# Gemini Backend Clone Assignment - Kuvaka Tech

Role: **Backend Developer at Kuvaka Tech**

## Objective:

Develop a Gemini-style backend system that enables user-specific chatrooms, OTP-based login, Gemini API-powered AI conversations, and subscription handling via Stripe. This project will assess your skills in backend architecture, authentication, third-party integration, and clean code practices.

## Requirements:

### 1. User Authentication:

- OTP-based login system (mobile number only).
- OTP should be triggered via API (no third-party SMS integration; just return/send OTP in response).
- Full authentication system using JWT tokens.

### 2. Chatroom Management:

- Users should be able to create and manage **multiple chatrooms**.
- Each chatroom supports conversations where user messages are passed to the **Google Gemini API**, and responses are returned.
- Use a **message queue** (RabbitMQ, Redis Queue, Celery, BullMQ, etc.) to handle Gemini API calls asynchronously.

### 3. Google Gemini API Integration:

- Integrate with Google Gemini API.

## 4. Subscription & Payments:

- Use **Stripe (sandbox mode)** for handling subscription payments.
- Define two tiers:
    - **Basic (Free)**: Limited daily usage (e.g., 5 prompts/day).
    - **Pro (Paid)**: Higher or unlimited usage.
- Include the following:
    - Subscription start API
    - Stripe webhook to handle events (success, failure, etc.)
    - Subscription status API for user

5. API Requirements:

| Endpoint | Method | Auth Required | Description |
|---|---|---|---|
| `/auth/signup` | `POST` | ❌ | Registers a new user with mobile number and optional info. |
| `/auth/send-otp` | `POST` | ❌ | Sends an OTP to the user's mobile number (mocked, returned in response). |
| `/auth/verify-otp` | `POST` | ❌ | Verifies the OTP and returns a JWT token for the session. |
| `/auth/forgot-password` | `POST` | ❌ | Sends OTP for password reset. |
| `/auth/change-password` | `POST` | ✅ | Allows the user to change password while logged in. |
| `/user/me` | `GET` | ✅ | Returns details about the currently authenticated user. |

| | | | |
|---|---|---|---|
| `/chatroom` | `POST` | ✅ | Creates a new chatroom for the authenticated user. |
| `/chatroom` | `GET` | ✅ | Lists all chatrooms for the user (**use caching here – justified below**). |
| `/chatroom/:id` | `GET` | ✅ | Retrieves detailed information about a specific chatroom. |
| `/chatroom/:id/message` | `POST` | ✅ | Sends a message and receives a Gemini response (via queue/async call). |
| `/subscribe/pro` | `POST` | ✅ | Initiates a Pro subscription via Stripe Checkout. |
| `/webhook/stripe` | `POST` | ❌ (Stripe only) | Handles Stripe webhook events (e.g., payment success/failure). |
| `/subscription/status` | `GET` | ✅ | Checks the user's current subscription tier (Basic or Pro). |

## Registration Notes:

- **JWT Authentication** should be included in the `Authorization` header for all protected routes (`Bearer <token>`).
- **Use caching/query caching** at least at one place where it feels necessary (must be **justified**).
- Endpoints must return consistent **JSON responses** with proper HTTP status codes.
- Implement middleware for token validation and error handling.
- Rate-limiting must be implemented for Basic tier users (e.g., daily message count check).

## Caching Requirement – Justification:

Use **query caching** on: `GET /chatroom`

- This endpoint is frequently accessed when loading the dashboard.
- Chatrooms don't change often compared to messages.
- Caching the chatroom list (per user) with a short TTL (e.g., 5–10 minutes) significantly improves **performance** and **reduces database load**.
- Use Redis, Node-cache, or any appropriate caching library.

## Technical Constraints:

- **Language:** Python (FastAPI preferred) or Node.js (Express/NestJS)
- **Database:** PostgreSQL
- **Queue:** RabbitMQ / Redis / Celery / BullMQ
- **Authentication:** JWT with OTP verification
- **Payments:** Stripe (sandbox environment)
- **External API:** Google Gemini
- **Deployment:** Any public cloud platform (Render, Railway, EC2, Fly.io, etc.)

## Deliverables:

### 1. Source Code

- Organized and modular project structure.
- Clear comments and appropriate naming conventions.

### 2. Postman Collection

- Folder-structured, well-labeled requests covering all APIs.

- Use JWT tokens where needed for authorization.

### 3. Deployment

- Deploy your app on any cloud service.
- Share a **public IP or URL** that works with Postman.

### 4. GitHub Repository

- Public GitHub repo with:
    - Complete source code
    - Readable commit history
    - Branches if used for feature isolation

### 5. Documentation

- A comprehensive **README.md** including:
    - How to set up and run the project
    - Architecture overview
    - Queue system explanation
    - Gemini API integration overview
    - Assumptions/design decisions
    - How to test via Postman
    - Access/deployment instructions

## Evaluation Criteria:

- Functional completion of all requirements

- Clean, readable, and maintainable code
- Proper use of async patterns or queues
- Error handling and edge case coverage
- Properly working APIs (tested via Postman)
- Deployment availability
- Documentation quality

## Submission Guidelines:

- Email your submission and your updated resume to **hr@kuvaka.io** with the subject: **"Gemini Backend Clone Assignment Submission - Kuvaka Tech [Your Full Name]"**

**Include the following in your email**:

- GitHub Repository Link
- Postman Collection File
- Deployment Link (Public IP or URL)
- Any setup instructions or environment notes

## Deadline:

- Submit the assignment **within 48 hrs - 72 hrs (2 - 3 days)** of receiving this prompt.