Software Engineer

Case Study

Welcome! Thank you for devoting your time to this case study. We greatly appreciate it.

We will review all materials you share. If you move forward in the process, keep in mind that not every detail will be discussed during the technical interview, so be sure to include all key written or visual information in your submission.

Important: Please submit your case study solutions as a GitHub repository link with a comprehensive README file.

AI Image Generation Backend System Design

Overview

This case study assesses your backend engineering skills through designing and implementing a scalable and robust AI image generation backend system with a credit-based economy.

You will build a system that:

- Manages user credits with deduction and refund logic
- Provides APIs to create generation requests and query user credit balances
- Handles image generation with failure management
- Includes scheduled weekly reporting for monitoring usage and credit consumption

Required Tech Stack

- Is fully implemented in Python, using Firebase Functions and Firebase Database (Firestore)
- Is runnable in the Firebase Local Emulator environment (firebase emulators:start)
- Deployment via Firebase CLI is optional but considered a plus.

Project Description & Requirements

Users customize AI image generation requests by selecting:

- **AI Model:** Model A or Model B (user selected)
- **Style:** realistic, anime, oil painting, sketch, cyberpunk, watercolor
- **Color:** vibrant, monochrome, pastel, neon, vintage
- **Image Size:** 512x512, 1024x1024, 1024x1792

Credit consumption is based **exclusively on the image size**, independent of model, style, or color.

The styles and colors **must match the values listed above** and will be stored in the generation request records. How you design their storage structure within Firestore is left to your discretion.

AI Model Simulation

As part of this project, **you will implement mock AI generation services for Model A and Model B**. Each model should:

- Return a fixed placeholder image URL unique to the model
- Include a configurable failure rate (e.g., ~5%) to simulate generation errors and test refund logic

Credit Usage

Credit cost depends solely on the selected image size, as per the table below:

Image Size	Credit Cost
512x512	1
1024x1024	3
1024x1792	4

Style & Color Options

- Styles and colors do **not** impact credit consumption but must be stored as part of generation requests for record-keeping and filtering.
- The system should validate that selected styles and colors are valid options and reject requests with invalid values.

API Design (Firebase Functions)

Endpoint	**Input Payload**	**Response**

createGenerationRequest	{
	userId,
	model,
	style,
	color,
	size,
	prompt
	}
		{
		generationRequestId,
		deductedCredits,
		imageUrl
		}

getUserCredits	{userId}	{
		currentCredits,
		transactions: [
		{
		id,
		type: (deduction/refund),
		credits,
		generationRequestId,
		timestamp
		}
		]

scheduleWeeklyReport	Triggered by scheduler no input	{ reportStatus: “success” }

createGenerationRequest

- Calculates credit cost based on size
- Deducts credits atomically and logs transaction
- Creates generation request record with all options
- Triggers generation simulation
- On success, updates request with placeholder image URL
- On failure, refunds credits and updates request status

getUserCredits

- Returns user's current credit balance and credit transaction history
- Transaction entries include type (deduction or refund), credits, timestamp, and generationRequestId

scheduleWeeklyReport

- Scheduled Firebase function runs weekly to aggregate usage data:
  - Generation counts and success/failure rates by model, style, size
  - Credit consumption summaries
  - Anomaly detection for unusual usage spikes

Testing

Your submission should include a suite of automated tests covering key functionality, including but not limited to:

- Credit deduction and refund logic correctness
- Validation of generation request inputs (model, style, color, size)
- Successful and failed AI generation simulations
- Scheduled weekly reporting execution

Clear instructions on how to run the tests should be included in your README.

Initial Data Setup and Emulator Import

To ensure consistent starting data for evaluation and development, you should provide a **Firebase Local Emulator-compatible export** containing essential collections and documents, such as:

- **users** — predefined test users with credit balances
- **colors** — valid color options (e.g., vibrant, neon, etc.)
- **styles** — valid style options (e.g., anime, realistic, etc.)
- **sizes** — size options with their credit costs
  Any other configuration collections you deem necessary.

Requirements for the Exported Data

- The export format must be compatible with Firebase Local Emulator import commands.
- You will submit this export folder or archive as part of your project repository.
- Evaluator, should be able to import data using:

Shell

```
●  firebase emulators:start --import=path_to_export_folder
```

- This ensures the local emulator starts with your predefined data, enabling seamless testing of the backend.

Evaluation

Criteria	Description	Weight
Functionality	Accurate credit management, API compliance	30%
Code Quality	Readability, maintainability, adherence to best practices	25%
API & DB Design	Clear endpoints with well-defined inputs, outputs, and error cases	20%
Documentation	Completeness and clarity of README	10%
Error Handling	Robustness in refund, and failure handling, coverage of different fail scenario	15%

Additional Notes

- Designing a flexible, extensible database schema that can evolve with business needs is essential.
- Functionality alone is not enough; clean, maintainable, and robust architecture is highly valued.

We genuinely appreciate the time and creativity you're putting into this case study. Your perspective matters, and we can’t wait to see what you come up with.

For any questions please reach out to basak@feraset.co