Software Requirements Specification

For

Lost Document Verifier for College Records

Version 1.0

Prepared by

MIRUDHULA R

30-09-2025

Table of Contents

Revision History

Name	Date	Reason For Changes	Version

1. Introduction

1.1 Purpose

This Software Requirements Specification (SRS) document describes the functional and non-functional requirements of the project titled Lost Document Verifier for College Records, version 1.0.

Lost Document Verifier is a secure, web-based system designed to help students and college administrators manage the reissue process of academic documents that have been reported lost. The platform enables students to submit reissue requests, upload verification details, and track the progress of their application. Admins can review requests, validate eligibility, and approve or reject document reissue.

The purpose of this document is to define the scope and expectations of the system and to serve as a reference for developers, testers, project managers, and college stakeholders. This SRS focuses on the core features of lost document handling and does not include extended features like integration with government databases or document blockchain validation, which may be considered for future releases.

1.2 Document Conventions

- Bold text is used to highlight important headings and terminology.
- <Placeholder> tags are used where further content may be added or refined.
- Requirements are labeled with unique identifiers for easy traceability.
- Priority levels are clearly defined for each requirement. There is no assumption that child requirements inherit priorities from parent ones.
- All diagrams follow UML (Unified Modeling Language) standards and will be presented where appropriate.

1.3 Intended Audience and Reading Suggestions

This document is intended for the following audience:

Students: To view the platform capabilities related to lost document management and understand the verification steps.

Suggested Reading Sequence:

1. Start with Section 1: Introduction for an overview of the project.
2. Continue with Section 2: Overall Description to understand the system’s context and environment.
3. Move to Section 3: Functional Requirements for detailed software functionality.
4. Refer to Section 4: Non-Functional Requirements for information on performance, security, and usability.
5. Use Appendices for diagrams, glossary, and reference materials.

1.4 Product Scope

Lost Document Verifier for College Records is intended to digitize and streamline the traditionally manual process of verifying and reissuing lost college documents such as mark sheets, ID cards, and certificates. It serves both students (requesters) and administrators (approvers) through a centralized web interface.

Key system objectives:

- Enable students to raise lost document requests with supporting proof.
- Allow admin staff to review, verify, and approve or reject the requests.
- Maintain records of previous requests and verification history.
- Ensure secure handling of sensitive academic records.
- Generate digitally signed approval receipts and status tracking.

This system supports a paperless, transparent, and faster approach to document recovery, aligning with institutional digital transformation goals.

1.5 References

The following documents and sources were referred to in preparing this SRS for the Lost Document Verifier for College Records:

- IEEE 830-1998 SRS Documentation Standard
- College administrative document policies (generic)
- UML Diagrams and Notation Guidelines
- Security best practices for academic data management
- Personal academic experience and typical college workflows

2. Overall Description

2.1 Product Perspective

The Lost Document Verifier is a web-based application designed to assist students, faculty, and administrative staff in verifying lost or duplicate academic documents. It integrates with the existing institutional database of student records to ensure authenticity. The system provides a secure, automated, and efficient way of validating whether a reported lost document is genuine or tampered. It functions as an independent module but can also be integrated into a broader college record management system.

2.2 Product Functions

The major functions of the Lost Document Verifier include:

- User authentication and secure login for students, administrators.
- Uploading scanned or digital copies of lost documents for verification.
- Automatic matching of uploaded documents with institutional records.

- Notification to users about the status of verification (valid, invalid, or duplicate).
- Administrative dashboard for handling verification requests and updating records.
- Generation of verification reports for official use.
- Search functionality for documents based on identifiers such as roll number, document ID, or name.

2.3 User Classes and Characteristics

- **Students:** Users who request verification of their lost documents. They require minimal technical knowledge and interact mainly with the upload and status-tracking modules.
- **Administrators:** Users with the highest privileges, responsible for managing verification requests, updating the database, and generating reports. They have moderate to advanced technical knowledge.

2.4 Operating Environment

- **Hardware:** Standard desktop or laptop computer with at least 4GB RAM and 2GHz processor; mobile devices for limited access.
- **Software:**
    - Server-side: FastAPI / Java / Python-based backend (depending on implementation).
    - Database: PostgreSQL or MySQL.
    - Client-side: HTML, CSS, JavaScript for the user interface.
- **Operating Systems:** Windows, Linux, macOS.
- **Browsers:** Compatible with Chrome, Firefox, Edge, and Safari.

2.5 Design and Implementation Constraints

- The system must comply with institutional security policies and data privacy regulations.
- The verification process depends on the completeness and accuracy of the existing student records database.
- The application should operate under network constraints and be accessible within campus intranet and over the internet.
- Implementation is limited to web-based access; no native mobile app is included in the current scope.

2.6 User Documentation

The system will include the following user documentation:

- **User Manual:** Instructions for students on how to log in, upload documents, and check status.

- **Administrator Guide:** Steps for handling verification requests, generating reports, and maintaining records.
- **Online Help/FAQ:** Built-in guidance available within the system interface for quick troubleshooting.

2.7 Assumptions and Dependencies

- It is assumed that all student records are stored in digital format within the institutional database.
- Users have access to an internet connection and a device capable of running modern browsers.
- Document scanning or uploading will be done by the user; the system does not provide scanning hardware.
- The accuracy of verification depends on the quality of the uploaded document and the reliability of stored records.
- Future extensions may depend on integration with third-party verification systems or government databases.

3. External Interface Requirements

3.1 User Interfaces

The system will provide a web-based user interface accessible through standard browsers.

- **Login Page:** Secure login for students, faculty, and administrators.
- **Student Dashboard:** Allows students to upload lost documents, track verification status, and download verification reports.
- **Admin Dashboard:** Enables administrators to view pending requests, approve or reject them, and generate reports.
- **Search Page:** Allows users/admins to search documents by roll number, document ID, or name.
- **Report Page:** Displays verification results in a printable/exportable format.

The interface will be designed for ease of use with consistent navigation, buttons, and forms.

3.2 Hardware Interfaces

- **User Devices:** Desktop/laptop with at least 4 GB RAM, 2 GHz processor.
- **Peripheral Devices:** Scanner or camera (used externally by users to digitize lost documents).
- **Mobile Devices:** Smartphones/tablets for limited access to basic features (status tracking, report view).
- No specialized hardware beyond standard institutional servers is required.

3.3 Software Interfaces

- **Operating System:** Windows/Linux/macOS for client and server machines.
- **Database:** PostgreSQL or MySQL for storing student and verification records.
- **Backend:** Implemented using FastAPI / Python (or equivalent Java stack, based on design).
- **Frontend:** HTML, CSS, JavaScript for user interaction.
- **Authentication:** Optionally interfaces with the institution's existing Single Sign-On (SSO) system.
- **File Storage:** Supports PDF/JPEG/PNG formats for uploaded documents.

3.4 Communications Interfaces

- All communication will take place over secure **HTTPS protocol**.
- The system should support operation both on the **institutional intranet** and via the **public internet**.
- API endpoints may be exposed for integration with external systems (such as government verification services in future extensions).
- The system must ensure reliable request/response handling with appropriate error messages in case of failure.

4. System Features

4.1 Feature 1 – User Authentication

Description:
The system will provide secure login functionality for students, faculty, and administrators.
Inputs: Username, password.
Outputs: User is redirected to the appropriate dashboard (Student, Faculty, Admin).
Preconditions: User must be registered in the system.
Postconditions: User session is created and access is granted.
Normal Flow:

1. User opens the login page.
2. User enters valid credentials.
3. System verifies credentials from the database.
4. System grants access to the appropriate dashboard.

Alternative Flow:

- If invalid credentials are entered, the system shows an error message and prompts for re-entry.

4.2 Feature 2 – Document Upload and Verification

Description:
Students can upload scanned or digital copies of their lost documents for verification against the institutional database.
Inputs: Scanned copy (PDF/JPEG/PNG), student ID, document ID.
Outputs: Verification status (Valid / Invalid / Duplicate).
Preconditions: User must be logged in as a student.
Postconditions: Verification request is logged in the system.
Normal Flow:

1. Student logs in and navigates to “Upload Document.”
2. Student uploads a scanned copy of the lost document.
3. System checks the document against institutional records.
4. System updates the status as “Pending Verification.”
5. Admin later approves or rejects the verification.

Alternative Flow:

- If the uploaded file format is not supported, the system shows an error message.
- If the document is tampered or does not match, status is marked as “Invalid.”

4.3 Feature 3 – Status Tracking

Description:
Students can track the progress of their verification requests in real time.
Inputs: Student ID or Request ID.
Outputs: Verification request status (Pending / Approved / Rejected).
Preconditions: Student must be logged in.
Postconditions: Student receives updated request information.
Normal Flow:

1. Student logs in and selects “Track Status.”
2. System fetches status of all pending and past requests.
3. System displays results on the student dashboard.

4.4 Feature 4 – Admin Management

Description:
Administrators manage all verification reuests, review uploaded documents, and update records.
Inputs: Verification request ID, document details.
Outputs: Updated status of the request (Approved / Rejected).
Preconditions: Admin must be logged in.
Postconditions: Verification database is updated.
Normal Flow:

1. Admin logs in and opens the pending requests list.

2. Admin reviews uploaded documents.
3. Admin updates the status to “Approved” or “Rejected.”
4. System notifies the student about the result.

4.5 Feature 5 – Report Generation

Description:
The system generates a verification report for each approved or rejected request.
Inputs: Verification request ID.
Outputs: PDF/printable report of the verification result.
Preconditions: Request must have been processed (Approved/Rejected).
Postconditions: Report is stored and made available for download.
Normal Flow:

1. Student/Admin requests a verification report.
2. System compiles the request details and result.
3. System generates a PDF or on-screen report.
4. User downloads or prints the report.

Use Cases :-

This section describes simplified use cases of the Lost Document Verifier for College Records system for SRS documentation. These use cases reflect the key interactions between users (students and admins) and the system to demonstrate core functionality.

UC01: Submit Lost Document Request

- Primary Actor: Student
- Secondary Actor: Lost Document System
- Precondition: Student is logged in
- Trigger: Student clicks “Report Lost Document”

Main Success Scenario:

1. Student navigates to “Lost Document Request” form.
2. Student selects the type of document lost (e.g., mark sheet, ID card).
3. Student uploads a valid proof document (e.g., affidavit).
4. Student submits the request.
5. System stores the request and shows a confirmation message.

Exception Scenario:

- If required fields are missing:
  → System displays: "Please fill all required details and upload the necessary documents."

UC02: Verify and Process Request

- Primary Actor: Admin
- Secondary Actor: Verification Engine
- Precondition: Admin is authenticated
- Trigger: Admin opens the list of new requests

Main Success Scenario:

1. Admin logs in and accesses the dashboard.
2. Admin reviews the details and attached proof.
3. Admin verifies document loss from records.
4. Admin approves or rejects the request.
5. System updates status and sends notification to the student.

Exception Scenario:

- If the supporting document is unclear or missing:
  → Admin rejects the request with a reason: "Insufficient verification. Please resubmit."

UC03: Track Request Status

- Primary Actor: Student
- Secondary Actor: Request Status Handler
- Precondition: Student has submitted a lost document request
- Trigger: Student clicks "Track My Request"

Main Success Scenario:

1. Student logs in and clicks "Track My Request."
2. System retrieves current request status (e.g., Pending, Approved, Rejected).
3. Status is displayed along with any admin remarks.

Exception Scenario:

- If no request is found:
  → System displays: "No active document request found for this account."

UC04: Generate Reissue Confirmation

- Primary Actor: Admin
- Secondary Actor: Document Generator
- Precondition: Request has been approved
- Trigger: Admin clicks “Generate Reissue Confirmation”

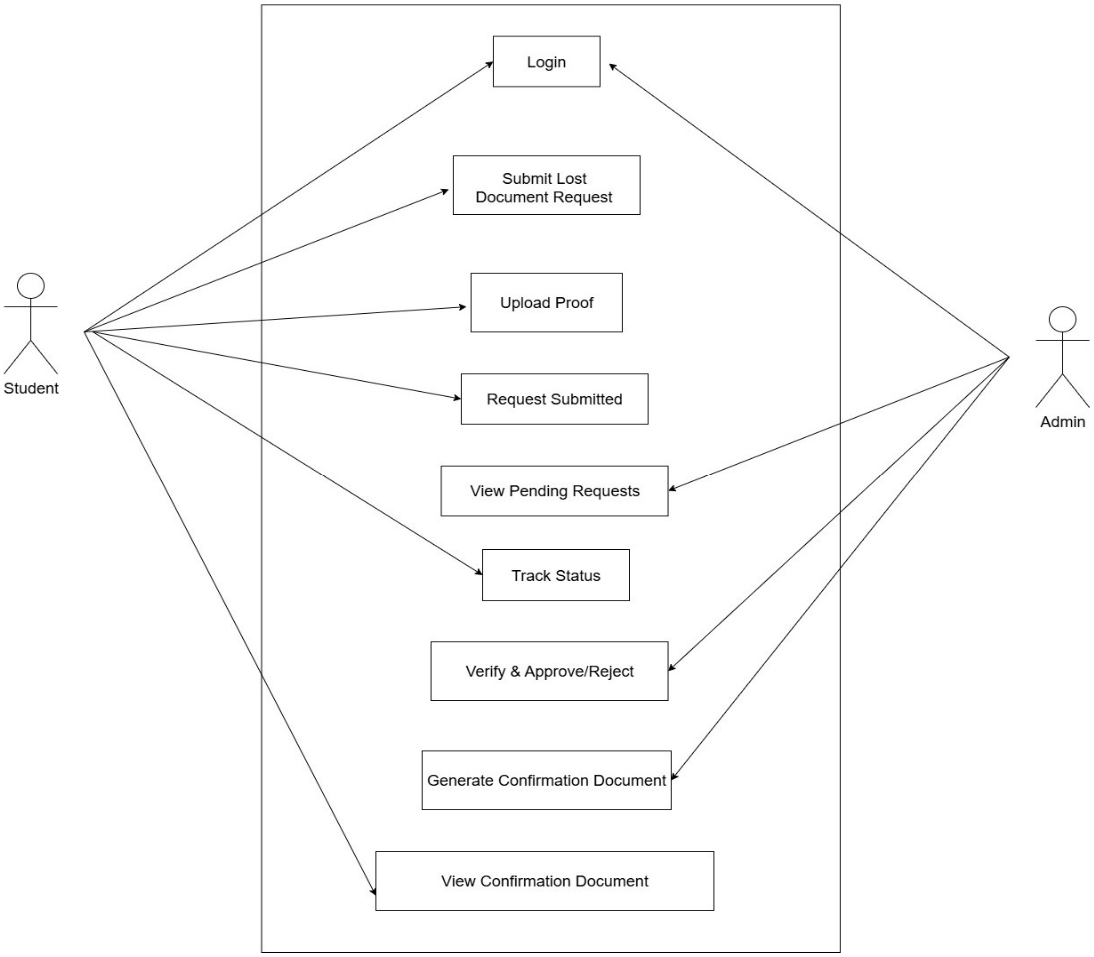
Main Success Scenario:

1. Admin opens approved request.
2. System generates a digital confirmation document (PDF or HTML).
3. Admin downloads or emails the document to the student.
4. Status is marked “Completed.”

Exception Scenario:

- If generation fails due to system error:
  → System displays: “Error generating document. Please try again later.”

USE CASE DIAGRAM



5. Other Nonfunctional Requirements

5.1 Performance Requirements

- The system shall process verification requests and return results within 3–5 seconds under normal load.
- The system shall support at least 100 simultaneous active users without performance degradation.
- The database shall be optimized to handle a minimum of 10,000 stored records without query delay exceeding 2 seconds.
- System downtime shall not exceed 2 hours per month for maintenance.

5.2 Safety Requirements

- The system shall maintain regular backups of student records and verification logs to prevent data loss.
- In the event of server failure, the system must recover data from the most recent backup.
- Only administrators can modify the database; students and staff have read-only access to institutional records.
- Any failed operation (e.g., interrupted upload) shall not corrupt or overwrite existing records.

5.3 Security Requirements

- All communication between client and server must occur over HTTPS.
- Passwords must be stored in the database using hashing and salting techniques.
- Role-based access control shall be enforced:
  - Students: Upload, track, and download reports.
  - Faculty/Staff: Assist with verification and view results.
  - Admin: Manage all records, approve/reject requests, generate reports.
- The system shall log all login attempts and verification actions for auditing.
- Document uploads must be scanned for malware and tampering before processing.

5.4 Software Quality Attributes

- **Usability:** The interface shall be simple and intuitive, requiring minimal training for first-time users.
- **Reliability:** The system shall achieve 99% uptime during operational hours.

- **Scalability:** The system shall allow future integration with government or third-party verification systems.
- **Maintainability:** Codebase and database structure shall support easy updates and bug fixes.
- **Portability:** The system shall be accessible on multiple browsers (Chrome, Firefox, Edge, Safari) and operating systems.

5.5 Business Rules

- Only registered students/staff may request document verification.
- Each lost document request shall be unique per document ID.
- Verification requests shall not be approved unless the document matches exactly with institutional records.
- Administrators have the final authority to approve or reject requests.
- Reports generated are official proof and must be stored for a minimum of 5 years.

6. Other Requirements

6.1 Database Requirements

- Use PostgreSQL 13+ with a fully normalized schema (3NF or higher) to ensure data integrity and avoid redundancy.
- Implement daily backups and maintain a disaster recovery plan to safeguard against data loss.
- Support efficient queries for document verification, status tracking, and report generation.

6.2 Internationalization Requirements

- The system shall support English as the primary language.
- The design shall allow future addition of multiple languages without major code refactoring.

6.3 Legal Requirements

- Ensure compliance with local/state student data protection regulations, including privacy and storage rules.
- Ensure copyright compliance for any shared learning materials or documents uploaded to the system.
- User consent must be obtained before storing or processing personal or document data.

6.4 Reuse Objectives

- Core code modules such as user authentication, file handling, and verification logic should be modular and reusable for future educational or institutional applications.
- The architecture should allow easy extension for other types of document verification systems.

6.5 Documentation Requirements

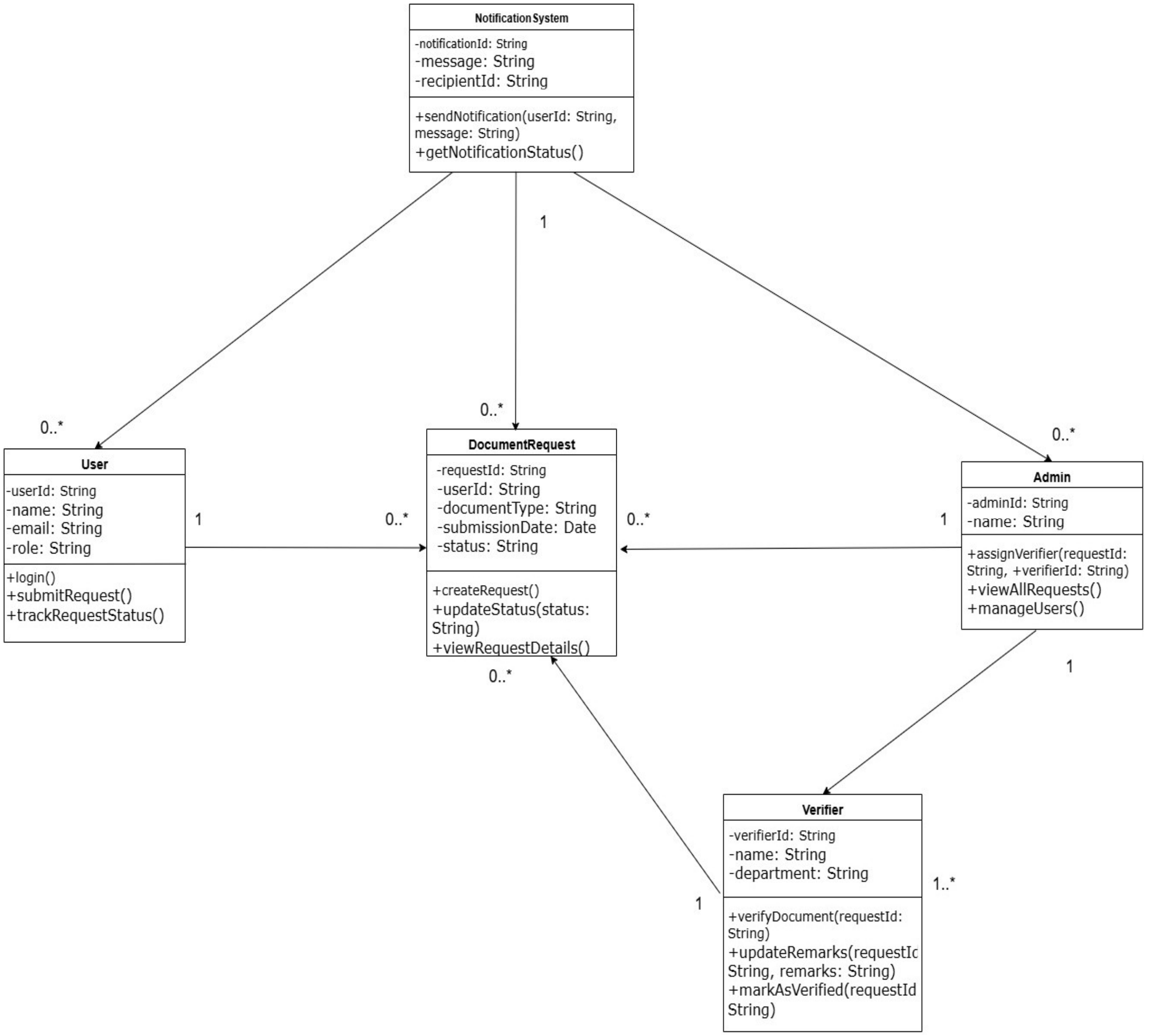
- Maintain user manuals for students, faculty, and administrators.
- Provide API documentation using tools like Swagger/OpenAPI for backend endpoints.
- Update all documentation with every major release, including tutorials, troubleshooting guides, and workflow diagrams.

6.6 Future Enhancements

- Integration of AI-based helper recommendations for students submitting documents.
- Gamification features to encourage timely submissions and engagement.
- Offline access for mobile users to track status and download reports.
- Integration with School LMS and other third-party educational platforms for streamlined document management.

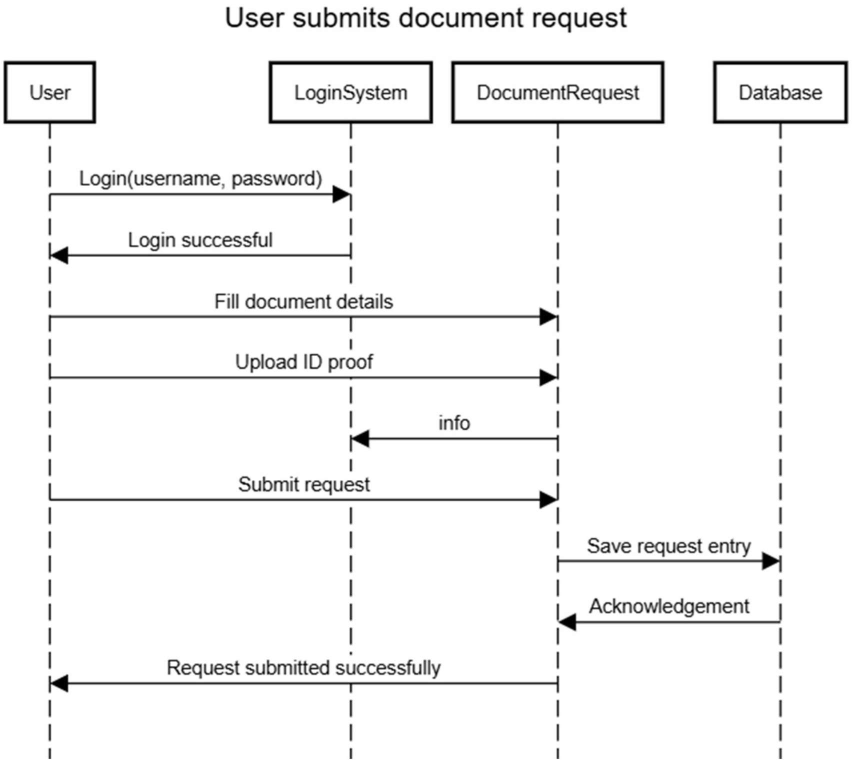
APPENDIX : ANALYSIS MODEL

CLASS DIAGRAM

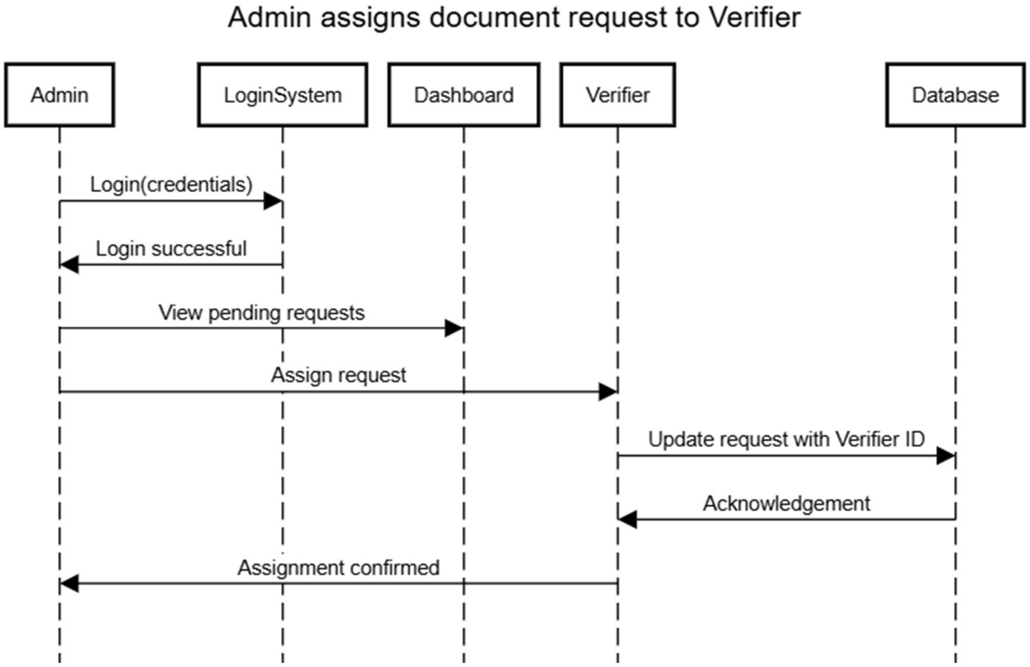


SEQUENCE DIAGRAM

SEQUENCE DIAGRAM : 1

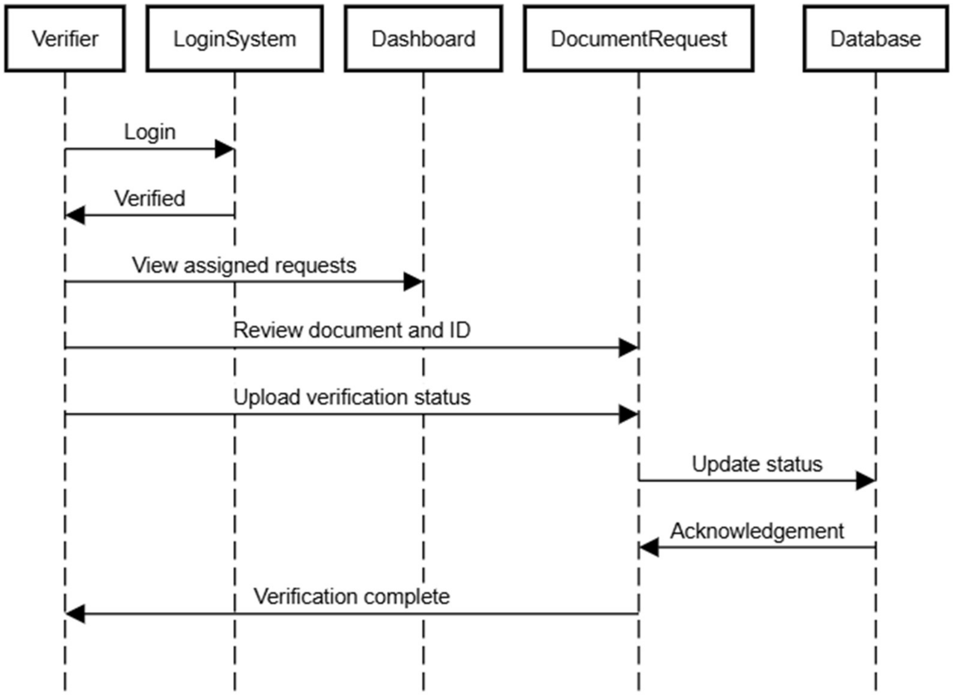


SEQUENCE DIAGRAM : 2



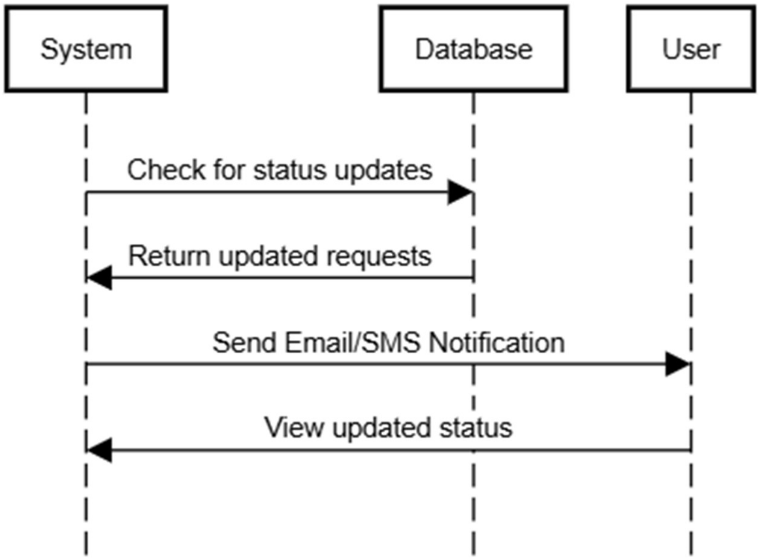
SEQUENCE DIAGRAM : 3

Verifier verifies the document request

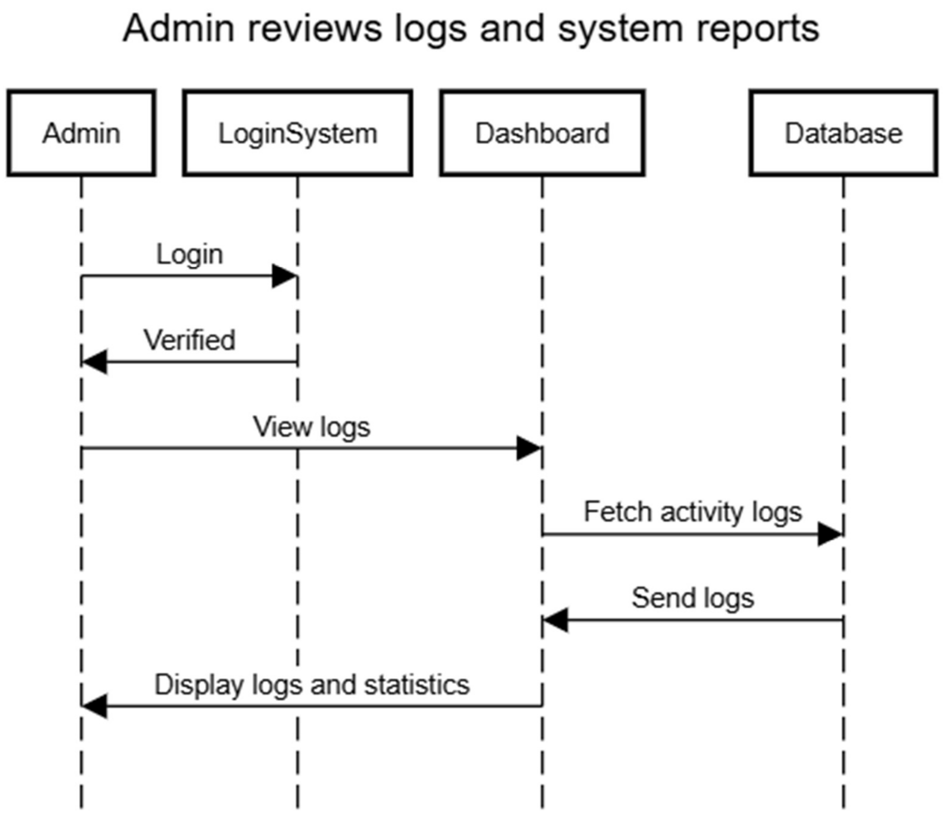


SEQUENCE DIAGRAM : 4

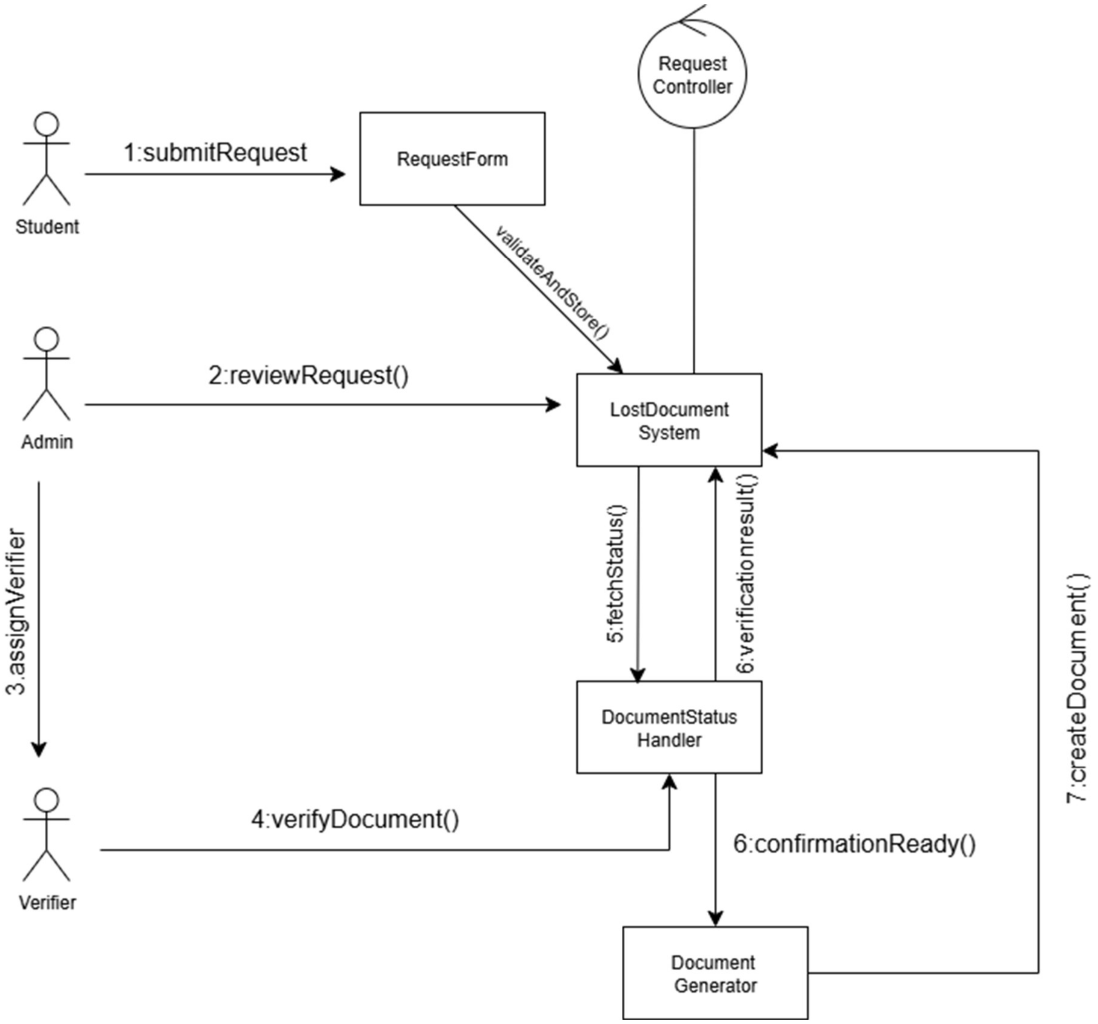
Verification Status Notification



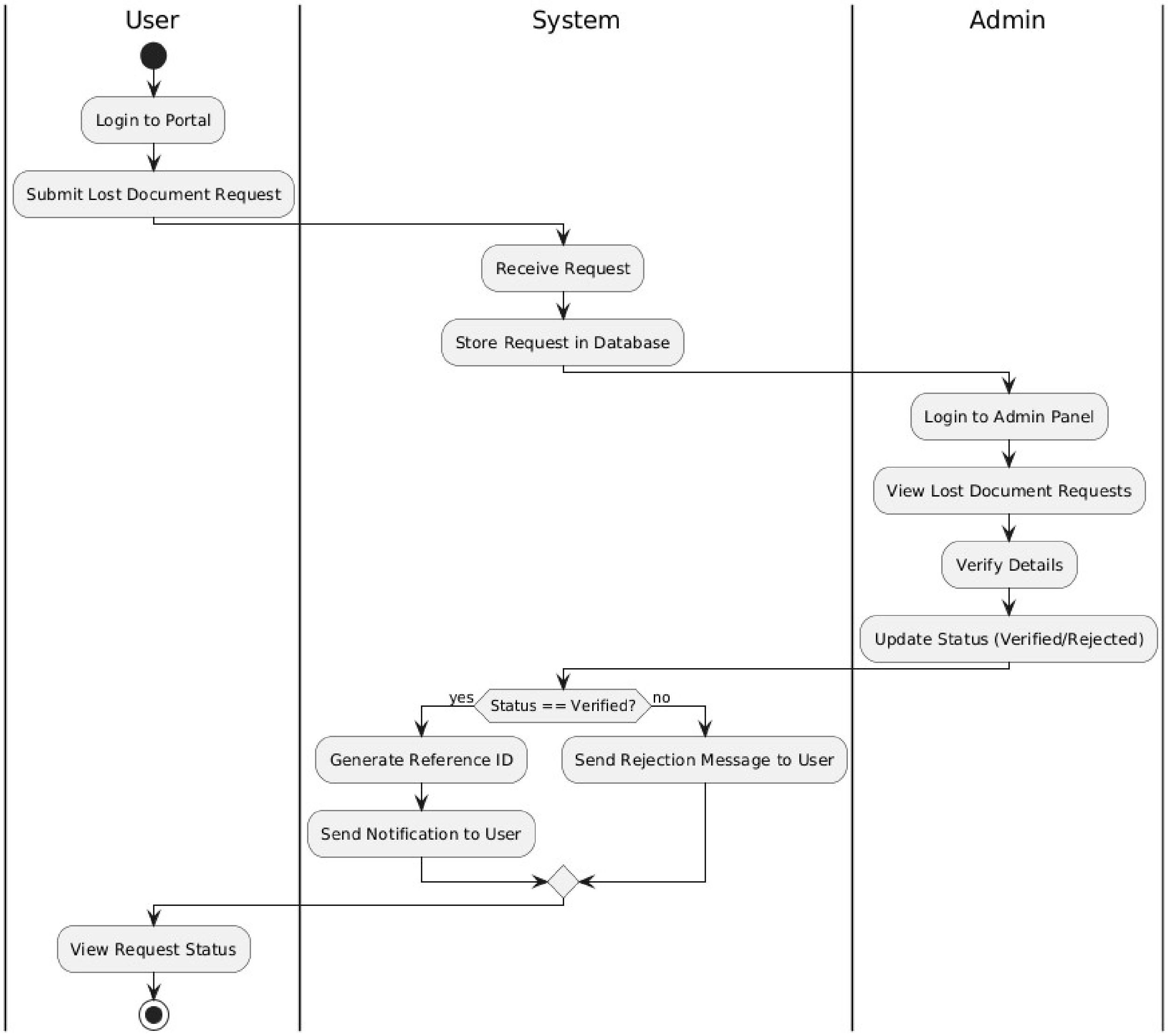
SEQUENCE DIAGRAM : 5



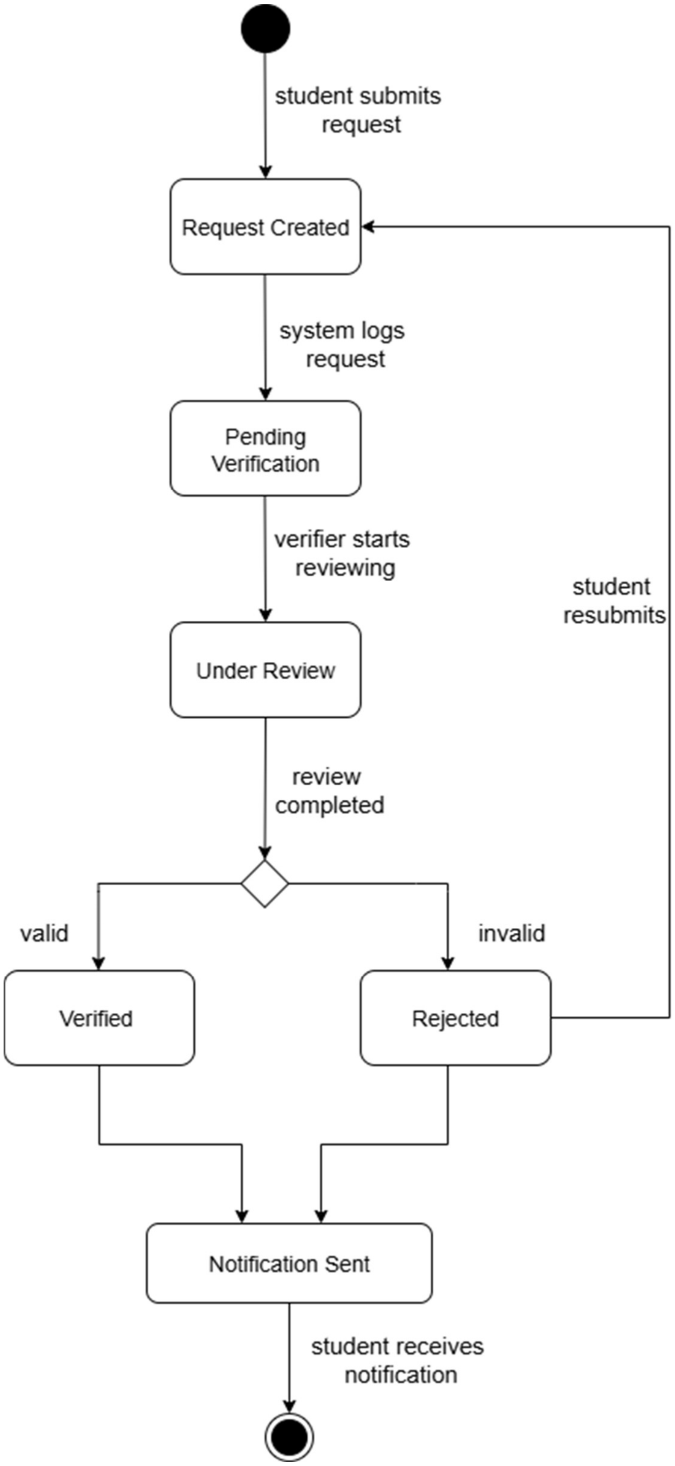
COLLABORATION DIAGRAM :-



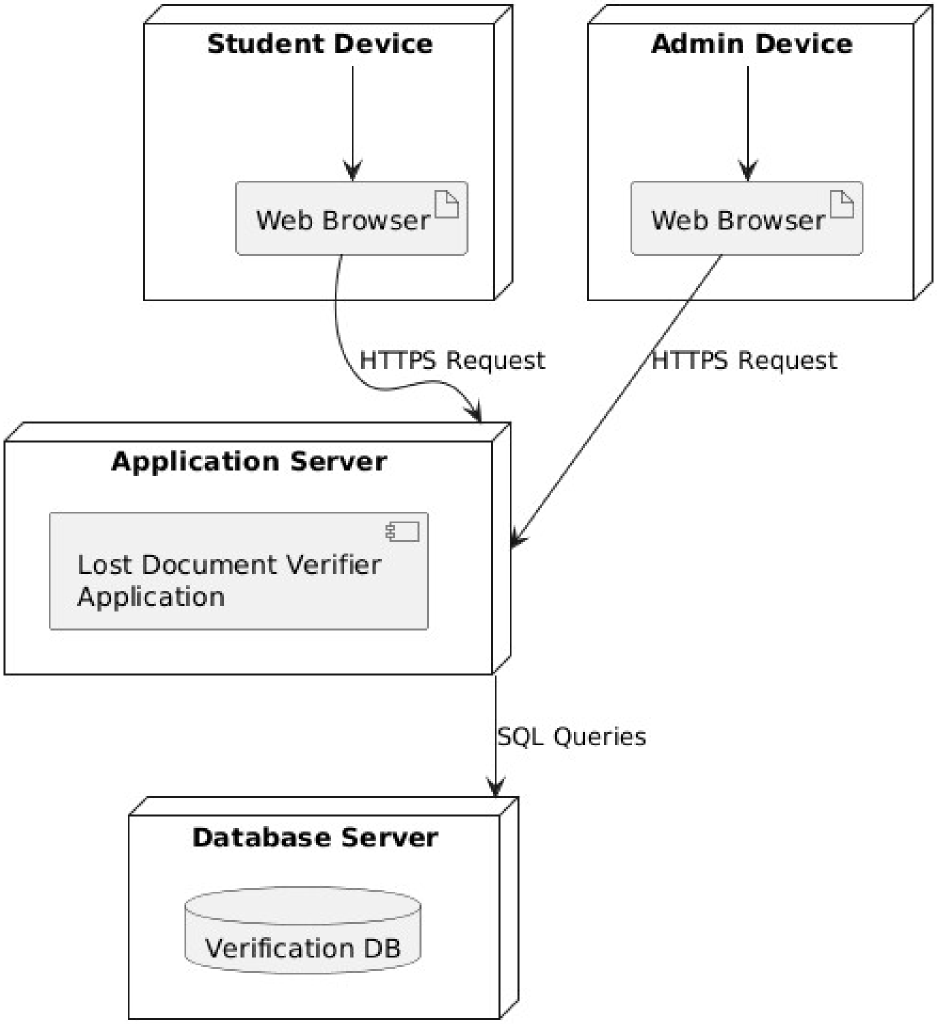
ACTIVITY DIAGRAM :-


User
System
Admin
Login to Portal
Submit Lost Document Request
Receive Request
Store Request in Database
Login to Admin Panel
View Lost Document Requests
Verify Details
Update Status (Verified/Rejected)
yes
Status == Verified?
no
Generate Reference ID
Send Rejection Message to User
Send Notification to User
View Request Status

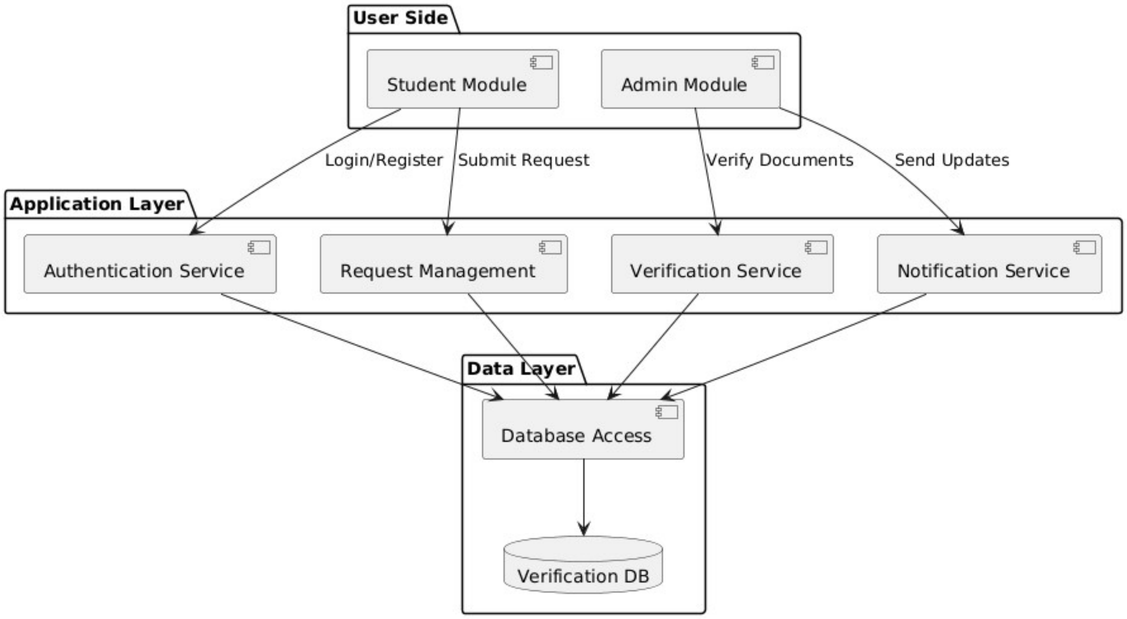
STATE DIAGRAM :-



DEPLOYMENT DIAGRAM :-



COMPONENT DIAGRAM :-



SOURCE CODE :-

app.py

```
import os
from flask import Flask, render_template, request, redirect, session, url_for, send_from_directory, flash,
make_response
from werkzeug.utils import secure_filename
from werkzeug.security import generate_password_hash, check_password_hash
from config import get_db_connection
UPLOAD_FOLDER = os.path.join(os.path.dirname(__file__), 'uploads')
ALLOWED_EXT = {'pdf','png','jpg','jpeg','doc','docx'}
if not os.path.exists(UPLOAD_FOLDER):
    os.makedirs(UPLOAD_FOLDER)
app = Flask(__name__)
app.secret_key = 'replace_with_a_strong_secret'
app.config['UPLOAD_FOLDER'] = UPLOAD_FOLDER
app.config['MAX_CONTENT_LENGTH'] = 8 * 1024 * 1024
def allowed_file(filename):
    return '.' in filename and filename.rsplit('.',1)[1].lower() in ALLOWED_EXT
@app.route('/')
def home():
    return render_template('index.html')
@app.route('/about')
def about():
    return render_template('about.html')
@app.route('/login', methods=['GET','POST'])
def login():
    if request.method == 'POST':
        email = request.form['email'].strip()
        password = request.form['password']
        conn = get_db_connection()
        cursor = conn.cursor(dictionary=True)
        cursor.execute("SELECT * FROM users WHERE email=%s", (email,))
        user = cursor.fetchone()
        cursor.close()
        conn.close()
        if user and check_password_hash(user['password'], password):
            session['user_id'] = user['id']
            session['role'] = user['role']
            session['name'] = user['name']
            session['email'] = user['email']
            if user['role'] == 'admin':
                return redirect(url_for('admin_dashboard'))
            return redirect(url_for('dashboard'))
        flash('Invalid credentials', 'danger')
    return render_template('login.html')
@app.route('/register', methods=['GET','POST'])
def register():
    if request.method == 'POST':
        name = request.form['name'].strip()
        email = request.form['email'].strip()
        password = request.form['password']
        role = request.form.get('role','student')
        enrollment_no = request.form.get('enrollment_no','').strip()
        department = request.form.get('department','').strip()
        hashed = generate_password_hash(password)
        conn = get_db_connection()
        cursor = conn.cursor()
        try:
            cursor.execute(
                "INSERT INTO users (name,email,password,role,enrollment_no,department) VALUES
(%s,%s,%s,%s,%s,%s)",
                (name,email,hashed,role,enrollment_no,department)
            )
            conn.commit()
            flash('Registration successful. Please login.', 'success')
            return redirect(url_for('login'))
        except Exception as e:
            conn.rollback()
            flash('Email already registered or DB error.', 'danger')
        finally:
            cursor.close()
            conn.close()
    return render_template('register.html')
@app.route('/logout')
def logout():
    session.clear()
    flash('Logged out', 'info')
    return redirect(url_for('login'))
@app.route('/dashboard')
def dashboard():
    if 'user_id' not in session or session.get('role') != 'student':
        return redirect(url_for('login'))
    user_id = session['user_id']
    conn = get_db_connection()
    cursor = conn.cursor(dictionary=True)
    cursor.execute("SELECT * FROM document_requests WHERE user_id=%s ORDER BY created_at DESC",
(user_id,))
    requests = cursor.fetchall()
    cursor.close()
    conn.close()
    return render_template('dashboard.html', requests=requests)
@app.route('/request', methods=['GET','POST'])
def request_form():
    if 'user_id' not in session or session.get('role') != 'student':
        return redirect(url_for('login'))
    if request.method == 'POST':
        doc_type = request.form['document_type'].strip()
        year = request.form.get('year_of_issue','').strip()
        details = request.form.get('additional_details','').strip()
        file_name = None
        enrollment_no = request.form.get('enrollment_no','').strip()
        department = request.form.get('department','').strip()

        uploaded = request.files.get('file')
        if uploaded and uploaded.filename:
            if allowed_file(uploaded.filename):
                filename = secure_filename(uploaded.filename)
                filename = f"{session['user_id']}_{int(__import__('time').time())}_{filename}"
                path = os.path.join(app.config['UPLOAD_FOLDER'], filename)
                uploaded.save(path)
                file_name = filename
            else:
                flash('File type not allowed.', 'danger')
                return redirect(request.url)
        conn = get_db_connection()
        cursor = conn.cursor()
        cursor.execute(
    "INSERT INTO document_requests (user_id, enrollment_no, department, document_type, year_of_issue, 
additional_details, file_name) "
    "VALUES (%s, %s, %s, %s, %s, %s, %s)",
    (session['user_id'], enrollment_no, department, doc_type, year, details, file_name)
)
        conn.commit()
        cursor.close()
        conn.close()
        flash('Request submitted successfully.', 'success')
        return redirect(url_for('dashboard'))
    return render_template('request_form.html')
@app.route('/verification_result')
def verification_result():
    if 'user_id' not in session or session.get('role') != 'student':
        return redirect(url_for('login'))
    user_id = session['user_id']
    conn = get_db_connection()
    cursor = conn.cursor(dictionary=True)
    cursor.execute("SELECT * FROM document_requests WHERE user_id=%s ORDER BY created_at DESC", 
(user_id,))
    requests = cursor.fetchall()
    cursor.close()
    conn.close()
    return render_template('verification_result.html', requests=requests)
@app.route('/download_certificate/<int:req_id>')
def download_certificate(req_id):
    if 'user_id' not in session:
        return redirect(url_for('login'))

    conn = get_db_connection()
    cursor = conn.cursor(dictionary=True)
    cursor.execute("""
    SELECT dr.*, u.name,
       COALESCE(dr.enrollment_no, u.enrollment_no) AS enrollment_no,
       COALESCE(dr.department, u.department) AS department
FROM document_requests dr
JOIN users u ON dr.user_id = u.id
WHERE dr.id = %s
""", (req_id,))
    rec = cursor.fetchone()
    cursor.close()
    conn.close()
    if not rec:
        flash('Record not found', 'danger')
        return redirect(url_for('dashboard'))
    if session.get('role') != 'admin' and session.get('user_id') != rec['user_id']:
        flash('Not authorized', 'danger')
        return redirect(url_for('dashboard'))

    # Render certificate template
    rendered = render_template('verification_result.html', single=rec, base_url=request.host_url)
    resp = make_response(rendered)
    resp.headers['Content-Type'] = 'text/html'
    resp.headers['Content-Disposition'] = f'attachment; filename=verification_{req_id}.html'
    return resp
@app.route('/admin_dashboard')
def admin_dashboard():
    if 'user_id' not in session or session.get('role') != 'admin':
        return redirect(url_for('login'))
    conn = get_db_connection()
    cursor = conn.cursor(dictionary=True)
    cursor.execute("""
  SELECT dr.*, u.name, u.email
  FROM document_requests dr
  JOIN users u ON dr.user_id = u.id
  ORDER BY dr.created_at DESC
""")
    requests = cursor.fetchall()
    cursor.close()
    conn.close()
    return render_template('admin_dashboard.html', requests=requests)
@app.route('/update_status/<int:req_id>/<status>')
def update_status(req_id, status):
    if 'user_id' not in session or session.get('role') != 'admin':
        return redirect(url_for('login'))
    if status not in ('Approved','Rejected'):
        flash('Invalid status', 'danger')
        return redirect(url_for('admin_dashboard'))
    conn = get_db_connection()
    cursor = conn.cursor()
    cursor.execute("UPDATE document_requests SET status=%s WHERE id=%s", (status, req_id))
    conn.commit()
    cursor.close()
    conn.close()
    flash(f'Request {status}.', 'success')
    return redirect(url_for('admin_dashboard'))
@app.route('/uploads/<filename>')
def uploaded_file(filename):
    if 'user_id' not in session:
        return redirect(url_for('login'))
    return send_from_directory(app.config['UPLOAD_FOLDER'], filename)
@app.route('/history')
def history():
    if 'user_id' not in session:
        return redirect(url_for('login'))
    conn = get_db_connection()
    cursor = conn.cursor(dictionary=True)
    cursor.execute("SELECT * FROM document_requests WHERE user_id=%s ORDER BY created_at DESC",
(session['user_id'],))
    records = cursor.fetchall()
    cursor.close()
    conn.close()
    return render_template('history.html', records=records)
if __name__ == '__main__':
    app.run(debug=True)
```

templates

about.html

```
<!DOCTYPE html>
<html>
<head>
    <title>About - Lost Document Verifier</title>
    <link rel="stylesheet" href="{{ url_for('static', filename='styles.css') }}">
</head>
<body>
    <header>
        Lost Document Verifier
    </header>
    <div class="navbar">
        <a href="/">Home</a>
        <a href="/about">About</a>
        <a href="/login">Login</a>
        <a href="/register">Register</a>
        <a href="/history">History</a>
    </div>
    <div class="container">
        <h1>About This Project</h1>
        <p>
            The Lost Document Verifier is designed to help institutions
            and individuals confirm the authenticity of documents and
            reduce fraud related to lost or re-issued documents.
        </p>
        <ul>
            <li>User-friendly login & registration</li>
            <li>Document verification system</li>
            <li>Complete verification history</li>
        </ul>
    </div>
    <footer>
        © 2025 Lost Document Verifier. All Rights Reserved.
    </footer>
</body>
</html>
```

admin_dashboard.html

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <title>Admin Dashboard</title>
  <link rel="stylesheet" href="{{ url_for('static', filename='styles.css') }}">
</head>
<body>
  <div class="container">
    <h1>Admin Dashboard</h1>
    <p>Welcome, {{ session.name }}</p>
    <a class="btn" href="{{ url_for('logout') }}">Logout</a>
    <h2>Requests</h2>
    <table>
      <tr><th>Name</th><th>Enrollment</th></th><th>Department</th><th>Document</th><th>Year</th><th>Status</th><th>File</th><th>Action</th></tr>
      {% for r in requests %}
```

```
   <tr>
    <td>{{ r.name }}<br><small>{{ r.email }}</small></td>
    <td>{{ r.enrollment_no }}</td>
    <td>{{ r.department }}</td>
    <td>{{ r.document_type }}</td>
    <td>{{ r.year_of_issue }}</td>
    <td>{{ r.status }}</td>
    <td>
     {% if r.file_name %}
      <a href="{{ url_for('uploaded_file', filename=r.file_name) }}" target="_blank">Open</a>
     {% else %}
      -
     {% endif %}
    </td>
    <td>
     <a href="{{ url_for('update_status', req_id=r.id, status='Approved') }}">Approve</a> |
     <a href="{{ url_for('update_status', req_id=r.id, status='Rejected') }}">Reject</a> |
     <a href="{{ url_for('download_certificate', req_id=r.id) }}">Download</a>
    </td>
   </tr>
   {% endfor %}
  </table>
 </div>
</body>
</html>
```

dashboard.html

```
<!DOCTYPE html>
<html>
<head>
 <meta charset="utf-8">
 <title>Student Dashboard</title>
 <link rel="stylesheet" href="{{ url_for('static', filename='styles.css') }}">
</head>
<body>
 <div class="container">
  <h1>Student Dashboard</h1>
  <p>Welcome, {{ session.name }}</p>
  <a class="btn" href="{{ url_for('request_form') }}">Submit Lost Document Request</a>
  <a class="btn" href="{{ url_for('verification_result') }}">View Verification Status</a>
  <a class="btn" href="{{ url_for('logout') }}">Logout</a>
  <h2>Your Requests</h2>
  {% if requests %}
  <table>
   <tr><th>Document</th><th>Year</th><th>Status</th><th>File</th><th>Action</th></tr>
   {% for r in requests %}
   <tr>
    <td>{{ r.document_type }}</td>
    <td>{{ r.year_of_issue }}</td>
    <td>{{ r.status }}</td>
    <td>
     {% if r.file_name %}
      <a href="{{ url_for('uploaded_file', filename=r.file_name) }}" target="_blank">View</a>
     {% else %}
      -
     {% endif %}
    </td>
    <td><a href="{{ url_for('download_certificate', req_id=r.id) }}">Download Certificate</a></td>
   </tr>
   {% endfor %}
  </table>
  {% else %}
   <p>No requests yet.</p>
  {% endif %}
 </div>
</body>
</html>
```

history.html

```
<!DOCTYPE html>
<html>
<head>
  <title>Verification History - Lost Document Verifier</title>
  <link rel="stylesheet" href="{{ url_for('static', filename='styles.css') }}">
</head>
<body>
  <header>
    Lost Document Verifier
  </header>
  <div class="navbar">
    <a href="/">Home</a>
    <a href="/about">About</a>
    <a href="/login">Login</a>
    <a href="/register">Register</a>
    <a href="/history">History</a>
  </div>
  <div class="container">
    <h1>Your Verification History</h1>
    {% if records %}
    <table>
      <tr>
        <th>ID</th>
        <th>Document Name</th>
        <th>Status</th>
        <th>Date</th>
      </tr>
      {% for row in records %}
      <tr>
        <td>{{ row.id }}</td>
        <td>{{ row.document_name }}</td>
        <td>{{ row.status }}</td>
        <td>{{ row.date }}</td>
      </tr>
      {% endfor %}
    </table>
    {% else %}
    <p>No verification records found.</p>
    {% endif %}
  </div>
  <footer>
    © 2025 Lost Document Verifier. All Rights Reserved.
  </footer></body></html>
```

index.html

```
<!DOCTYPE html>
<html>
<head>
  <title>Lost Document Verifier - Home</title>
  <link rel="stylesheet" href="{{ url_for('static', filename='styles.css') }}">
</head>
<body>
  <header>
    Lost Document Verifier
  </header>
  <div class="navbar">
    <a href="/">Home</a>
    <a href="/about">About</a>
    <a href="/login">Login</a>
    <a href="/register">Register</a>
    <a href="/history">History</a>
  </div>
  <div class="container">
    <h1>Welcome to Lost Document Verifier</h1>
    <p>
      A secure platform to verify lost or duplicate documents.
      Users can register, log in, and track verification history.
    </p>
    <a href="/login" class="btn">Get Started</a>
  </div>
  <footer>
    © 2025 Lost Document Verifier. All Rights Reserved.
  </footer>
</body>
</html>
```

login.html

```
<!DOCTYPE html>
<html>
<head>
  <title>Login - Lost Document Verifier</title>
  <link rel="stylesheet" href="{{ url_for('static', filename='styles.css') }}">
</head>
<body>
  <header>
    Lost Document Verifier
  </header>
  <div class="navbar">
    <a href="/">Home</a>
    <a href="/about">About</a>
    <a href="/login">Login</a>
    <a href="/register">Register</a>
    <a href="/history">History</a>
  </div>
  <div class="container">
    <h1>User Login</h1>
    <form method="POST">
      <label>Email:</label>
      <input type="email" name="email" required>
      <label>Password:</label>
      <input type="password" name="password" required>
      <button type="submit">Login</button>
    </form>
    <p>Don’t have an account? <a href="/register">Register here</a></p>
  </div>
  <footer>
    © 2025 Lost Document Verifier. All Rights Reserved.
  </footer>
</body>
</html>
```

register.html

```
<!DOCTYPE html>
<html>
<head>
  <title>Register - Lost Document Verifier</title>
  <link rel="stylesheet" href="{{ url_for('static', filename='styles.css') }}">
</head>
<body>
  <header>
    Lost Document Verifier
  </header>
  <div class="navbar">
    <a href="/">Home</a>
    <a href="/about">About</a>
    <a href="/login">Login</a>
    <a href="/register">Register</a>
    <a href="/history">History</a>
  </div>
  <div class="container">
    <h1>Create an Account</h1>
    <form method="POST">
      <label>Name:</label>
      <input type="text" name="name" required>
      <label>Email:</label>
      <input type="email" name="email" required>
      <label>Password:</label>
      <input type="password" name="password" required>
      <button type="submit">Register</button>
    </form>
    <p>Already have an account? <a href="/login">Login here</a></p>
  </div>
  <footer>
    © 2025 Lost Document Verifier. All Rights Reserved.
  </footer>
</body>
</html>
```

request_form.html

```
<!DOCTYPE html>
<html>
<head>
 <meta charset="utf-8">
 <title>Submit Lost Document Request</title>
 <link rel="stylesheet" href="{{ url_for('static', filename='styles.css') }}">
</head>
```

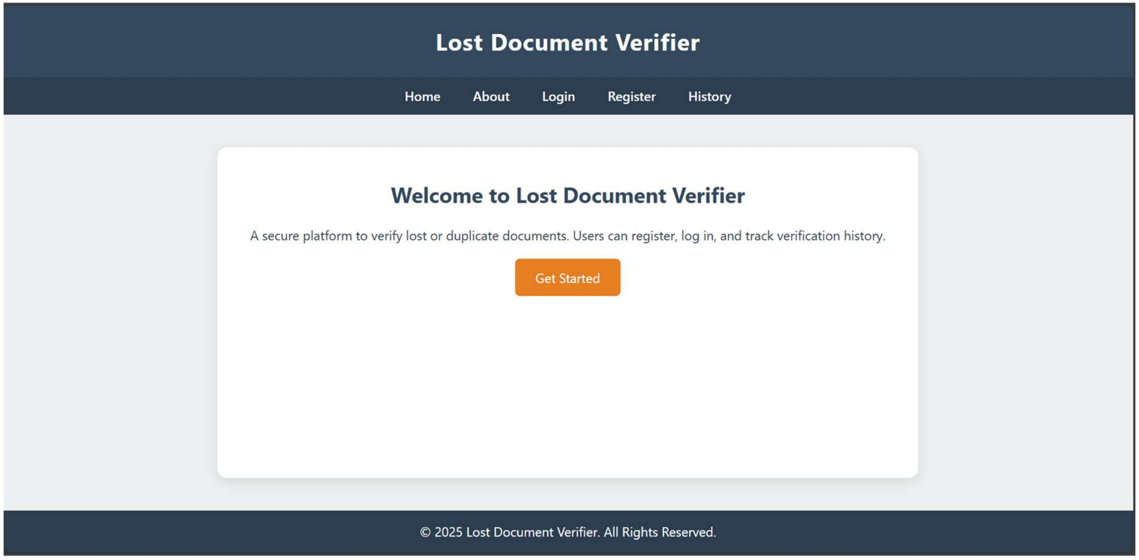
```
<body>
  <div class="container">
    <h1>Lost Document Request</h1>
    {% with messages = get_flashed_messages(with_categories=true) %}
      {% if messages %}
        <div class="messages">
          {% for cat, msg in messages %}
            <div class="msg {{cat}}">{{ msg }}</div>
          {% endfor %}
        </div>
      {% endif %}
    {% endwith %}
    <form method="POST" enctype="multipart/form-data">
      <label>Enrollment Number</label>
      <input type="text" name="enrollment_no" placeholder="Your enrollment number" required>

      <label>Department</label>
      <input type="text" name="department" placeholder="Your Department" required>

      <label>Document Type</label>
      <input type="text" name="document_type" placeholder="e.g., Marksheet, ID Card" required>

      <label>Year of Issue</label>
      <input type="text" name="year_of_issue" placeholder="e.g., 2022">

      <label>Additional details</label>
      <textarea name="additional_details" placeholder="Any extra info "></textarea>

      <label>Optional upload (scan)</label>
      <input type="file" name="file" accept=".pdf,.png,.jpg,.jpeg,.doc,.docx">

      <button type="submit">Submit Request</button>
    </form>
    <a href="{{ url_for('dashboard') }}">Back to Dashboard</a>
  </div>
</body>
</html>
```

verification_result.html

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <title>Verification Certificate</title>
  <style>
    body {
      font-family: 'Segoe UI', sans-serif;
      background: #f4f7f9;
      padding: 30px;
    }
    .certificate-container {
      max-width: 700px;
      margin: auto;
      border: 2px solid #2c3e50;
      padding: 40px;
      background: #fff;
      box-shadow: 0 0 20px rgba(0,0,0,0.1);
      text-align: center;
    }
    .certificate h1 {
      color: #2c3e50;
      margin-bottom: 30px;
    }
    .certificate h2 {
      color: #1abc9c;
      margin-bottom: 20px;
    }
    .certificate p {
      font-size: 18px;
      margin: 8px 0;
    }
    .certificate .status {
      font-weight: bold;
      color: #1abc9c;
      font-size: 20px;
      margin: 15px 0;
    }
    a.back-link {
      display: inline-block;
      margin-top: 20px;
      padding: 10px 15px;
      background: #2c3e50;
      color: #fff;
      text-decoration: none;
      border-radius: 5px;
    }
    a.back-link:hover {
      background: #1abc9c;
    }
  </style>
</head>
<body>
  <div class="certificate-container">
    {% if single %}
      <div class="certificate">
        <h1>College Verification Certificate</h1>
        <h2>Verification Details</h2>
        <p><strong>Name:</strong> {{ single.name }}</p>
        <p><strong>Enrollment No:</strong> {{ single.enrollment_no }}</p>
        <p><strong>Department:</strong> {{ single.department }}</p>
        <p><strong>Document:</strong> {{ single.document_type }}</p>
        <p><strong>Year of Issue:</strong> {{ single.year_of_issue }}</p>
        <p class="status"><strong>Status:</strong> {{ single.status }}</p>
        <p><em>Submitted on: {{ single.created_at }}</em></p>
        <p><em>Generated by Lost Document Verifier (College)</em></p>
      </div>
      <a href="{{ base_url }}dashboard">Back to Dashboard</a>
    {% else %}
      <p>No certificate data available.</p>
    {% endif %}
  </div>
</body>
</html>
```

DEMONSTRATION :-

Home Page



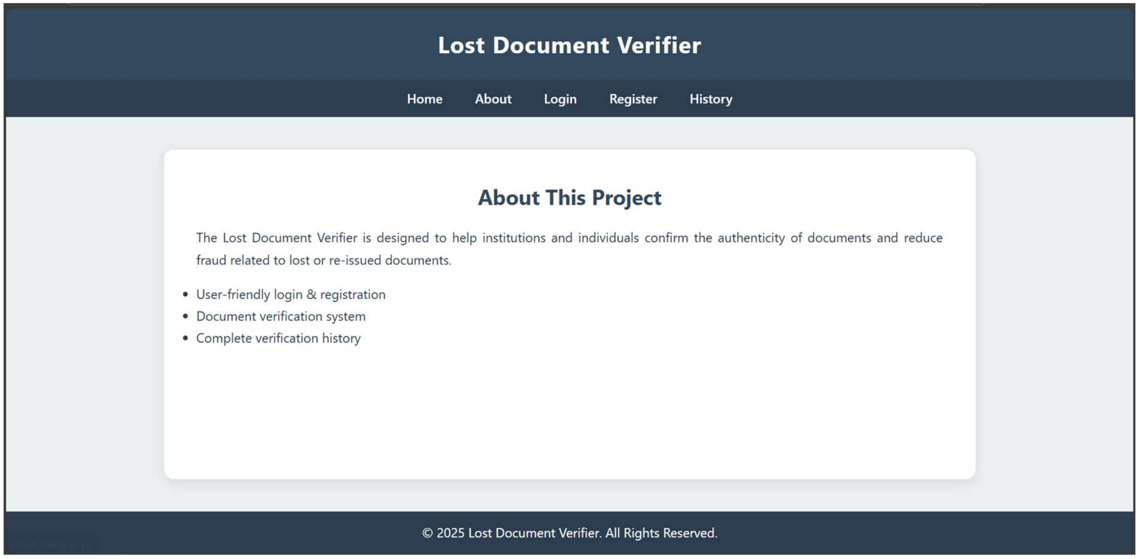
Lost Document Verifier

Welcome to Lost Document Verifier

A secure platform to verify lost or duplicate documents. Users can register, log in, and track verification history.

© 2025 Lost Document Verifier. All Rights Reserved.

About Page



Lost Document Verifier

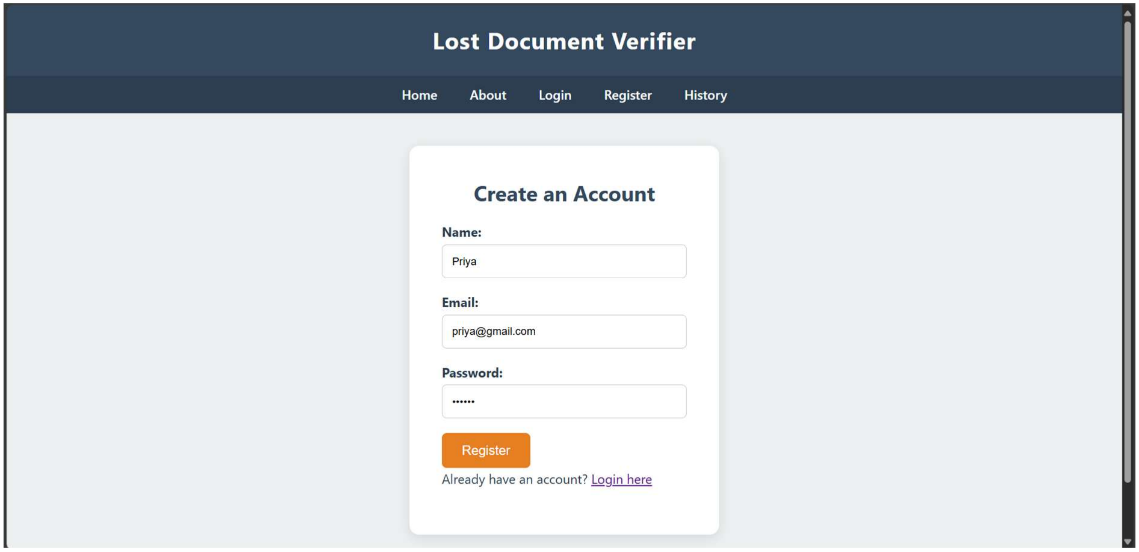
About This Project

The Lost Document Verifier is designed to help institutions and individuals confirm the authenticity of documents and reduce fraud related to lost or re-issued documents.

- User-friendly login & registration
- Document verification system
- Complete verification history

© 2025 Lost Document Verifier. All Rights Reserved.

Student Register Page



Lost Document Verifier

Create an Account

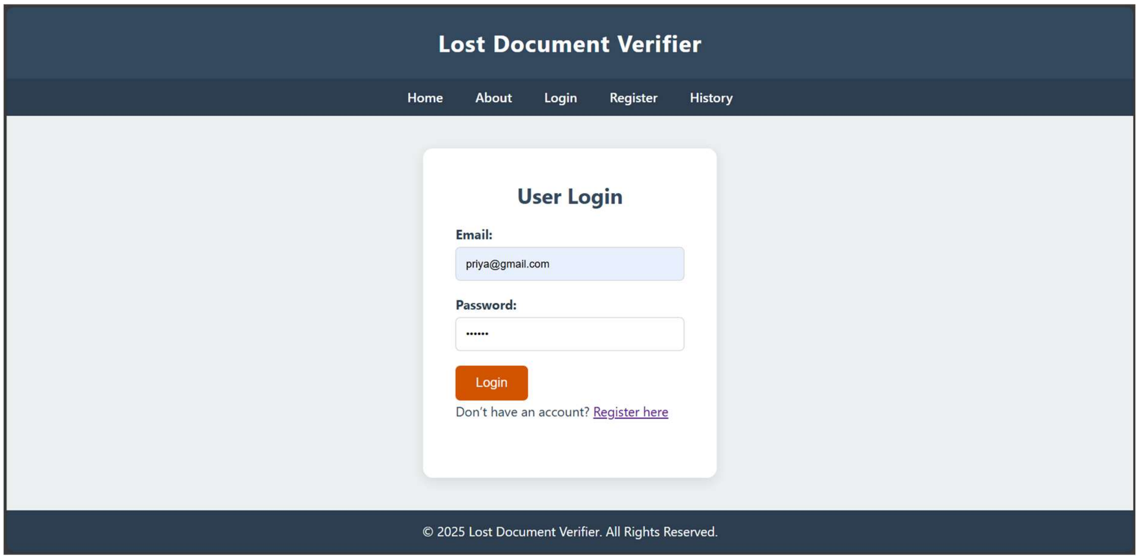
Name: Priya

Email: priya@gmail.com

Password: ••••••

Already have an account? Login here

Student Login Page



Lost Document Verifier

User Login

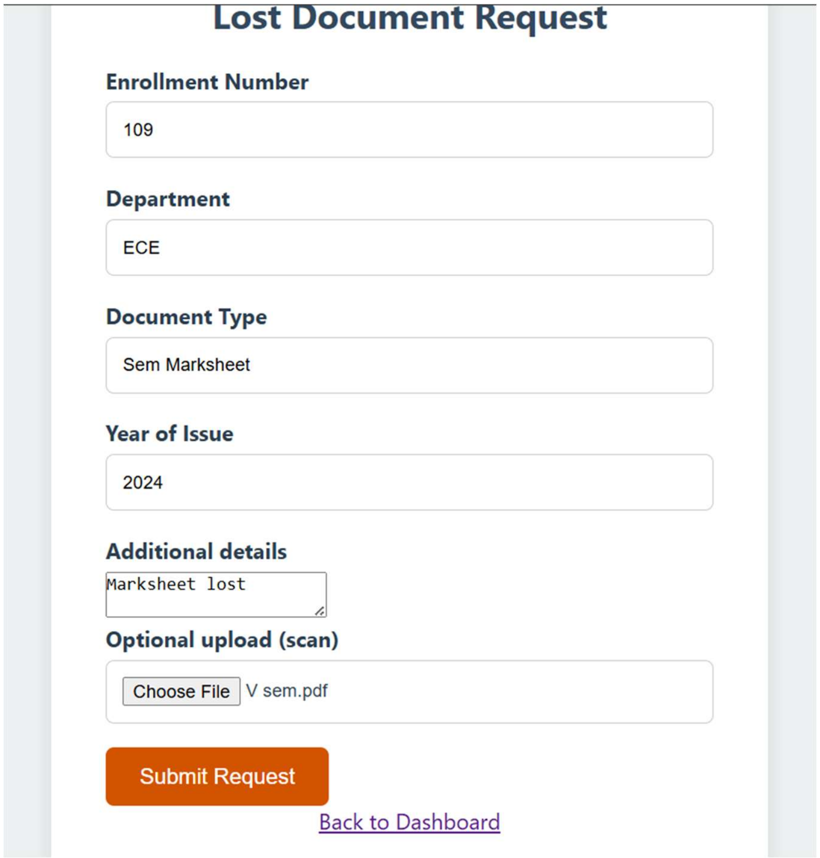
Email: priya@gmail.com

Password: ••••••

Don't have an account? Register here

© 2025 Lost Document Verifier. All Rights Reserved.

Request for Document



Lost Document Request

Enrollment Number: 109

Department: ECE

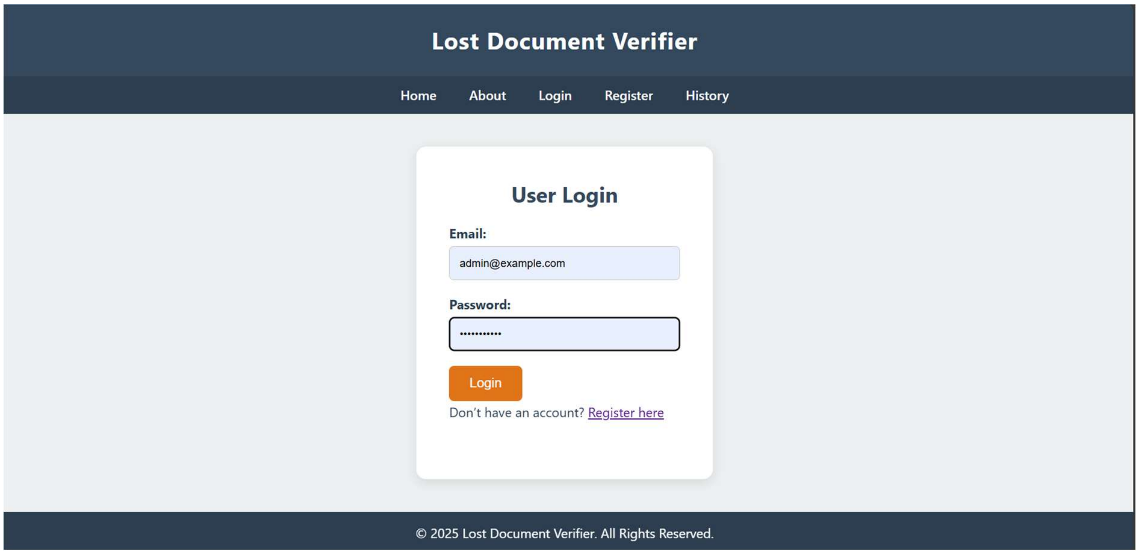
Document Type: Sem Marksheet

Year of Issue: 2024

Additional details: Marksheet lost

Optional upload (scan): Choose File V sem.pdf

Admin Login Page



Lost Document Verifier
Home
About
Login
Register
History

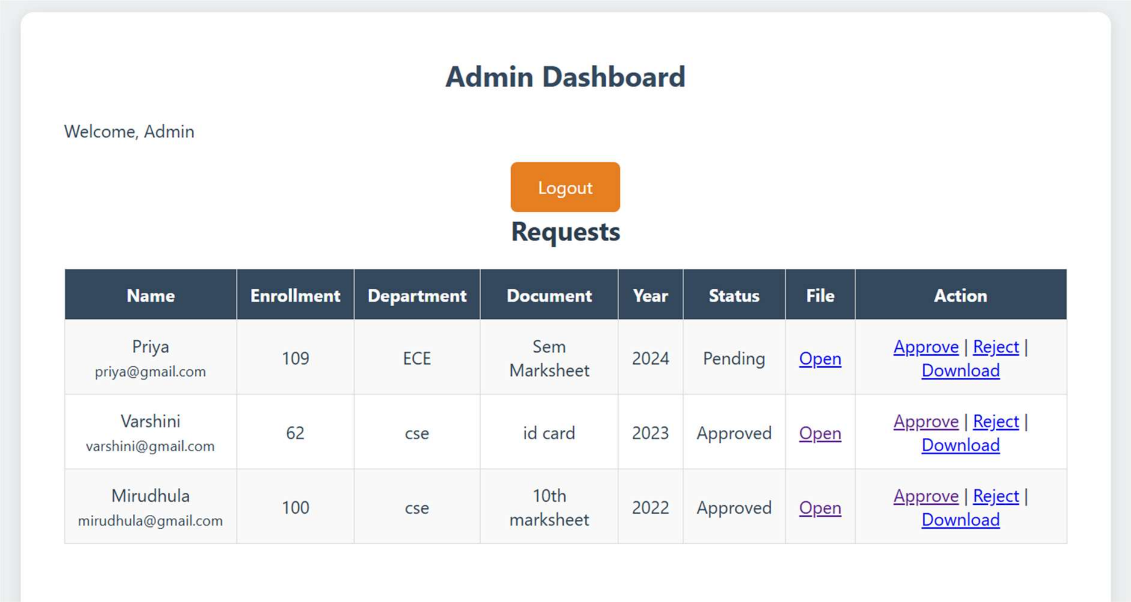
User Login

Email: admin@example.com

Password: ••••••••••••

Login
Don't have an account? Register here
© 2025 Lost Document Verifier. All Rights Reserved.

Admin Dashboard page



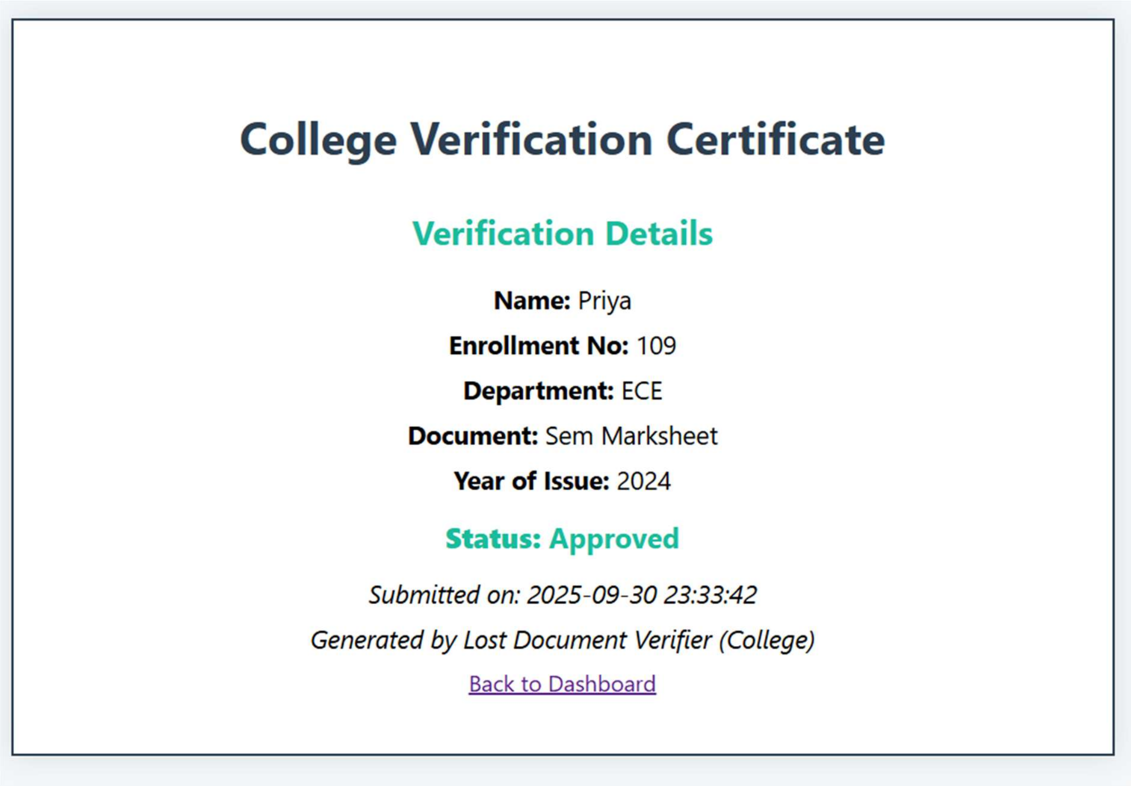
Admin Dashboard

Welcome, Admin

Requests

Name	Enrollment	Department	Document	Year	Status	File	Action
Priya priya@gmail.com	109	ECE	Sem Marksheet	2024	Pending	Open	Approve | Reject | Download
Varshini varshini@gmail.com	62	cse	id card	2023	Approved	Open	Approve | Reject | Download
Mirudhula mirudhula@gmail.com	100	cse	10th marksheet	2022	Approved	Open	Approve | Reject | Download

Admin Approval Certificate page



College Verification Certificate

Verification Details

Name: Priya

Enrollment No: 109

Department: ECE

Document: Sem Marksheet

Year of Issue: 2024

Status: Approved

Submitted on: 2025-09-30 23:33:42

Generated by Lost Document Verifier (College)

Back to Dashboard