National College of Ireland

Configuration Manual

MSc Research Project
FinTech

Biswadeep Sarkar
Student ID: 23315652

School of Computing
National College of Ireland

Supervisor: Prof. Brian Byrne

National College of Ireland
Project Submission Sheet
School of Computing

National College of Ireland

Student Name:	Biswadeep Sarkar
Student ID:	23315652
Programme:	FinTech
Year:	2024
Module:	MSc Research Project
Supervisor:	Prof. Brian Byrne
Submission Due Date:	11/08/2025
Project Title:	Configuration Manual
Word Count:	889
Page Count:	9

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	
Date:	8th August 2025

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Biswadeep Sarkar
23315652

1 Introduction

Please find the system specification used to perform this research project. I have highlighted the software, hardware specifications, along with the steps followed to perform the research Context Aware Token Optimization for Personalized Financial Advisory with LLMs.

2 System Requirements

Find the key system software and hardware requirements to run this project.

2.1 Software Requirements

- **IDE:** We used VS Code, an open-source, highly configurable code editor built by Microsoft.
- **OS:** We used Windows 11 as our system's operating system.
- **Code:** We used Python, HTML, CSS, and JavaScript in our code.
- **Database:** We used a JSON file as a dummy database.
- **Synthetic Data Generator:** We used ChatGPT to generate historical conversational synthetic data.
- **LLM:** We used Gemini 1.5 Flash by Google as our core LLM engine.
- **Token Calculator:** We used the Gemini Token counter by DBSwan.
- **Browser:** We used Google Chrome to run our web application.

2.2 Hardware Requirements

- **Processor:** AMD Ryzen 5 5500U with Radeon Graphics 2.10 GHz
- **Installed RAM:** 20.0 GB (17.8 GB usable)
- **System Type:** 64-bit operating system, x64-based processor
- **Pen and touch:** No pen or touch input is available for this display

3 Development Environment

3.1 IDE: VS Code

- **Version:** 1.102.3
- **Electron:** 35.6.0
- **Chromium:** 134.0.6998.205
- **Node.js:** 22.15.1
- **OS:** Windows_NT x64 10.0.22631

3.2 Libraries

- **os:** Provides operating system functions.
- **json:** Handles JSON data.
- **datetime:** Works with dates and times.
- **google-generativeai:** Interacts with Google's Generative AI models e.g. Gemini 1.5 Flash.
- **google-api-core:** Manages Google API exceptions.
- **Flask:** The web application framework.
- **Flask-Cors:** Manages Cross-Origin Resource Sharing.

4 Steps to Run the Project

Following these steps, this project can be run.

- **Download VS Code:** Get VS Code IDE from the official website.
- **Install Python:** Download the latest version and ensure "Add Python to PATH" is checked.
- **Install Extension:** In VS Code, install the official Python extension from Microsoft.
- **Create Environment:** Run the 'Python: Create Environment' command in the Command Palette.
- **Open Project Folder:** Open the project folder containing the code artefacts.
- **Open Command Line Interface:** In the opened project folder, right-click on an empty space while holding the Shift key and select **Open PowerShell window here** or **Open command window here**.

- **Clone the Repository:** Use the git clone command followed by the repository's URL to download a copy of the project. (URL: [1])
- **Navigate to Project Directory:** Use the cd (change directory) command to enter the newly cloned project folder.

```
cd financial_advisor
```

- **Fetch Latest Changes:** If you already have the repository cloned, use git fetch to download the latest changes without merging them.

```
git fetch origin
```

- **Merge the Changes:** Use git merge to integrate the fetched changes into your current branch.

```
git merge origin/main
```

- **Pull All Changes (Alternative):** As a shortcut for fetching and merging, you can use git pull to update your local branch with the remote's latest changes.

```
git pull origin main
```

- **Install Libraries:** Use the terminal command 'pip install google-generativeai google-api-core Flask Flask-Cors' to download the required libraries.
- **Generate Your API Key:** Use Google AI studio to generate your API Key (Link: [2])
- **Replace Your API Key:** Paste your API Key in the financial advisor logic and memory file.
- **Run The Server:** Run the python memory.py to first optimize the memory and then python app.py to start the Flask server on localhost port 5000 on your terminal. (See Image 1)

5 Development

5.1 Chat Data

We have used a synthetic dataset to perform this research. By a synthetic dataset, we mean we have created a fake conversation between a customer and an insurance advisor that seems exactly like a real one. We have used ChatGPT, a publicly available LLM Chat solution by OpenAI, to create this synthetic dataset. We have used this Prompt (see image: 2) to create the data. Due to the limitation of output tokens in the GPT model, we had to perform multiple iterations to generate a significantly large enough chronological chat dataset (See Image: 3).

[1] `https://github.com/biswadeepsarkar/financial_advisor`
[2] `https://aistudio.google.com/`

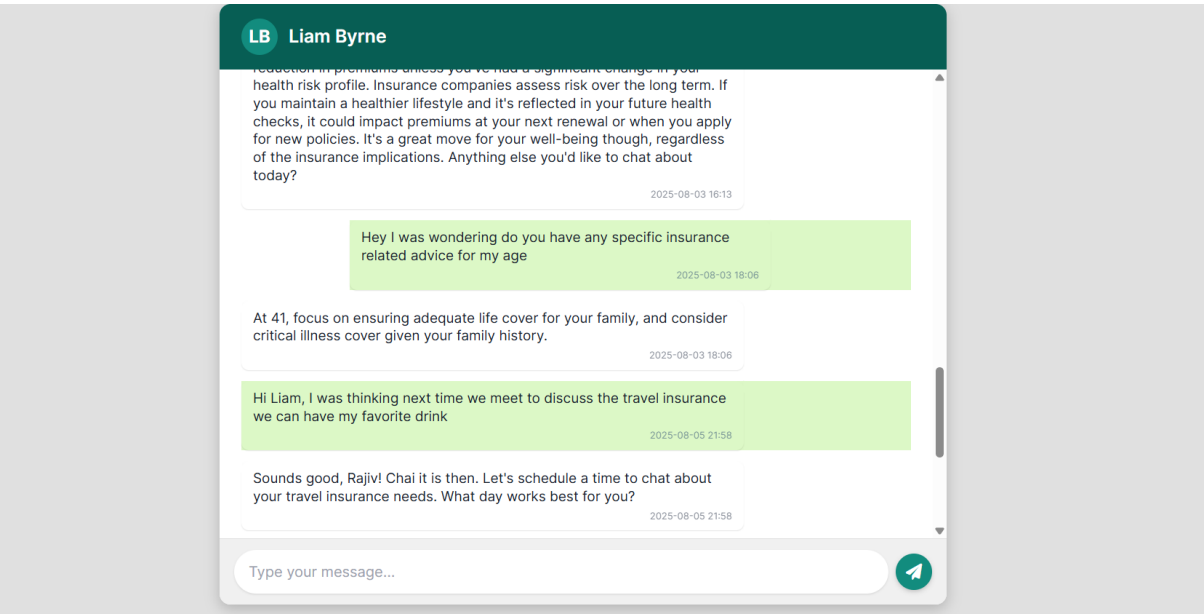

Figure 1: Chat Window

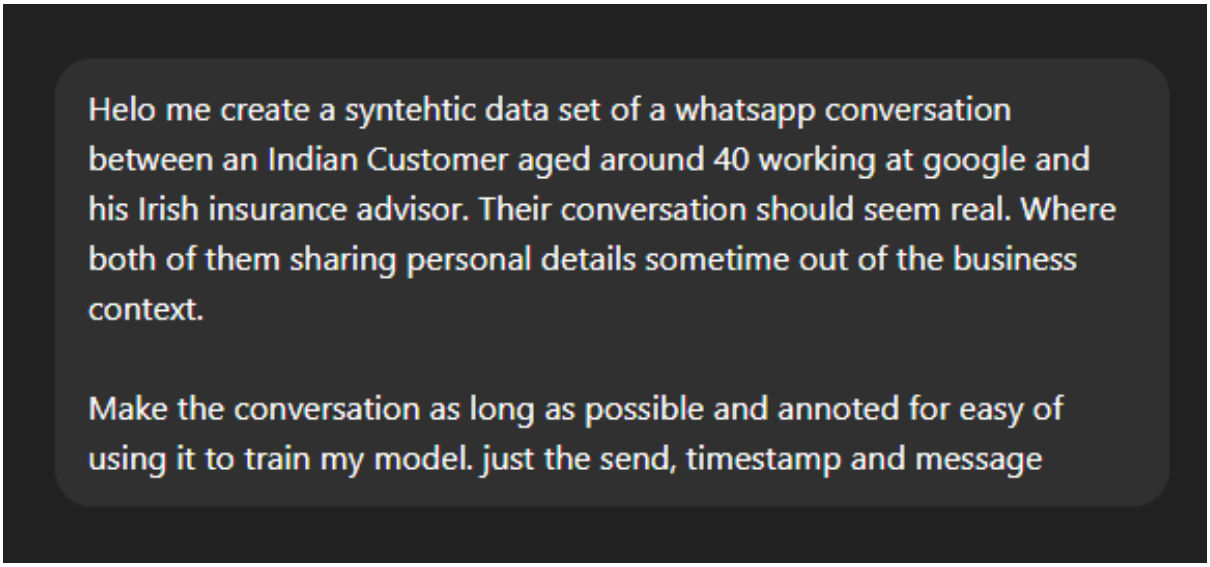

Figure 2: Prompt

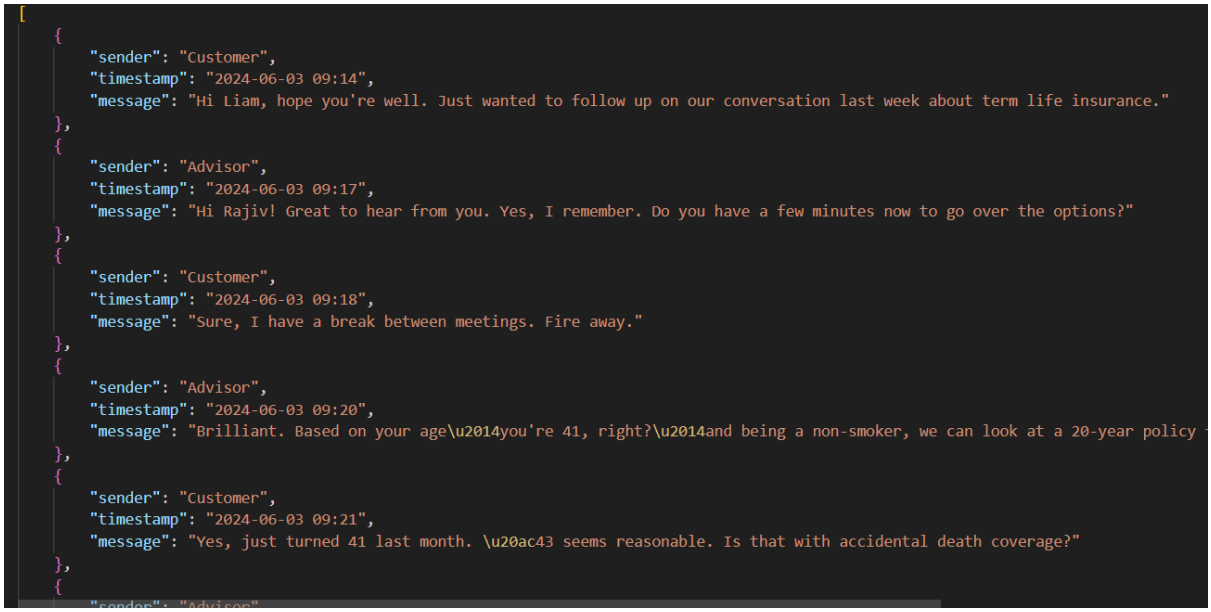

Figure 3: Chat Data

5.2 Code

For the Development of the code, we have used Python as the base since it is widely adopted and multiple documentations are available on the internet from the community, as we integrate several APIs and libraries. We have used the Flask server engine to run our app and developed the user interface index.html using HTML, CSS, and JavaScript.

We first created the long-term and short-term JSON memory files and then created the memory optimization engine (See Image: 5). This memory optimization engine works in a repetitive manner to process the chat history of 180 days till the final optimized version (See Image: 8). After that, I created the chat engine with financial advisory logic, which handles customer chat. In this engine, I am using 20 recent chats and the short-term memory as the context to make the conversation continuous and relevant. This also ensures the user gets personalized financial advice (See Image: 1). After that, I finally built the Flask-based server and the user interface mimicking WhatsApp (See Image: 7).

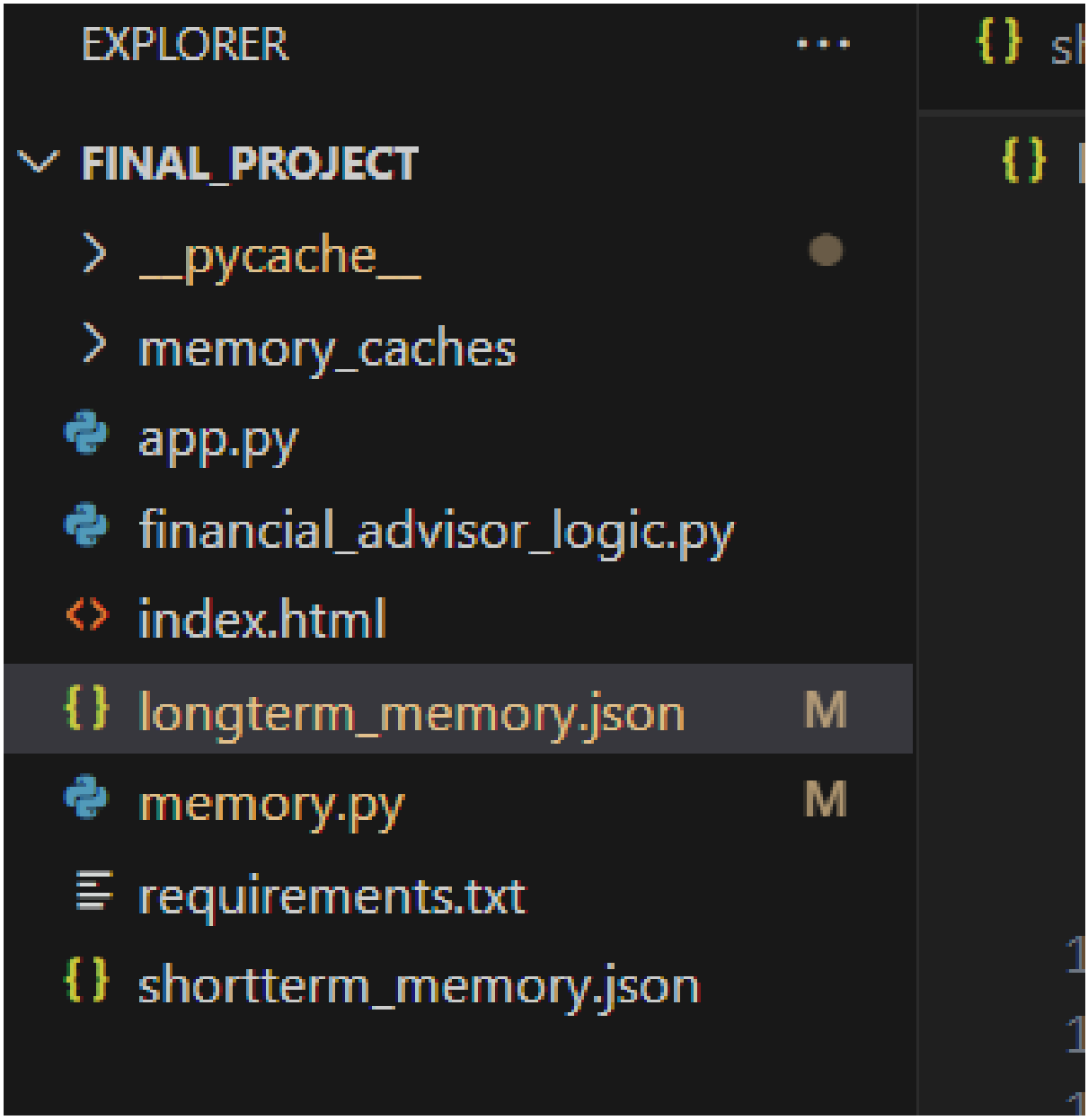

Figure 4: Project Code Structre

memory.py > ...

```python
        "response_mime_type": "application/json"
    }
)

LONGTERM_PATH = "longterm_memory.json"
SHORTTERM_PATH = "shortterm_memory.json"
CACHE_DIR = "memory_caches"
WINDOW_DAYS = 180

# --- MY HELPER Functions ---

def load_messages(path):
    """Loads and sorts messages from a JSON file."""
    try:
        with open(path, "r", encoding="utf-8") as f:
            msgs = json.load(f)
        for m in msgs:
            m["dt"] = datetime.strptime(m["timestamp"], "%Y-%m-%d %H:%M")
        return sorted(msgs, key=lambda m: m["dt"])
    except FileNotFoundError:
        print(f"Error: The file {path} was not found.")
        return []
    except (json.JSONDecodeError, KeyError, UnicodeDecodeError) as e:
        print(f"Error parsing messages from {path}: {e}")
        return []

def call_gemini_api(messages, system_prompt, model=GEMINI_MODEL, max_tokens=512, temperature=0.0):
    """A reusable function to call the Gemini API with error handling."""
```

Figure 5: Memory.py

app.py > ...

```python

# Memory handling
if not os.path.exists(SHORTTERM_PATH):
    with open(SHORTTERM_PATH, 'w', encoding='utf-8') as f:
        json.dump(mock_customer_profile_data, f, indent=4)
    print(f"Created initial {SHORTTERM_PATH} with mock data.")

# index.html file route
@app.route('/')
def serve_index():
    return send_from_directory(os.getcwd(), 'index.html')

@app.route('/chat', methods=['POST'])
def chat():
    data = request.json
    user_message = data.get('user_message')
    current_session_history = data.get('current_session_history', [])

    if not user_message:
        return jsonify({"error": "No message provided"}), 400

    try:
        # Past chat loading (last 20 messages before today)
        past_chat_history = load_chat_history(LONGTERM_PATH)

        # Customer profile loading (Short-term memory)
        customer_profile = load_customer_profile(SHORTTERM_PATH)

```

Figure 6: App.py

```
<html lang="en">
<body class="flex items-center justify-center min-h-screen">
    <script>
        async function handleSendMessage() {


            const loadingWrapper = document.createElement('div');
            loadingWrapper.classList.add('message-wrapper', 'message-advisor');
            const loadingBubble = document.createElement('div');
            loadingBubble.classList.add('loading-indicator');
            loadingBubble.textContent = 'Liam Byrne is typing...';
            loadingWrapper.appendChild(loadingBubble);
            chatMessagesDiv.appendChild(loadingWrapper);
            chatMessagesDiv.scrollTop = chatMessagesDiv.scrollHeight;
            console.log("Loading indicator displayed.");

            try {
                console.log("Attempting to fetch from backend...");

                const response = await fetch(`${BACKEND_URL}/chat`, {
                    method: 'POST',
                    headers: {
                        'Content-Type': 'application/json',
                    },
                    body: JSON.stringify({
                        user_message: userMessage,
                        current_session_history: currentSessionHistory
                    }),
```

Figure 7: Index.html

```
"goals": {
  "mid-term": [
    "Child's university education"
  ],
  "long_term": [
    "Retirement in Goa"
  ]
},
"facts": {
  "life_insurance": true,
  "travel_insurance": true,
  "claims": [
    {
      "incident": "Son's sprained ankle",
      "date": "2024-08",
      "location": "Valencia, Spain",
      "cost": "\u20ac230"
    }
  ],
  "income_protection_policy": true,
  "health_insurance": true,
  "review_coverage": true,
  "family_event": {
    "cousin_passing": "2025"
  }
},
"dates": {
  "birth_date": "1983-05",
```


Figure 8: Optimized Memory