Software Engineer

Case Study

Thank you for taking the time to participate in our case study. This exercise is designed to simulate a realistic workflow where you build a small product and collaborate with modern tools (including AI assistants).

1) Introduction & Objective

Objective:
Develop a demo mobile application that simulates the Hexa – AI Logo & Art Generator.

Your application should:

- Follow the provided Figma design (link: *Figma Case Study*).
- Implement:
    - Input Screen – where the user enters the required information.
    - Output Screen – where the generated (mock) result is displayed.
- Include a status chip / indicator that reflects the generation status.

Status Chip Behavior

- When the user starts the generation process:
    - Show a chip in a “processing” state.
- The chip should stay in “processing” for a random delay between 30–60 seconds.
- After the delay:
    - The chip changes to “done” (or “failed” in a mock failure scenario, if you choose to support that).

- In the “done” state:
  - Tapping the chip navigates to the Output Screen, displaying a mock image.

2. Collaboration with AI Tools (Required to Document)

We expect that you may use tools such as ChatGPT, GitHub Copilot, Claude, etc. This is allowed and encouraged. We care about how you use them, not whether you use them.

Rules & Expectations

1. Transparency (Required)
   In the root of your repository, add a file named `AI_USAGE.md` that briefly covers:
   - Which tools you used (e.g., ChatGPT, Copilot, others).
   - What you used them for (e.g., “generate initial Firestore rules”, “help debug X”, “suggest test cases”).
   - 2–3 example prompts/questions you found especially useful.
   - A short reflection (5–10 sentences total) on:
     - What AI did well for you in this task.
     - What you had to significantly adjust or fix.
     - Anything you deliberately did *without* AI.
2. Ownership of Code
   - You must understand every line you submit.
   - In the follow-up conversation, we may ask you to:

        - Explain parts of the code.
        - Modify or extend a feature live.

It's completely fine if an AI helped draft some of it, we just want to see that you can work with and reason about it.

3. Spirit of the Exercise

    - Please don't paste the entire assignment into an AI tool and ship the very first full-project answer as-is.
    - Treat AI as a pair programmer:
        - Ask for help on specific pieces.
        - Use it to explore trade-offs, improve code, or generate variations.
    - We trust you to follow this in good faith.

You will not be penalized for using AI; on the contrary you *will* be evaluated on how thoughtfully and effectively you collaborated with it.

3. Technologies & Requirements

Design

- Use the design available at Figma Case Study.
- Aim for good visual alignment (layouts, spacing, typography, colors) within reasonable time.

Frontend (Required)

- React Native Expo (Managed Workflow)
- Implement:
  - Input Screen
  - Output Screen with mock image
  - Status chip with the required state machine (**idle** → **processing** → **done** / **failed**)

Database (Required)

- Firestore

Backend (Required)

- Firebase Cloud Functions (Python) for managing update operations.

We intentionally do not fully specify the backend logic. We want to see how you:

- Implement the mock generation logic.
- Decide what should live in the backend vs. frontend.
- Design the interaction between frontend and backend.

You should structure your solution so that integrating a real AI generation step in the future would be straightforward (even though no real AI inference is required now).

4. Suggested Backend Flow (You May Adapt)

You are free to design the backend, but a good pattern (and a nice-to-have for us) is something like:

1. **Generation Request**
    - Frontend writes a "job" document to Firestore (e.g., `status: "processing"`, input parameters, timestamps).
2. **Cloud Function Trigger**
    - A Firebase Cloud Function (Python) reacts to the new document.
    - It:
        - Waits for a random delay between 30–60 seconds.
        - Updates the document to `status: "done"` or `status: "failed"` (mock).
        - Optionally writes a mock image URL or reference.
3. **Frontend Listener**
    - The frontend listens to changes on that document.
    - It updates the chip state (`processing`, `done`, `failed`) accordingly.
    - When `done`, tapping the chip navigates to the Output Screen showing the mock result.

Nice-to-Have Backend Enhancements

- Simulate a webhook-like architecture, as if an external AI service called back when done.

- Upload the mock output image to an S3-compatible storage (R2, MinIO, Wasabi, etc.) and store the URL in Firestore.

5. Duration & Deliverables

Total Time

- You have 3 days from the moment you receive this assignment.

What to Submit

1. **GitHub Repository**
    - Complete project code (frontend + backend).
    - Clear, concise README that includes:
        - How to run the app (Dev setup: Expo, env vars, Firestore/Functions).
        - Brief architecture overview (frontend state management, backend structure).
        - Any assumptions or trade-offs you made.
        - Known limitations / things you would improve with more time.
    - A meaningful commit history that reflects your development process.
        - Avoid a single “initial commit with everything”.
2. **Screen Recording**
    - A short video (Loom, local recording, etc.) showing:
        - Input → generation start.

        - Status chip transitions.
        - Navigation to Output Screen and viewing the mock result.
    - If you implemented failure scenarios, please show them too.

3. **AI Collaboration Summary (`AI_USAGE.md`)**
    - As described in Section 2.

6. Guidelines & Evaluation Criteria

We will evaluate your submission on:

1. **Functionality**
    - Does the flow work end-to-end?
    - Is the async delay handled cleanly?
    - Do the chip states and navigation behave as described?
2. **Code Quality**
    - Structure and modularity (separation of concerns).
    - Readability and naming.
    - Handling of async logic and error states.
    - Use of TypeScript (if you choose to use it) and types.
3. **Design Accuracy**
    - Reasonable adherence to the Figma design.
    - Consistent spacing, typography, and components.

4. **Architecture & Backend Decisions**
    - How you used Firestore and Cloud Functions.
    - How you modeled the generation job and status transitions.
    - How “future AI integration” would fit in.
5. **AI Collaboration**
    - Clarity and honesty in AI_USAGE.md.
    - Whether you seem to understand and own the code, even when AI assisted.
    - Whether AI helped you get better structure, tests, or edge cases — instead of just dumping a giant unmodified codebase.

7. Additional “Nice to Have” Client-Side Enhancements

These are not required, but they’ll be appreciated if you have time.

1. **CI/CD Integration**
    - GitHub Actions pipeline that runs:
        - TypeScript checks.
        - Linting (ESLint, etc.).
        - Basic tests (if present).
    - (Bonus) EAS Build + EAS Submit configuration.

2. **Automated UI Testing with Maestro (Bonus)**

   - Integrate Maestro UI tests and wire them into GitHub Actions.
   - Example scenarios:

     - Chip → Done
       When status changes to done, tapping should navigate to the Output Screen.

     - Chip → Failed
       Mock a failed status and ensure failure flow is handled.

     - Chip → Processing
       While processing, pressing “Create” shows a Toast/Alert.

     - General Flow
       Input → Generation → Chip states → Output.

8. Contact

If you have questions or need clarification, feel free to contact: **basak@feraset.co**

We look forward to seeing how you approach the problem and how you collaborate with modern tools.