School ID Photo Capture Mobile Ecosystem

Master Technical Specification Document (v2.0)

1. Executive Summary

This document serves as the comprehensive technical guide for the **School ID Photo Capture Mobile App**. It details the backend architecture, database schema, and API protocols required to support an **Offline-First**, high-volume photography workflow.

Core Architecture Principles:

- **Offline Tolerance:** The app must support fully offline capture for up to 2,000 photos .
- **Delta Sync:** Downstream data fetches only download records modified since the last sync to save bandwidth.
- **Multipart Batching:** Upstream uploads use chunks of 10-20 records, separating high-res binaries from JSON metadata.
- **Security:** All transport is HTTPS with SSL Pinning . Data at rest on the device is encrypted (AES-256) .

2. Database Architecture (Normalized)

To support the syncing logic and reduce data redundancy, the legacy flat table st_student is normalized into a relational structure.

2.1 Required Schema Changes (Sync & Audit)

Regardless of normalization, the following columns are **mandatory** for the sync logic to function properly.

Column Name	Type	Purpose
thumbnail_url	VARCHAR(255)	Path to compressed image for fast mobile list loading.
capture_timestamp	DATETIME	Audit: The exact time the photo was taken on the device.
edited_by	INT	Audit: ID of the photographer who took the

		photo.
updated_at	TIMESTAMP	**Critical:** Auto-updates on modification. Used for Delta Sync.

2.2 Normalized Schema (3NF)

A. Table: centres (Schools)
Maps to the order_id in the API.

SQL

```sql
CREATE TABLE centres (
  id INT AUTO_INCREMENT PRIMARY KEY,
  centre_code VARCHAR(20) UNIQUE NOT NULL, -- (API: order_id)
  name VARCHAR(150) NOT NULL
);
```

B. Table: students (Core Table)
Lightweight and optimized for sync.

SQL

```sql
CREATE TABLE students (
  id BIGINT AUTO_INCREMENT PRIMARY KEY, -- (API: student_id)
  adm_no VARCHAR(30) UNIQUE NOT NULL,
  name VARCHAR(100) NOT NULL,

  -- Relationships
  centre_id INT NOT NULL,
  class_id INT NOT NULL,

  -- Media & Sync (Mapped from legacy)
  photo_url VARCHAR(255),         -- (Legacy: sphoto_1)
  thumbnail_url VARCHAR(255),       -- New: Mobile optimization
  status TINYINT DEFAULT 1,         -- (Legacy: status_1)
  -- Audit & Sync Control
  capture_timestamp DATETIME,
  edited_by INT,
  updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,

  FOREIGN KEY (centre_id) REFERENCES centres(id)
);
```

---

3. API Specification

3.1 Authentication

- **Endpoint:** POST /api/v1/auth/login
- **Purpose:** Authenticates user and caches the student database .

Request:

JSON

```json
{
  "username": "photographer_01",
  "password": "secure_password",
  "device_id": "android_hw_id"
}
```

Response (200 OK):

JSON

```json
{
  "status": "success",
  "data": {
    "user_id": 105,
    "token": "eyJhbGciOiJIUzI1Ni...",
    "order_ids": ["1001", "1002"] // List of assigned School Codes
  }
}
```

3.2 Downstream Sync (Fetch)

- **Endpoint:** GET /api/v1/students/fetch
- **Logic: Delta Sync**. If last_sync_timestamp is provided, return only changed records.

Query Params:

- order_id (Required): The School Code (e.g., "1001").
- last_sync_timestamp (Optional): The server_sync_timestamp from the previous fetch.

Response (200 OK):

JSON

```json
{
  "status": "success",
  "order_id": "1001",
  "server_sync_timestamp": "2026-01-21T14:00:00Z", // SAVE THIS for next request
  "students": [
    {
      "student_id": 5501,
      "name": "Arun Kumar",
      "photo_url": "uploads/5501.jpg",
      "thumbnail_url": "uploads/5501_thumb.jpg",
      "status": 1,
      "updated_at": "2026-01-21T13:45:00Z" // Logic: updated_at > last_sync_timestamp
    }
  ]
}
```

3.3 Upstream Sync (Batch Upload)

- **Endpoint:** POST /api/v1/students/sync-batch
- **Logic: Multipart/Form-Data** with **Exception Reporting**.
    - **Images:** Sent as binary files (saving 33% overhead vs Base64).
    - **Metadata:** Sent as a JSON string.
    - **Thumbnails:** Sent as Base64 inside the JSON (since they are tiny, <5KB).

Payload (Form Data):

Key	**Value Type**	**Description**
order_id	String	e.g., "1001"
batch_data	**JSON String**	Serialized metadata array (see below).
file_5501	Binary File	High-res image for Student 5501.
file_5502	Binary File	High-res image for Student 5502.

Structure of batch_data String:

```JSON
[
 {
  "student_id": 5501,
  "capture_timestamp": "2026-01-21T14:30:00Z",
  "edited_by": 105,
  "status": 1,
  "thumbnail_base64": "/9j/4AAQSk...",
  "has_high_res_update": true // Server looks for 'file_5501'
 }
]
```

Response (200 OK - Exception Reporting):
Returns only failures. Empty list = 100% Success.

JSON

```
{
  "status": "success",
  "failed_ids": [
    {
      "student_id": 5503,
      "error": "File corrupted"
    }
  ]
}
```

---

4. Backend Implementation Reference (PHP)

File: api/v1/students/sync-batch.php

PHP

```php
<?php
// CONFIG: Allow large uploads (Ensure php.ini post_max_size > 50M)
header("Content-Type: application/json");

// 1. Validation
$orderId = $_POST['order_id'] ?? null;
$batchDataString = $_POST['batch_data'] ?? null;

if (!$orderId || !$batchDataString) {
    http_response_code(400);
    echo json_encode(["status" => "error", "message" => "Missing Data"]);
    exit;
}

$students = json_decode($batchDataString, true);
$failedIds = [];
$uploadDir = "uploads/" . $orderId . "/";

if (!is_dir($uploadDir)) mkdir($uploadDir, 0755, true);

// 2. Processing Loop
foreach ($students as $student) {
    $id = $student['student_id'];
    // A. Handle High-Res Binary
    if ($student['has_high_res_update'] ?? false) {
        $fileKey = "file_" . $id; // Matches Flutter's key
        if (isset($_FILES[$fileKey]) && $_FILES[$fileKey]['error'] == 0) {
            move_uploaded_file($_FILES[$fileKey]['tmp_name'], $uploadDir . $id . ".jpg");
        } else {
            $failedIds[] = ["student_id" => $id, "error" => "File Missing"];
            continue;
        }
    }

    // B. Handle Thumbnail (Base64)
    if (!empty($student['thumbnail_base64'])) {
        file_put_contents($uploadDir . $id . "_thumb.jpg",
base64_decode($student['thumbnail_base64']));
    }

    // C. Database Update (Pseudo-code)
    // UPDATE students SET photo_url=..., capture_timestamp=... WHERE id = $id
}

// 3. Response
echo json_encode([
    "status" => empty($failedIds) ? "success" : "partial_success",
    "failed_ids" => $failedIds
]);
?>
```

---

5. Frontend Logic Reference (Flutter)

Class: BackgroundSyncWorker

1. **Selection:** Query local DB for WHERE sync_status = 'pending' LIMIT 10.
2. **Manifest Construction:** Create the batch_data JSON string.
3. **Multipart Request:**
   Dart
   ```
   var request = http.MultipartRequest('POST', Uri.parse(url));
   request.fields['batch_data'] = jsonEncode(batchList);
   request.fields['order_id'] = "1001";

   // Attach binaries
   ```

```
for (var s in batchList) {
 if (s.hasPhoto) {
   request.files.add(await http.MultipartFile.fromPath('file_${s.id}', s.localPath));
 }
}
```

4. **Process Response:**
   - If failed_ids is empty $\rightarrow$ Mark ALL 10 records as synced.
   - If failed_ids has items $\rightarrow$ Mark successes as synced, keep failures as pending.

---

6. Security & Infrastructure Checklist

1. **SSL Pinning:** The mobile app must embed the server's public certificate to prevent MITM attacks .
2. **Server Config:**
   - client_max_body_size 50M; (Nginx)
   - post_max_size = 50M (PHP.ini)
   - upload_max_filesize = 50M (PHP.ini) .
3. **Conflict Resolution:** If the server receives a photo with an older capture_timestamp than what is already stored, it should reject the update (First-Writer-Wins or Last-Writer-Wins policy).