the specific location where the item was lost or found, and optionally, an image upload feature. Furthermore, the system must empower administrators to efficiently manage these items. This includes functionalities to review new reports, update item details if necessary, and crucially, change the status of an item (e.g., from 'found' to 'pending claim', then to 'claimed', or eventually 'archived' or 'disposed' after a defined period) based on the claim and verification outcomes
4. **To Integrate a Secure Verification Process:** Implement a "Secret Question" mechanism as a core security feature. This objective is key to solving the primary flaw in manual systems. It ensures that claimed items are returned only to their rightful owners by empowering administrators with a reliable, system-integrated method to verify ownership with confidence before approving a claim.

CHAPTER 3

REQUIREMENT SPECIFICATION

This chapter defines the technical and functional requirements necessary for the development and operation of the Lost and Found web application.

3.1 Functional Requirements

- User Management:
  - The system shall allow new users (students/staff) to register with a full name, email, and password.
  - The system shall allow existing users to log in securely.
  - The system shall differentiate between standard 'user' roles and 'admin' roles.
- Item Management:
  - An authenticated user shall be able to report a 'lost' item with all relevant details.
  - An authenticated user shall be able to report a 'found' item with all relevant details.
  - When reporting a 'lost' item, the user must be prompted to create a 'secret question' and 'secret answer' for verification.
- Search and Claim Process:
  - All users shall be able to view a public, searchable list of all 'found' items.
  - Users shall be able to submit a claim for a 'found' item.
  - If an item has a verification question, the user must provide an answer to submit the claim.
- Admin Module:
  - An admin shall have a dashboard to view all pending claims.
  - For each claim, the admin shall be able to see the original secret question, the correct secret answer, and the claimant's submitted answer.
  - The admin shall have the ability to 'approve' or 'reject' a claim.
  - Approving a claim shall change the item's status to 'claimed' and remove it from the public 'found' list.

3.2 Software Requirements

- **Operating System:** Windows, macOS, or Linux (for development and deployment)
- **Web Server:** Apache Tomcat 9.0 or higher
- **Database:** MySQL 8.0 or higher
- **Backend Technology:** Java 8 (JDK 1.8) or higher, Java Servlets 4.0, JDBC (Java Database Connectivity)
- **Frontend Technology:** JavaServer Pages (JSP) 3.0, JSTL 1.2, HTML5, CSS3
- **IDE (Development):** Visual Studio Code (with Java Extension Pack) or Eclipse IDE for Java EE Developers
- **Build Tool:** Apache Maven

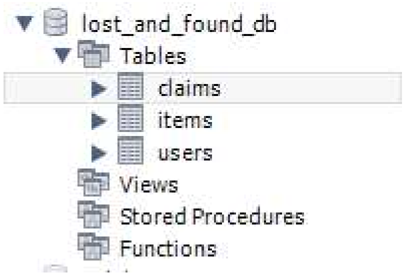
3.3 Hardware Requirements

- **Processor:** Intel Core i3 (or equivalent) 64-bit processor or better
- **RAM:** 4 GB (minimum), 8 GB (recommended)
- **Hard Disk Space:** 10 GB of free space (for IDE, tools, and server)

CHAPTER 4

DATABASE DESIGN

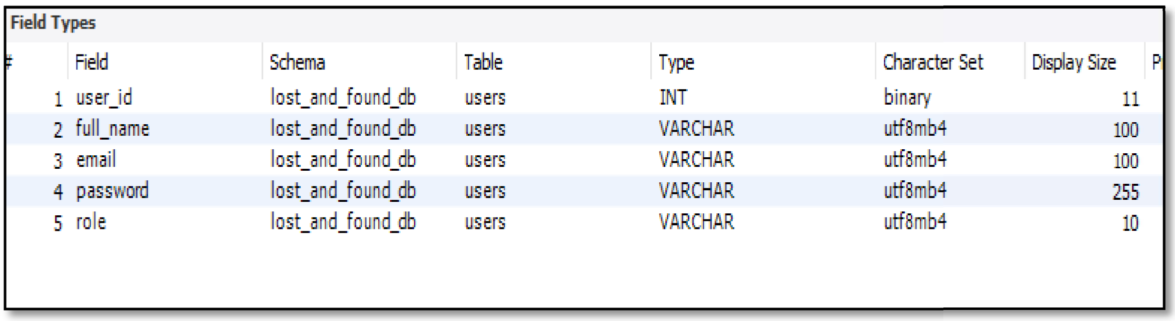
The database is the backbone of the application, designed to store all information logically and efficiently. We chose MySQL as our Relational Database Management System (RDBMS) for its reliability, performance, and widespread industry support. The schema is designed following Third Normal Form (3NF) to reduce data redundancy and improve data integrity.



```
▼ lost_and_found_db
  ▼ Tables
    ▶ claims
    ▶ items
    ▶ users
    Views
    Stored Procedures
    Functions
```

4.1 Users Table This table stores the authentication and identification details for every user of the system, including administrators.

- user_id (Primary Key, INT, AUTO_INCREMENT): A unique numerical identifier for each user.
- full_name (VARCHAR): The user's full name.
- email (VARCHAR, UNIQUE): The user's email address, which doubles as their username and is constrained to be unique.
- password (VARCHAR): Stores the user's password (in a real-world scenario, this would be hashed).
- role (VARCHAR): Defines the user's permissions, e.g., 'user' or 'admin'.

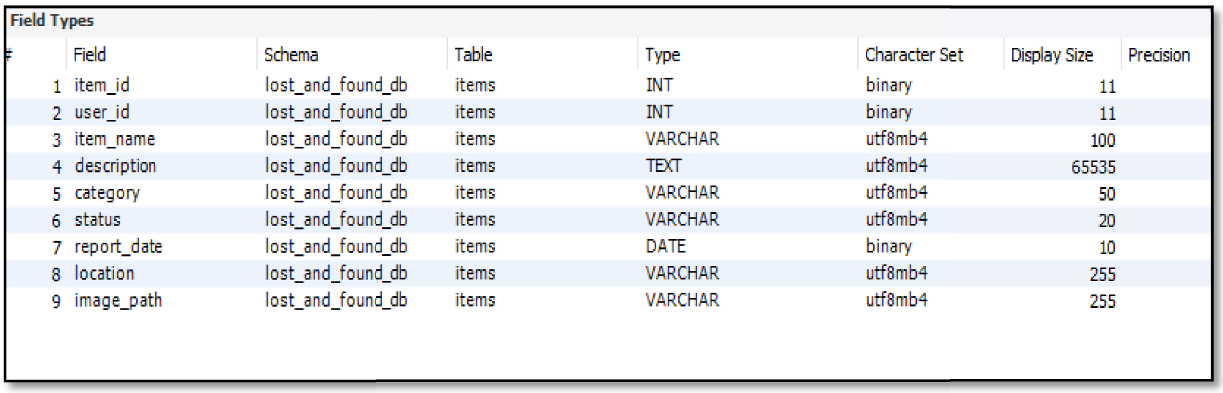


Field Types

#	Field	Schema	Table	Type	Character Set	Display Size	P
1	user_id	lost_and_found_db	users	INT	binary	11	
2	full_name	lost_and_found_db	users	VARCHAR	utf8mb4	100	
3	email	lost_and_found_db	users	VARCHAR	utf8mb4	100	
4	password	lost_and_found_db	users	VARCHAR	utf8mb4	255	
5	role	lost_and_found_db	users	VARCHAR	utf8mb4	10	

4.2 Items Table This table is the central repository, holding all details about every reported lost or found item.

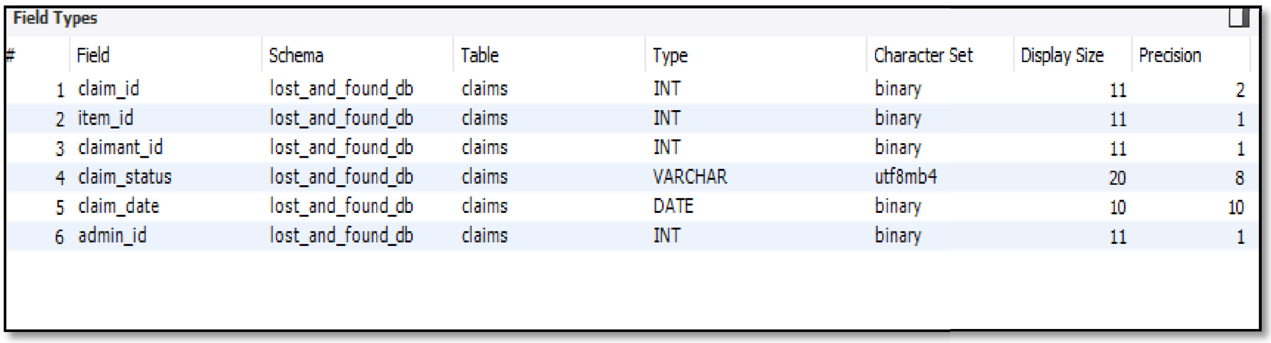
- item_id (Primary Key, INT, AUTO_INCREMENT): A unique numerical identifier for each item.
- user_id (Foreign Key to Users table): Links the item to the user who reported it. This creates a one-to-many relationship (one user can report many items).
- item_name (VARCHAR): The title of the item, e.g., "Black Dell Laptop".
- category (VARCHAR): A predefined category for sorting, e.g., "Electronics", "Books", "Apparel".
- description (TEXT): A detailed description of the item.
- location (VARCHAR): The location where the item was lost or found.
- date_reported (DATE): The date the item was submitted to the system.
- status (VARCHAR): The current state of the item, e.g., 'lost', 'found', 'claimed'.
- verification_question (VARCHAR, NULLABLE): The secret question set by the user who reported a 'lost' item. It is nullable because 'found' items do not have this.
- verification_answer (VARCHAR, NULLABLE): The corresponding secret answer for the verification question.


Field Types

#	Field	Schema	Table	Type	Character Set	Display Size	Precision
1	item_id	lost_and_found_db	items	INT	binary	11	
2	user_id	lost_and_found_db	items	INT	binary	11	
3	item_name	lost_and_found_db	items	VARCHAR	utf8mb4	100	
4	description	lost_and_found_db	items	TEXT	utf8mb4	65535	
5	category	lost_and_found_db	items	VARCHAR	utf8mb4	50	
6	status	lost_and_found_db	items	VARCHAR	utf8mb4	20	
7	report_date	lost_and_found_db	items	DATE	binary	10	
8	location	lost_and_found_db	items	VARCHAR	utf8mb4	255	
9	image_path	lost_and_found_db	items	VARCHAR	utf8mb4	255	

4.3 Claims Table This table acts as a bridge, tracking all claims made by users on items.

- claim_id (Primary Key, INT, AUTO_INCREMENT): A unique numerical identifier for each claim.
- item_id (Foreign Key to Items table): Links the claim to a specific item. This forms a one-to-many relationship (one item can have many claims).
- user_id (Foreign Key to Users table): Links the claim to the user who made it. This is another one-to-many relationship (one user can make many claims).
- claim_status (VARCHAR): The current state of the claim, e.g., 'pending', 'approved', 'rejected'.
- date_claimed (DATE): The date the claim was submitted.
- claimant_answer (VARCHAR, NULLABLE): The answer to the secret question provided by the claimant.



Field Types

#	Field	Schema	Table	Type	Character Set	Display Size	Precision
1	claim_id	lost_and_found_db	claims	INT	binary	11	2
2	item_id	lost_and_found_db	claims	INT	binary	11	1
3	claimant_id	lost_and_found_db	claims	INT	binary	11	1
4	claim_status	lost_and_found_db	claims	VARCHAR	utf8mb4	20	8
5	claim_date	lost_and_found_db	claims	DATE	binary	10	10
6	admin_id	lost_and_found_db	claims	INT	binary	11	1

CHAPTER 5

SYSTEM IMPLEMENTATION

The application was implemented using a standard, robust technology stack and following the Model-View-Controller (MVC) architectural pattern. This pattern is fundamental to modern Java web development as it separates application logic, data, and presentation, making the code much easier to manage, debug, and scale.

- **Model:** This layer represents the data and the business logic of the application.
    - **POJOs (Plain Old Java Objects):** Classes like User.java, Item.java, and Claim.java were created to model the data, acting as simple data containers with getters and setters.
    - **DAOs (Data Access Objects):** Classes like UserDAO.java, ItemDAO.java, and ClaimDAO.java were implemented to encapsulate all database interactions. This is a key part of our Object-Oriented design, abstracting the SQL logic away from the main application. All JDBC (Java Database Connectivity) code—such as creating connections, preparing statements, and handling ResultSets—is located *only* in this layer.
- **View:** This is the presentation layer, responsible for rendering the user interface.
    - **JSP (JavaServer Pages):** We used JSP files like login.jsp, dashboard.jsp, reportItem.jsp, and adminDashboard.jsp to create the dynamic HTML pages.
    - **JSTL (JSP Standard Tag Library):** We exclusively used JSTL (e.g., <c:forEach>, <c:if>) to handle all dynamic logic within the JSPs. This is a best practice that avoids messy and hard-to-maintain Java "scriptlet" code (<% ... %>) in the view files.
- **Controller:** This layer acts as the "brain" of the application, handling all user requests and coordinating between the Model and the View.
    - **Java Servlets:** We implemented several servlets, such as LoginServlet.java, DashboardServlet.java, ItemServlet.java, and AdminServlet.java.

- Each servlet is mapped to specific URL patterns (e.g., /login, /reportItem).
- They use the doPost() method to handle form submissions and doGet() to handle page loads, retrieving data from the Model (DAOs) and forwarding the request (along with the data) to the appropriate View (JSP). They also manage the user's HttpSession to maintain login state.

The project is managed and built using **Apache Maven**, which automatically handles all our dependencies (like the MySQL JDBC driver and JSTL libraries). Maven compiles all Java source code, packages it with the JSP files and web.xml descriptor, and creates a single deployable .war (Web Application Archive) file. This .war file is then deployed to the **Apache Tomcat** web server, which hosts the application.

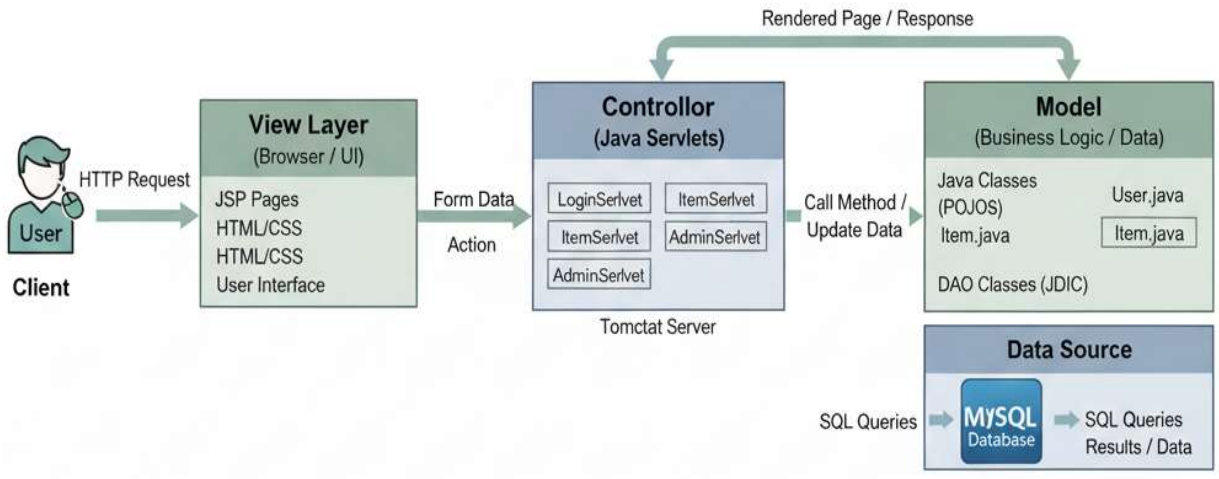


Fig 5.1 MVC

CHAPTER 6

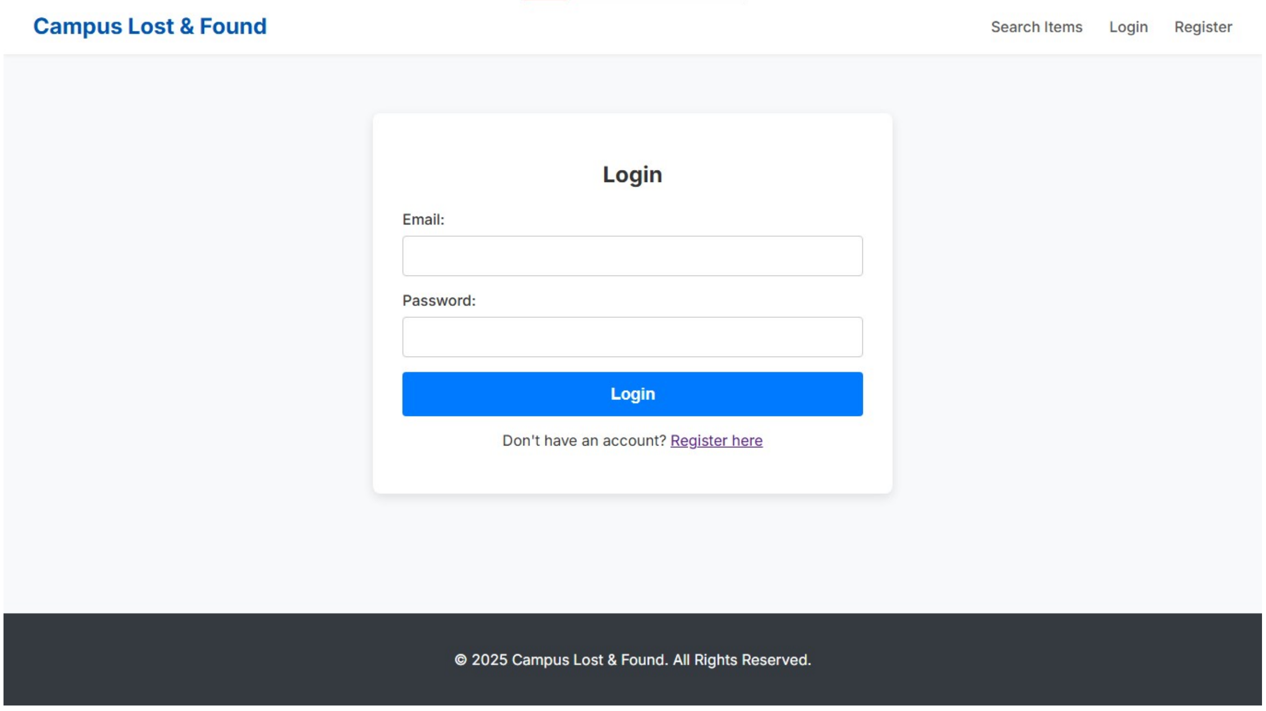
SYSTEM OUTPUT / SCREENSHOTS

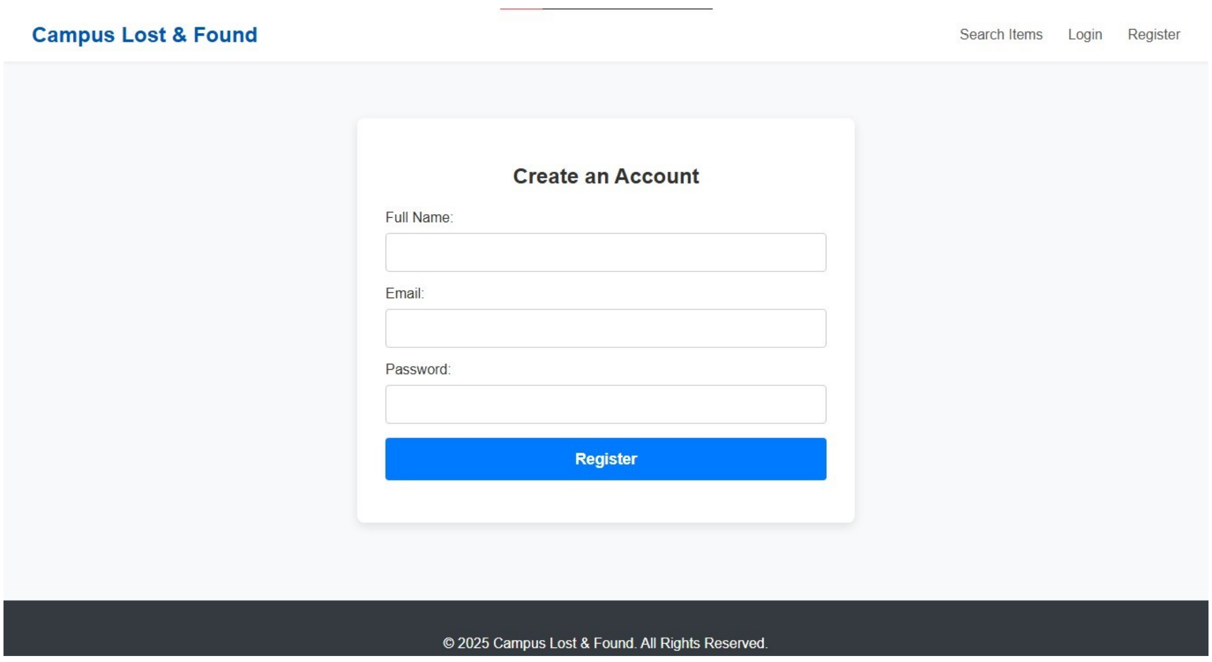
This chapter presents the key visual components of the web application. Each screenshot corresponds to a core feature, demonstrating the user flow from initial login to final claim verification.

6.1 Login and Registration Page

This is the primary entry point for all users. The interface is clean and minimal, featuring forms for both new user registration and existing user login. The registration form collects the user's full name, email address, and a password. The login form authenticates users against the users table in the database. An admin user logs in using the same form but will be redirected to a different dashboard based on their 'role'.

Figure 6.1 - User Login and Registration Interface


Campus Lost & Found
Search Items
Login
Register
Login
Email:
Password:
Login
Don't have an account? Register here
© 2025 Campus Lost & Found. All Rights Reserved.


Campus Lost & Found

Search Items Login Register

Create an Account

Full Name:

Email:

Password:

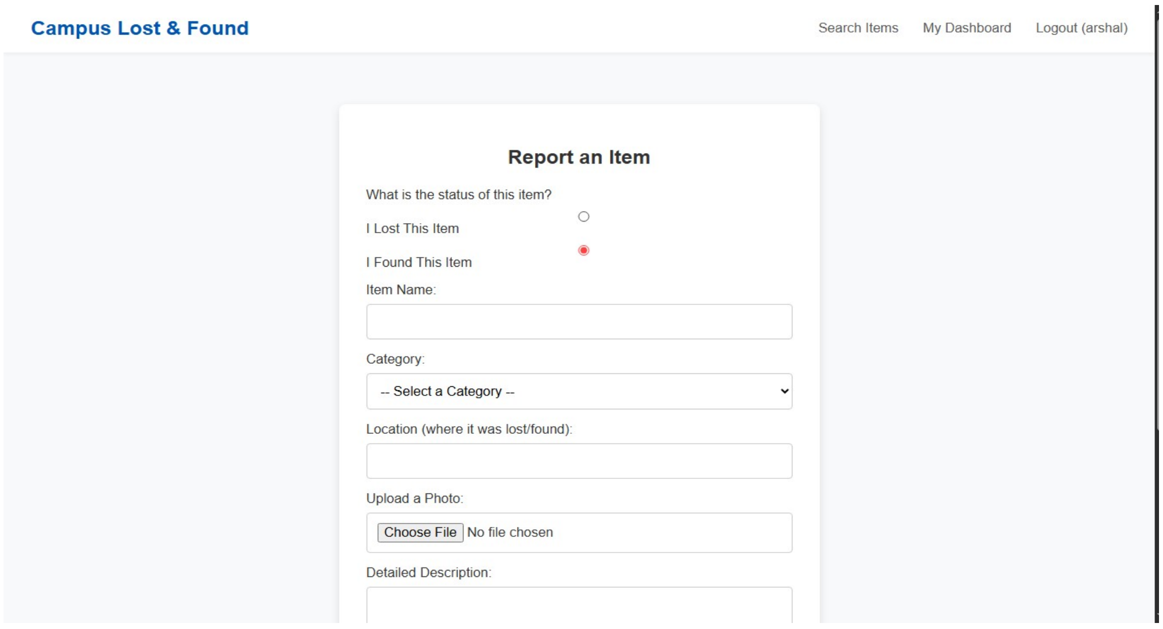
Register

© 2025 Campus Lost & Found. All Rights Reserved.

6.2 Report Item Page

When an authenticated user wants to report an item, they are directed to this form. The page uses radio buttons to toggle between "I Lost This Item" and "I Found This Item." If the user selects "I Lost This Item," JavaScript dynamically unhides the fields for the "Secret Question" and "Secret Answer." This is a critical feature, as this data is used later to verify their ownership.

Figure 6.2 - Report Item Form (showing Secret Question fields)


Campus Lost & Found

Search Items My Dashboard Logout (arshal)

Report an Item

What is the status of this item?

I Lost This Item ○

I Found This Item [✓ ●]

Item Name:

Category:

-- Select a Category --

Location (where it was lost/found):

Upload a Photo:

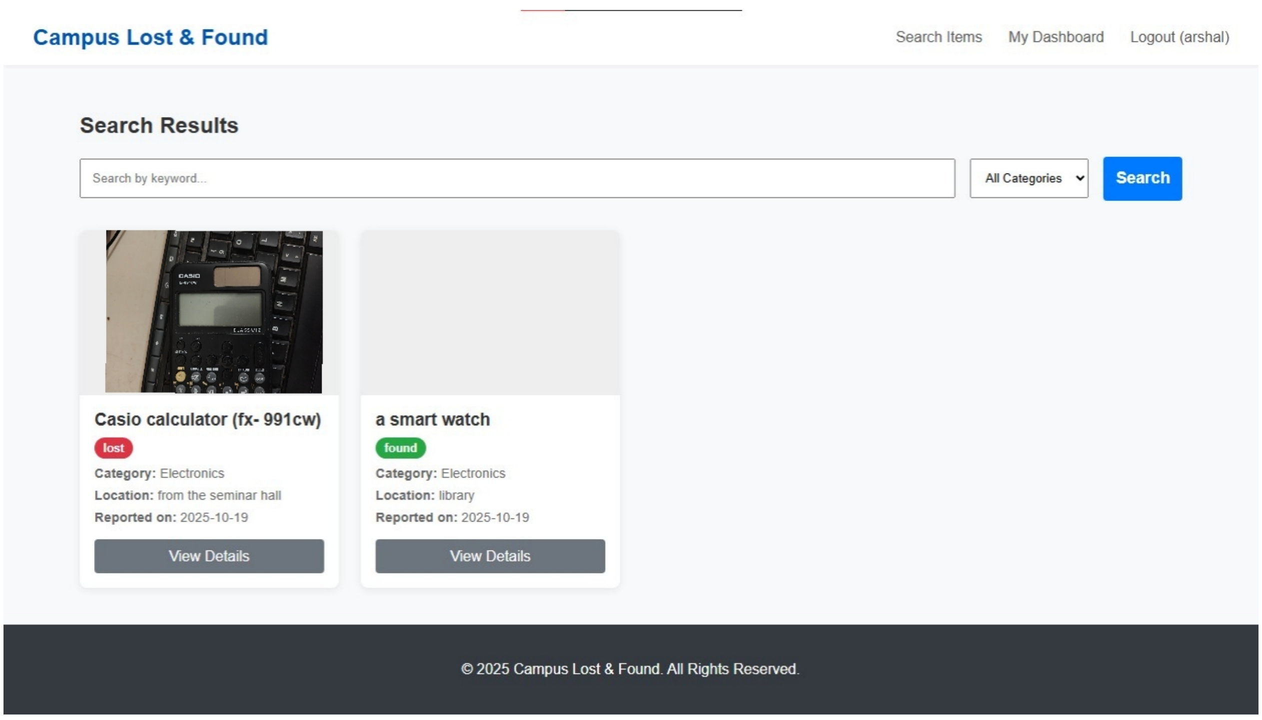
Choose File No file chosen

Detailed Description:

6.3 Search Results Page

This page displays all currently "found" items in a clean, card-based grid. Users can see a brief overview of each item, including its name, image, and status. This page also includes search and filter functionality, allowing users to narrow down the results by keyword (searching the item name and location) or by category, making it easy to find a specific item.

Figure 6.3 - Item Search and Results Grid



Search Results

Casio calculator (fx- 991cw)
lost
Category: Electronics
Location: from the seminar hall
Reported on: 2025-10-19

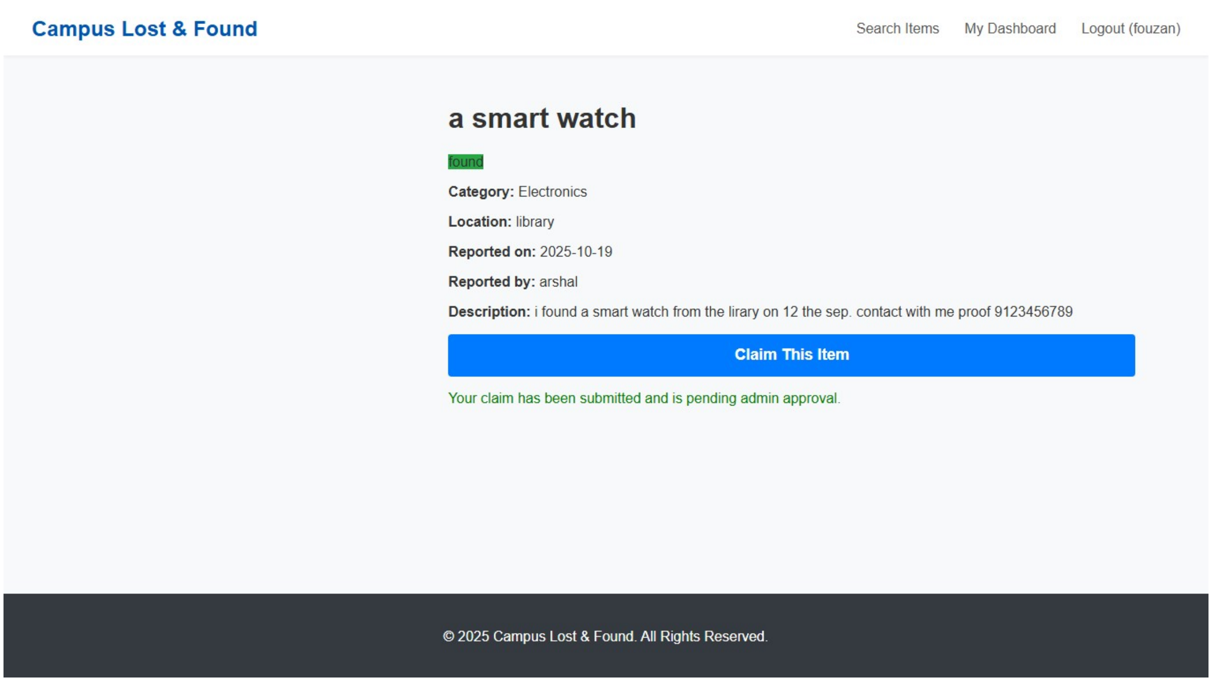
a smart watch
found
Category: Electronics
Location: library
Reported on: 2025-10-19

© 2025 Campus Lost & Found. All Rights Reserved.

6.4 Item Details and Claim Page

Clicking "View Details" on a search result leads the user to this page. It displays all information about the found item, such as its full description, location, and the date it was reported. If the item has a "Secret Question" associated with it (meaning it was first reported as 'lost' and then 'found'), the claim form will dynamically display the question and a text box for the claimant to enter their answer.

Figure 6.4 - Item Details Page with Claim Verification Question



a smart watch

found

Category: Electronics

Location: library

Reported on: 2025-10-19

Reported by: arshal

Description: i found a smart watch from the lirary on 12 the sep. contact with me proof 9123456789

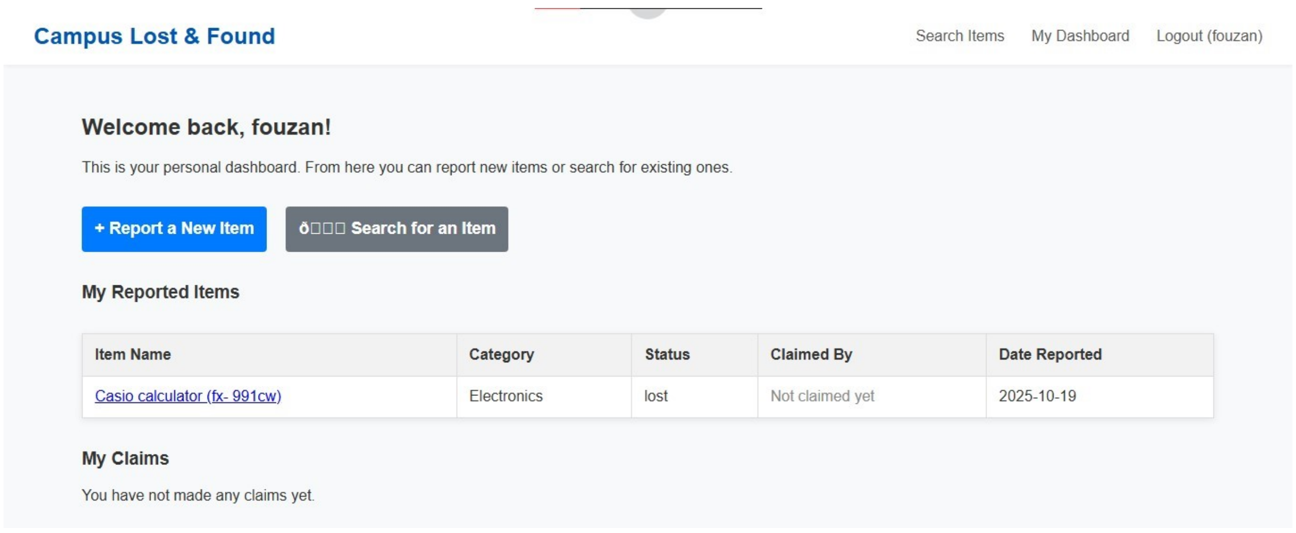
Your claim has been submitted and is pending admin approval.

© 2025 Campus Lost & Found. All Rights Reserved.

6.5 User Dashboard

After logging in, a standard user is directed to their personal dashboard. This page is the user's central hub and is dynamically populated with data specific to them. It features two main tables: "My Reported Items" (showing the status of items they've reported) and "My Claims" (showing the status of claims they've made on other items).

Figure 6.5 - Dynamic User Dashboard



Welcome back, fouzan!

This is your personal dashboard. From here you can report new items or search for existing ones.

My Reported Items

Item Name	Category	Status	Claimed By	Date Reported
Casio calculator (fx- 991cw)	Electronics	lost	Not claimed yet	2025-10-19

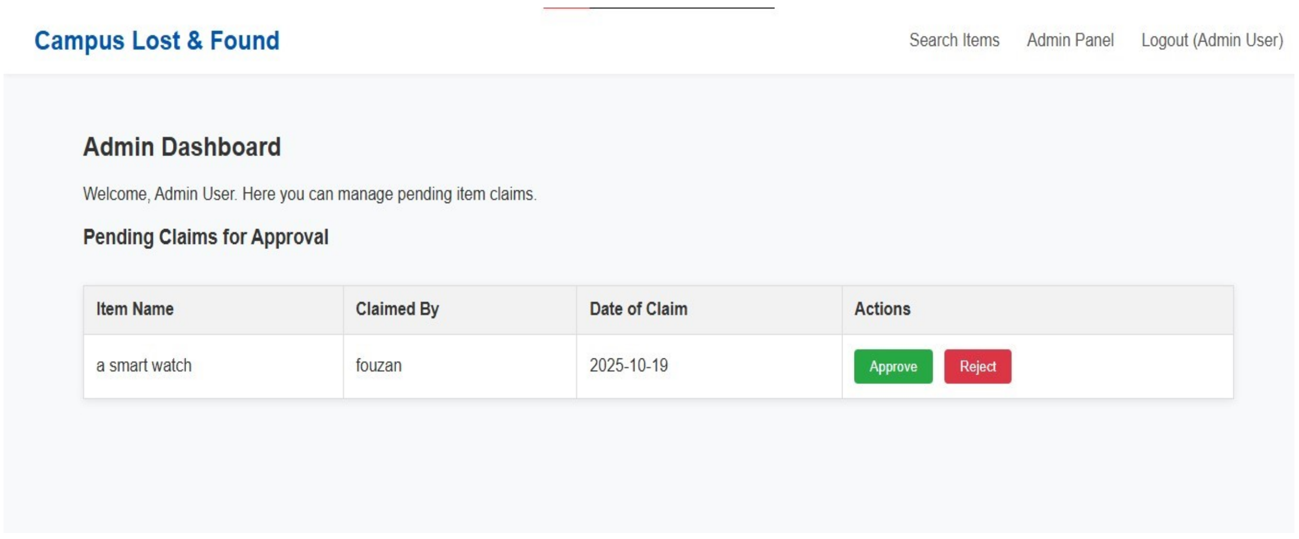
My Claims

You have not made any claims yet.

6.6 Admin Dashboard

The admin dashboard provides a clear, high-level view of all pending claims that require action. This is the system's core verification screen. For each pending claim, the admin can clearly see the item's name, the claimant's name, the original **Verification Question**, the **Correct Answer** (stored in the database), and the **Claimant's Submitted Answer.** This side-by-side comparison allows the admin to confidently approve or reject the claim with the click of a button.

Figure 6.6 - Admin Dashboard with Claim Verification Details



Admin Dashboard

Welcome, Admin User. Here you can manage pending item claims.

Pending Claims for Approval

Item Name	Claimed By	Date of Claim	Actions
a smart watch	fouzan	2025-10-19	Approve Reject

Chapter 7

CODE LISTING

1)ClaimDAO.java

```
package com.example.lostandfound.dao;

import com.example.lostandfound.db.DatabaseConnection;

import com.example.lostandfound.model.Claim;

import java.sql.*;

import java.util.ArrayList;

import java.util.List;

public class ClaimDAO {

  public boolean addClaim(Claim claim) throws SQLException {

     String sql = "INSERT INTO claims (item_id, claimant_id, claim_date, claim_status) VALUES (?, ?, ?, 
'pending')";

    try (Connection conn = DatabaseConnection.getConnection();

         PreparedStatement stmt = conn.prepareStatement(sql)) {

       stmt.setInt(1, claim.getItemId());

       stmt.setInt(2, claim.getClaimantId());

       stmt.setDate(3, claim.getClaimDate());

       return stmt.executeUpdate() > 0;

    }

  }

  public List<Claim> getPendingClaims() throws SQLException {

     List<Claim> claims = new ArrayList<>();

     String sql = "SELECT c.*, i.item_name, u.full_name AS claimant_name " +

             "FROM claims c " +
        "JOIN items i ON c.item_id = i.item_id " +
          "JOIN users u ON c.claimant_id = u.user_id " +
          "WHERE c.claim_status = 'pending'";
  try (Connection conn = DatabaseConnection.getConnection();
     Statement stmt = conn.createStatement();
     ResultSet rs = stmt.executeQuery(sql)) {
    while (rs.next()) {
      Claim claim = new Claim();
      claim.setClaimId(rs.getInt("claim_id"));
      claim.setItemId(rs.getInt("item_id"));
      claim.setClaimantId(rs.getInt("claimant_id"));
      claim.setClaimDate(rs.getDate("claim_date"));
      claim.setItemName(rs.getString("item_name"));
      claim.setClaimantName(rs.getString("claimant_name"));
      claims.add(claim);
    }
  }
  return claims;
}
public void updateClaimStatus(int claimId, String status, int adminId) throws SQLException {
  String sql = "UPDATE claims SET claim_status = ?, admin_id = ? WHERE claim_id = ?";
  try (Connection conn = DatabaseConnection.getConnection();
     PreparedStatement stmt = conn.prepareStatement(sql)) {
    stmt.setString(1, status);
    stmt.setInt(2, adminId);
    stmt.setInt(3, claimId);
    stmt.executeUpdate();   }
  }

  public List<Claim> getClaimsByUserId(int userId) throws SQLException {

    List<Claim> claims = new ArrayList<>();

    String sql = "SELECT c.*, i.item_name " +

           "FROM claims c JOIN items i ON c.item_id = i.item_id " +

           "WHERE c.claimant_id = ? ORDER BY c.claim_date DESC";

    try (Connection conn = DatabaseConnection.getConnection();

       PreparedStatement stmt = conn.prepareStatement(sql)) {

      stmt.setInt(1, userId);

      ResultSet rs = stmt.executeQuery();

      while (rs.next()) {

        Claim claim = new Claim();

        claim.setClaimId(rs.getInt("claim_id"));

        claim.setItemName(rs.getString("item_name"));

        claim.setClaimStatus(rs.getString("claim_status"));

        claim.setClaimDate(rs.getDate("claim_date"));

        claims.add(claim);

      }

    }

    return claims;

  }

}
```

2) ItemDAO.java

```
package com.example.lostandfound.dao;

import com.example.lostandfound.db.DatabaseConnection;
import com.example.lostandfound.model.Item;

import java.sql.*;

import java.util.ArrayList;

import java.util.List;

public class ItemDAO {

 public boolean addItem(Item item) throws SQLException {

     String sql = "INSERT INTO items (user_id, item_name, description, category, status, report_date, location,
image_path) VALUES (?, ?, ?, ?, ?, ?, ?, ?)";

     try (Connection conn = DatabaseConnection.getConnection();

          PreparedStatement stmt = conn.prepareStatement(sql)) {

        stmt.setInt(1, item.getUserId());

        stmt.setString(2, item.getItemName());

        stmt.setString(3, item.getDescription());

        stmt.setString(4, item.getCategory());

        stmt.setString(5, item.getStatus());

        stmt.setDate(6, item.getReportDate());

        stmt.setString(7, item.getLocation());

        stmt.setString(8, item.getImagePath());

        return stmt.executeUpdate() > 0;

     }

  }

 public List<Item> searchItems(String keyword, String category) throws SQLException {

     List<Item> items = new ArrayList<>();

     // Build the SQL query dynamically

     String sql = "SELECT i.*, u.full_name FROM items i JOIN users u ON i.user_id = u.user_id WHERE i.status !=
'claimed'";

     if (keyword != null && !keyword.isEmpty()) {

        sql += " AND (i.item_name LIKE ? OR i.description LIKE ?)";
```

```
    }
    if (category != null && !category.isEmpty() && !category.equals("All")) {
      sql += " AND i.category = ?";
    }
    sql += " ORDER BY i.report_date DESC";
try (Connection conn = DatabaseConnection.getConnection();
       PreparedStatement stmt = conn.prepareStatement(sql)) {
      int paramIndex = 1;
      if (keyword != null && !keyword.isEmpty()) {
        stmt.setString(paramIndex++, "%" + keyword + "%");
        stmt.setString(paramIndex++, "%" + keyword + "%");
      }
      if (category != null && !category.isEmpty() && !category.equals("All")) {
        stmt.setString(paramIndex++, category);
      }

      ResultSet rs = stmt.executeQuery();
      while (rs.next()) {
        Item item = new Item();
        item.setItemId(rs.getInt("item_id"));
        item.setItemName(rs.getString("item_name"));
        item.setDescription(rs.getString("description"));
        item.setCategory(rs.getString("category"));
        item.setStatus(rs.getString("status"));
        item.setReportDate(rs.getDate("report_date"));
        item.setLocation(rs.getString("location"));
        item.setImagePath(rs.getString("image_path"));
```

```
            item.setReporterName(rs.getString("full_name"));

            items.add(item);

          }

        }

        return items;

      }

      public Item getItemById(int itemId) throws SQLException {

        String sql = "SELECT i.*, u.full_name FROM items i JOIN users u ON i.user_id = u.user_id WHERE i.item_id =
?";

        try (Connection conn = DatabaseConnection.getConnection();

             PreparedStatement stmt = conn.prepareStatement(sql)) {

          stmt.setInt(1, itemId);

          ResultSet rs = stmt.executeQuery();

          if (rs.next()) {

            Item item = new Item();

            item.setItemId(rs.getInt("item_id"));

            item.setUserId(rs.getInt("user_id"));

            item.setItemName(rs.getString("item_name"));

            item.setDescription(rs.getString("description"));

            item.setCategory(rs.getString("category"));

            item.setStatus(rs.getString("status"));

            item.setReportDate(rs.getDate("report_date"));

            item.setLocation(rs.getString("location"));

            item.setImagePath(rs.getString("image_path"));

            item.setReporterName(rs.getString("full_name"));

            return item;

          }
```

```
    }
    return null;
}


public void updateItemStatus(int itemId, String status) throws SQLException {
    String sql = "UPDATE items SET status = ? WHERE item_id = ?";
    try (Connection conn = DatabaseConnection.getConnection();
         PreparedStatement stmt = conn.prepareStatement(sql)) {
        stmt.setString(1, status);
        stmt.setInt(2, itemId);
        stmt.executeUpdate();
    }
}
public List<Item> getItemsByUserId(int userId) throws SQLException {
    List<Item> items = new ArrayList<>();
    // This query now uses a LEFT JOIN to connect items to claims and then to the users table
    // to get the claimant's name. A LEFT JOIN is used so we still get the item even if it has no claim.
    String sql = "SELECT i.*, u_claimant.full_name AS claimant_name " +
                 "FROM items i " +
                 "LEFT JOIN claims c ON i.item_id = c.item_id AND c.claim_status = 'approved' " +
                 "LEFT JOIN users u_claimant ON c.claimant_id = u_claimant.user_id " +
                 "WHERE i.user_id = ? ORDER BY i.report_date DESC";


    try (Connection conn = DatabaseConnection.getConnection();
         PreparedStatement stmt = conn.prepareStatement(sql)) {
        stmt.setInt(1, userId);
        ResultSet rs = stmt.executeQuery();
```

```
          while (rs.next()) {
            Item item = new Item();
            item.setItemId(rs.getInt("item_id"));
            item.setItemName(rs.getString("item_name"));
            item.setCategory(rs.getString("category"));
            item.setStatus(rs.getString("status"));
            item.setReportDate(rs.getDate("report_date"));
            item.setLocation(rs.getString("location"));
             // NEW: Set the claimant's name. If no one claimed it, this will be null.
           item.setClaimantName(rs.getString("claimant_name"));
items.add(item);
          }
      }
      return items;
    }
}
```

3) UserDAO.java

```
package com.example.lostandfound.dao;
import com.example.lostandfound.db.DatabaseConnection;
import com.example.lostandfound.model.User;
import java.sql.*;
public class UserDAO {
public User login(String email, String password) throws SQLException {
      String sql = "SELECT * FROM users WHERE email = ? AND password = ?";
      try (Connection conn = DatabaseConnection.getConnection();
           PreparedStatement stmt = conn.prepareStatement(sql)) {
          stmt.setString(1, email);
stmt.setString(2, password); // Note: In a real app, hash the password!
          ResultSet rs = stmt.executeQuery();
          if (rs.next()) {
              User user = new User();
              user.setUserId(rs.getInt("user_id"));
              user.setFullName(rs.getString("full_name"));
              user.setEmail(rs.getString("email"));
              user.setRole(rs.getString("role"));
              return user;
          }
      }
      return null;
  }
public boolean register(User user) throws SQLException {
      String sql = "INSERT INTO users (full_name, email, password, role) VALUES (?, ?, ?, 'user')";
      try (Connection conn = DatabaseConnection.getConnection();
           PreparedStatement stmt = conn.prepareStatement(sql)) {
          stmt.setString(1, user.getFullName());
          stmt.setString(2, user.getEmail());
          stmt.setString(3, user.getPassword()); // Hashing should be done here
          return stmt.executeUpdate() > 0;
      }
  }
}
```

4) DatabaseConnection.java

```
package com.example.lostandfound.db;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.SQLException;

public class DatabaseConnection {

   private static final String URL = "jdbc:mysql://localhost:3306/lost_and_found_db";

   private static final String USER = "root"; // Change if you have a different user

   private static final String PASSWORD = "160106"; // Change to your MySQL password

 public static Connection getConnection() throws SQLException {

      try {

         Class.forName("com.mysql.cj.jdbc.Driver");

         return DriverManager.getConnection(URL, USER, PASSWORD);

      } catch (ClassNotFoundException e) {

         throw new SQLException("MySQL JDBC Driver not found.", e);

      }

   }

}
```

5) Claim.java

```
package com.example.lostandfound.model;

import java.sql.Date;

public class Claim {

   private int claimId;

   private int itemId;

   private int claimantId;
```

```
    private String claimStatus;

    private Date claimDate;

    private String itemName; // To display in admin view

    private String claimantName; // To display in admin view


    // Getters and Setters

    public int getClaimId() { return claimId; }

    public void setClaimId(int claimId) { this.claimId = claimId; }

    public int getItemId() { return itemId; }

    public void setItemId(int itemId) { this.itemId = itemId; }

    public int getClaimantId() { return claimantId; }

    public void setClaimantId(int claimantId) { this.claimantId = claimantId; }

    public String getClaimStatus() { return claimStatus; }

    public void setClaimStatus(String claimStatus) { this.claimStatus = claimStatus; }

    public Date getClaimDate() { return claimDate; }

    public void setClaimDate(Date claimDate) { this.claimDate = claimDate; }

    public String getItemName() { return itemName; }

    public void setItemName(String itemName) { this.itemName = itemName; }

    public String getClaimantName() { return claimantName; }

    public void setClaimantName(String claimantName) { this.claimantName = claimantName; }

}
```

6) item.java

```
package com.example.lostandfound.model;

import java.sql.Date;

public class Item {

    private int itemId;
  private int userId;

    private String itemName;

    private String description;

    private String category;

    private String status;

    private Date reportDate;

    private String location;

    private String imagePath;

    private String reporterName; // For displaying who reported it

    private String claimantName;


    // Getters and Setters

    public String getClaimantName() {

       return claimantName;

    }

    public void setClaimantName(String claimantName) {

       this.claimantName = claimantName;

    }


    public int getItemId() { return itemId; }

    public void setItemId(int itemId) { this.itemId = itemId; }

    public int getUserId() { return userId; }

    public void setUserId(int userId) { this.userId = userId; }

    public String getItemName() { return itemName; }

    public void setItemName(String itemName) { this.itemName = itemName; }

    public String getDescription() { return description; }

    public void setDescription(String description) { this.description = description; }
    public String getCategory() { return category; }

    public void setCategory(String category) { this.category = category; }

    public String getStatus() { return status; }

    public void setStatus(String status) { this.status = status; }

    public Date getReportDate() { return reportDate; }

    public void setReportDate(Date reportDate) { this.reportDate = reportDate; }

    public String getLocation() { return location; }

    public void setLocation(String location) { this.location = location; }

    public String getImagePath() { return imagePath; }

    public void setImagePath(String imagePath) { this.imagePath = imagePath; }

    public String getReporterName() { return reporterName; }

    public void setReporterName(String reporterName) { this.reporterName = reporterName; }

  }
```

7) ClaimServlet.java

```
package com.example.lostandfound.servlet;

import com.example.lostandfound.dao.ClaimDAO;

import com.example.lostandfound.model.Claim;

import com.example.lostandfound.model.User;

import javax.servlet.*;

import javax.servlet.http.*;

import javax.servlet.annotation.*;

import java.io.IOException;

import java.sql.Date;

import java.sql.SQLException;


@WebServlet("/claim")
```

```
public class ClaimServlet extends HttpServlet {

   private ClaimDAO claimDAO = new ClaimDAO();



   protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException,
IOException {

       HttpSession session = request.getSession(false);

       if (session == null || session.getAttribute("user") == null) {

           response.sendRedirect("login.jsp");

           return;

       }



       User user = (User) session.getAttribute("user");

       int itemId = Integer.parseInt(request.getParameter("itemId"));

 Claim claim = new Claim();

       claim.setItemId(itemId);

       claim.setClaimantId(user.getUserId());

       claim.setClaimDate(new Date(System.currentTimeMillis()));



       try {

           claimDAO.addClaim(claim);

           // Redirect with a success message

           response.sendRedirect("items?action=view&id=" + itemId + "&claim_success=true");

       } catch (SQLException e) {

           throw new ServletException("Database error when making a claim.", e);

       }

   }

}
```

8) DashboardServlet.java

```
package com.example.lostandfound.servlet;


import com.example.lostandfound.dao.ClaimDAO;

import com.example.lostandfound.dao.ItemDAO;

import com.example.lostandfound.model.Claim;

import com.example.lostandfound.model.Item;

import com.example.lostandfound.model.User;


import javax.servlet.ServletException;

import javax.servlet.annotation.WebServlet;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

import javax.servlet.http.HttpSession;

import java.io.IOException;

import java.sql.SQLException;

import java.util.List;


@WebServlet("/dashboard")

public class DashboardServlet extends HttpServlet {

  private ItemDAO itemDAO = new ItemDAO();

  private ClaimDAO claimDAO = new ClaimDAO();
```

```
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException,
IOException {

        HttpSession session = request.getSession(false);

        if (session == null || session.getAttribute("user") == null) {

            response.sendRedirect("login.jsp");

            return;

        }



        User user = (User) session.getAttribute("user");



        try {

            List<Item> reportedItems = itemDAO.getItemsByUserId(user.getUserId());

            request.setAttribute("reportedItems", reportedItems);

            List<Claim> userClaims = claimDAO.getClaimsByUserId(user.getUserId());

            request.setAttribute("userClaims", userClaims);

            request.getRequestDispatcher("dashboard.jsp").forward(request, response);



        } catch (SQLException e) {

            throw new ServletException("Database error while loading dashboard data.", e);

        }

    }

}
```

CONCLUSION

This project successfully addressed the inefficiencies and security concerns inherent in traditional, manual lost and found systems by developing a modern, web-based Campus Lost & Found application. The primary aim—to create a secure, centralized, and efficient platform for managing lost and found items within a college environment—was effectively achieved. By replacing outdated methods with a digital solution, this application provides a significant improvement in the item recovery process for the entire campus community.

The system utilizes Java Servlets and JSP built upon the Model-View-Controller (MVC) architecture, connected to a MySQL database via JDBC. This approach resulted in a well-structured, scalable, and maintainable application. Key functional achievements include secure user authentication, intuitive interfaces for reporting lost or found items, dynamic search capabilities, and personalized user dashboards. The most significant innovation is the implementation of the "Secret Question" verification mechanism. This feature directly tackles the problem of verifying ownership, empowering administrators to return items confidently and securely, thereby increasing trust in the system.

The development process provided valuable practical experience in full-stack Java web development, database design, MVC implementation, and addressing real-world application requirements. While the current system fulfills its core objectives, future enhancements could include features like image uploads, automated email notifications, advanced search filters, location-based services, or even a dedicated mobile application to further improve user experience and functionality.

In summary, the Campus Lost & Found web application represents a successful transition to a digital service, offering a robust, user-friendly, and secure solution. It serves as a valuable tool that demonstrably saves time, reduces administrative workload, and fosters a more connected and responsible campus environment.

REFERENCES

[1] GeeksforGeeks, "Servlet and JSP Tutorial," [Online]. Available: [https://www.geeksforgeeks.org/servlet-jsp-tutorial/](https://www.google.com/search?q=https://www.geeksforgeeks.org/servlet-jsp-tutorial/).

[2] Baeldung, "Introduction to JSP," [Online]. Available: [https://www.baeldung.com/intro-to-jsp](https://www.google.com/search?q=https://www.baeldung.com/intro-to-jsp).

[3] Oracle, "Java Servlet Technology," [Online]. Available: [https://www.oracle.com/java/technologies/java-servlet-technology.html](https://www.google.com/search?q=https://www.oracle.com/java/technologies/java-servlet-technology.html).

[4] MySQL Documentation, "MySQL 8.0 Reference Manual," [Online]. Available: https://dev.mysql.com/doc/refman/8.0/en/.