System Design Document: Campus Lost & Found System

1. Introduction

This System Design Document (SDD) outlines the architectural design, component breakdown, technology stack, and implementation considerations for the Campus Lost & Found System. Building upon the established functional and non-functional requirements, this document provides a technical blueprint for the development team, ensuring a robust, scalable, secure, and user-friendly solution. The primary goal is to address the inefficiencies of manual lost and found processes by providing a centralized digital platform.

1.1 Purpose

The purpose of this document is to detail the technical design of the Campus Lost & Found System, serving as a guide for development, testing, and deployment. It ensures that the system meets all specified functional requirements (FRs) and non-functional requirements (NFRs) outlined in the Requirements Document.

1.2 Scope

This SDD covers the design of the mobile application (Android), the web portal, the backend API, the database, and integrated third-party services. It details how the system will facilitate item reporting, searching, secure messaging, and administrative management.

1.3 Definitions and Acronyms

- API: Application Programming Interface
- DB: Database
- FR: Functional Requirement
- IDE: Integrated Development Environment
- ISMS: Information Security Management System
- NFR: Non-Functional Requirement
- PII: Personally Identifiable Information
- SDD: System Design Document
- UI/UX: User Interface/User Experience

2. System Architecture

The Campus Lost & Found System will employ a three-tier architecture (or multi-tier architecture) to ensure separation of concerns, scalability, and maintainability.

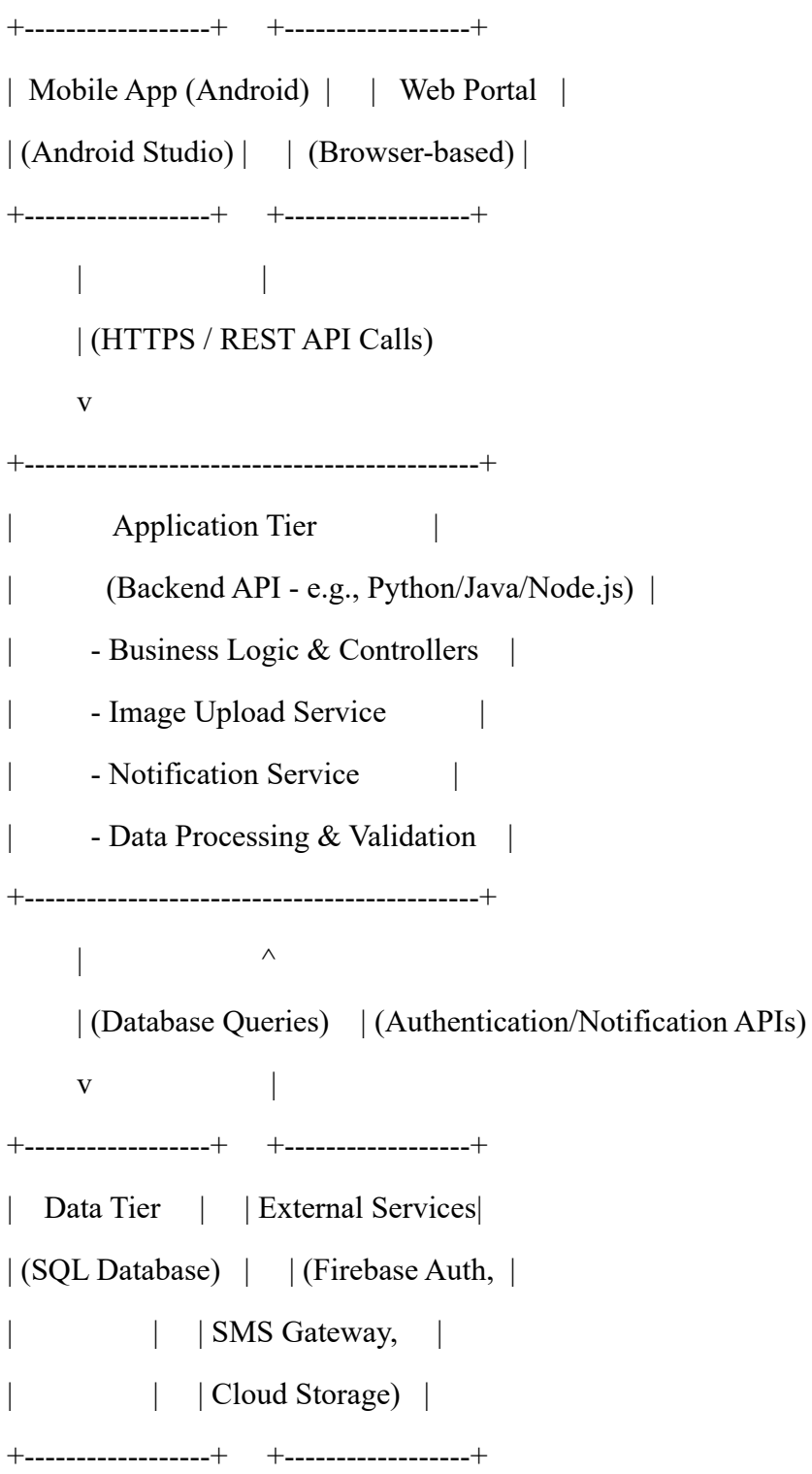
- Presentation Tier (Client-side): User interfaces for interacting with the system.

    Mobile Application (Android): Native application for students and staff.

    Web Portal: Browser-based interface for students, staff, and administrators.

- Application Tier (Backend API): Business logic, data processing, and communication with the database and external services.
- Data Tier: Persistent storage for all system data.

High-Level Architecture Diagram:



3. Component Design

3.1 Presentation Tier

- 3.1.1 Mobile Application (Android)

  Technology: Developed using Android Studio (Kotlin/Java).

  Modules:

  - Authentication Module: User registration and login (integrates with Firebase Auth).
  - Item Reporting Module: Forms for lost/found items with fields for details, categories, location, and image upload.
  - Search & Browse Module: UI for filtering and displaying item listings.
  - Messaging Module: In-app chat interface.
  - Notifications Module: Handles in-app notifications and push notifications.
  - Profile Management Module: User profile and settings.

  - UI/UX: Adherence to Material Design guidelines for intuitiveness and usability (NFR-05: Usability).

- 3.1.2 Web Portal

  Technology: Frontend framework (e.g., React, Angular, Vue.js) with HTML, CSS, JavaScript.

  Modules: Similar to mobile app, but with additional administrative functionalities.

  - User Module: Registration/Login.
  - Item Reporting/Search: Web forms and search interface.
  - Messaging: Web-based chat interface.
  - Admin Dashboard: Comprehensive interface for staff (security, cleaning supervisors) to manage items, update status, view logs, and manage user claims (FR-06: Item Status Management).

  - UI/UX: Responsive design to ensure optimal viewing across various devices, meeting NFR-05: Usability.

3.2 Application Tier (Backend API)

- Technology: A robust backend framework (e.g., Python with Django/Flask, Node.js with Express, Java with Spring Boot).
- Architecture: RESTful API endpoints for all client-server communication.

- Key Services/Modules:

    Authentication & Authorization Service: Manages user sessions, token validation (integrates with Firebase Auth results), and role-based access control (e.g., distinguishing students from security staff).

    Item Management Service: Handles CRUD (Create, Read, Update, Delete) operations for LostItemsand FoundItems, including data validation (FR-02, FR-03, FR-06).

    Search & Matching Service: Implements the core search logic, applying filters and algorithms for matching lost and found items. This service will interact heavily with the SQL Database (FR-04).

    Messaging Service: Manages the secure storage and retrieval of in-app messages between users, ensuring anonymity initially (FR-05).

    Image Upload Service: Handles secure storage of image files (e.g., leveraging cloud storage like Google Cloud Storage or AWS S3), processing uploads, and providing secure URLs.

    Notification Service: Triggers and manages notifications (e.g., email, SMS, push notifications) based on matching items or new messages (FR-07).

    Admin Service: Provides specific endpoints for administrative tasks (e.g., user management, item auditing, analytics).

3.3 Data Tier

- Technology: SQL Database (e.g., PostgreSQL, MySQL).
- Database Schema Design (Conceptual/Logical):

    Users Table: user_id (PK), campus_email (Unique), password_hash, first_name, last_name, user_type (student/staff/admin), registration_date.

    Items Table: item_id (PK), reporter_user_id (FK to Users), item_type (lost/found), category, name, description, last_seen_location, date_lost/found, status (Lost, Found, Claimed, Returned, Archived), image_url, reported_date.

    Messages Table: message_id (PK), sender_id (FK to Users), receiver_id (FK to Users), item_id (FK to Items), message_content, timestamp, is_read.

    Categories Table: category_id (PK), category_name (e.g., 'Electronics', 'ID Cards', 'Books').

    Locations Table: location_id (PK), location_name, campus_building.

    AuditLogs Table (for NFR-01, NFR-05 - if implemented): log_id (PK), user_id (FK), action, timestamp, details.

- Indexing: Appropriate indexing on frequently queried columns (e.g., item_type, category, date_lost/found, location) to ensure fast search performance (NFR-03: Performance).

4. Security Design (NFR-01 & NFR-02)

- Authentication & Authorization:

  Firebase Auth (FR-01): Handles primary user authentication. User IDs from Firebase Auth will be stored in the SQL database for linking to user-specific data.

  Session Management: Secure, short-lived tokens (e.g., JWTs) managed by the backend after Firebase authentication.

  Role-Based Access Control (RBAC): Backend logic to enforce permissions based on user_type(e.g., only admin or security staff can update item status).

- Data Encryption:

  In Transit: All communication between clients (mobile app, web portal) and the backend API will be secured using HTTPS/SSL/TLS.

  At Rest: Sensitive data (e.g., user credentials – handled by Firebase Auth, but if any PII is stored directly in SQL, it would be encrypted) in the SQL database will be encrypted. Database backups will also be encrypted.

- Input Validation & Sanitization: All user inputs will be rigorously validated on both client and server sides to prevent common vulnerabilities (e.g., SQL Injection, XSS).

- Data Privacy (NFR-02):

  Anonymized Messaging (FR-05): Personal contact details will not be exposed directly between matching users until they mutually agree to share them outside the system.

  PII Protection: Strict access controls on PII. Data minimization principles applied (collecting only necessary data).

  Compliance: Design aligns with Kenya Data Protection Act and principles of GDPR (e.g., right to access, right to be forgotten).

- Verification: Implementing verification mechanisms for item claims (e.g., requiring users to answer specific questions about the item's unique features, or photo verification by staff).

5. Performance & Scalability Design (NFR-03 & NFR-04)

- Database Optimization:

  Strategic indexing on frequently searched fields.

Optimized SQL queries.

Connection pooling for efficient database access.

- Backend Scalability:

  Stateless API: The backend API will be designed to be stateless, allowing for easy horizontal scaling (adding more server instances as demand grows).

  Load Balancing: A load balancer will distribute incoming requests across multiple backend instances.

  Microservices (Future): For very large scale, complex services (e.g., search, notifications) could be spun off into separate microservices.

- Caching: Implementing caching mechanisms (e.g., Redis) for frequently accessed, less volatile data to reduce database load.
- CDN for Images: Using a Content Delivery Network (CDN) for serving user-uploaded images to ensure fast loading times globally.
- Cloud Infrastructure: Deploying the system on a scalable cloud platform (e.g., Google Cloud Platform, AWS, Azure) to leverage their auto-scaling and managed services.

6. Deployment Strategy

The system will be deployed on a cloud infrastructure platform.

- Backend API: Containerized using Docker and deployed on a container orchestration service (e.g., Kubernetes on GCP/AWS ECS) for easy scaling and management.
- Database: A managed SQL database service (e.g., Google Cloud SQL, AWS RDS) for high availability, backups, and simplified administration.
- Image Storage: Cloud storage buckets (e.g., Google Cloud Storage, AWS S3) for storing user-uploaded item images securely and scalably.
- Mobile App: Published on the Google Play Store for Android users.
- Web Portal: Hosted via a web server (e.g., Nginx, Apache) or a managed web hosting service on the cloud platform.
- CI/CD: Continuous Integration/Continuous Deployment (CI/CD) pipelines will be set up to automate testing and deployment processes, ensuring rapid and reliable updates.

7. Technology Stack Summary

- Mobile App Development (Android): Android Studio (Kotlin/Java)
- Backend API: Python (e.g., Django REST Framework or Flask) / Node.js (Express) / Java (Spring Boot)
- Database: SQL Database (e.g., PostgreSQL or MySQL)

- Authentication: Firebase Auth
- Cloud Storage (for images): Google Cloud Storage / AWS S3
- Messaging (Push Notifications): Firebase Cloud Messaging (FCM) or similar service.
- Version Control: Git / GitHub

This SDD provides a solid foundation for the technical implementation of the Campus Lost & Found System, aligning design choices with the project's requirements and vision.