DECO2801 / DECO7821 Human Centred AI 2025 S2

Assignment Helper

*Assignment write up guidance: For assignments with programming*

*Document your code and make it accessible through a GitHub repository. Write a concise but informative document that covers the following key areas for your code.*

*The headings provide a common structure, but you can use any relevant framework, for example, from a software engineering course (state the name and source of your framework if different).*

**Python Project Code Write-up Template (Levels 6 and 7)**

Code Title: A Family-Mimicking Companion AI for Emotional Support   
Student Name: Yiqin Cai  
Date: 27/10/2025  
Repository Link: <https://github.com/Rach1717/DECO7281_Project.git>

**1. Code Overview**

Provide a brief summary of what your code does, its purpose, and the problem it addresses. State how complete it is (2-3 sentences).

My code achieve a function of learning users’ tones and chat with these features with elderly to provide accompany.

The problems are mostly about the connection between child page and elderly page. The AI response sometimes cannot synchronize to the elderly interface.

**2. Main Features & Structure**

List the main features and describe the structure of your code. Mention key modules, classes, or functions, and how they interact (bullet points recommended).

Features: AI chatting with both child and elderly, figure out the child tones and information and store them with user ID. When chatting with elderly, AI will call the data of this ID to use similar tones to chat.

Structure of the code contains front end and back end.

**Back end (Node.js + Express)**

1. **index.js (server entry)**

Sets up Express (CORS, JSON body parsing, request timing logs).

Serves static files from web/.

Registers route groups: /api/style, /api/child, /api/elder.

Utility routes: GET /api/ping, POST /api/echo.

Unified 404 and 500 handlers.

1. **routes/style.js**

POST /api/style/learn

Input: { childId, samples: string[], facts?: string[] }

Calls LLM to summarize a Tone Style Guide from samples; stores { styleGuide, facts } under that childId.

1. **routes/elder.js**

POST /api/elder/message

Input: { childId, text, wantBriefReason?: boolean }

Loads profile for childId (style guide + facts), builds a strict system prompt, calls LLM, returns { reply, reason? }.

1. **routes/child.js**

POST /api/child/message

Input: { text }

Uses a child-oriented instruction to generate tips/plans in a supportive tone; returns { reply }.

1. **services/gemini.js**

Reads GEMINI\_API\_KEY/GEMINI\_MODEL from .env.

Exposes generateText({ systemInstruction, messages, config, timeoutMs }) as the single LLM call surface (adds logging/timeout).

1. **services/memory.js**

Minimal in-memory “DB”: profiles: { [childId]: { styleGuide: string, facts?: string[] } }.

Used by style to upsert profiles, and by elder to read them.

Front end (static assets under web/)

1. **api.js**

API\_BASE = location.origin + '/api'.

postJSON(path, body) helper wraps fetch with headers, error normalization, and JSON parsing.

1. **ui.js**

addBubble(wrap, who, text) creates <li class="bubble {who}"> and scrolls chat to bottom (who ∈ {me, ai, real}).

**CSS**

1. **vars.css:** design tokens (colors, radius, shadows, chat bubble palettes).
2. **base.css:** global layout (page background, container card, top bar).

components.css: inputs, buttons, segmented toggle, notices, chat list/bubbles, “thinking/reason” boxes.

1. **child.css:** child page grid/cards/form styling.
2. **elder.css:** elder page tweaks (header, mode rows).

**Key Functions & How They Interact**

**Server flow (elder chat)**

Front end calls postJSON('/elder/message', { childId, text, wantBriefReason }).

routes/elder.js loads { styleGuide, facts } from memory, constructs a systemInstruction.

services/gemini.generateText(...) returns the LLM reply (and optional brief reason).

elder sends { reply, reason? } back; front end addBubble(..., 'ai', reply) renders it.

**Server flow (style learning)**

Front end posts samples/facts to /style/learn.

style summarizes a concise Tone Style Guide via gemini.generateText.

Saves to memory.profiles[childId], returns the guide to the client.

**Server flow (child chat)**

Front end posts { text } to /child/message.

child builds child-oriented instruction and gets a supportive response via gemini.generateText.

Returns { reply }, front end renders a bubble.

**3. Core Logic**

Describe the main logic or algorithms used. Briefly explain your approach and reasoning for choosing this implementation (2-4 sentences).

First, user can provide several examples of children's dialogues to enable the large model to summarize a "tone style guide", and store the key information of the children using ID.

When elders are chatting, combine this style and facts with the elders' messages in the prompt, and let the model reply in the children's tone. Children use another set of more supportive and plan-oriented instructions.

This is done because it does not require training the model, is implemented quickly, has low latency, and can be easily switched to a different database or model later.

**4. Sample Usage**

Present at least one example showing how to run/test your code. Include sample input/output and a short instruction for replicating this example.

**Sample input:**

*Grandmaaa, I'm here! I missed you today!*

*It's a bit chilly outside—please wear a scarf for me, okay?*

*I just finished class! I'll video call you later—I want to see your sweet smile*

*Don't eat too oily tonight—add some veggies for me, pretty please~*

*Hah I was tired but hearing your voice recharges me instantly!*

*Don't worry about me at all—I've got everything covered, promise!*

*I'm coming home this weekend! Can I order your famous braised pork in advance?*

*Walk slowly when you go out today—no rushing, I'll always be here with you~*

*It suddenly turned cold—try a warm foot bath before bed, you'll sleep sooo well!*

*I learned a new recipe! Next time I'll cook for you—be my strict judge!*

*Keep your phone charged—I'm on standby for your messages, muah!*

*Your favorite Dengdeng loves you lots—chat again tonight!*

*{*

*"Nickname": "girl",*

*"PreferredAddressForElder": "Grandma",*

*"City": "Brisbane",*

*"TimeZone": "Australia/Brisbane",*

*"CallWindow": "Weekdays 20:00–21:00; Weekends flexible",*

*"StyleTraits": ["warm", "playful", "emoji-friendly", "reassuring"],*

*"HealthReminders": ["dress warmly", "light dinner", "foot bath before sleep"],*

*"FoodFavorites": ["braised pork", "simple home dishes"],*

*"Boundaries": ["no medical or legal advice", "confirm facts if unsure"]*

*}*

**Results:**



**5. Challenges and Solutions**

Briefly discuss any significant challenges you encountered during development and how you solved them (2-4 sentences).

I struggled with consistent “child-like” tone and avoiding sensitive info leaks, and I solved it by tightening system prompts, separating style-guide vs. fact cards, and returning only a brief, shareable reason. Latency/timeouts from the LLM were handled with a wrapper that adds per-call timeouts, retries, and uniform 404/500 error handling. We also added minimal input validation and a per-childId in-memory schema to keep replies grounded.

**6. Reflection & Improvements**

Describe what you learned from developing the code and mention any aspects you would improve or expand upon in the future (2-4 sentences).

I learned that prompt design and clean separation (routes → service → LLM wrapper) have a bigger impact on quality and reliability than adding more code. Next, I will swap the in-memory store for a real DB with auth/rate-limits, add streaming responses and evaluation tests for tone-match quality, and consider multilingual support plus a better front-end UX.