Team G - Arc 42 Documentation

1) Introduction and Goals

1.1 Requirements Overview

Name	Description
User Authentication	A user can sign up, log in, and log out via Google OAuth (Firebase Auth).
Image Upload	An authenticated user can upload photos to Firebase Storage.
Image Editing	A user can launch an image in the integrated Pixlr editor and save edits.
Gallery View	Users can browse their uploaded and edited images in a responsive gallery.
Sharing & Link Publish	Users can generate a public link to view or download an edited image.
Download	A user can download any image they own in its latest edited form.

Driving Forces & Constraints

- **GDPR-Compliance:** All data must remain in the EU.
- **Scalability:** Firebase Hosting + Functions auto-scale to meet demand.
- **Mobile-First:** Primary usage on smartphones and tablets.
- **Budget:** MVP relies solely on open-source libraries (no paid third-party).
- **Team Expertise:** Frontend in React for rapid, maintainable development.

Motivation From the end-user's perspective, PhotoApp must make it effortless to manage and enhance personal photos on any device without worrying about data privacy or performance. The streamlined workflow—login, upload, edit, share—should feel instantaneous, intuitive, and secure.

1.2 Quality Goals

- Usability
  - Users expect a seamless and intuitive experience on both mobile and web
    - Image upload in under 30 seconds
- Performance
  - Images should load quickly, even with large files or under high traffic
    - 95% of pictures should load in under 2 seconds
- Maintainability
  - The codebase should be easy to update and extend as the app evolves and scales.
    - 80% test coverage

1.3 Stakeholder

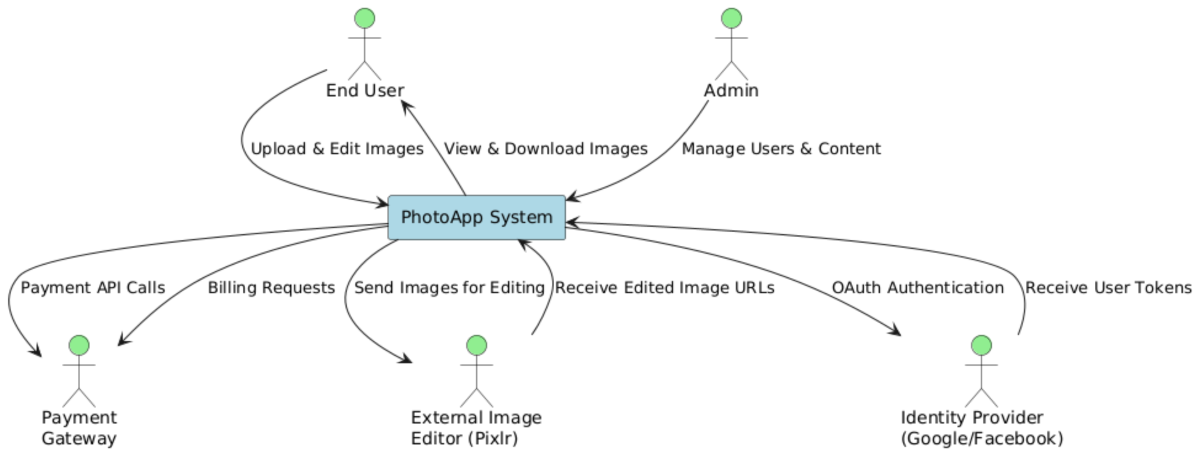
- Sarah Chen – Marketing Manager
    - Effective tools to promote the app and grow a loyal user base
- Michael Nguyen – Project Manager
    - Clear scope, timeline and smooth collaboration across teams
- Ava Patel – Lead Developer
    - Extendable application and maintainable architecture
- Investor
    - Clear USP and scalable business models
- Startup Founders
    - Cost efficient products and long-term income

2) Architecture Constraints

- **Must use Firebase Hosting**: Chosen for scalability and simplicity; it aligns with the team's expertise and fits our MVP timeline.
- **Frontend must be built in React**: The team has prior experience with React, which boosts productivity and ensures a maintainable codebase.
- **Data must not leave the EU**: To comply with GDPR regulations, all data will be stored and processed within the European Union.
- **No paid third-party libraries**: Budget constraints in the MVP phase mean we rely solely on open-source solutions and avoid any licensed, paid libraries.
- **Must support mobile-first design**: Since most end users are expected to access the application from mobile devices, we prioritize a responsive, mobile-optimized UI.

3) Context and Scope

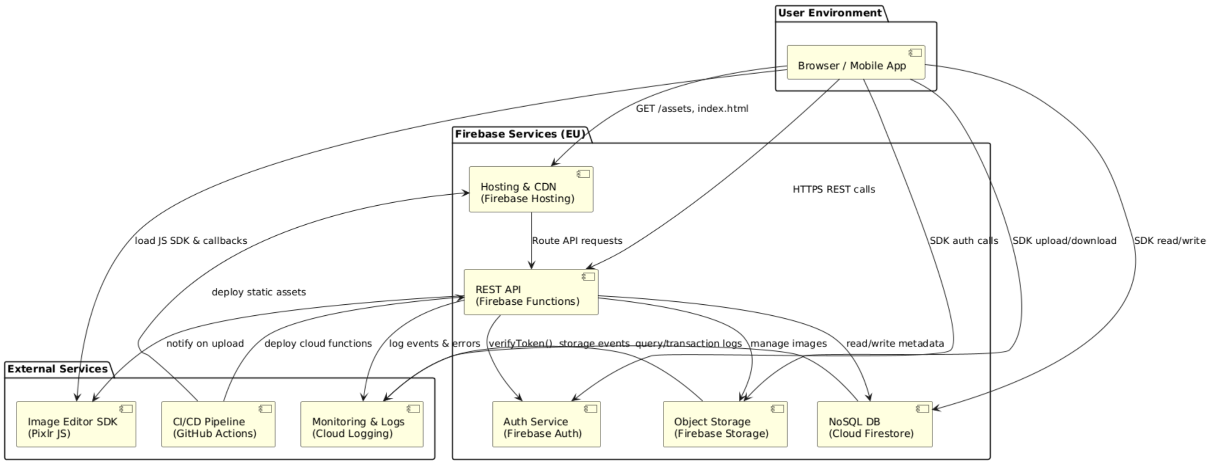
3.1 Business context



Purpose: Shows the system "in the world" – who its users and external systems are, and the primary interactions.

- **Primary Actors:**
  - **End User** – uploads, edits, views and downloads photos.
  - **Admin** – manages users, moderates content.
  - **Payment Gateway** – handles any billing calls.
  - **External Image Editor (Pixlr)** – invoked to perform rich in-browser editing.
  - **Identity Provider (Google/Facebook)** – issues OAuth tokens for login.
- **System Boundary:** "PhotoApp System" encapsulates all your Firebase-based services plus the SPA.
- **Key Relationships:**
  - User ↔ PhotoApp: upload/edit/view/download
  - PhotoApp ↔ Pixlr: send images out for editing + receive edited URLs
  - PhotoApp ↔ IdP: OAuth handshake for authentication
  - PhotoApp ↔ Payment: billing API calls

3.2 Technical context



Component	Technology	Responsibility	Constraints/Notes
Client	React SPA	UI, SDK calls to Auth/Firestore/Storage	Mobile-first, modern browsers, HTTPS only
Hosting & CDN	Firebase Hosting	Static asset delivery, SSL, CDN caching	EU-edge origins, SPA routing support
Auth Service	Firebase Auth	OAuth2 / JWT issuance & verification	Short-lived ID Tokens, SDK + Admin SDK
REST API	Firebase Functions (Node)	Protected endpoints, event triggers	Cold-start latency, auto-scaling
Object Storage	Firebase Storage	Binary image persistence + trigger events	EU-only buckets, IAM per-file rules
NoSQL Database	Cloud Firestore	Metadata persistence, real-time listeners	EU region, security rules per document
Image Editor SDK	Pixlr JS / REST	In-browser editing canvas, transform ops	CORS, external API latency
CI/CD	GitHub Actions + Firebase	Build, test, deploy SPA & Functions	Secrets in GitHub, preview on PRs
Monitoring & Logs	Cloud Logging, Crashlytics	Centralized logs, error & perf monitoring	GDPR-safe logging, alerting via Slack/Email

4) Solution Strategy

Goals:

Architecture Style: Monolith for the MVP phase – simpler deployment, faster development

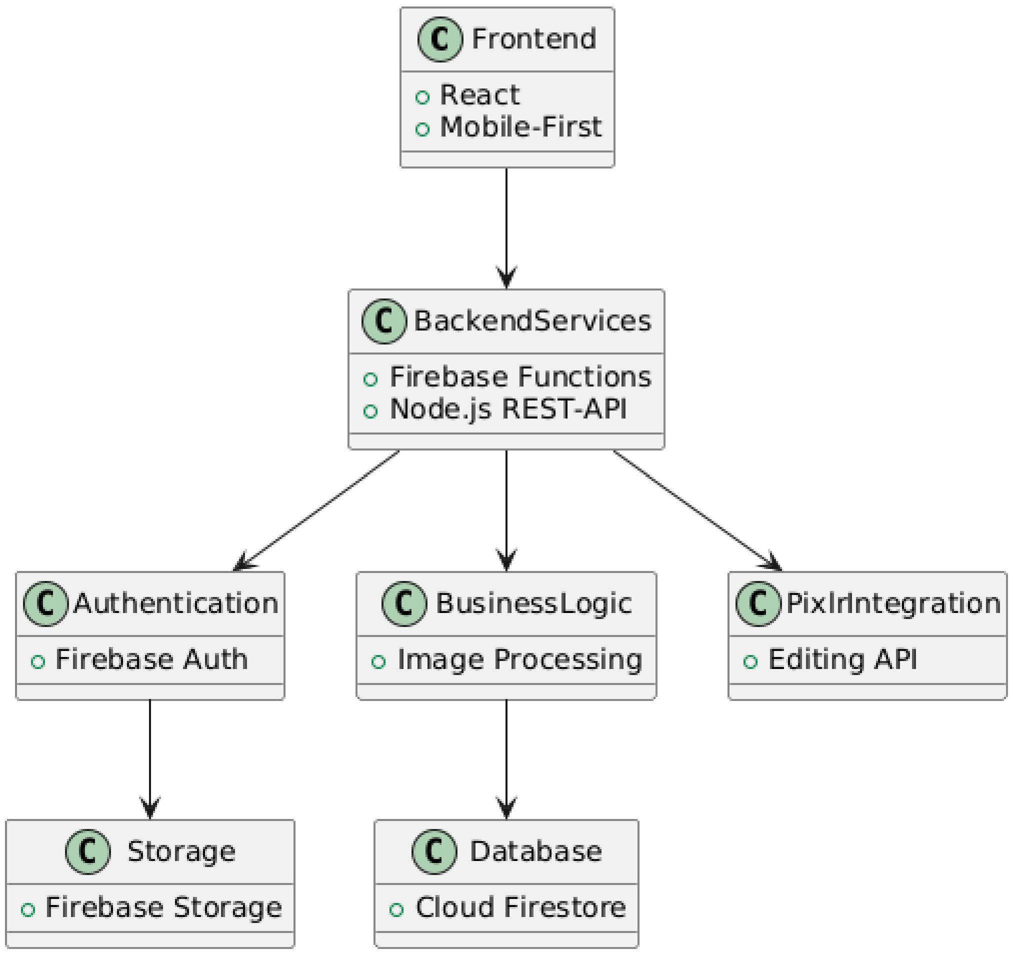
Technology Stack: Frontend: React (cross-platform), Backend: Firebase / Node.js, API: RESTful

Deployment/Operations: Cloud-based deployment using Google Firebase Hosting and Firebase Functions

Quality assurance approach: Unit tests for core logic, stakeholder review meetings

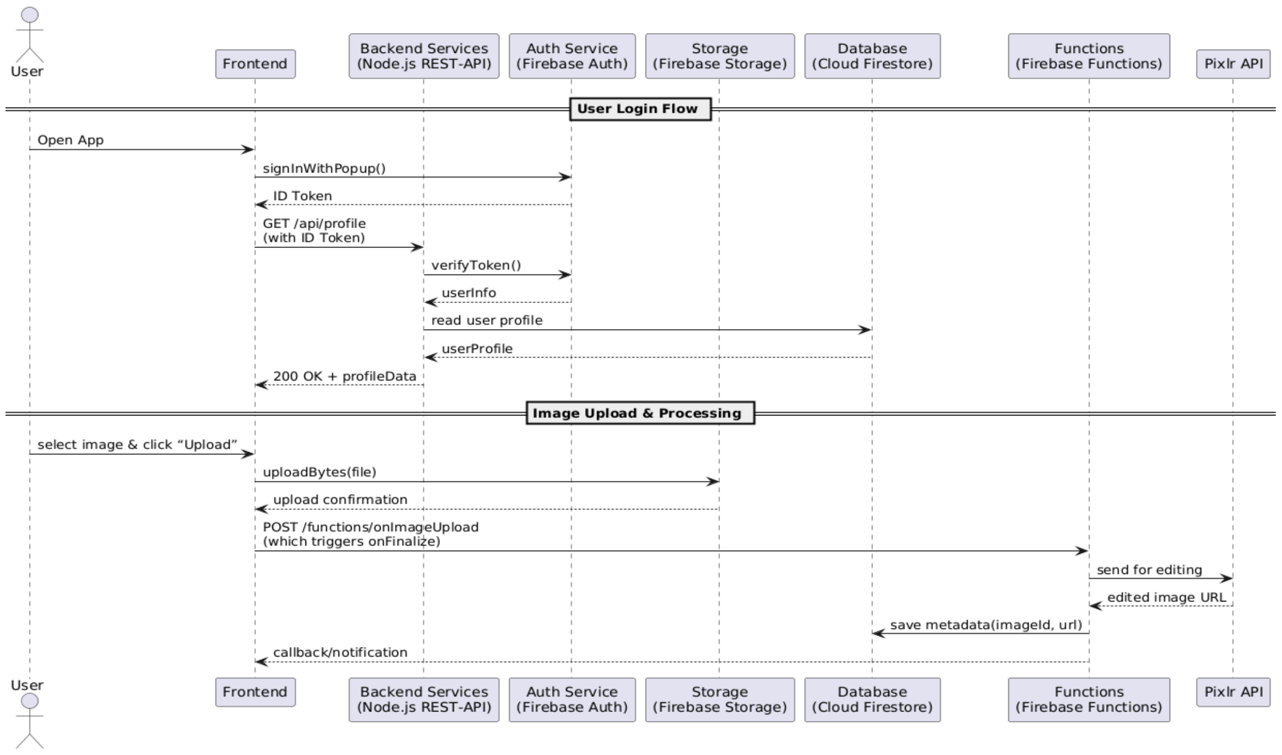
- 80% test coverage, weekly review meetings

5) Building Block view



- **Frontend** – the React SPA, designed mobile-first.
- **Backend Services** – your Node.js REST API running as Firebase Functions.
- **Authentication Module** – Firebase Auth, plugged into both frontend and backend for token issuance & validation.
- **Business Logic Module** – image-specific workflows: upload handling, metadata management, share-link generation.
- **Pixlr Integration** – the adapter that sends images into/receives them back from Pixlr's SDK or REST API.
- **Database** – Cloud Firestore storing user profiles, image metadata, share links.
- **Storage** – Firebase Storage buckets holding the binary image files themselves.

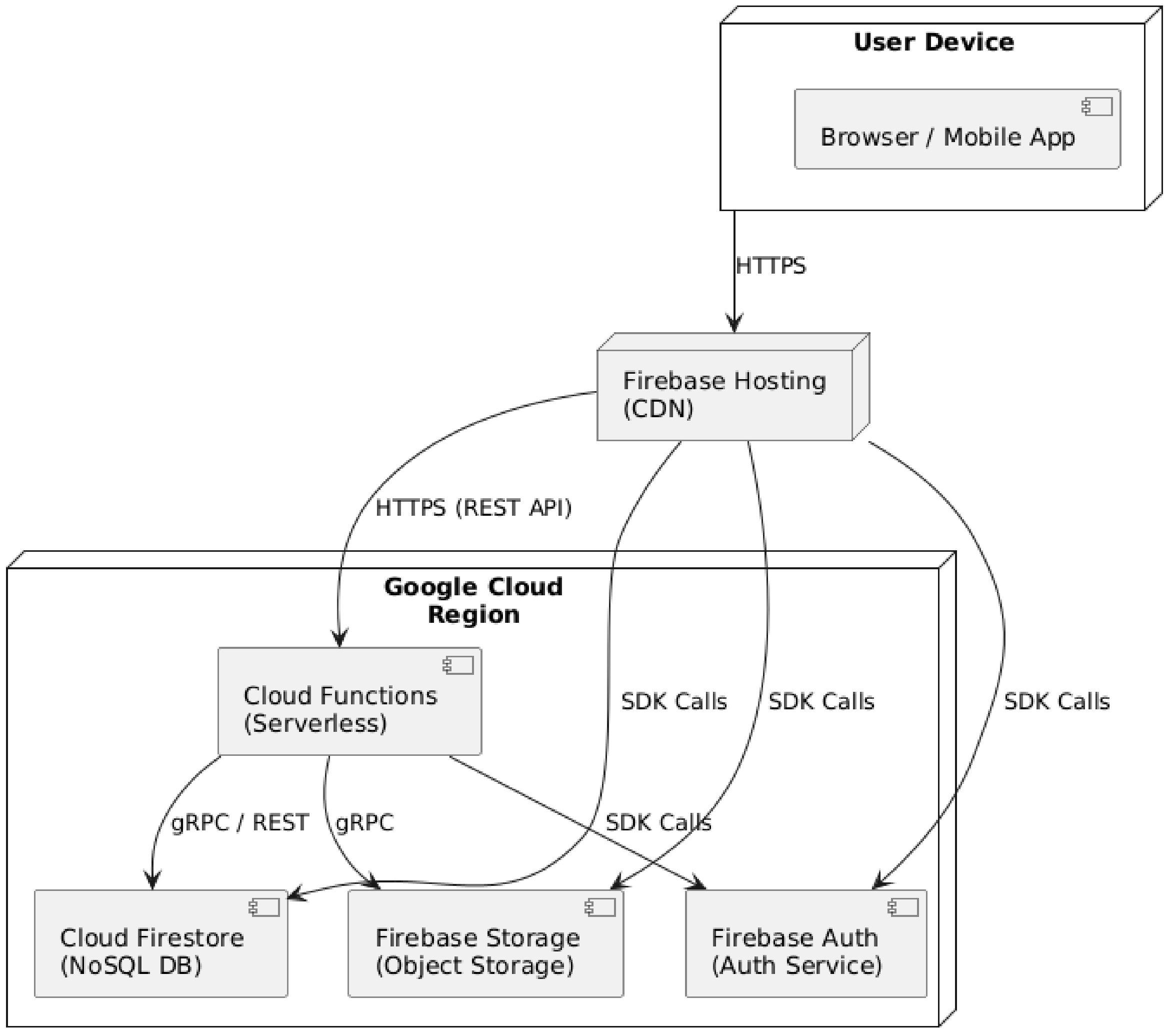
6) Runtime view


User
Frontend
Backend Services (Node.js REST-API)
Auth Service (Firebase Auth)
Storage (Firebase Storage)
Database (Cloud Firestore)
Functions (Firebase Functions)
Pixlr API
User Login Flow
Open App
signInWithPopup()
ID Token
GET /api/profile (with ID Token)
verifyToken()
userInfo
read user profile
userProfile
200 OK + profileData
Image Upload & Processing
select image & click "Upload"
uploadBytes(file)
upload confirmation
POST /functions/onImageUpload (which triggers onFinalize)
send for editing
edited image URL
save metadata(imageId, url)
callback/notification

Purpose: Walks through a few key scenarios step-by-step to show message flows.

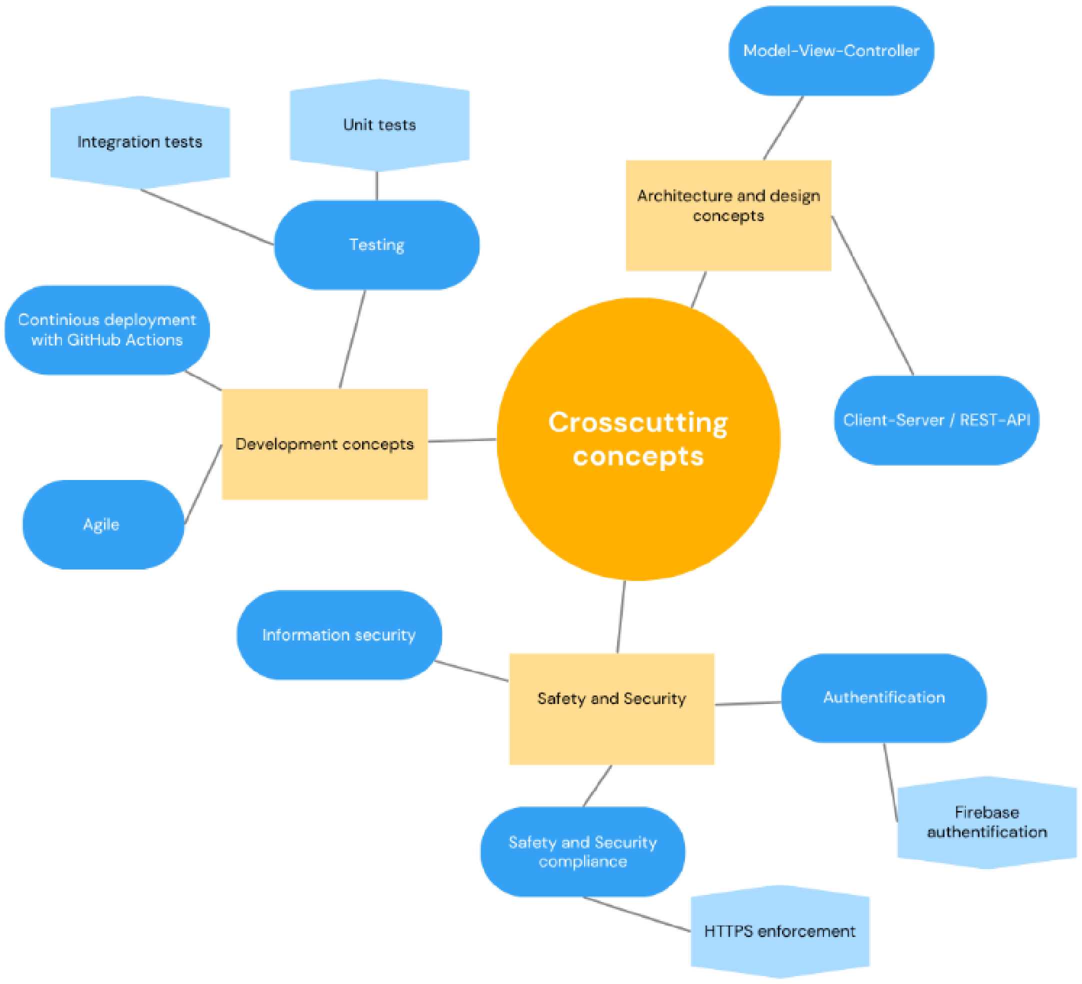
1. **User Login Flow:**
   a. User → Frontend: opens the app.
   b. Frontend → Auth SDK: signInWithPopup().
   c. Auth → Frontend: returns ID token.
   d. Frontend → Backend (Functions): GET /api/profile with token.
   e. Backend → Auth: verifyToken().
   f. Auth → Backend: userInfo.
   g. Backend → Firestore: fetch profile.
   h. Firestore → Backend → Frontend: 200 + profile data.
2. **Image Upload & Processing:**
   a. User → Frontend: selects image + "Upload."
   b. Frontend → Storage: uploadBytes(file).
   c. Storage → Frontend: confirmation + URL.
   d. Storage → Functions: onFinalize trigger fires.
   e. Functions → Pixlr: hand off image for editing.
   f. Pixlr → Functions: edited image URL callback.
   g. Functions → Firestore: save updated metadata.
   h. Functions → Frontend: notify upload/edit success.

7) Deployment view



- **User Device** (Browser or Mobile App)
- **Firebase Hosting (CDN)**
- **Google Cloud Region** containing:
  - Cloud Functions
  - Cloud Firestore
  - Firebase Storage
  - Firebase Auth
- **Arrows / Protocols:**
  - HTTPS from User → Hosting → Functions (API)
  - SDK calls from Hosting or Functions → Firestore, Storage, Auth
- **Notes:** all services are deployed in EU-only regions to satisfy GDPR.

8) Crosscutting Concepts



Decisions:

- We need fast deployment and versioning -> GitHub Actions and Firebase CI/CD -> Streamlines deployment process, fewer manual steps
- Authentication -> Firebase Authentication -> Simple login, but limits to Firebase-supported auth flows
- Project needs must be met within 3 months -> Monolith for MVP -> Fast delivery, harder to scale

9) Architecture decisions

We need fast deployment and versioning: We will use GitHub Actions and Firebase CI/CD: Streamlined deployment process, fewer manual steps

User authentication is required: We will use Firebase Authentication: Simplifies login, but limits to Firebase-supported auth flows

Project needs must be met within 3 months: We will implement a Monolith for MVP: Faster delivery, but harder to scale in the long term

10) Quality Goals and Scenarios

Interoperability: A user edits an image in FocusFrame and shares it via a public link that opens correctly on any platform.

Efficiency: A user uploads a high-resolution image and completes editing with Pixlr; the system saves and displays the result within 2 seconds.

Testability: The image upload and editing modules are covered by unit and integration tests and can be tested without requiring access to Pixlr’s live API.

11) Risks and Technical Debt

Scalability Limitations: As the user base grows, scaling the entire application can become inefficient compared to scaling individual services.

Codebase Complexity Over Time: As features increase, a monolithic codebase becomes harder to maintain, test, and understand (risk of “spaghetti code”).

Limited Flexibility for Teams: Different developers or teams cannot work independently on services – changes in one module may affect the entire system.

12) Glossary

Term	Definition
Architecture Style	The overarching structural pattern of a software solution.
Image Editing Tools	Tools or integrations used for image editing.
GDPR Compliance	Adherence to the EU General Data Protection Regulation: all user data must be stored and processed within the European Union.
Monolith	A software architecture where all components (UI, business logic, data access) are delivered as a single application.
Firebase Hosting	Google's hosting service for static and dynamic web content with automatic scaling.
Firebase Functions	Google's serverless functions platform for executing backend logic in response to events, automatically scaled.
Google OAuth (Firebase Auth)	Authentication protocol using Google accounts via Firebase Authentication, enabling secure sign-in without managing your own password store.
Firebase Storage	Firebase's object storage for storing and retrieving files (e.g. uploaded images), tightly integrated with Firebase Security Rules.
RESTful API	An architectural style for web services using HTTP methods (GET, POST, PUT, DELETE) to expose resources in a well-defined format (typically JSON).
CI/CD	Continuous Integration and Continuous Deployment: automated pipelines (e.g. GitHub Actions, Firebase CI/CD) for code integration, testing, and release.
Mobile-First Design	A design approach that optimizes the user experience for mobile devices first, then adapts to larger screens; prioritizes touch-friendly interfaces and performance on smartphones/tablets.
Scalability	The ability of a system to handle increasing loads (users, data volume) by scaling horizontally or vertically without performance degradation.
Performance	A measure of an application's responsiveness and speed (e.g. image load times under high traffic).
Usability	The ease and intuitiveness with which users can interact with a system, especially across different devices.
Maintainability	The ease with which the codebase can be updated, refactored, and extended as the application evolves.
Interoperability	The capability of a system to exchange and correctly display data and functions across different platforms (e.g. public image links working in any browser).
Efficiency	Resource efficiency, such as rapid image processing and minimal overhead during upload/download operations.
Testability	The degree to which modules can be tested in isolation (unit tests) and together (integration tests) without requiring live external dependencies (e.g. Pixlr API).

Technical Debt	The extra maintenance and refactoring work incurred when shortcuts are taken in code or architecture to deliver quickly, which accumulate over time.
Scalability Limitations	Risks inherent to a monolithic architecture: difficulty scaling individual components independently, leading to inefficiencies as user base grows.
Codebase Complexity	The risk that, as more features are added, a monolithic codebase becomes tangled (“spaghetti code”) and makes refactoring and testing more cumbersome.