**LOW LEVEL DESIGN (LLD)**

**1. Introduction**

1.1 **What is Low-Level Design Document?**  
A Low-Level Design (LLD) document provides the internal design details of each component in the application. It outlines class structures, relationships, and methods, making it easier for developers to implement the code.

1.2 **Scope**  
This document will describe the architecture, detailed workflows, database design, API structures, and user interactions for the **Child Safety Application**.

**2. Architecture**

**Workflow Overview**

1. Parent and child register/login.
2. Child updates geolocation periodically.
3. Parent views the child’s geolocation and receives alerts.
4. SOS mechanism sends alerts via SMS/email to guardians.

**Components:**

* **Frontend:** HTML, CSS, JS (UI for parent and child dashboards).
* **Backend:** Node.js and Express.js (API handling, business logic).
* **Database:** MongoDB (user data, geolocation history, guardian details).
* **Third-Party APIs:** Google Maps API (geolocation), Twilio/SendGrid (SMS/Email).

**3. Architecture Description**

**3.1 Data Description**

* **Parent Data:** Registration details, child association, preferences.
* **Child Data:** Registration details, geolocation updates, guardians.
* **Guardian Data:** Email, phone, relationship to child.
* **SOS Data:** Trigger time, geolocation, guardian contact information.

**3.2 API Workflow**

* Registration/Login APIs (authentication and data validation).
* Geolocation Update API (child updates current location).
* SOS Trigger API (sends SMS/email alerts).

**3.3 Data Flow:**

1. Child app sends periodic geolocation updates to the backend.
2. Backend stores geolocation in MongoDB.
3. Parent app fetches geolocation data from the database.
4. SOS sends immediate notifications to guardians.

**4. Database Design**

**Schema Examples:**

1. **User Collection**

{

"userId": "unique\_id",

"name": "string",

"email": "string",

"password": "hashed\_string",

"role": "parent/child",

"createdAt": "timestamp"

}

1. **Geolocation Collection**

{

"childId": "unique\_id",

"latitude": "number",

"longitude": "number",

"timestamp": "datetime"

}

1. **Guardian Collection**

{

"childId": "unique\_id",

"guardianName": "string",

"email": "string",

"phone": "string",

"relation": "string"

}

**5. Unit Test Cases**

| **Test Case Description** | **Pre-Requisite** | **Expected Result** |
| --- | --- | --- |
| Verify Parent/Child Registration | Application is accessible | User should successfully register. |
| Verify Parent Login | Registered user data exists | Parent logs in successfully. |
| Verify Geolocation Update API | Child account exists | Location updates successfully stored. |
| Verify SOS Trigger | SOS API integration is configured | Guardians receive SMS/email notifications. |

**6. Deployment**

* **Cloud Service:** AWS/GCP/Azure
* **Components Deployed:**
  + **Frontend:** Static hosting (AWS S3, GCP Cloud Storage).
  + **Backend:** Dockerized Node.js server.
  + **Database:** MongoDB Atlas for scalability.

**7.Wireframes**

Below are descriptions for the key wireframes. I’ll generate a simple representation for each:

**1. Parent Dashboard**

* **Features:**
  + View registered children.
  + Display geolocation on a map.
  + SOS alerts and recent notifications.
  + Manage guardian details and blocked sites.

**2. Child Dashboard**

* **Features:**
  + Update geolocation.
  + Trigger SOS manually via a button.
  + Manage guardian contact list.
  + Search blocked sites or unsafe areas.

**3. Admin Dashboard (Optional)**

* **Features:**
  + Monitor all active SOS events.
  + Contact parents or guardians in emergencies.
  + View historical SOS logs.

**Workflow Diagrams**

**1. SOS Workflow**

* **Trigger:** Child presses the SOS button.
* **Flow:**
  1. SOS button triggers an API call to the backend.
  2. Backend fetches child geolocation and guardian details from MongoDB.
  3. SMS/email notification sent to all listed guardians.
  4. Parent notified via push notification.

**2. Geolocation Update Workflow**

* **Trigger:** Periodic or manual geolocation update by the child.
* **Flow:**
  1. Child app sends current GPS location to the backend.
  2. Backend updates MongoDB geolocation collection.
  3. Parent app fetches updated location on demand.

