TOBB ETÜ

University of Economics & Technology

Design Document

EYAY

Ali Türkücü

Esra Alparslan

Muhammed Yusuf Kartal

Yağız Can Akay

CONTENTS

1) System Overview

Kid-safe web app where children upload a photo of their drawing, select a theme, get an AI-reimagined image, and share it with friends. Parents can oversee activity. Safety and privacy are first-class (avatars only, minimal PII, preset comments).

Primary actors

• Child user (default role)

• Parent user (restricted management actions)

• System services (AI adapter, moderation)

Quality goals (selected)

• Time to transform (upload→AI result shown): ≤120 s avg / ≤180 s p95.

• UI action latency (like/comment): ≤1 s p95.

• Availability during operating hours: ≥95%.

• No free-text comments; preset only.

• No public visibility until moderation (and parental rules if enabled) passes.

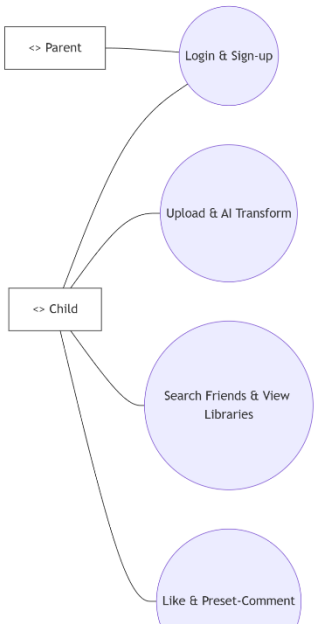


Figure 1: Use Case Diagram

2) Architecture & Stack

Architecture. Layered monolith with modular services and 3rd-party adapters.

```
[React SPA]
  ↓ HTTPS (JWT)
[FastAPI Backend]
  ├─ Auth & RBAC (Child/Parent/Admin)
  ├─ Upload Service → [Object Storage]
  ├─ AI Orchestrator → [Image Gen API]
  ├─ Moderation Service → [Moderation API]
  ├─ Social (posts/likes/preset-comments/friendship)
  └─ Search (users, posts)
[MongoDB + CDN/Object Storage]
```

Figure 2: Architecture

Tech. React (Vite/Next optional), FastAPI (Python 3.11), MongoDB, PyTest, GitHub Actions CI.

Security & Privacy.

• Minimal PII; avatars only for children.

• Strong password hashing; no plaintext secrets.

• RBAC on every endpoint.

• Moderation gate before public visibility.

• Rate-limits on auth, search, likes/comments.

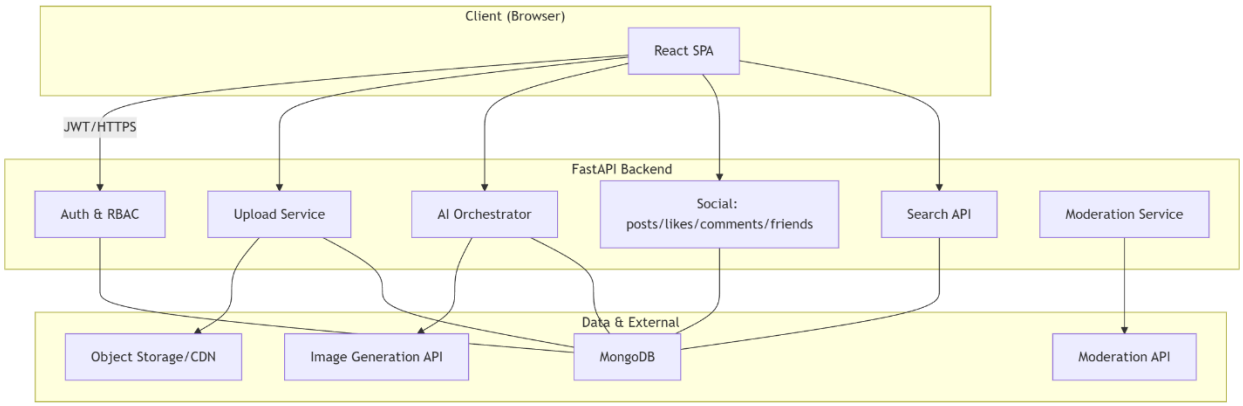


Figure 3: Component Diagram

3) Domain Model (Key Entities)

• **User** { id, username, role['child'|'parent'|'admin'], email?, avatarUrl, parentOf?: [childIds], createdAt }

• **AuthCredential** { userId, passwordHash, passwordAlgo, createdAt }

• **Friendship** { id, requesterId, addresseeId, status['pending'|'accepted'|'blocked'], createdAt }

• **Image** { id, ownerId, kind['original'|'ai'], url, sizeBytes, mime, safe:boolean, createdAt }

• **Post** { id, ownerId, originalImageId, aiImageId, visibility['public'|'friends'], status['pending_mod'|'approved'|'rejected'], createdAt }

• **Like** { id, postId, userId, createdAt } (unique on (postId,userId))

• **Comment** { id, postId, userId, presetId, createdAt } (no free-text)

• **PresetComment** { id, text, locale } (curated list)

• **AuditLog** { id, actorId, action, targetType, targetId, meta, at }

Indexes on User.username (unique, case-insensitive), Friendship pairs, Post(ownerId, status, visibility, createdAt desc), Like(postId), Comment(postId).

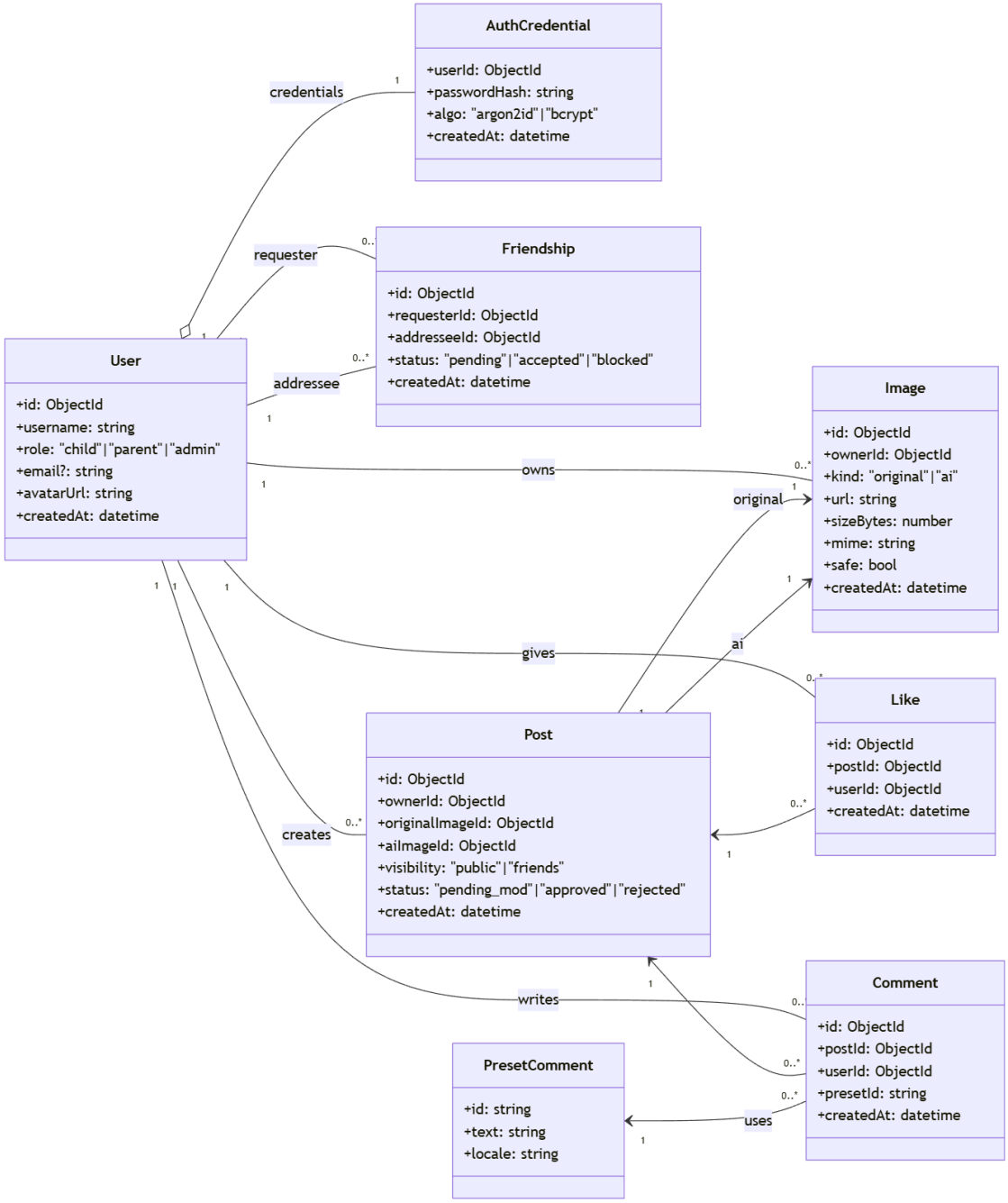


Figure 4: Class Diagram

4) Public API (Selected Endpoints)

Auth.

• POST /api/auth/register → {username, password, role, parentLink?} → 201

• POST /api/auth/login → {username, password} → {accessToken, refreshToken}

• POST /api/auth/refresh → {refreshToken} → {accessToken}

• POST /api/auth/logout → 204

Upload & AI

• POST /api/uploads (auth) multipart {image} → {imageId}

• POST /api/ai/transform (auth) {imageId, theme} → {postId, aiImageUrl, status:'pending_mod'|'approved'}

• GET /api/library?owner=<userId|me> → user's images/posts

Friends & Search

• GET /api/users/search?q=<username> → list of {userId, username, avatarUrl}

• POST /api/friends/{userId}/request → 201

• POST /api/friends/{userId}/accept → 200

• GET /api/users/{userId}/posts?visibility=public → posts for profile

Likes & Preset Comments

• POST /api/posts/{postId}/like → 204 (idempotent)

• DELETE /api/posts/{postId}/like → 204

• GET /api/presets → curated preset list

• POST /api/posts/{postId}/comment {presetId} → 201

• All endpoints enforce RBAC and safety rules (no visibility before approval).

Figure 5: Login Screen

5) Use-Case Designs

UC-1: Login & Sign-up

Goal. Let a user create an account and sign in securely.

Actors. Child, Parent.

Preconditions. User not authenticated.

Main Flow:

1. Sign-up: Client posts to /api/auth/register. Backend validates username uniqueness, password strength, and role constraints (child vs parent).

2. Password is hashed with Argon2id (or bcrypt≥12); user & credential records created.

3. Login: Client posts to /api/auth/login; on success returns JWT access + refresh.

4. Client stores tokens in memory (refresh in httpOnly cookie if desired) and navigates to Library.

Alternate/Exceptions:

• Duplicate username → 409.

• Too many failed logins → 429 (temporary lockout).

• Parent-child linking (optional): parent invites or verifies via code.

Postconditions. Valid session; audit log for registration & login.

Non-functional hooks. Rate limits; SAST checks; unit tests for hashing; p95 login API latency ≤300 ms.

Traceability. SWC-005/007 (security), privacy minimality.

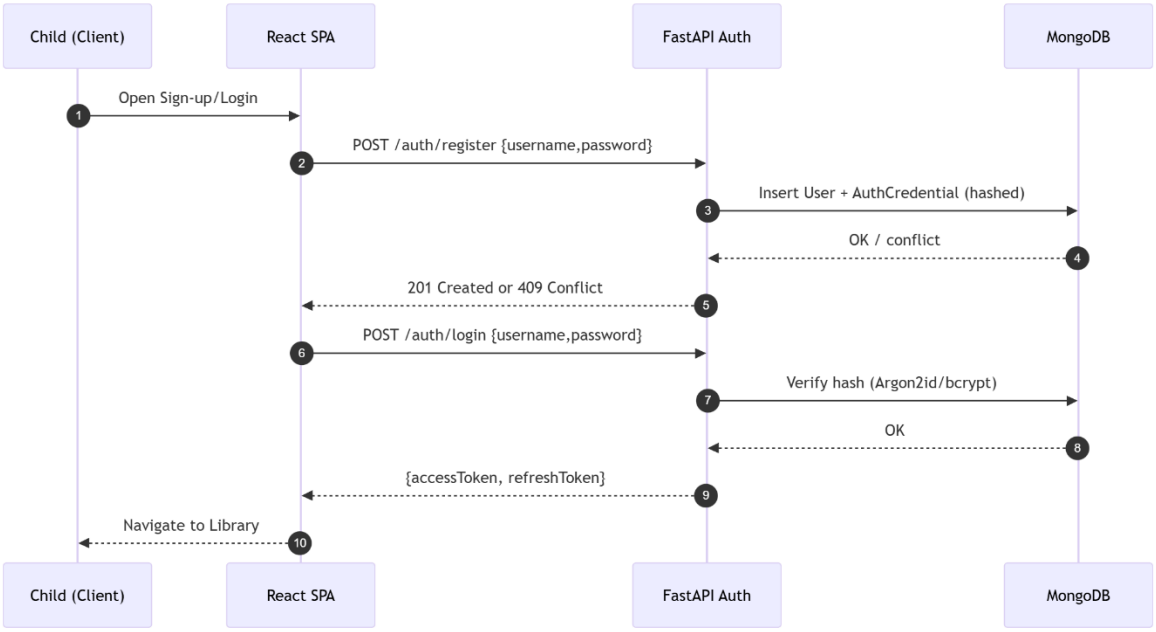


Figure 6: UC-1 Sequence Diagram

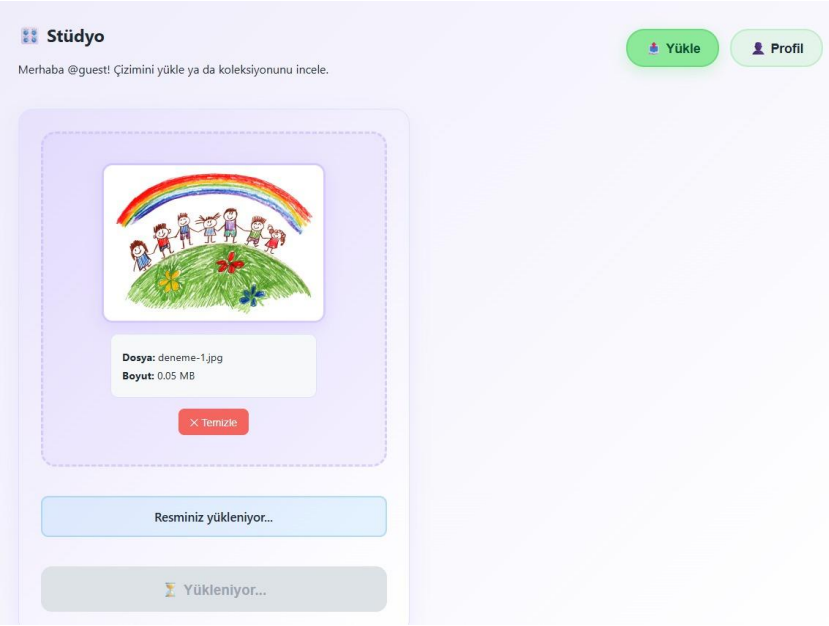


Figure 7: Image Upload Screen

UC-2: Upload Drawings & Transform with AI

Goal. Child uploads a photo of a drawing and gets an AI-reimagined image.

Actors. Child (primary), System services (AI, Moderation).

Preconditions. Authenticated child; image ≤10 MB; MIME ∈ {jpg,png}.

Main Flow:

1. Client → POST /api/uploads with multipart image. Backend validates size/type; stores to Object Storage; creates Image(kind='original', safe=false).

2. Client triggers POST /api/ai/transform {imageId, theme}.

3. Backend AI Orchestrator builds prompt/template, calls Image Gen API; stores AI result as Image(kind='ai').

4. Moderation Service evaluates both images; result approved or pending/rejected.

5. Backend creates a Post linking original & AI images with initial status.

6. Client's Library shows pair; if approved, Share/Visibility controls are enabled.

Alternate/Exceptions:

• AI service timeout → job requeue (backoff); user sees "processing" state.

• Moderation rejects → post stays private with "Needs review" banner; parent can appeal in dashboard (future).

Postconditions. Post record exists; visibility depends on moderation outcome.

Non-functional hooks. E2E timing logs ensure ≤120 s avg transform; upload timeout ≤30 s with 2 retries; storage virus scan optional.

Traceability. SWC-001 (avatars/PII), SWC-004 (≤10 MB), SWC-008 (moderation), performance targets.

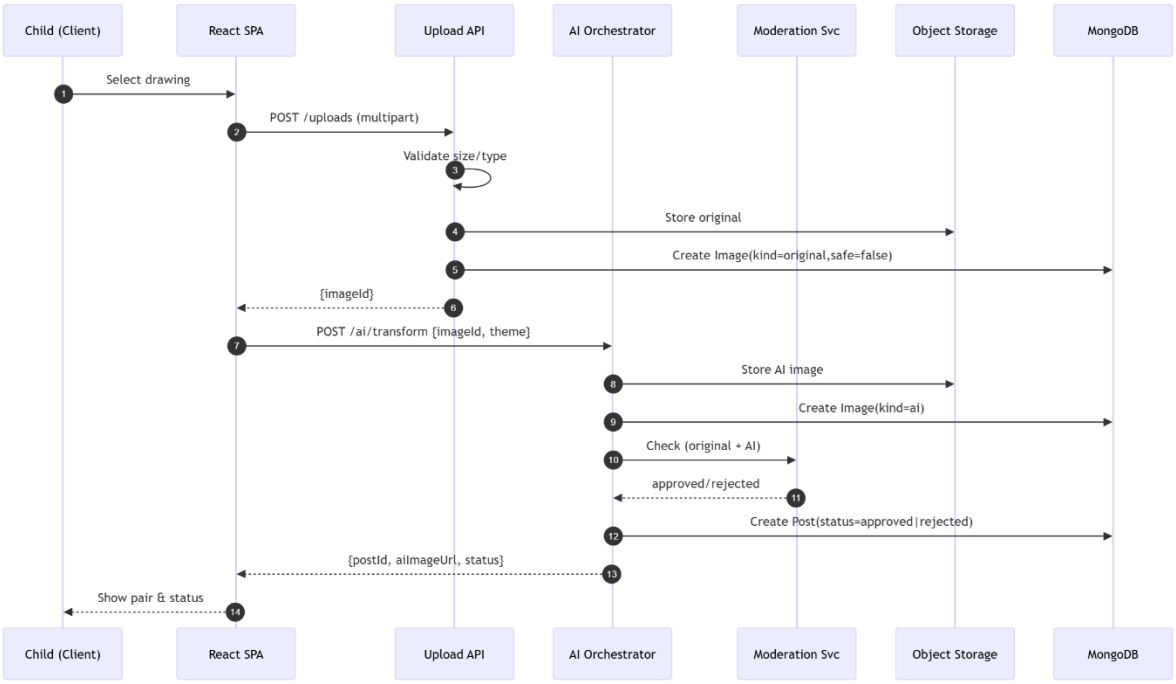


Figure 8: UC-2 Sequence Diagram

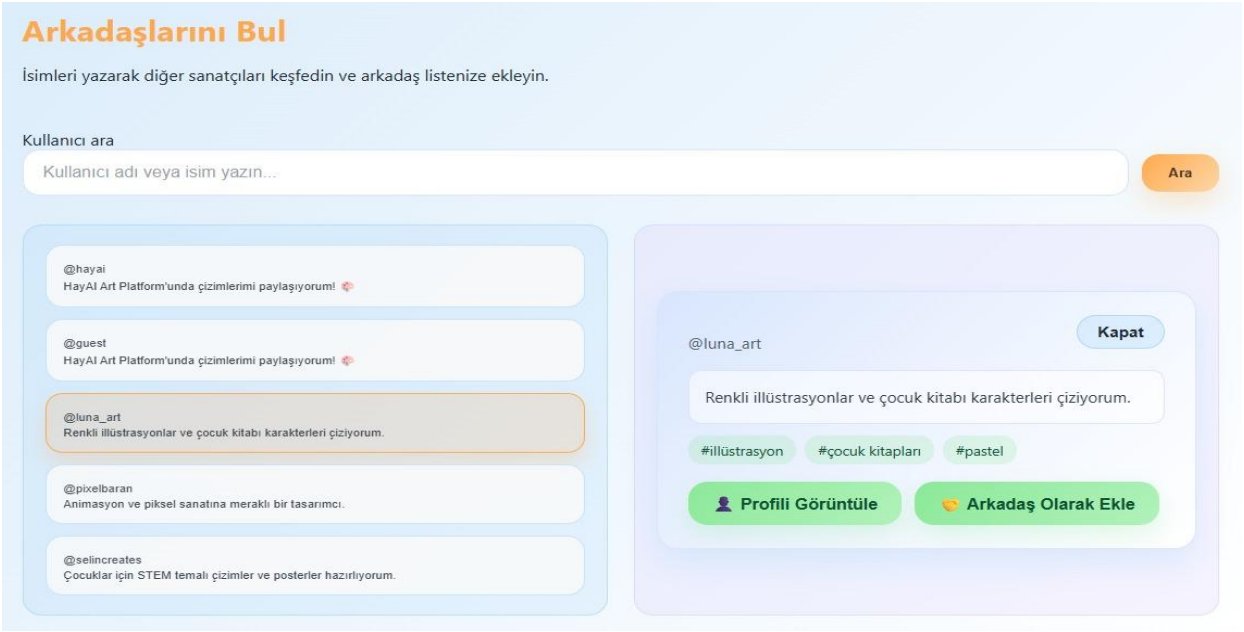


Figure 9: Friend Search Screen

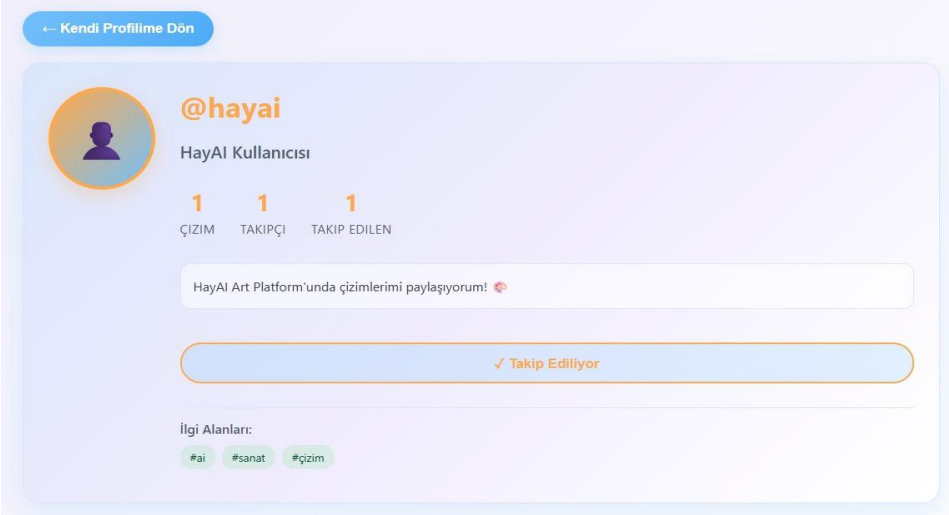


Figure 10: View Friend's Library

UC-3: Search Friends & View Their Libraries

Goal. Find other users and view their approved/public images.

Actors. Child.

Preconditions. Authenticated; search string entered.

Main Flow:

1. Client calls GET /api/users/search?q=… (prefix match; debounced).

2. Results list shows username + avatar. Tap opens Profile.

3. Profile screen calls GET /api/users/{userId}/posts?visibility=public (or friends if friendship accepted).

4. Grid gallery shows only approved posts (moderation-passed).

Alternate/Exceptions:

• No results → “No users found”.

• Private/friends-only posts → prompt to send friend request.

• Blocked relationship → 403.

Postconditions. None (read-only).

Non-functional hooks. Search rate-limit; index on usernameLower; p95 search ≤400 ms; accessibility for keyboard/touch.

Traceability. SWC-011 (compatibility), SWC-008 (visibility after moderation).

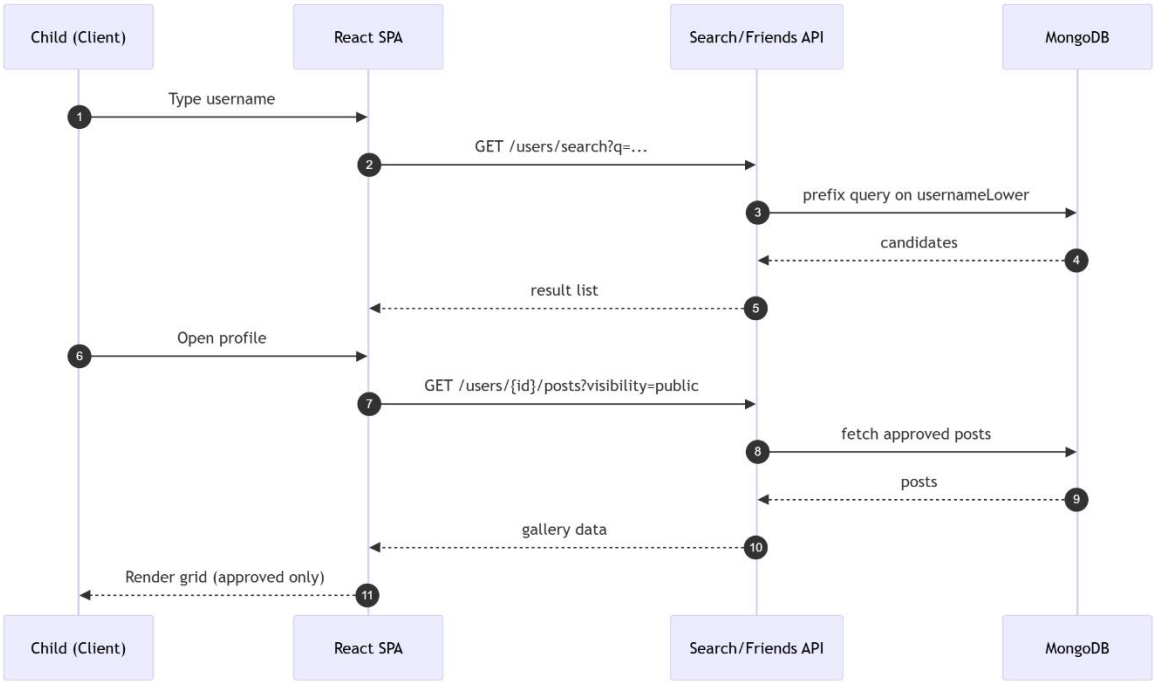
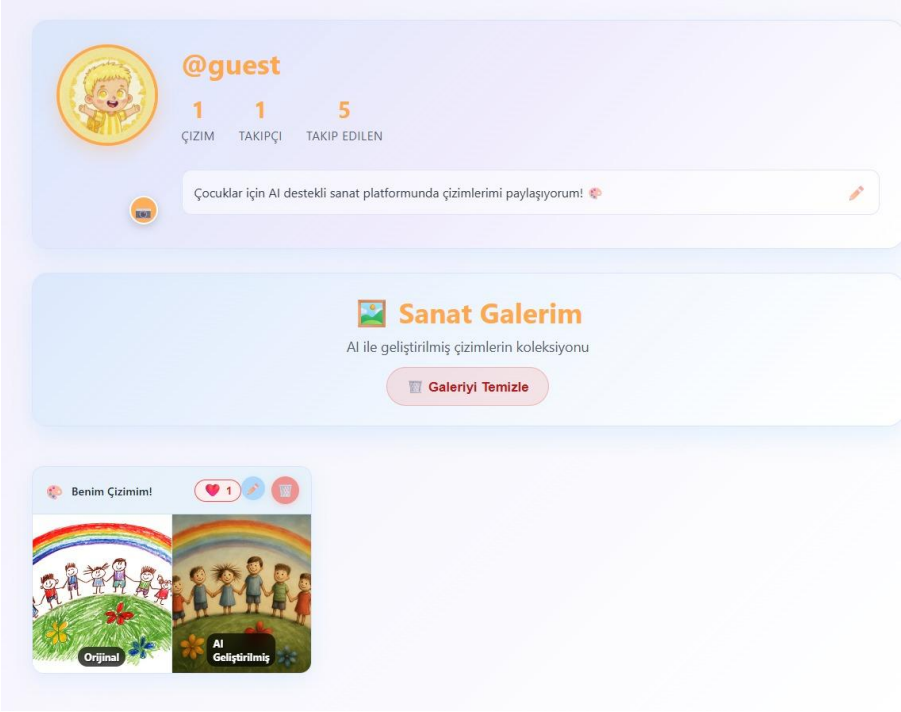


Figure 11: UC-3 Sequence Diagram



UC-4: Like & Preset-Comment on Friends' Images

Goal. Lightweight social interaction with safety.

Actors. Child.

Preconditions. Authenticated; viewing an approved post with permission (public or friends).

Main Flow:

1. Tap Like → POST /posts/{postId}/like (idempotent). Count updates optimistically.

2. Tap Comment → preset menu opens (GET /api/presets cached).

3. Select a preset → POST /posts/{postId}/comment {presetId}. New comment appears.

Alternate/Exceptions:

• Double-tap like toggles off via DELETE /posts/{postId}/like.

• Attempting free-text via devtools → backend rejects (400/403).

• Posting on unapproved/hidden post → 403.

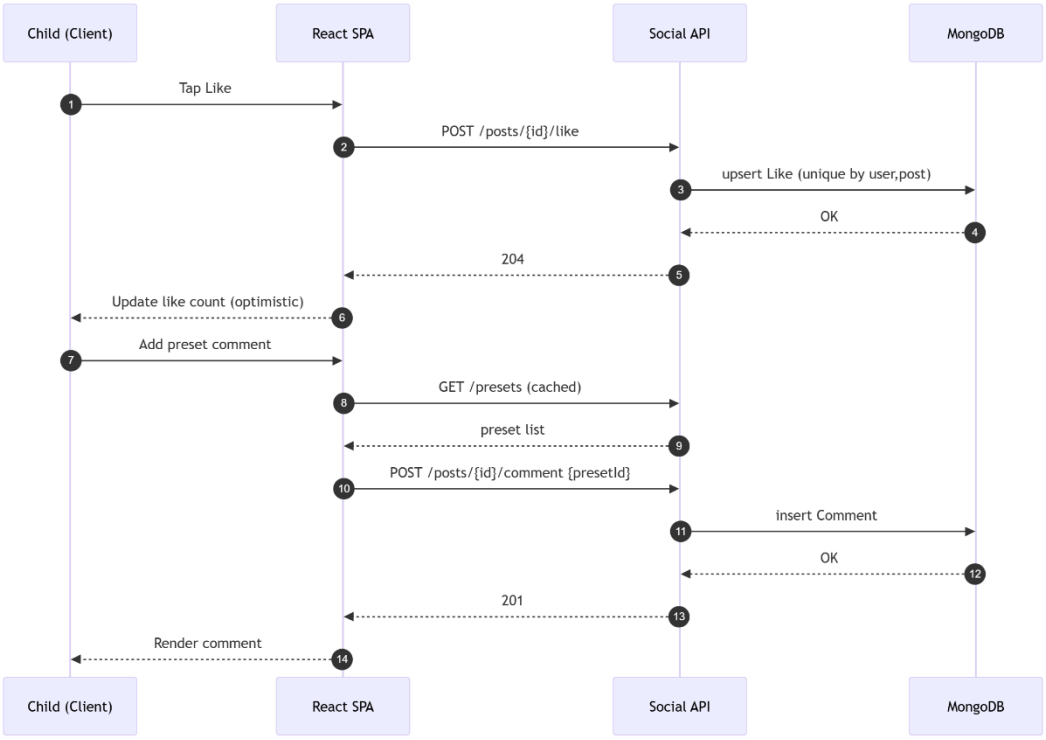


Figure 13: UC-4 Sequence Diagram

Postconditions. Like/comment persisted; audit entry for moderation-sensitive actions.

Non-functional hooks. p95 like/comment ≤1 s; abuse controls (per-user rate-limit); unique like constraint.

Traceability. SWC-002 (no free-text), SWC-003 (UI latency), SWC-008 (approved-only).

6) State & Process Diagrams

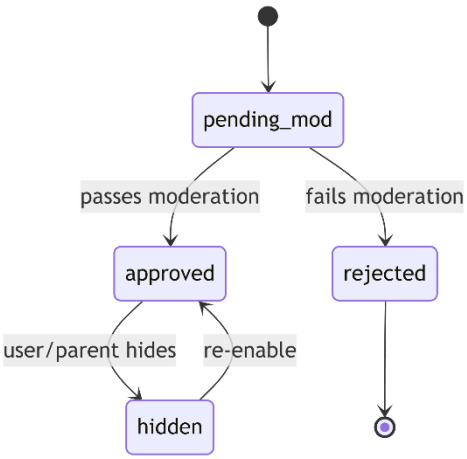


Figure 14: State Diagram

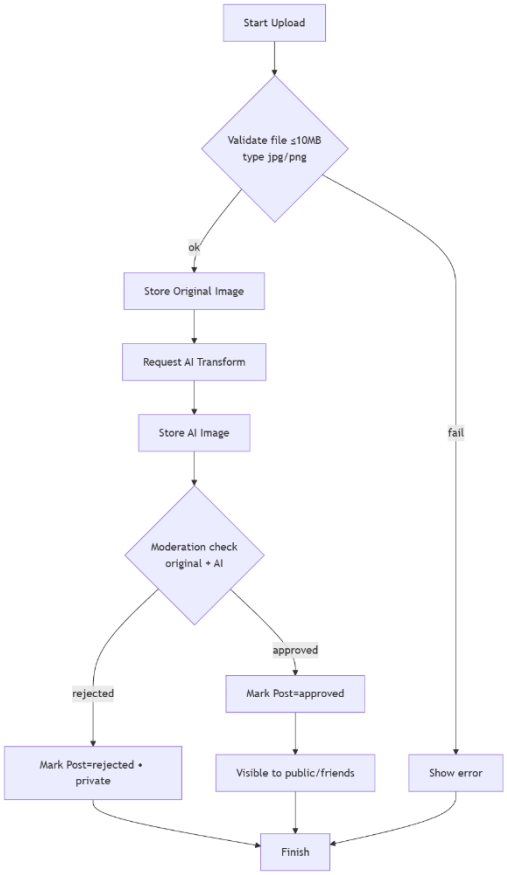


Figure 15: Activity Diagram

7) Design Decisions — Technology Comparisons & Justifications

This section compares at least two options for each key choice and states the decision with a short rationale. Choices align with our constraints (e.g., SWC-004 ≤10 MB uploads, SWC-007 password hashing, SWC-008 moderation gate, UI p95 ≤1 s).

Frontend framework — React vs Vue

- *React:* largest ecosystem, strong testing (RTL/Playwright), team already uses it.
- *Vue:* simpler SFC ergonomics but fewer ready plugins our team knows.
  Decision: React. **Why:** fastest velocity for us; plenty of UI/accessibility libs.

Backend framework — FastAPI vs Express/Nest

- *FastAPI:* type-safe, Pydantic validation, great for mocking AI, Python fits ML/moderation code.
- *Express/Nest:* mature but either too manual (Express) or heavier ceremony (Nest).
  Decision: FastAPI. **Why:** concise endpoints + easy validation to enforce input constraints.

Database — MongoDB vs PostgreSQL

- *MongoDB:* flexible documents for posts/images; quick iteration; Atlas free tier.
- *PostgreSQL:* stronger relational guarantees but more schema work for evolving shapes.
  Decision: MongoDB. **Why:** document model matches posts/images; unique indexes cover integrity (e.g., one like per user+post).

AI provider — Hosted API vs Self-hosted SD

- *Hosted:* safer defaults, predictable latency; pay per generation.
- *Self-hosted:* cheaper per-gen but ops + safety burden.
  Decision: Hosted (behind an adapter). **Why:** safety and speed now; adapter lets us swap later.

Moderation — Hosted API vs Custom model

- *Hosted:* ready models and audits.
- *Custom:* full control but heavy effort.
  Decision: Hosted moderation + manual overrides. **Why:** supports SWC-008 quickly.

Deployment — Managed (Vercel + Render/Fly + Atlas) vs Self-managed VPS

- *Managed:* CI hooks, free tiers, low ops.
- *VPS:* full control but higher maintenance.
  Decision: Managed services. **Why:** quicker to reach targets without SRE overhead.

Risk mitigations (summary)

- AI cost spikes → per-user daily quota and monitoring.
- Moderation misses → preset comments only + parental controls.
- Vendor lock-in → adapter interfaces for AI/moderation/storage, with unit tests.

8) Document-Specific Task Matrix

Document Task	Yusuf	Esra	Ali	Yağız
Drafting main sections	✅			
Domain modeling	✅			
Use-case preparation	✅			
API outline	✅			
Diagrams	✅			
Final editing	✅	✅	✅	✅