Lost and Found Items Management System

Report

By

Abdul Raffay Bin Ilyas	**2023021**
M. Shaheer	**2023508**
M. Usman Nazir	**2023546**

Lost and Found Management System: Features and Descriptions

1. **User Authentication**
    a. **Description**: Secure login, signup, and password reset for users and admins. Users log in with email/password, with separate workflows for regular users (`dashboard.py`) and admins (`admin.py`). Password resets update the `users` table. Session management uses `current_user.txt`.
    b. **Files**: `login.py`, `forget.py`
2. **User Dashboard**
    a. **Description**: Central hub for users to claim items, post lost/found items, or log out. Validates user session via `current_user.txt` and redirects to appropriate scripts (`claim.py`, `post.py`, `login.py`).
    b. **File**: `dashboard.py`
3. **Claim Lost Items**
    a. **Description**: Users browse non-returned items (title, location, image) and submit claims with loss date/location. Claims are stored in the `claims` table as "Pending" for admin review.
    b. **File**: `claim.py`
4. **Post Lost/Found Items**
    a. **Description**: Users report lost or found items via a form (title, description, location, date, status, image). Images are saved to an `images` folder, and data is inserted into the `items` table.
    b. **File**: `post.py`
5. **Admin Dashboard**
    a. **Description**: Admins manage items, claims, users, and deleted records via a tabbed interface. Features include approving/deleting claims, deleting items/users, restoring deleted records, and syncing data to Firebase Firestore. Actions are logged for auditability.
    b. **File**: `admin.py`
    c. **Tabs**: Posted Items, Claim Requests, Users, Deleted Records (Items, Users, Claims), Recent Actions
6. **PostgreSQL Integration**

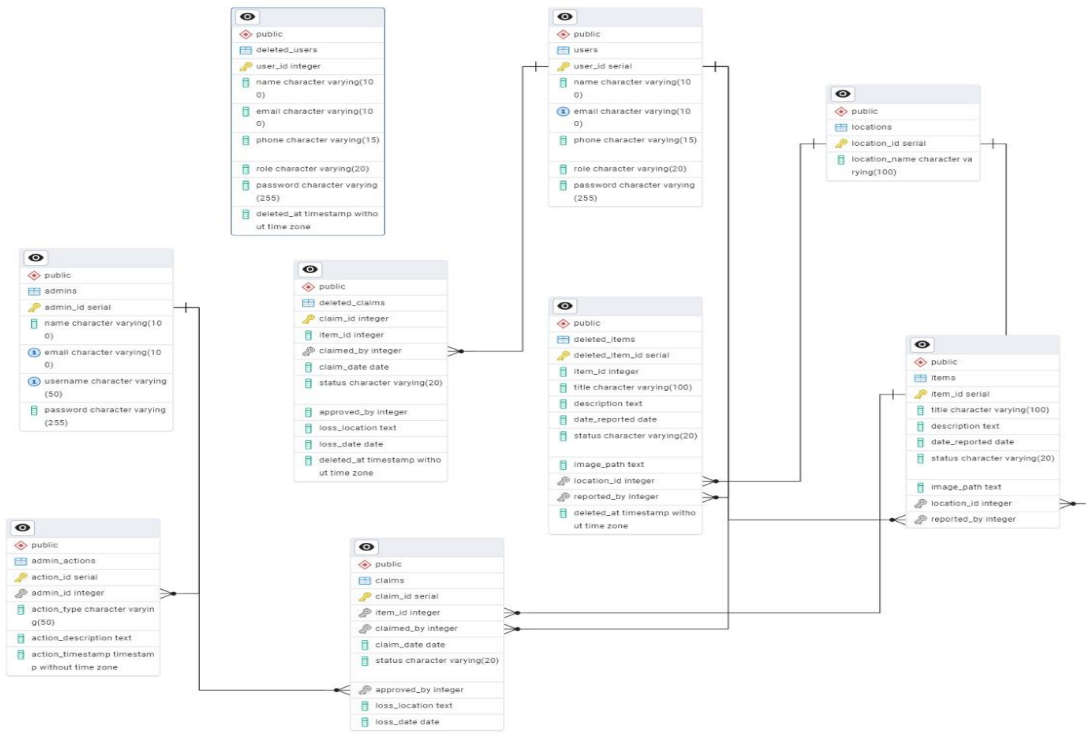
a. **Description**: Stores data in tables (`users`, `admins`, `items`, `claims`, `locations`, `deleted_items`, `deleted_users`, `deleted_claims`, `admin_actions`) with secure CRUD operations. Configuration is read from `db.ini`.
b. **Files**: All scripts

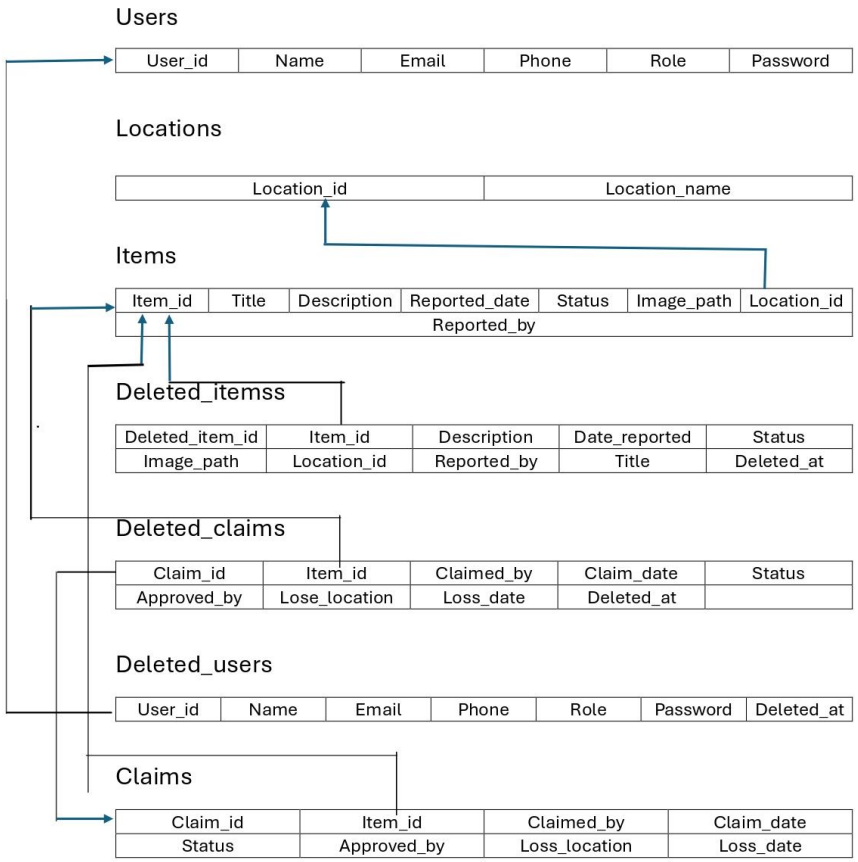
7. **Firebase Firestore Backup**
   a. **Description**: Admins upload data (items, users, claims, deleted records, admin actions) to Firestore collections for cloud storage/backup, using credentials from `fb.json`.
   b. **File**: `admin.py`
8. **Image Handling**
   a. **Description**: Supports image uploads for items (saved to `images` folder) and displays them in claim/admin interfaces. Fallbacks to gray placeholders for missing images.
   b. **Files**: `claim.py`, `post.py`, `admin.py`
9. **Error Handling & Feedback**
   a. **Description**: Robust error handling with message boxes for database errors, invalid inputs, or session issues, ensuring clear user feedback.
   b. **Files**: All scripts
10. **Responsive Tkinter GUI**
   a. **Description**: Modern, non-resizable GUI with scrollable lists, tabs, dropdowns, date pickers, and styled buttons. Uses a purple/white color scheme (`#3b1d5e`, `#5e3aca`) and includes branding images (`pic.jpg`, `logo.png`).
   b. **Files**: All scripts
11. **Session Management**
   a. **Description**: Maintains user sessions via `current_user.txt`. Redirects to login if no valid session exists. Logout deletes the session file.
   b. **Files**: `claim.py`, `dashboard.py`, `post.py`

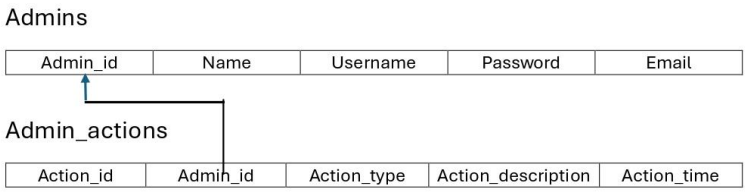
12. **Admin Action Audit Trail**

a. **Description**: Logs admin actions (e.g., claim approvals, deletions) in the `admin_actions` table and displays them in the "Recent Actions" tab for transparency.
b. **File**: `admin.py`

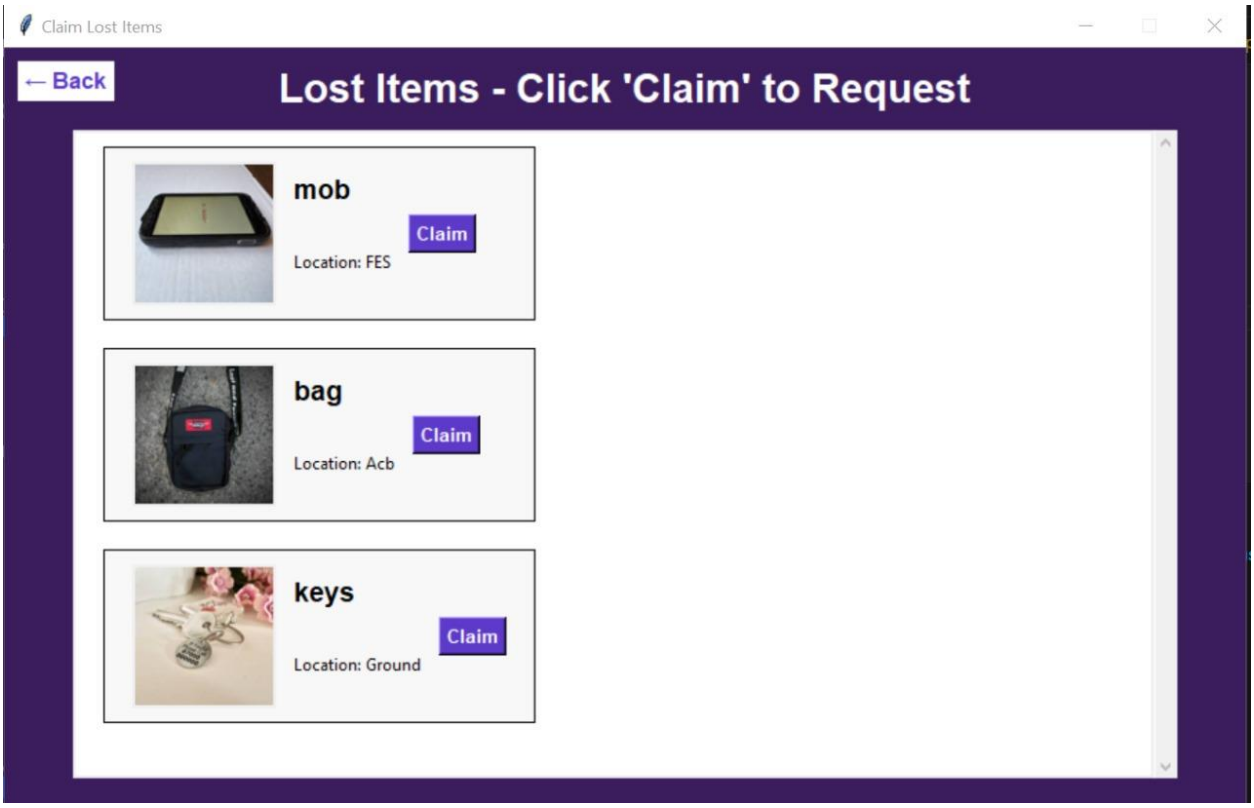
ER Diagram and Schema Mappings

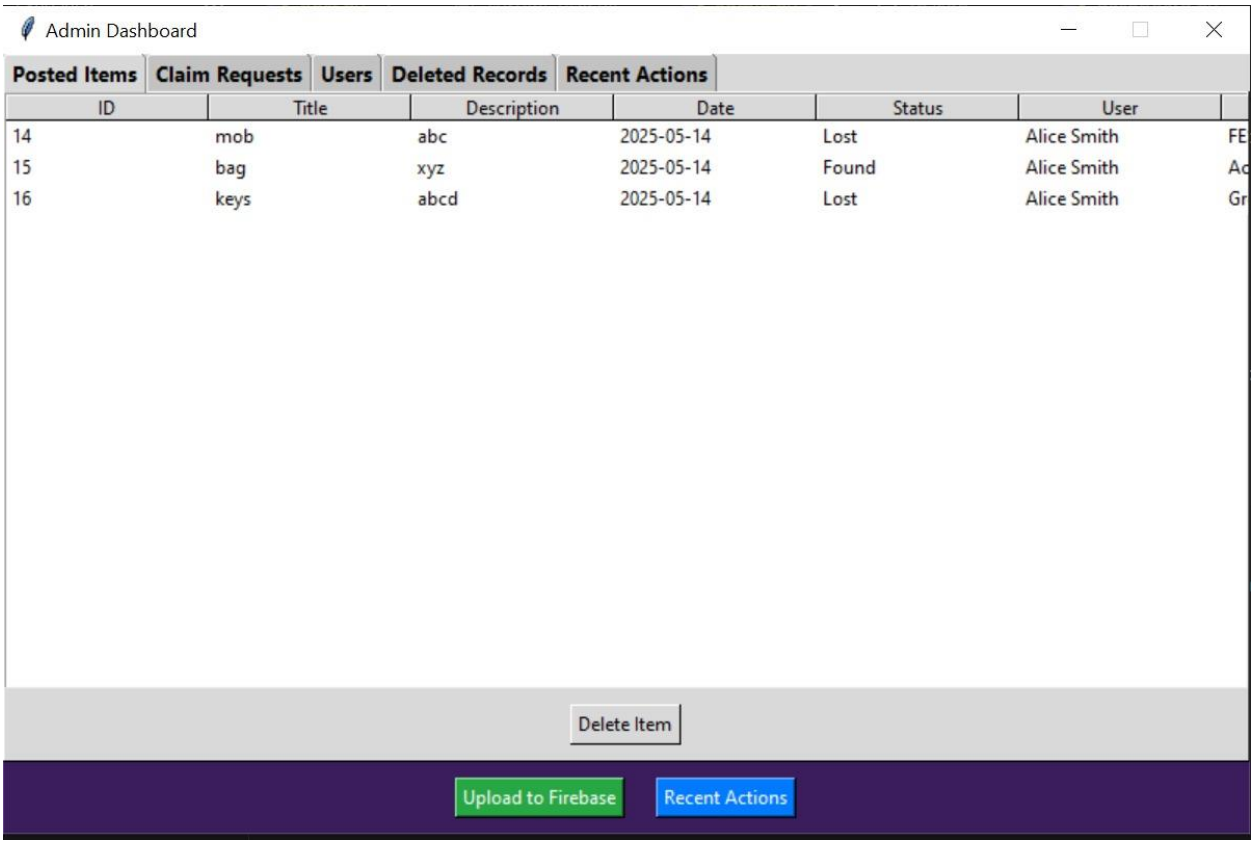


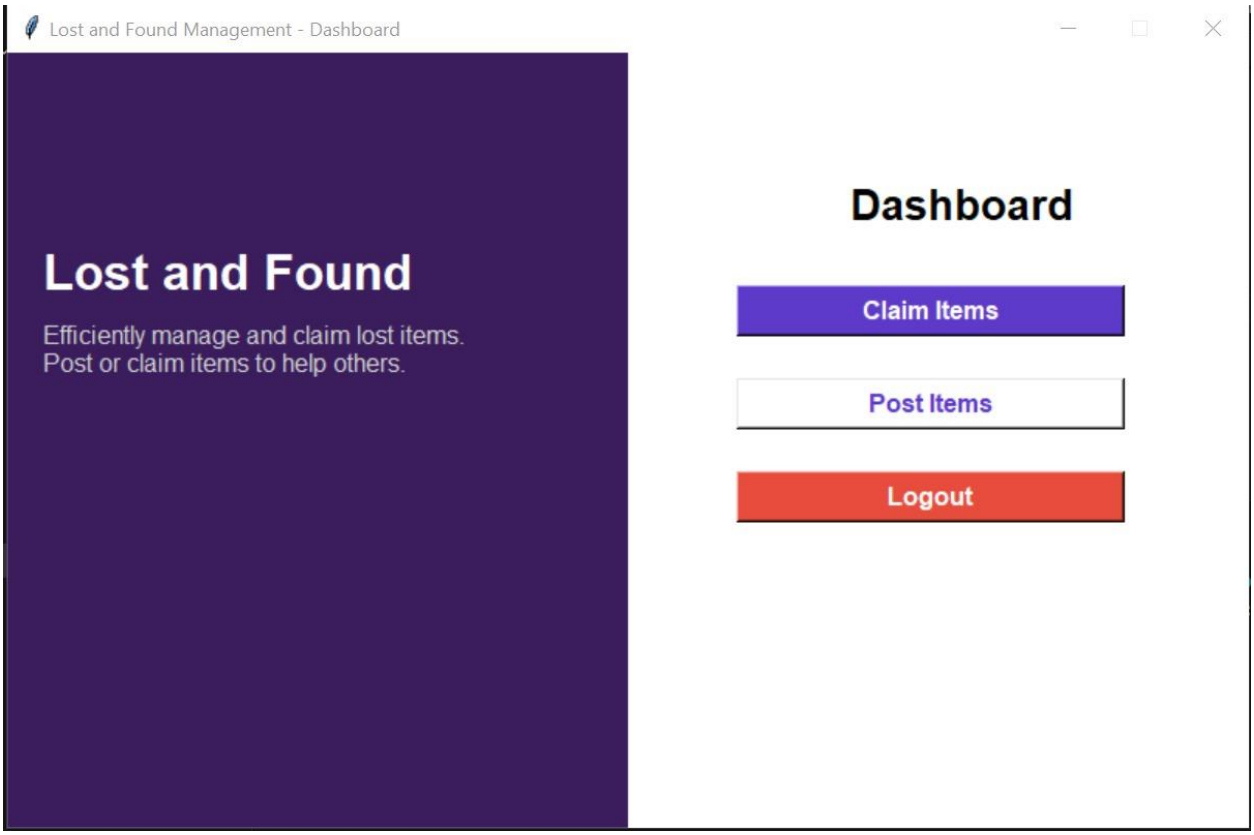


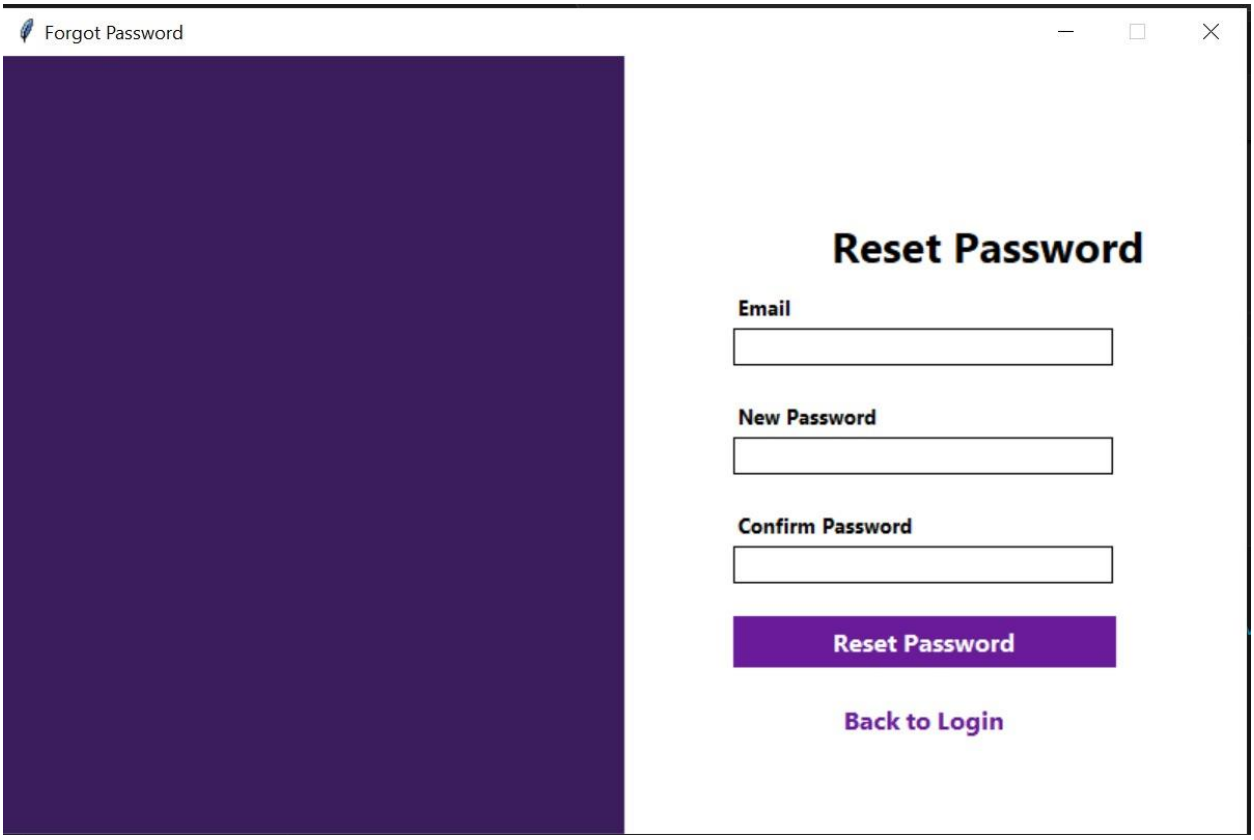


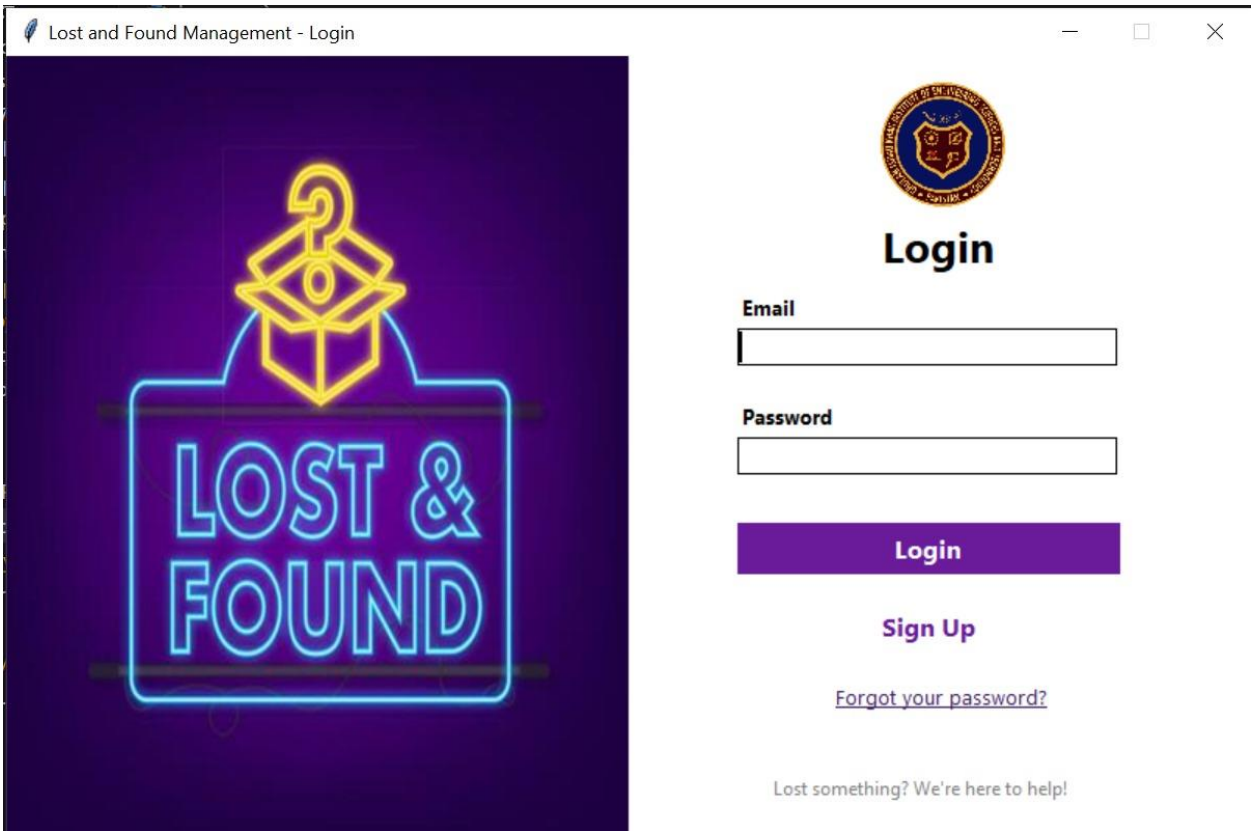
Screenshots

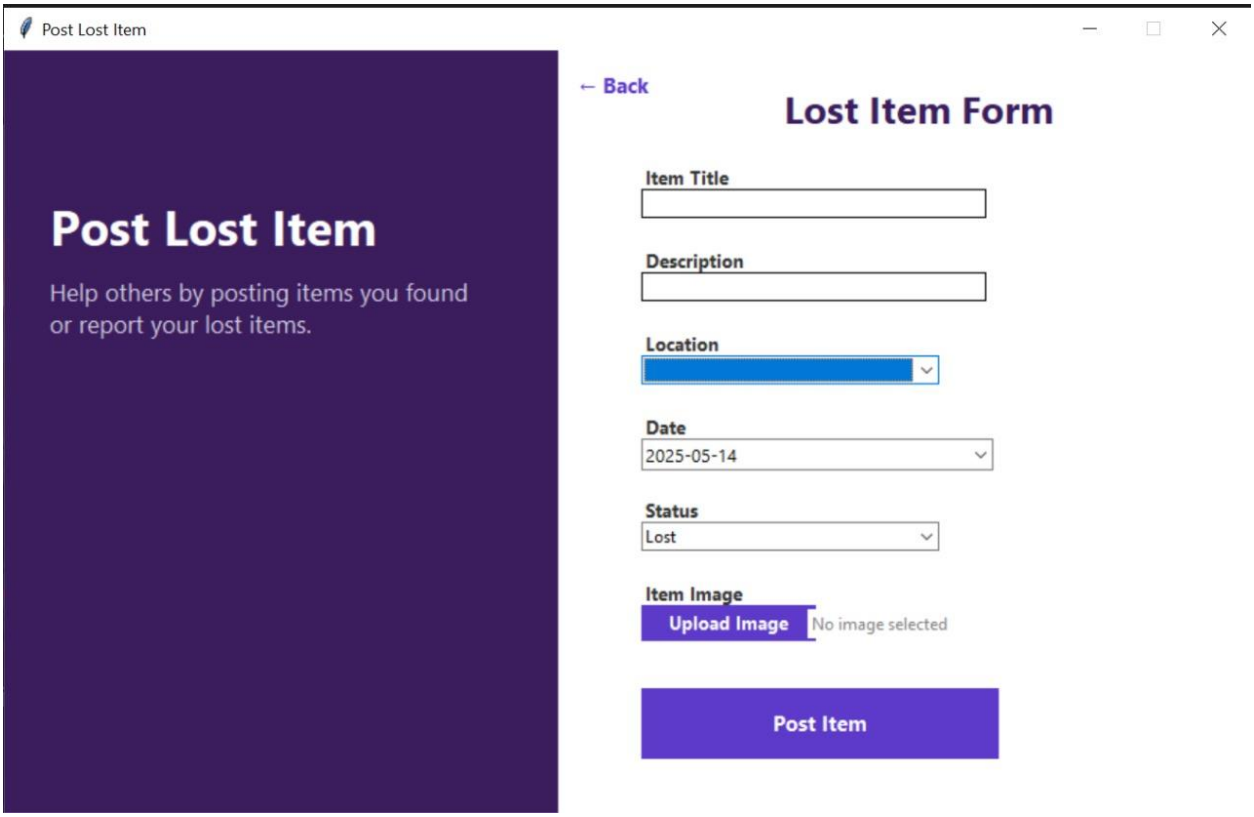

Claim Lost Items
← Back
Lost Items - Click 'Claim' to Request
mob
Claim
Location: FES
bag
Claim
Location: Acb
keys
Claim
Location: Ground

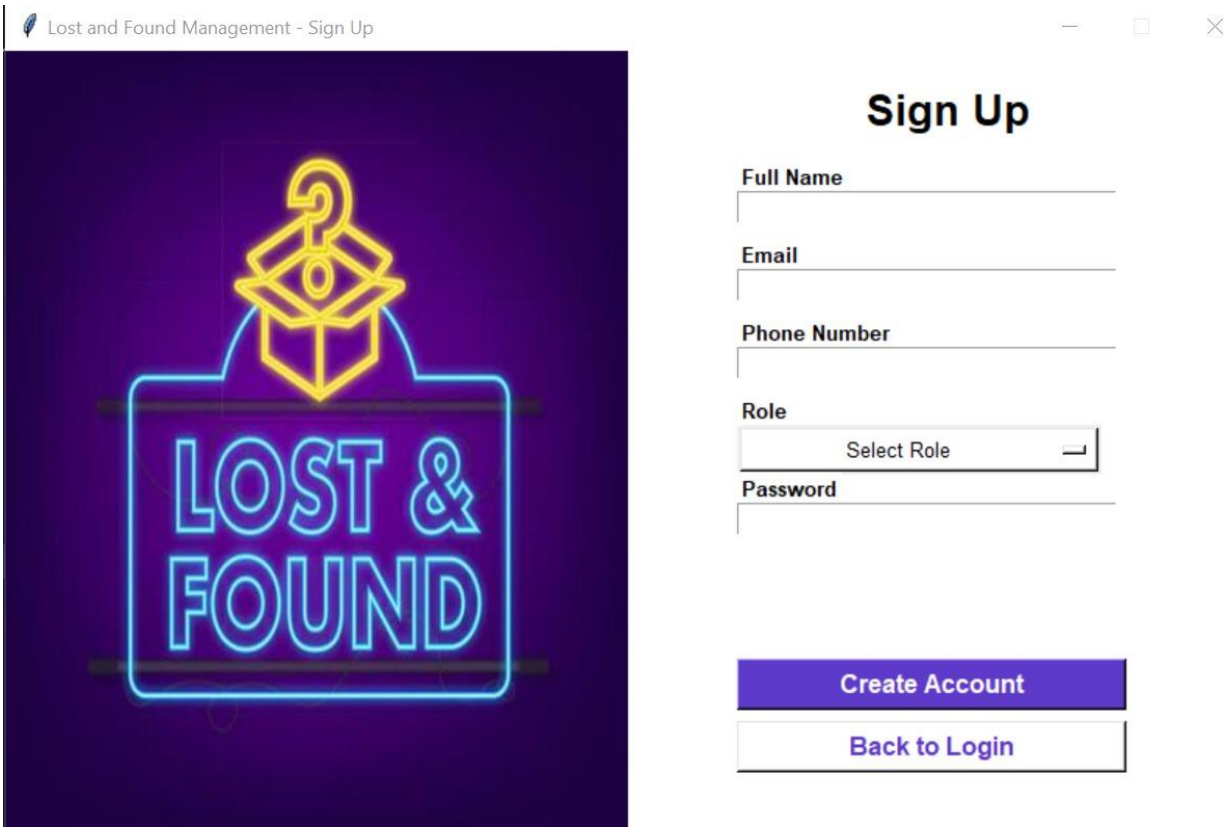

Admin Dashboard
Posted Items
Claim Requests
Users
Deleted Records
Recent Actions
ID | Title | Description | Date | Status | User
14 | mob | abc | 2025-05-14 | Lost | Alice Smith | FE
15 | bag | xyz | 2025-05-14 | Found | Alice Smith | Ac
16 | keys | abcd | 2025-05-14 | Lost | Alice Smith | Gr
Delete Item
Upload to Firebase
Recent Actions

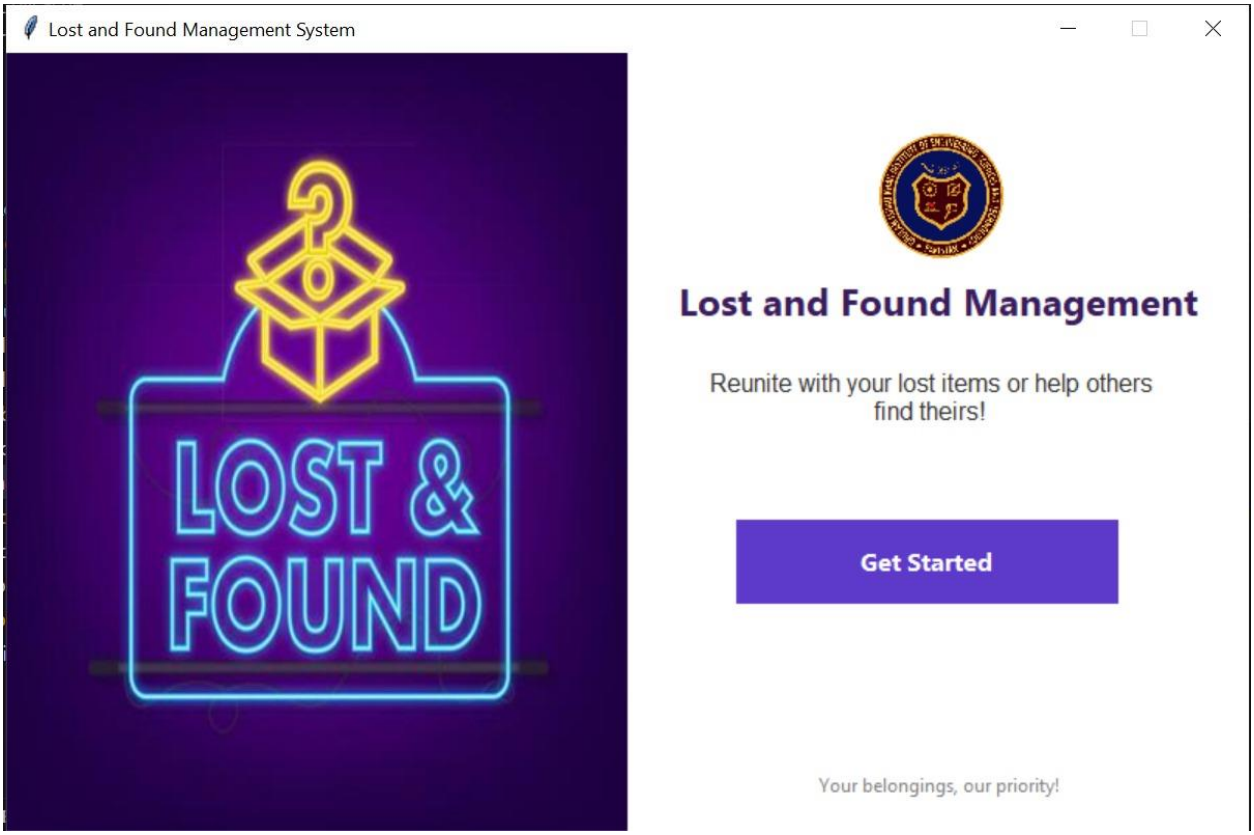

Lost and Found Management - Dashboard
Lost and Found
Efficiently manage and claim lost items.
Post or claim items to help others.
Dashboard
Claim Items
Post Items
Logout


Forgot Password
Reset Password
Email
New Password
Confirm Password
Reset Password
Back to Login


Lost and Found Management - Login
LOST &
FOUND
Login
Email
Password
Login
Sign Up
Forgot your password?
Lost something? We're here to help!


Post Lost Item
Post Lost Item
Help others by posting items you found
or report your lost items.
← Back
Lost Item Form
Item Title
Description
Location
Date
2025-05-14
Status
Lost
Item Image
Upload Image
No image selected
Post Item


Lost and Found Management - Sign Up
LOST &
FOUND
Sign Up
Full Name
Email
Phone Number
Role
Select Role
Password
Create Account
Back to Login


Lost and Found Management System
LOST &
FOUND
Lost and Found Management
Reunite with your lost items or help others find theirs!
Get Started
Your belongings, our priority!

```sql
-- USERS
CREATE TABLE users (
    user_id SERIAL PRIMARY KEY,
    name VARCHAR(100) NOT NULL CHECK (name ~ '^[A-Za-z\s]+$'),
    email VARCHAR(100) UNIQUE NOT NULL CHECK (email ~ '^[A-Za-z0-9._%+-]+@gmail\.com$'),
    phone VARCHAR(15) CHECK (phone ~ '^03\d{9}$'),
    role VARCHAR(20) CHECK (role IN ('Student', 'Staff', 'Guest')) NOT NULL,
    password VARCHAR(255) NOT NULL
);

-- DELETED USERS
CREATE TABLE deleted_users (
    user_id INTEGER PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    email VARCHAR(100) NOT NULL,
    phone VARCHAR(15),
    role VARCHAR(20),
    password VARCHAR(255),
    deleted_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

```sql
-- ADMINS
CREATE TABLE admins (
    admin_id SERIAL PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    email VARCHAR(100) UNIQUE NOT NULL,
    username VARCHAR(50) UNIQUE NOT NULL,
    password VARCHAR(255) NOT NULL
);

-- LOCATIONS
CREATE TABLE locations (
    location_id SERIAL PRIMARY KEY,
    location_name VARCHAR(100) NOT NULL
);
```

```sql
-- ITEMS
CREATE TABLE items (
    item_id SERIAL PRIMARY KEY,
    title VARCHAR(100) NOT NULL,
    description TEXT,
    date_reported DATE NOT NULL,
    status VARCHAR(20) NOT NULL DEFAULT 'Lost' CHECK (status IN ('Lost', 'Found', 'Returned')),
    image_path TEXT,
    location_id INTEGER REFERENCES locations(location_id) ON DELETE CASCADE,
    reported_by INTEGER REFERENCES users(user_id) ON DELETE SET NULL
);
```

```sql
-- DELETED CLAIMS
CREATE TABLE deleted_claims (
    claim_id INTEGER PRIMARY KEY,
    item_id INTEGER,
    claimed_by INTEGER REFERENCES users(user_id) ON DELETE SET NULL,
    claim_date DATE,
    status VARCHAR(20),
    approved_by INTEGER,
    loss_location TEXT,
    loss_date DATE,
    deleted_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- Indexes for performance
CREATE INDEX idx_items_reported_by ON items(reported_by);
CREATE INDEX idx_claims_item_id ON claims(item_id);
CREATE INDEX idx_claims_claimed_by ON claims(claimed_by);
CREATE INDEX idx_deleted_items_reported_by ON deleted_items(reported_by);
CREATE INDEX idx_deleted_claims_claimed_by ON deleted_claims(claimed_by);
```

```sql
-- DELETED ITEMS
CREATE TABLE deleted_items (
    deleted_item_id SERIAL PRIMARY KEY,
    item_id INTEGER,
    title VARCHAR(100) NOT NULL,
    description TEXT,
    date_reported DATE NOT NULL,
    status VARCHAR(20),
    image_path TEXT,
    location_id INTEGER REFERENCES locations(location_id) ON DELETE CASCADE,
    reported_by INTEGER REFERENCES users(user_id) ON DELETE SET NULL,
    deleted_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- CLAIMS
CREATE TABLE claims (
    claim_id SERIAL PRIMARY KEY,
    item_id INTEGER REFERENCES items(item_id) ON DELETE CASCADE,
    claimed_by INTEGER REFERENCES users(user_id) ON DELETE SET NULL,
    claim_date DATE NOT NULL DEFAULT CURRENT_DATE,
    status VARCHAR(20) NOT NULL DEFAULT 'Pending' CHECK (status IN ('Pending', 'Approved', 'Rejected')),
    approved_by INTEGER REFERENCES admins(admin_id) ON DELETE SET NULL,
    loss_location TEXT,
    loss_date DATE
);
```