Product Requirements Document (PRD): RapidPhotoUpload — AI-Assisted High-Volume Photo Upload System

1. Introduction and Project Goal

1.1 Project Goal

The goal of the RapidPhotoUpload project is to design and implement a high-performance, asynchronous photo upload system capable of reliably handling up to **100 concurrent media uploads**. This project challenges candidates to demonstrate architectural excellence, mastery of concurrency, and exceptional user experience design across both mobile and web clients.

1.2 Context

This system simulates a high-volume media platform (similar to Google Photos or Drive) where users upload large batches of images while expecting the application to remain fully responsive. The project focuses on handling load, providing real-time feedback, and ensuring a clean, scalable design suitable for production environments.

2. Business Functionality

2.1 Problem Statement

Users expect seamless, high-speed media uploads without application freezing. The system must address the architectural complexities of concurrent file handling, status tracking, and efficient storage integration to deliver a reliable, non-blocking experience.

2.2 Core Functional Requirements

1. **High-Volume Concurrency:** The system **MUST** support the simultaneous uploading of up to 100 photos per user session.
2. **Asynchronous UI:** Users **MUST** be able to continue navigating and interacting with the application (both web and mobile) while uploads are in progress.
3. **Real-Time Status:** Display individual and batch upload progress using responsive indicators (e.g., progress bars) with real-time status updates (Uploading, Failed, Complete).

4. **Web Interface:** A dedicated web interface for viewing, tagging, and downloading previously uploaded photos.
5. **Mobile Interface:** A dedicated mobile application (React Native or Flutter) that mirrors the upload and viewing functionality.
6. **Backend Handling:** The backend must manage concurrent requests, store file metadata, and efficiently stream/store the binary files in cloud object storage.
7. **Authentication:** Basic authentication (mocked or JWT-based) is required to secure access for both mobile and web clients.
8. **Project Scope:** This is a two-part project consisting of **one mobile application** and **one web application**, both integrated with the same shared backend API.

3. Architecture and Technical Requirements

3.1 Architectural Principles (Mandatory)

The backend architecture is the core of the assessment and **MUST** adhere to the following principles:

1. **Domain-Driven Design (DDD):** Core concepts (e.g., Photo, Upload Job, User) must be modeled as robust Domain Objects.
2. **CQRS (Command Query Responsibility Segregation):** Implement a clear separation between handling upload/mutation commands and querying photo status/metadata.
3. **Vertical Slice Architecture (VSA):** Organize the backend code around features (e.g., `UploadPhotoSlice`, `GetPhotoMetadataSlice`).

3.2 Technical Stack

- **Back-End (API):** Java with Spring Boot. Must handle large, asynchronous requests efficiently.
- **Web Front-End:** TypeScript with React.js.
- **Mobile Front-End:** React Native or Flutter.
- **Cloud Storage (Mandatory):** Files **MUST** be stored in a scalable object storage solution: **AWS S3 or Azure Blob Storage**.
- **Database:** PostgreSQL is required for persisting metadata (User, Photo, Upload Job Status).
- **Cloud Platforms:** Deployment target flexibility: **AWS or Azure**.

3.3 Performance Benchmarks

- **Concurrency Load:** The system **MUST** handle the concurrent upload of 100 photos (average size 2MB each) within **90 seconds** on a standard broadband connection.
- **UI Responsiveness:** Both the mobile and web interfaces **MUST** remain fluid and fully responsive during peak upload operations.

4. Code Quality and AI Acceleration

4.1 Code Quality Standards (Mandatory)

- **Architecture:** Clean separation of concerns across Domain, Application, and Infrastructure layers.
- **Backend:** Must demonstrate robust handling of concurrency, including mechanisms for retries and efficient streaming of large file uploads.
- **Frontend:** Both React/Next.js and React Native/Flutter apps must use a clean, component-based architecture and adhere strictly to TypeScript standards.
- **Readability:** Consistent naming conventions, modularity, and comprehensive documentation are required.

4.2 Testing (Mandatory)

- **Integration Tests: MUST** implement integration tests that validate the complete upload process, from the client (simulated mobile/web) through the backend services and ending with successful persistent storage in the cloud object store.

4.3 AI Tool Utilization

AI tools (Cursor, Copilot, v0.dev, Locofy) are optional. If used, they should be applied intelligently for tasks such as:

- Image categorization (e.g., tagging the uploaded image contents).
- Compression optimization algorithms.
- Upload prioritization logic.

The evaluation will measure the candidate's effective use of AI for quality acceleration.

5. Project Deliverables and Constraints

5.1 Time Constraint

- **Recommended Completion Time:** 5 days.

5.2 Submission Requirements

1. **Code Repository:** Complete, functional code repository (GitHub preferred), containing all three components (backend, web client, mobile client).
2. **Brief Technical Writeup (1-2 pages):** Documenting the chosen concurrency strategy, asynchronous design, cloud storage interaction (S3/Blob), and the division of logic across the three application components.

3. **Demo:** A video or live presentation demonstrating the simultaneous upload of a batch of images and the real-time progress indicators on both client platforms.
4. **AI Tool Documentation:** Detailed documentation of any AI tools used, including example prompts and a justification for their impact.
5. **Test Cases and Validation Results:** Evidence of passing integration tests validating the end-to-end upload flow.