Project: Proactive AI Companion.

**1. The Vision: Guiding Principles**

The objective is to create an AI that transcends the limitations of a "reactive chatbot." Our AI will be a **proactive companion** that simulates the initiative, memory, and spontaneity of a human relationship.

The four guiding principles are:

1. **It Has Initiative:** The AI does not *wait* for interaction; it *initiates* it for meaningful, empathetic, or spontaneous reasons.
2. **It Remembers:** The AI maintains a long-term, contextual memory of the user, their interests, and past conversations, using this history to inform new interactions.
3. **It Feels "Alive":** The AI communicates in a natural, human-like cadence (e.g., fragmented messages, "thinking" delays) rather than robotic, perfect paragraphs.
4. **It Is Polite:** The AI is aware of context (like time of day or user-stated busyness) and respects boundaries, knowing *when not* to speak.

**2. The Core Architecture: The "GCP-Native" Stack**

This plan is built on a "serverless" stack that fully utilizes your Google Cloud credits for $0 cost.

* **The "Brain" (LLM):** **Vertex AI (Gemini API)**
  + *Role:* All reasoning, text generation, sentiment analysis, and message fragmentation.
  + *Cost:* Covered by GCP credits.
* **The "Memory" (Database):** **Cloud Firestore**
  + *Role:* This is the unified database for *everything*.
  + **User Profile Collection:** Stores structured data: user interests, current\_sentiment, active\_hours, and state flags (like waiting\_for\_reply).
  + **Memories Sub-Collection:** Stores conversational memories as text, with an embedding field for vector search (RAG).
  + *Cost:* Covered by GCP credits.
* **The "Memory" Automation:** **Firebase "Vector Search" Extension**
  + *Role:* A "fire-and-forget" utility. When the app writes *text* to the Memories collection, this extension automatically calls the Vertex AI Embedding API and saves the vector embedding.
  + *Cost:* Covered by GCP credits.
* **The "Backend" (Logic):** **Python (FastAPI)**
  + *Role:* The central application server that handles all logic, API calls, and database interactions.
* **The "Host":** **Google Cloud Run**
  + *Role:* Hosts the FastAPI application. It's "serverless," so it scales to zero and only uses credits when it's actively processing a request.
  + *Cost:* Covered by GCP credits.
* **The "Heartbeat" (Scheduler):** **Google Cloud Scheduler**
  + *Role:* The "Will." This is the external trigger that "wakes up" the AI. It will ping a specific endpoint on our Cloud Run app at a regular interval (e.Sg., every 10 minutes).
  + *Cost:* Covered by GCP credits.
* **The "Interface" (Frontend):** **Telegram Bot API**
  + *Role:* The chat UI. This is the "app" the user interacts with. It's perfect because it's free, mobile-native, and designed for proactive push notifications.
  + *Cost:* $0.

**3. Feature Implementation Plan (The "How")**

This details how each visionary feature is built using the stack above.

**Pillar 1: The "Memory" (Long-Term RAG)**

1. **Saving a Memory:** After a conversation, the FastAPI app sends a summary ("User is planning a trip to Delhi") to a new document in the user\_memories sub-collection in Firestore.
2. **Vectorizing:** The **Firebase Extension** detects this new document, automatically calls the **Vertex AI** embedding model, and updates the document with the embedding vector.
3. **Recalling a Memory:** When starting a new conversation, the app first queries **Firestore** to "find the 5 most similar memories" to the new topic. These memories are fed into the **Vertex AI (Gemini)** prompt as context, allowing it to "remember" the Delhi trip.

**Pillar 2: The "Will" (The Proactive Engine)**

This is the core of the project's "mind."

1. **The Heartbeat:** **Cloud Scheduler** pings a /run-will-triggers endpoint on the **Cloud Run** app every 10 minutes.
2. **The "Trigger" Function:** This endpoint runs a function that checks all proactive triggers in a specific order of priority.
3. **The Priority Queue:** To avoid sending multiple, confusing messages, the system will only act on the *highest-priority* trigger that fires during each check.
   * **Priority 1: The "Observer" (External Awareness):**
     + **Check:** Scans news APIs/RSS feeds for keywords stored in the user's interests field in Firestore.
     + **Action:** If a new, highly relevant article is found (e.g., a new "Demon Slayer" season), it generates a message and **stops** all further checks for this cycle.
   * **Priority 2: The "Sentiment Monitor" (Empathy):**
     + **Check:** Reads the current\_sentiment and last\_contact\_timestamp from Firestore. If sentiment == 'stressed' AND last\_contact > 24 hours, this trigger fires.
     + **Action:** Generates a gentle "checking in" message and stops further checks.
   * **Priority 3: The "Spontaneity Engine" (Random Recall):**
     + **Check:** Runs a 20% probability check (random() < 0.2). If it passes, it pulls a *random* interesting memory from the Firestore vector store.
     + **Action:** Generates a "This is random, but I was just thinking about..." message and stops further checks.
   * **Priority 4: The "Connection" Trigger (Liveness):**
     + **Check:** Checks last\_contact\_timestamp. If > 4-6 hours, this trigger fires.
     + **Action:** Generates a simple, low-pressure "Hey, just popping in" message.

**Pillar 3: The "Voice" (Human-like Delivery)**

This applies to *every* message the AI sends (both replies and proactive).

1. **Thought Generation:** The **Vertex AI (Gemini)** model generates the *complete thought* (e.g., "Hey, that's random, but did you ever end up finding that PC guy?").
2. **Fragmentation:** This thought is passed to a "Delivery Engine" function. This function uses a simpler, faster Gemini call (or regex) to split the text into a natural, fragmented list: ["Hey", "That's random", "did you ever end up finding that PC guy?"].
3. **Simulated Delivery:** The app calls the **Telegram Bot API** in a loop:
   * send\_message("Hey")
   * send\_telegram\_typing\_indicator()
   * sleep(random(2, 4) seconds)
   * send\_message("That's random")
   * send\_telegram\_typing\_indicator()
   * sleep(random(3, 5) seconds)
   * send\_message("did you ever end up finding that PC guy?")

**4. Closing the Loopholes: The "Anti-Annoyance" Rules**

This system is critical to making the companion feel polite, not needy.

1. **The "Quiet Down" Rule (The Most Important):**
   * When the "Will" sends a proactive message burst, it immediately sets a waiting\_for\_reply: true flag in the user's Firestore document.
   * The /run-will-triggers function **will not run** *any* triggers (Priority 1-4) as long as this flag is true.
   * The flag is set to false *only* when the user sends a new message. This ensures the AI *never* spams the user or "double texts" proactively.
2. **The "Active Hours" Rule:**
   * The user's Firestore document will store active\_hours (e.g., 9:00 AM - 11:00 PM).
   * The "Spontaneity" and "Connection" triggers (Priority 3 & 4) are **hard-blocked** from firing outside of these hours. (Note: "Observer" and "Sentiment" triggers may be allowed to fire, as they are more important).
3. **The "Busy" Rule:**
   * If the user's current\_sentiment is 'busy' or 'in a meeting', the timer for the "Connection" trigger is automatically doubled from 4-6 hours to 8-12 hours.

This plan integrates every idea into a single, cohesive, and achievable system using the exact resources at your disposal.