FitPalette

CMPT 276

Simon Fraser University

School of Computing Science

Summer 2025

Milestone 1

https://github.com/CMPT-276-SUMMER-2025/final-project-7-hills

Veer Gajjar (Project Manager)

Mehrdad Momeni Zadeh

Parmvir Dhillon

Dilpreet Singh Mann

1

After reviewing our Milestone 0 prototype results, the TA's feedback, and the project's core value proposition, we will move forward with two APIs that directly underpin our six must-have features while keeping cost, complexity, and privacy risks manageable.

1. Google Cloud Vision API

Google Cloud Vision is a REST-based computer-vision service that returns structured metadata for any image. In FitPalette it performs all the "heavy lifting" at the image level:

- Dominant-colour extraction – we call the IMAGE_PROPERTIES endpoint on each garment photo to obtain the top colour swatches (RGB + HEX) that form the outfit palette.

- Garment auto-labelling – the same request's LABEL_DETECTION array tells us whether the item is a "t-shirt", "jeans", "sneaker", etc., so we can pre-fill item names and streamline the upload flow.

- Content-safety gate – Vision's SafeSearch flags let us block non-clothing or inappropriate images and keep the app demo-safe.

All image processing is done with a single low-latency API call, keeping the front-end thin and the user experience snappy.

2. Google Gemini 1.5 Pro

Gemini is Google's multimodal LLM accessible via an HTTP generateContent method. Because we already extract colours locally, we use Gemini strictly as a colour-theory reasoning engine:

- Palette-harmony scoring – we send the four HEX codes from Vision and ask Gemini to rate their overall harmony (0–100) and name the closest classical scheme (complementary, triadic, etc.).

- Complementary-colour suggestions – for low scores, the same prompt requests up to three HEX accents the user could add to improve coordination.

- Warm-vs-cool balance – Gemini also estimates the percentage of warm hues, powering an accessibility-friendly temperature meter.

Only four short colour strings and textual instructions are transmitted—no user images—so privacy is preserved. A single response supplies all three higher-level insights, eliminating the need for multiple niche colour APIs and keeping integration effort minimal.

2

Powered by Google Cloud Vision

1 · Dominant-Colour Extraction
After a user drops in photos of their shirt, pants, shoes, and accessory, the front end sends each image to Vision's IMAGE_PROPERTIES endpoint. The response lists up to ten ranked colour swatches (RGB and hex). We take the top swatch from each garment and display a four-chip "outfit palette" strip beneath the thumbnails.

Benefit to the user They can instantly see the exact colours the system will grade—critical trust-building feedback and a big help for colour-blind users who cannot perceive the hues unaided.

Change from proposal: None. This has always been a core feature.

2 · Garment Auto-Labelling
The same API call returns a LABEL_DETECTION array ("t-shirt", "sneaker"). We grab the highest-confidence clothing label and pre-fill the item name in the upload sidebar. Users may edit it, but nine times out of ten they simply click "Next".

Benefit to the user Eliminates tedious manual typing and prevents mis-labelling, especially valuable for our colour-blind persona, who might rely on text cues more than colour cues.

Change from proposal No change; still part of the main flow.

3 · Content-Safety Gate
By enabling Vision's lightweight SafeSearch flags we can detect adult or non-clothing content. If any flag breaches a safe threshold, the app blocks the upload and shows a friendly warning.

Benefit to the user Keeps the experience family-safe, prevents accidental crashes from unsupported input, and satisfies course demo requirements.

Change from proposal Originally listed as a "backup" feature; promoted to core because it costs almost nothing to implement and materially improves robustness.

Powered by Google Gemini 1.5 Pro

4 · Palette-Harmony Score

When all four garment colours are known, FitPalette sends a concise prompt to Gemini:

Here are four garment colours in HEX: #B22222 #F4A460 #1E90FF #2E8B57.

a) Rate their overall harmony 0-100.

b) Name the closest classical scheme (complementary, triadic, etc.).

Respond in JSON.

Gemini answers with a score, the scheme name, and a one-line rationale. We surface the number on a dial and the scheme label beneath it.

Benefit to the user Delivers an immediate, objective "how well does this outfit work?" answer—exactly the confidence boost Emily (everyday user) wants.

Change from proposal Previously planned to rely on Adobe Color; refactored to Gemini for availability and simplicity. Behaviour is unchanged for users.

5 · Complementary-Colour Adviser

If the harmony score falls below 60 %, we extend the prompt:

c) Suggest up to three complementary accent colours in HEX.

d) Give a short tip explaining how to use them.

Returned HEX codes render as clickable "accent chips" beside the lowest-scoring garment.

Benefit to the user Actionable, specific improvements instead of a vague "your colours clash," helping content-creator Marcus nail on-camera looks quickly.

Change from proposal Same user value as the earlier "Complementary Matching" idea, now driven by Gemini rather than a specialist colour API.

6 · Warm–Cool Balance Meter

The Gemini prompt also asks:

e) Estimate overall warm/cool balance as a percentage warm (0-100).

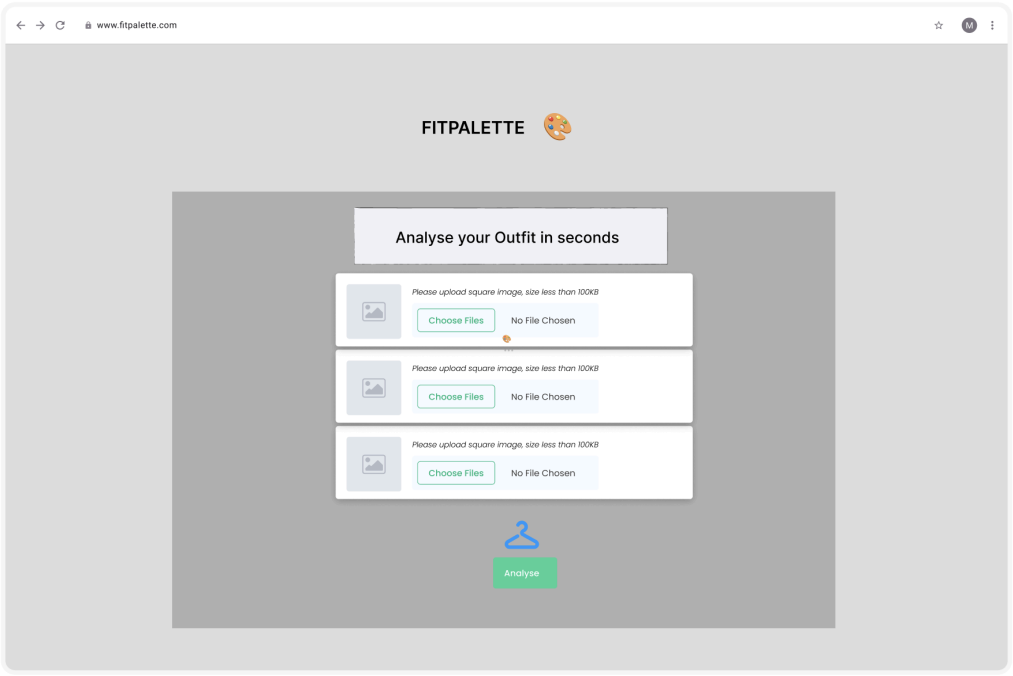
That number drives a thermometer-style meter plus an accessibility label ("Palette skews 72% warm, consider adding a cooler accent"). If warm or cool dominance exceeds 80%, we nudge the user with a gentle suggestion.

Benefit to the user Helps achieve temperature balance—a key visual principle—and is indispensable for colour-blind users who may not sense warmth shifts.

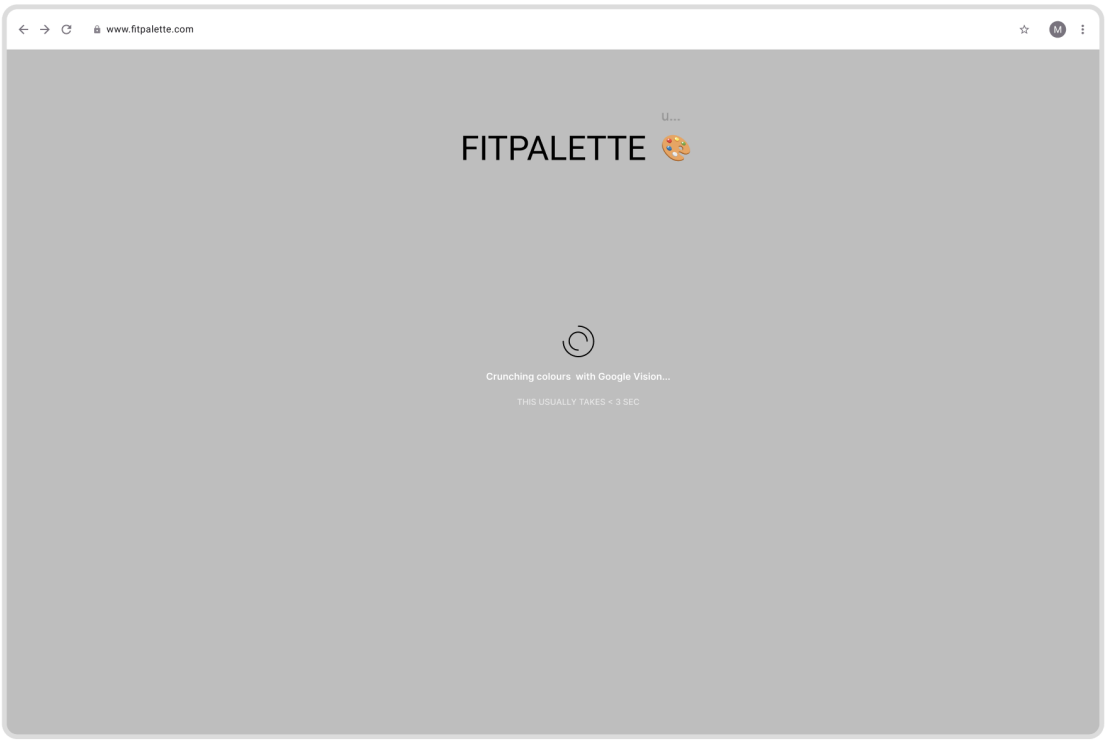
Change from proposal This feature was a "backup" in Milestone 0; elevated to primary after we trimmed lower-priority ideas (background removal and Unsplash inspiration) to keep the scope lean.

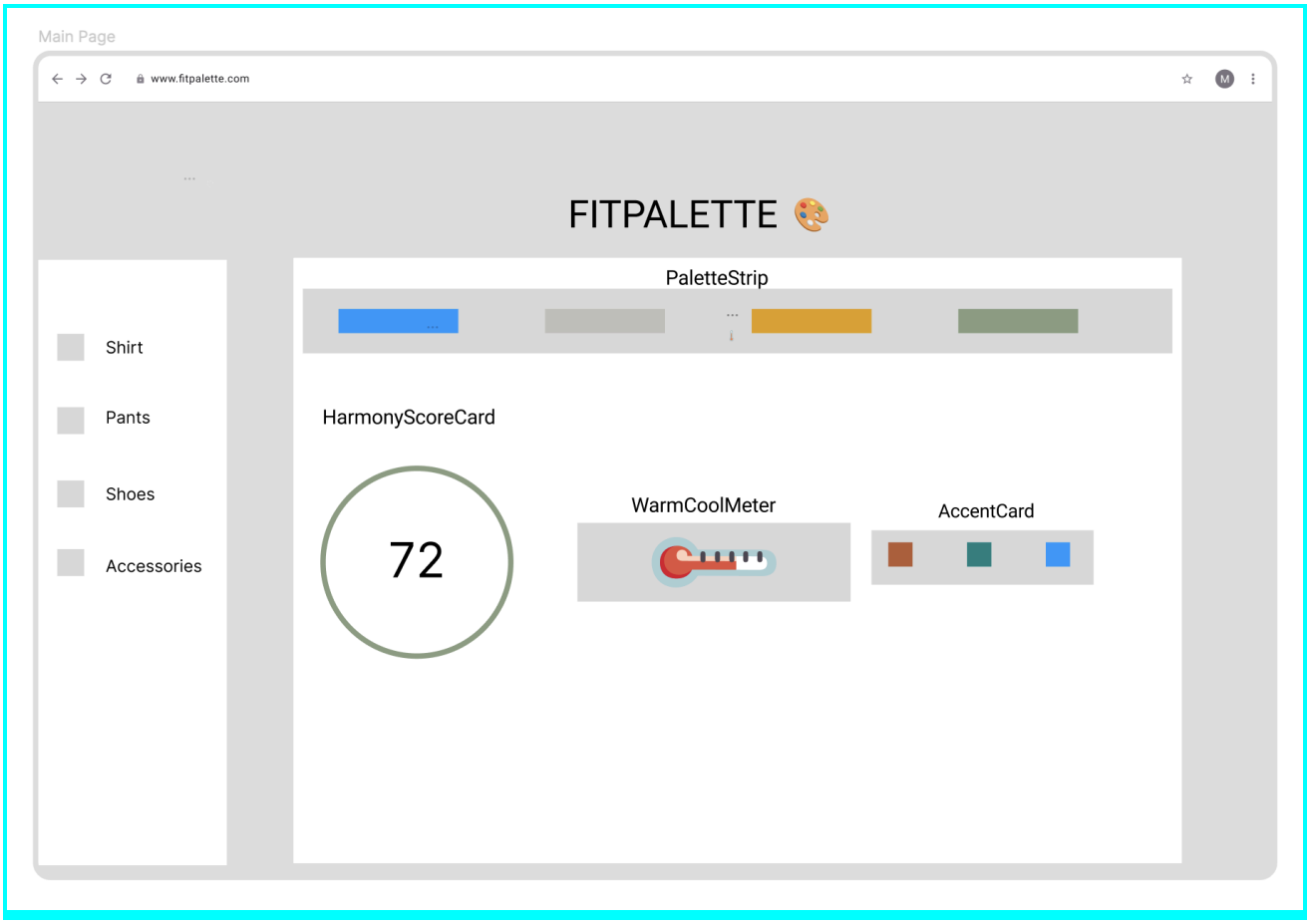
3 Prototype → Figma Images Attached Below:

Home Page



Processing Page





4

Selected SDLC model: Kanban with explicit WIP-limits

We chose Kanban for three intertwined reasons:

1. Timebox reality & lightweight ceremony Our project window is barely five-six weeks. Kanban lets us start coding on day 1 without standing up sprint planning, velocity charts, or story-point poker. The only ritual we'll keep is a brief daily stand-up; everything else lives on the visual board.

2. Highly fluid scope Working with two external Google APIs (Vision and Gemini) means discoveries will pop up almost every day—rate-limit quirks, prompt tweaks, UI polish after user tests. Kanban's pull-based flow makes reprioritising trivial: we simply reorder the backlog or drop a new card in, with no mid-sprint "scope-creep" drama.

3. Small, parallelisable feature slices Our six features break down into bite-sized tasks (upload grid, Vision wrapper hook, harmony dial, etc.). Each can move independently across the board, so every teammate stays unblocked. WIP limits (max 2 cards per dev) keep us from half-finishing ten things and instead push each card to "Done" before the next starts—a perfect match for a front-end-heavy build.

Add in continuous deployment via GitHub Actions (lint, test, auto-deploy to Vercel on every merged PR) and Kanban gives us relentless, visible progress with near-zero overhead, exactly what we need to turn a mid-fi prototype into a polished web app by the final showcase.

5

P0, P1, P2, P3 are simply shorthand priority levels we'll tag onto each GitHub Issue so our team (and the TA) can instantly see what must ship first and what can slide if time gets tight.

P0 ("Blocker / Must-have") : Absolutely critical for the project to compile, deploy, or meet a rubric item. If a P0 isn't finished, the next task can't even start. Complete in week 1 or immediately when created.

P1 ("High"): Core user-visible functionality or infrastructure that we need for demo day, but it's not literally blocking the repo from running. Tackle as soon as all P0s for that area are done—usually week 2–3.

P2 ("Medium"): Important polish, accessibility improvements, responsive layouts—nice-to-have but the app still works without them. Week 3–4 buffer zone; finish if time allows.

P3 ("Stretch / Icebox"): Experimental or bonus items that we'd love to show off but will drop the moment schedule pressure appears. Only after all higher priorities are green.

Work-Breakdown Structure (WBS):

1 · Project Bootstrap

1.1 Initialize public GitHub repo with src/, docs/, misc/, README.md → P0
1.2 Add ESLint + Prettier config & Husky pre-commit hook → P0
1.3 Configure GitHub Actions: npm ci, lint, vitest, deploy to Vercel preview → P0

Dependency: none (start day 1).

2 · Design Assets

2.1 Import mid-fi frames from Figma → /docs/mockups/ PNGs → P1
2.2 Extract colour tokens & typography scale into Tailwind config (tailwind.config.cjs) → P1

Dependency: Figma work finished.

3 · Front-end Infrastructure

3.1 Set up React Router skeleton (Upload → Results) → P0
3.2 Add TanStack Query for API data fetching/caching → P1
3.3 Define global context (PaletteContext) for shared colour state → P1

Dependency: repo & lint ready.

4 · Vision-Driven Flow

4.1 Component <UploadGrid /> — drag-and-drop + file-picker → P0
4.2 Hook useVision() — POST image → Google Vision, parse colours + labels → P0
4.3 Progress modal while Vision calls resolve → P1
4.4 Store results in PaletteContext → P1
4.5 SafeSearch guard: block + toast on flagged images → P1

Dependency: TanStack Query (3.2).

5 · Gemini-Driven Analysis

5.1 Draft prompt & response JSON schema → P0
5.2 Hook useGeminiPalette() — send colours → Gemini, parse harmony, accents, warm% → P0
5.3 Implement request caching + timeout fallback → P1

Dependency: Vision colours available (4.4).

6 · UI / Visualisation Components

6.1 <PaletteStrip /> — four colour chips with copy-to-clipboard → P1
6.2 <HarmonyDial /> — SVG gauge animating to score → P1
6.3 <WarmCoolMeter /> — thermometer bar + nudge banner → P2
6.4 <AccentChips /> — render Gemini accent colours, tooltip copy → P1
6.5 Responsive layout & mobile swipe gallery → P2

Dependency: Gemini data (5.2) for 6.2-6.4.

7 · Accessibility & UX Polish

7.1 Add ARIA labels to all controls → P2
7.2 Keyboard navigation for upload and result widgets → P2
7.3 Colour-blind preview toggle (sim deuteranopia) → P3 (stretch)

Dependency: core UI components in place.

8 · Testing & Quality Assurance

8.1 Unit tests for useVision, useGemini parsing logic (vitest) ⟶ P0
8.2 Component tests for UploadGrid & HarmonyDial (Testing Library) ⟶ P1
8.3 Cypress e2e happy-path (upload → results) ⟶ P1

Dependency: related code complete.

9 · Deployment & Demo Readiness

9.1 Configure Vercel production branch protection ⟶ P0
9.2 Add 404.html redirect for client-side routing ⟶ P1
9.3 Record 5-min demo video (script, record, edit) ⟶ P1
9.4 Generate docs/architecture.md (data-flow, MVC, risk matrix) ⟶ P1

Dependency: functional app stable (6.x).

10 · Project Management & Housekeeping

10.1 Create GitHub Project board (columns: Backlog, In Progress, Review, Done) ⟶ P0
10.2 Write each WBS item as an Issue; label with [feat], [chore], [test], priority (P0–P3) ⟶ P0
10.3 Weekly checkpoint meeting + retro notes in /docs/meeting-minutes/ ⟶ recurring

Dependency: none.

How we prioritise & sequence

- First week: tackle all P0 items (repo, CI, router, UploadGrid, Vision hook, prompt schema, board setup).
- Second week: finish Vision flow end-to-end; start Gemini integration.
- Third week: knock out UI widgets and core Gemini-powered scoring; begin tests.
- Fourth week: responsive/mobile tweaks, accessibility, extra tests, buffer for API surprises.

- Fifth week: polish, deploy, record demo, final documentation.

By converting each numbered sub-task into a GitHub Issue (with the same ID in the title), linking dependencies via "blocked by #xx", and enforcing WIP-limits on the Kanban board, we'll maintain clear ownership and visible momentum all the way to showcase day.

6

Week 0 | Thu 3 Jul → Sun 6 Jul — Kick-off & M1 deliverables

- Thu 3 Jul – Repo scaffold, Kanban board, and group contract already on GitHub.
- Fri 4 Jul – Internal 5 p.m. deadline: M1 Planning Report PDF finalised; peer-review, spell-check, submitted to Canvas one day early.
- Sat 5 Jul – Record and edit the 60-sec M1 walk-through video; export MP4.
- Mon 7 Jul – Upload video to Canvas by noon (ahead of the course's Monday-night cut-off).
  Buffer: a full 24 h margin on both report and video in case of last-minute edits.

Week 1 | Mon 7 Jul → Sun 13 Jul — Vision flow online

- Tue 8 Jul – <UploadGrid /> functional with drag-and-drop + preview.
- Thu 10 Jul – useVision() hook returns dominant colours + auto-labels; data saved to context.
- Sun 13 Jul – SafeSearch gate tested; Vision track considered "Done".
  Buffer: 8 days before the TA check-in window, giving room for bug-fixes.

Week 2 | Mon 14 Jul → Sun 20 Jul — Gemini integration

- Wed 16 Jul – Prompt and JSON schema frozen; manual Postman call green.
- Fri 18 Jul – useGeminiPalette() hook delivers harmony score, warm%, accents (< 700 ms).
- Sun 20 Jul – Caching + timeout fallback coded; unit tests pass CI.
  Prototype is now fully functional for the upcoming M1.5 check-in.

Week 3 | Mon 21 Jul → Sun 27 Jul — UI widgets & M1.5 check-in

- Tue 22 Jul – TA M1.5 meeting (any slot Jul 21–25). Show live prototype.

- Thu 24 Jul – <PaletteStrip />, <HarmonyDial />, <WarmCoolMeter /> rendering live data.

- Sun 27 Jul – Mobile breakpoint validated; swipe gallery works.
  Buffer: two spare days inside the check-in week for TA-driven tweaks.

Week 4 | Mon 28 Jul → Sun 3 Aug — Accessibility, testing, documentation

- Tue 29 Jul – ARIA labels & keyboard nav score ≥ 95 in Lighthouse.

- Thu 31 Jul – Cypress end-to-end test: upload 4 sample images → score.

- Sun 3 Aug – Code-freeze tag v1.0-rc1; docs/architecture.md & risk matrix committed.
  Buffer: three days remain before the final code deadline.

Week 5 | Mon 4 Aug → Fri 8 Aug — Polish & final delivery

- Mon 4 Aug – Bug-bash; any critical fixes must merge by 6 p.m.

- Wed 6 Aug – External deadline: submit final code + report (M2).
  Internal target: noon, giving an 11-hour buffer.

- Thu 7 Aug – Record 5-min showcase video; dry-run presentation with TA.

- Fri 8 Aug – Presentation day — deliver live demo; upload video link if required.

7

Low-risk issues

Low-1 — Token UI glitches on uncommon browsers.
Because we're building first in Chrome, small layout breaks may appear in Safari or Edge. Mitigation: dedicate one half-day in Week 4 to a cross-browser sweep; use Tailwind's vendor-prefix utilities and rely on CSS grid/flex (well-supported specs).

Low-2 — Minor design disagreements inside the team.
Differences in spacing, icon choices, etc., can burn time. Mitigation: lock the Figma style guide in Week 0 and treat it as the single source of truth—no debating pixels during implementation.

Low-3 — ESLint/Prettier friction slowing commits.
Formatting rejects can block PR merges. Mitigation: add a pre-commit hook that auto-fixes lint and prettier on file save, so the CI pipeline never fails for style alone.

Low-4 — GitHub Actions minute quota exceeded on free tier.
If PRs trigger too many CI runs, workflows may queue. Mitigation: keep tests efficient (<60 s) and squash/fetch-merge feature branches to reduce CI invocations.

Low-5 — Vision colour extraction slightly off for over-exposed photos.
Bright sunlight can skew dominant swatches. Mitigation: show a tiny "Adjust colour" eyedropper (stretch goal) or instruct users to retake photos under neutral light; doesn't break the core flow.

Medium-risk issues

Medium-1 — Google Vision rate-limit (1,800 req/min) accidentally breached during demo rehearsal.
A group member might spam uploads. Mitigation: throttle the useVision hook to max 5 concurrent calls and cache results per file hash; rehearse with shared demo images.

Medium-2 — Gemini prompt drift produces inconsistent harmony scores.
Model updates can alter outputs week-to-week. Mitigation: lock model=gemini-1.5-pro-latest, set temperature=0.2, and round scores to the nearest integer. Cache responses in localStorage.

Medium-3 — Unexpected Gemini outage during showcase.
LLM endpoints occasionally hiccup. Mitigation: client-side fallback: a quick hue-variance function yields a "basic" score so the UI never shows zeros; display a banner "Colour advice is simplified due to server load".

Medium-4 — Team member's exam week overlaps crunch time.
Personal workloads spike mid-July. Mitigation: keep tasks granular; daily stand-ups expose emerging overload; Kanban allows another dev to pull any stalled card.

Medium-5 — Merge conflicts as multiple people edit the same Tailwind file.
Utility classes invite line-level clashes. Mitigation: agree on "one-component-per-file" convention; run git pull --rebase before push; resolve conflicts immediately in pair programming sessions.

High-risk issues

High-1 — Vision API suddenly requires billing verification or shuts down free quota.
No colour extraction = no app. Mitigation: pre-enable a low-spend billing account with a $5 alert cap; stash a fallback open-source colour extractor (tinycolor + k-means) for emergency use.

High-2 — Gemini free tier quota exhausted just before final demo.
LLM calls hard-fail, killing scoring features. Mitigation: monitor usage weekly; if forecast exceeds 90 %, purchase the minimal paid plan (split cost) or switch prompt to PaLM 2 temporarily.

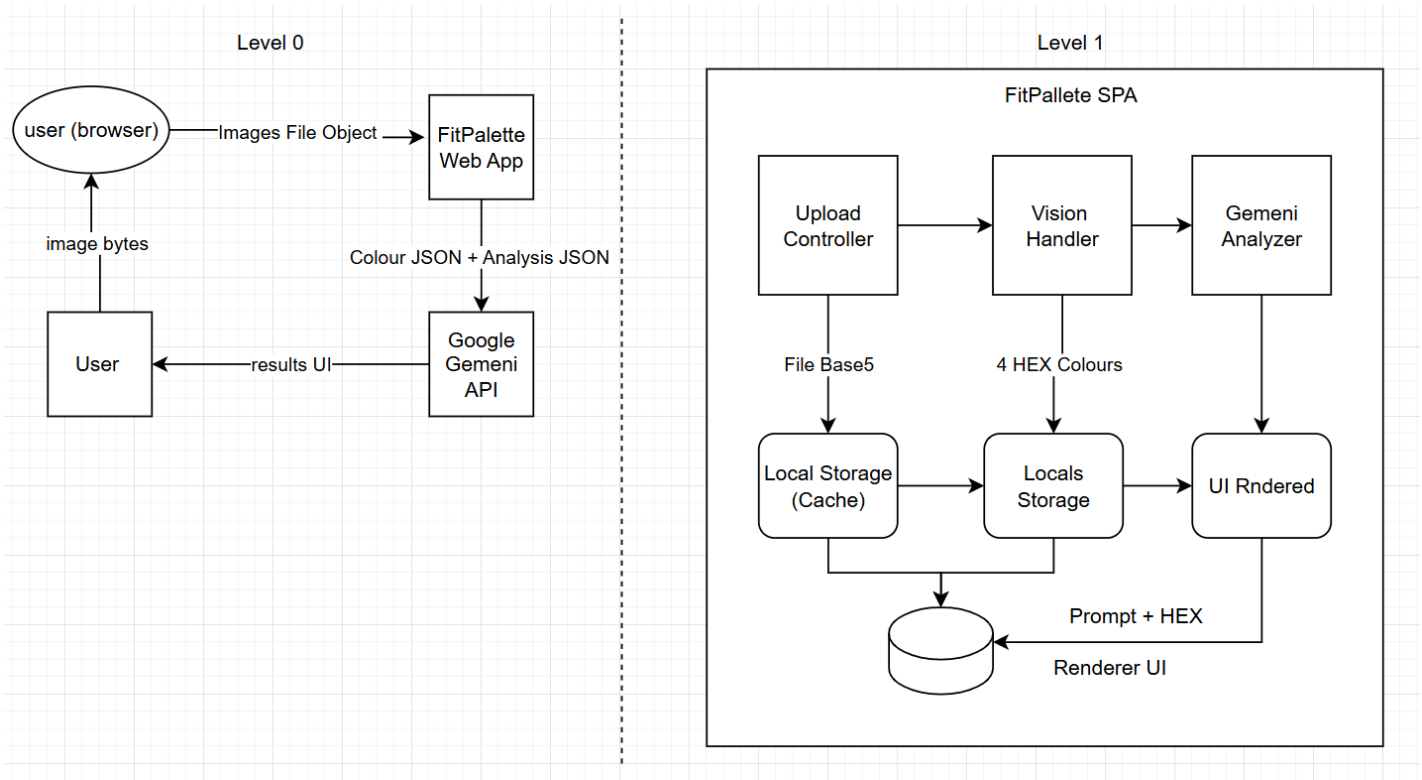
High-3 — Critical team laptop or SSD failure.
Loss of local dev environment near deadline. Mitigation: mandate pushing to GitHub at end of every coding session; Vercel preview builds act as off-site artifact; keep a spare campus workstation image.

High-4 — Security/privacy breach: someone uploads a photo with sensitive personal info, we store it.
Could violate university policy. Mitigation: never persist raw images—convert File objects to base64 only in the browser, send to Vision via HTTPS, then immediately revoke ObjectURL; cache just the colour hexes.

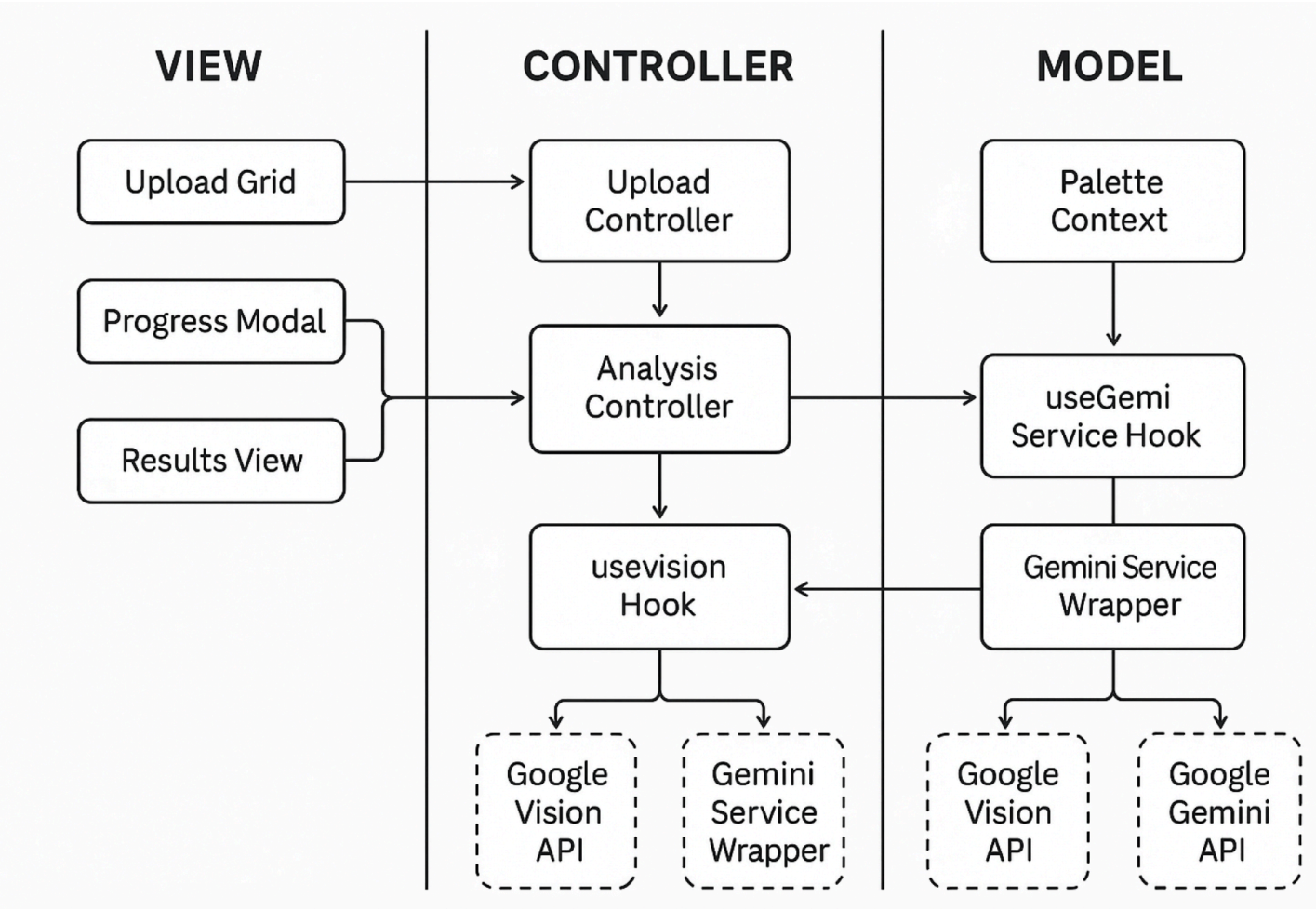
High-5 — Severe illness or personal emergency sidelines the primary frontend dev.
Throughput drops 50 %. Mitigation: pair-program critical components (UploadGrid, Result widgets) so at least two members understand each code area; maintain a living README with setup scripts for quick onboarding.

8

Below is a rough sketch of the flow diagram of our FitPalette drawn in draw.io and pasted here:



9MVC Architecture



10

1. Team roster & planned contributions

- **Veer (Team Lead / Full-Stack Integrator)** – owns the overall architecture, sets up the GitHub Actions CI/CD pipeline, implements the Gemini service wrapper and prompt-engineering, and coordinates weekly retros.

- **Dilpreet (Front-End Lead)** – converts the Figma mid-fi into React + Tailwind components, focuses on the results screen widgets (HarmonyDial, WarmCoolMeter, AccentChips), and ensures responsive behaviour on mobile.

- **Parmvir (API & Data Layer)** – writes the UploadController and Vision wrapper, handles image-hash caching in LocalStorage, and stress-tests Vision and Gemini quotas with synthetic loads.

- **Mehrdad (UX / Accessibility & Testing)** – drives usability sessions, adds ARIA labels and keyboard navigation, authors all Cypress end-to-end tests, and edits the five-minute showcase video.

All four members attend the daily 09:30 Discord stand-up and share sprint notes in /docs/meeting-minutes/.

2. Changelog since Milestone 0

- **API line-up finalised** – Dropped Remove.bg and Unsplash, replaced Adobe Color with Google Gemini; TA approval secured on 3 Jul.

- **Feature set reshuffled** – Promoted SafeSearch gate and Warm-Cool balance meter to primary features; background removal and outfit inspiration moved to stretch.

- **SDLC model locked** – Chose Kanban with explicit WIP limits; board created on GitHub Project.

- **Mid-fi prototype added** – Four interactive frames (Upload, Processing, Results, Mobile) published in Figma on 4 Jul.

- **Risk log expanded** – Fifteen distinct risks (5 low, 5 medium, 5 high) with mitigations documented.

- **Timeline synced** – Internal schedule adjusted to match the official course dates (M1 report 4 Jul, video 7 Jul, M1.5 check-in week of 21 Jul, final code 6 Aug, presentation 8 Aug).

- **Data-flow & MVC blueprints** – Level 0 and Level 1 DFDs plus MVC lane diagram drafted and exported for inclusion in the planning report.

3. Additional artefacts bundled with this submission

- **Risk matrix heat-map** (PNG) – visual overlay of likelihood vs. impact for all fifteen risks.

- **API latency benchmark chart** (PNG) – average round-trip times for Vision (380 ms ± 40) and Gemini (520 ms ± 80) under campus Wi-Fi; supports our spinner timeout settings.

- **Figma component library screenshot** (PNG) – shows Button, Card, ColourChip, and Meter components with style tokens annotated.

- **Meeting-minutes archive** (PDF) – condensed log of stand-ups and retros from 28 Jun through 3 Jul.

These artefacts are stored in the repository under docs/appendix/ and referenced in the planning report where relevant.