Personal Finance Chatbot: Intelligent Guidance for Savings, Taxes, and Investments

Project Description:

The "Personal Finance Chatbot" project aims to develop an intelligent conversational AI system that leverages IBM's generative AI models and Watson services to provide personalized financial guidance. This initiative will harness advanced natural language processing (NLP) capabilities within the IBM ecosystem to answer user questions about savings, taxes, and investments, offer generated budget summaries, and suggest actionable spending insights. A core innovation will be the chatbot's ability to adapt its tone and complexity through IBM's AI to suit different user demographics, specifically distinguishing between students and professionals. By providing accessible, tailored financial information and analysis powered by IBM's cutting-edge AI, this project seeks to empower individuals to make more informed financial decisions and improve their overall financial literacy and well-being.

Scenarios:

Scenario 1: Consider a user who wants to know "How can I save while repaying student loans?" In such a scenario, the chatbot clarifies the issue, deciphers the query, and provides straightforward, pertinent, and customized answers. This makes it simpler for anybody, especially those who are unsure of how to phrase their worries, to ask for assistance with routine financial issues.

Scenario 2: The chatbot can provide a summary of how they are spending their money when a user provides their income and list of costs. Essentials like their monthly savings, the primary use of their money, and what remains after costs are highlighted. Users may rapidly grasp their financial situation and take charge of their budget with the help of this type of feedback.

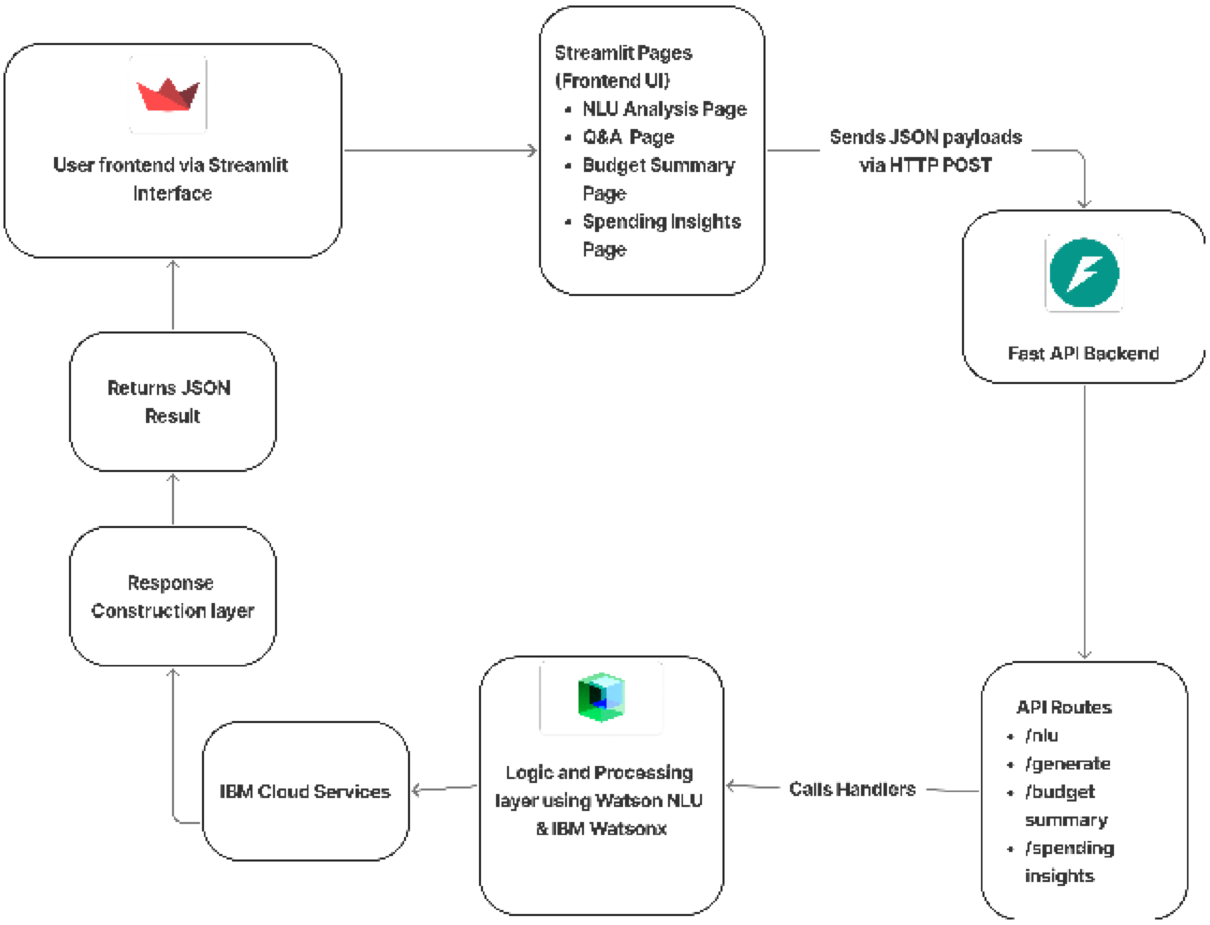
Scenario 3: Let's say a customer want to accumulate an emergency fund, purchase a laptop, or save money for a trip. In order to assist them, the chatbot displays the amount they must consistently save as well as whether their present spending patterns permit it. It serves as a useful manual for converting long-term objectives into precise, doable tasks.

Scenario 4: The chatbot helps users pick their top spending categories, such as food, shopping, or transportation, if they want to discover where their money is going each month. Users can more easily adapt and save more efficiently because it raises awareness of habits that are sometimes overlooked.

Scenario 5: Following a budget or spending review, consumers may wonder what to do next. In these situations, the chatbot makes practical recommendations, such as how to reduce expenses, save more effectively, or reconsider purchases. These brief suggestions enable people to move forward with confidence without feeling overburdened.

Scenario 6: Users may just wish to discuss their financial concerns, such as stress about spending or doubt about saving. In addition to helping customers find answers, the chatbot can provide a comforting and useful conversational response that makes them feel heard and supported.

Technical Architecture:



Pre-requisites:

1. Python Programming – https://docs.python.org/3/
2. FastAPI Integration - https://fastapi.tiangolo.com/
3. IBM watsonx.ai - https://cloud.ibm.com/apidocs/watsonx-ai, https://www.ibm.com/docs/en/software-hub/5.1.x?topic=services-watsonxai
4. Streamlit - https://docs.streamlit.io/
5. LangChain - https://langchain.readthedocs.io/en/latest/
6. Watson NLU - https://cloud.ibm.com/apidocs/natural-language-understanding
7. Granite3.28B Instruct - https://www.ibm.com/docs/en/watsonx/w-and-w/2.1.0
8. IBM Cloud - https://cloud.ibm.com/docs

Project Workflow:

Activity 1: Model Selection and Architecture

- Activity 1.1: Research and Select Suitable Language Models for Financial Understanding that could interpret financial information, generate summaries, and provide meaningful recommendations. Emphasis was placed on models capable of both structured data understanding and contextual reasoning.

- Activity 1.2: Define Application Architecture Including Backend, Frontend, and Processing Layers. A high-level architecture was designed to connect user inputs with AI-generated insights through a well-structured pipeline.
- Activity 1.3: Set Up Development Environment and Dependencies to ensure smooth integration between the frontend, backend, and model layers during iterative development and testing.
- Activity 1.4: Create IBM NLU and Watsonx Credentials Using IBM Cloud as part of enabling the AI capabilities. This involved setting up cloud resources, configuring authentication, and securely connecting the chatbot with IBM's model APIs to support semantic analysis and financial reasoning.

Activity 2: Core Functionalities Development

- Activity 2.1: Build Core Features for Budget Summary, Spending Insights, and Personalized Advice. These features aimed to mimic a financial advisor by simplifying complex inputs and offering actionable suggestions for saving and budgeting.
- Activity 2.2: Implement FastAPI Backend for Routing and Feature Management using a modular API framework, enabling the system to handle multiple user queries while maintaining clarity between different processing components. This structure supported clean routing of user requests and model-driven responses.

Activity 3: Main App Logic

- Activity 3.1: Write Core Logic in Main Backend Script and Integrate Model Responses to orchestrate communication between the UI, models, and internal logic. It handled each interaction in a consistent manner, coordinating inputs, invoking the correct processing path, and formatting results for display.

Activity 4: Frontend Development

- Activity 4.1: Design Streamlit UI for User Interaction and Input Collection using simple and visually appealing user interface using Streamlit, offering users a familiar chat-like experience. This interface allowed users to either ask open-ended financial questions or paste in structured input for budget analysis.
- Activity 4.2: Display Financial Summaries and Smart Responses Visually through structured summaries, budget cards, and visual containers that highlighted the most relevant insights. This helped users quickly understand their spending patterns and areas for improvement.
- Activity 4.3: Add Summary Cards, Recommendations, and Page Enhancements such as bullet-point summaries, financial highlights, and layout refinements were introduced to elevate the user experience and give a clearer snapshot of their financial health.

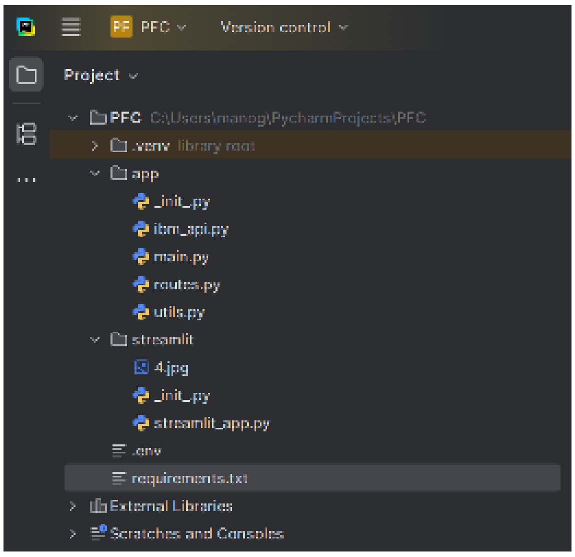
Activity 5: Deployment

- Activity 5.1: Prepare the Application for Deployment and Environment Setup for external access by organizing files, checking compatibility, and ensuring that service keys, environment variables, and configurations were securely handled and well documented.
- Activity 5.2: Deploy the Chatbot on a Hosting Platform with streamlit community cloud, making it accessible for real-time interaction and testing. This step enabled broader feedback collection and confirmed the system’s readiness for production use.

Activity 6: Conclusion

Milestone 1: Model Selection and Architecture

- Activity 1.1: Research and Select Suitable Language Models for Financial Understanding
    1. Understand the Project Requirements: The project's objective is to create an intelligent chatbot that can assist users in managing their money by comprehending their input regarding their earnings, outlays, saves, and financial objectives. In addition to providing correct responses, the assistant should offer users insightful advice.
    2. Explore IBM Watsonx models on IBM Cloud: Investigated the cloud-based linguistic tools offered by IBM Watson in order to achieve these objectives. These technologies are ideal for this type of helper since they are made to comprehend human language, including topics, emotions, and financial concepts.
    3. Evaluate the capabilties and select optimal set of models: Following a thorough analysis, decided on IBM Granite 3-2-8b instruct for producing insightful responses and IBM Watson NLU for comprehending user input. These models were selected due to their dependability, strong support, and capacity to manage actual financial discussions.
- Activity 1.2: Define Application Architecture Including Backend, Frontend, and Processing Layers
    1. Draft an Architectural Diagram: Developed a basic framework that illustrates the system's construction. It consists of an intelligence layer that comprehends and reacts, a middle layer that processes information, and a user interface.
    2. Detail Frontend Functionality and Backend responsibilities: Through a web interface, the user can discuss their earnings or expenses, ask questions, and get summary. This input is sent to a backend system, which links to the language models and returns intelligent responses or suggestions.
    3. Describe AI Integration Points: Connecting the system to strong language models gives it intelligence. Natural Language Understanding model aids in comprehending the user's words, regardless of how informal or sophisticated they are, while Granite 3-2-8b instruct aids in producing understandable and beneficial answers. The assistant feels natural and perceptive thanks to the seamless integration of these clever tools.
- Activity 1.3: Set Up Development Environment and Dependencies
    1. Install Python and Pip - Ensure you have Python 3.8+ and pip available on your machine.
    2. Install FastAPI, Uvicorn & Streamlit
    3. Install All Other Required Libraries including langchain, transformers, ibm-watsonx-ai.
    4. Set Up the Personal Finance Chatbot Folder Structure



```
PFC C:\Users\manog\PycharmProjects\PFC
  .venv library root
  app
    _init_.py
    ibm_api.py
    main.py
    routes.py
    utils.py
  streamlit
    4.jpg
    _init_.py
    streamlit_app.py
  .env
  requirements.txt
External Libraries
Scratches and Consoles
```

- Activity 1.4: Create IBM NLU and Watsonx Credentials Using IBM Cloud

1. IBM Cloud Account: To begin using IBM's AI services, first create an IBM Cloud account. This account gives access to various AI and language tools needed for our financial assistant.
2. Watson NLU: Set up the Watson NLU service, which helps the system analyze user messages—like identifying important keywords or understanding how the user feels. After setup, there is a unique key and URL that allows the chatbot to securely access this service.
3. Watsonx.ai studio: Next, activate Watsonx.ai, which includes a powerful language model called Granite. This model helps the assistant generate natural, helpful responses. Generate secure credentials for Granite, including a model ID, key, and project ID, and connected them to backend.

Milestone 2: Core Functionalities Development

- Activity 2.1: Build Core Features for Budget Summary

1. ibm_api.py

IBM Watson NLU for Sentiment and Entity Extraction function extracts semantic information from unprocessed user input by directly interacting with IBM Watson NLU's REST API. Named entity recognition (NER), sentiment analysis, and keyword extraction are its three main functions. Watson NLU can parse the input and return structured insights because the API payload is set up with a JSON schema. In order to maximize performance and relevancy, the function restricts the number of keywords and entities to five. For use later, the response is condensed into a concise dictionary. Because of its modular design, the function is a perfect preprocessing tool for financial NLP pipelines. It may be used to improve downstream LLM prompt creation or user profiling by adding metadata like as tone or subject classification to raw input.
Cached Watsonx LLM Model Loader using credentials unique to the project and model parameters kept in environment variables, this function initializes and returns a WatsonxLLM instance. By ensuring that the costly model initialization is only carried out once every runtime session, @lru_cache() greatly reduces latency and maximizes resource efficiency for deployments with large traffic. To avoid silent failures, all configuration variables—model ID, API key, project ID, and endpoint URL—are verified during runtime. The model is set up to accommodate long-form creation, such financial summaries, by allowing response sampling (decoding_method: sample) with a wide output window (max_new_tokens: 6500). Model instantiation is stored and separated from the prediction logic in this architecture, which follows scalable design conventions.
Prompt-based LLM Invocation serves as a centralized interface for leveraging the Watsonx Granite LLM to carry out prompt-based inference. The code guarantees a clear division between business logic and model interaction by enclosing the prompt submission logic in a distinct method. In accordance with LangChain's API specifications, the prompt is supplied to llm.generate() as a singleton list. The function is resilient to runtime errors like network problems or invalid prompts since it records exceptions and provides structured error diagnostics. Other prompt variations, model versions, or fallback logic can be supported by this paradigm, which is very adaptable and reusable. Applications needing deterministic interaction with generative models in production are a good fit for it.
LLM-based Budget Summarization uses the build_budget_prompt() utility to create a prompt, which is then fed into the Watsonx model to produce natural language summaries of structured financial information. The question is constructed to elicit a logical, intelligible reply that represents the user's financial metadata (such as income, expenses, and user type). In order to isolate the prompt's essential data portion for transparency or debugging, the model output is post-processed using extract_cleaned_prompt(). Frontend programs can use the function because it is exception-safe and returns a dictionary including the complete prompt, response text, and error (if any). This method improves decision support and user trust by making automated budget analysis workflows interpretable.
Deep Spending Behavior Analysis via LLM uses build_spending_insight_prompt() to synthesize monthly expense data into a structured prompt, operationalizing LLM-

based financial insight production. The prompt is designed to generate detailed insights that are divided into areas including recommendations, abnormalities, and expenditure trends. To enforce structural limitations, the output is parsed, with a tidy finish following section. In long completions, this prevents model drift or hallucinations. Consistency in downstream UI rendering is ensured by the post-processing logic, which exhibits exact control over the model's textual boundaries. This function offers a scalable, language-based method of financial coaching and reporting by showcasing an advanced usage of generative models for behavioral analytics.

2. utils.py

Static Prompt Builder is a simple, instruction-guided prompt template designed for lightweight LLM inference without contextual augmentation is produced by this function. By employing the persona description to formalize user input into a specified schema, it facilitates quick standardization among different sorts of assistants (e.g., professional, student). The output string is perfect for stateless conversational agents or backup situations where external NLU is either not available or not required because it has direct input echoing and a fixed behavioral directive ("respond clearly and concisely").
Contextualized Prompt with NLU Enrichment uses metadata collected from Watson NLU to enhance quick creation. To extract semantic sentiment, named entities, and keyword characteristics from user_text, it internally calls analyze_nlu(). The LLM can then tailor its answer based on user emotion and financial issue specificity thanks to the integration of these aspects into a context-aware prompt scaffold. The output enforces persona-adaptive recommendations (e.g., simpler procedures for students, brief strategies for professionals) and domain limits (e.g., excludes references to mental health or treatment). It is best suited for topically limited and emotionally intelligent generations.
Budget Summary Prompt (Student Persona) financial summary customized for student users is serialized by this function. It calculates post-savings surplus, disposable income, and total annual expenses as part of deterministic preprocessing. Additionally, it calculates monthly equivalents and percentages of expenses by category. Five output sections—ranked expenditure categories, qualitative recommendations, summary, hints, and a conclusion—are rigorously defined by the created prompt and must be followed in a particular order. Its instructional style, which directs LLMs toward straightforward, prescriptive counsel suitable for inexperienced financial users, places an emphasis on psychological readability, clarity, and fundamental financial literacy.
Budget Summary Prompt (Professional Persona) this variation modifies its reasoning for a more sophisticated financial audience. A closing admonition for a "professional takeaway," more assertive language framing, and three top expenditure categories (as opposed to two) are some of the variations. By maintaining interpretability bounds and avoiding unsupported recalculations, the prompt enforces domain fidelity. It is best suited for customers with greater incomes and analytically minded personalities looking for financial recommendations based on performance.
Persona-Based Prompt Dispatcher is where persona-specific prompt creation is abstracted away by this wrapper. Using a lowercase-checked user_type switch, it dynamically assigns execution to either build_professional_prompt() or build_student_prompt(). By offering a single interface and separating user segmentation logic from downstream systems, this function streamlines the creation

of external APIs. It can be expanded to accommodate new persona kinds through registry pattern insertion or straightforward function expansion.
Structured Insight Generator for Spending Behavior This function, which creates an eight-part prompt for in-depth budget analytics, is the most complicated in the module. It carries out multifaceted financial classification, which includes:

√Decomposition of Spending: Variable versus Fixed
√Needs vs Wants: Classification of the Budget
√Comparing benchmarks to static heuristics (e.g., 5% for eating out)
√Time-to-goal tracking based on surplus after savings
√Risk flagging is the process of using rules to identify high-risk trends, such as a rent-to-income ratio of greater than 30%.

Title blocks, interpretation prompts, and action directives are enforced by the output format's high level of structure. By integrating all quantitative inputs into the prompt itself and telling the LLM to rely only on interpretation and recommendation logic, the function reduces the possibility of hallucinations. Use cases involving automated budget feedback loops, AI financial coaching systems, and behavioral finance analysis are ideal for it.

- Activity 2.2: Implement FastAPI Backend for Routing and Feature Management

Across the codebase, prompt templates live in utils.py, model orchestration lives in ibm_api.py, routes live in routes.py, and presentation lives in streamlit_app.py. This strict modularization—combined with persona-driven templates, NLU-powered context enrichment, and LLM result post-processing—yields a system that is easy to extend (add a new persona or report type), debug (inspect prompts in isolation), and maintain (swappable LLM models, frontends). Each feature encapsulates a clear single responsibility, promoting code clarity and long-term adaptability.

Milestone 3: Main App Logic

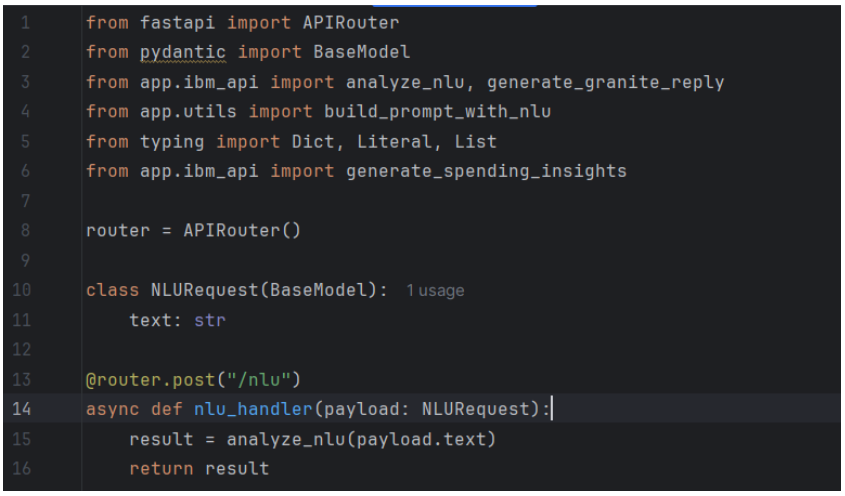
1. Initialization (App Configuration) - FastAPI first creates the application instance (app = FastAPI(...)) when main.py is run. In this stage, the OpenAPI schema is automatically updated with metadata such as the API title, description, and version. Autogenerated interactive documents (/docs or /redoc) use these definitions.

2. Environment Setup - Configuration variables (such as API keys and service URLs) are injected into the environment via the load_dotenv() function, which reads from the.env file. This makes it possible to safely access external services through environment-driven integration,such as Granite or IBM Watson NLU.

3. Middleware Registration - The application is enhanced by middleware, like CORSMiddleware in this instance. Every request and response are intercepted by this layer. Before the request reaches the route handler, it conducts pre-processing (such as CORS checks, authentication, logging, etc.) and post-processing (such as sending the response). Cross-origin queries from http://localhost:8000 are enabled here.

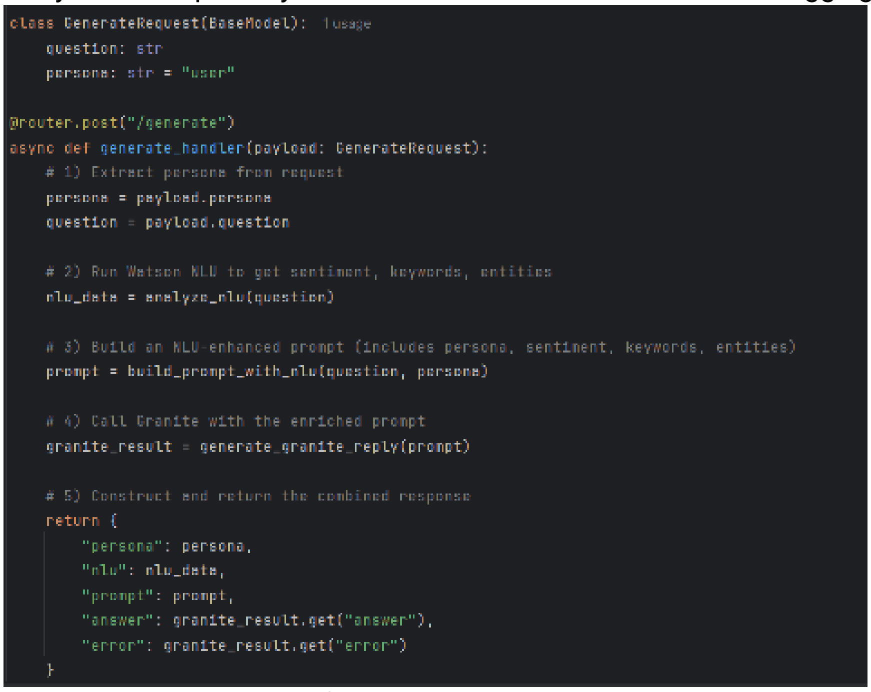
4. Route Mounting – app.include_router(router) mounts the router from app.routes. This allows the running API to access the registered endpoints, such as /nlu, /generate, /budget-summary, and /spending-insights. Based on the HTTP method and URL path, a route is assigned to each incoming request.

√IBM Watson's Natural Language Understanding service is used to interpret plain text input at the /nlu endpoint and extract structured insights like sentiment, keywords, and entities. This path is mostly intended for internal diagnostics or pre-processing, which enables developers to assess the semantic interpretation of user input prior to its transfer into generative pipelines. It is a lightweight utility endpoint for testing or frontend previews since it employs a straightforward NLURequest architecture and returns Watson's raw output unaltered.



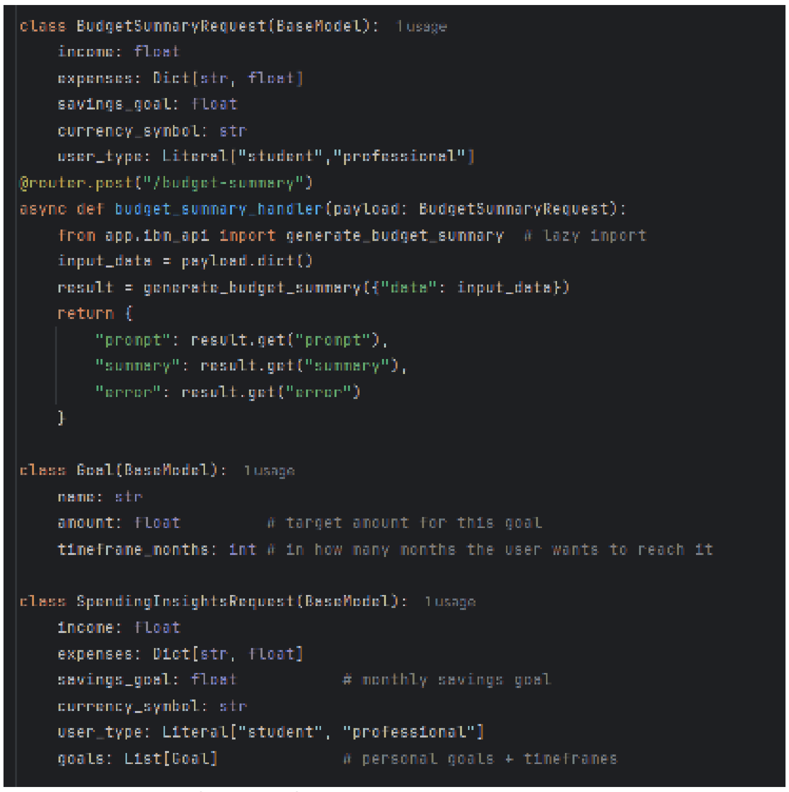
```python
from fastapi import APIRouter
from pydantic import BaseModel
from app.ibm_api import analyze_nlu, generate_granite_reply
from app.utils import build_prompt_with_nlu
from typing import Dict, Literal, List
from app.ibm_api import generate_spending_insights

router = APIRouter()

class NLURequest(BaseModel):  1 usage
    text: str

@router.post("/nlu")
async def nlu_handler(payload: NLURequest):
    result = analyze_nlu(payload.text)
    return result
```

√Persona-driven LLM production and NLU analysis are combined in the composite pipeline known as the /generate endpoint. It first uses Watson NLU to extract semantic indicators and emotional tone from the text after receiving a user inquiry and an optional persona label. These characteristics are incorporated into a personalized prompt created using build_prompt_with_nlu(), which is subsequently supplied to an LLM (Granite) to provide a human-like, context-aware response. The persona, enriched prompt, raw NLU data, and the LLM's response are all included in the response payload, which offers complete traceability and transparency for UX modification or downstream debugging.



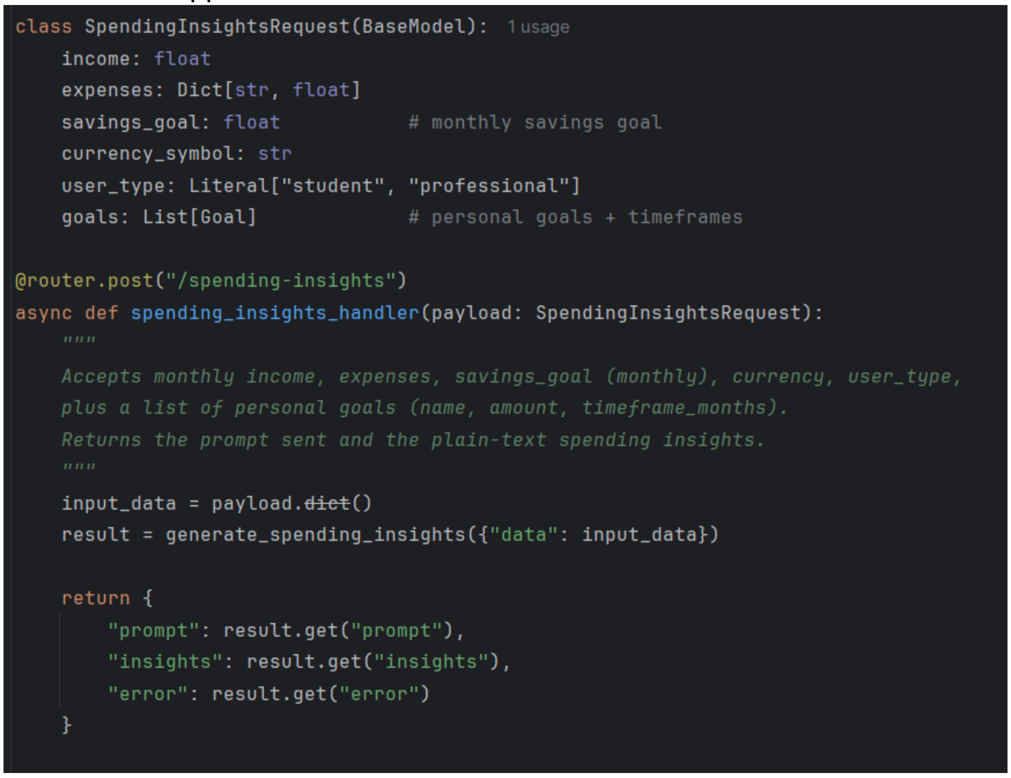
```
class GenerateRequest(BaseModel):  1 usage
    question: str
    persona: str = "user"

@router.post("/generate")
async def generate_handler(payload: GenerateRequest):
    # 1) Extract persona from request
    persona = payload.persona
    question = payload.question

    # 2) Run Watson NLU to get sentiment, keywords, entities
    nlu_data = analyze_nlu(question)

    # 3) Build an NLU-enhanced prompt (includes persona, sentiment, keywords, entities)
    prompt = build_prompt_with_nlu(question, persona)

    # 4) Call Granite with the enriched prompt
    granite_result = generate_granite_reply(prompt)

    # 5) Construct and return the combined response
    return {
        "persona": persona,
        "nlu": nlu_data,
        "prompt": prompt,
        "answer": granite_result.get("answer"),
        "error": granite_result.get("error")
    }
```

√This tool creates a condensed financial picture by processing structured budget data, such as monthly income, categorized expenses, savings objectives, and user profile. The endpoint creates a descriptive prompt with the user's financial information and persona context using a lazy import of generate_budget_summary. This prompt is then sent to the LLM for summarization. The output is helpful for reporting dashboards or advising tools since it provides the final prompt together with a narrative summary that explains the user's financial situation.



```
class BudgetSummaryRequest(BaseModel):  1 usage
    income: float
    expenses: Dict[str, float]
    savings_goal: float
    currency_symbol: str
    user_type: Literal["student","professional"]
@router.post("/budget-summary")
async def budget_summary_handler(payload: BudgetSummaryRequest):
    from app.ibm_api import generate_budget_summary  # lazy import
    input_data = payload.dict()
    result = generate_budget_summary({"data": input_data})
    return {
        "prompt": result.get("prompt"),
        "summary": result.get("summary"),
        "error": result.get("error")
    }

class Goal(BaseModel):  1 usage
    name: str
    amount: float          # target amount for this goal
    timeframe_months: int # in how many months the user wants to reach it

class SpendingInsightsRequest(BaseModel):  1 usage
    income: float
    expenses: Dict[str, float]
    savings_goal: float            # monthly savings goal
    currency_symbol: str
    user_type: Literal["student", "professional"]
    goals: List[Goal]              # personal goals + timeframes
```

√Based on the user's profile and financial objectives, the /spending-data endpoint offers detailed financial insights. A savings goal, income, costs, and a list of future goals with deadlines are among the many inputs it allows. Using this information, the system creates a prompt that directs the LLM to provide tailored, forward-looking guidance, emphasizing goal achievability, budget leakage, and surplus potential. Use cases requiring in-depth contextual interpretation of financial habits, such as goal monitoring, financial coaching, or embedded fintech assistants, are well suited for this approach.



```python
class SpendingInsightsRequest(BaseModel):  1 usage
    income: float
    expenses: Dict[str, float]
    savings_goal: float           # monthly savings goal
    currency_symbol: str
    user_type: Literal["student", "professional"]
    goals: List[Goal]             # personal goals + timeframes

@router.post("/spending-insights")
async def spending_insights_handler(payload: SpendingInsightsRequest):
    """
    Accepts monthly income, expenses, savings_goal (monthly), currency, user_type,
    plus a list of personal goals (name, amount, timeframe_months).
    Returns the prompt sent and the plain-text spending insights.
    """
    input_data = payload.~~dict~~()
    result = generate_spending_insights({"data": input_data})

    return {
        "prompt": result.get("prompt"),
        "insights": result.get("insights"),
        "error": result.get("error")
    }
```

5. Request Handling - For each HTTP request: FastAPI uses the relevant Pydantic model (e.g., GenerateRequest, BudgetSummaryRequest) to parse and validate the incoming payload. The verified data is then injected into the appropriate async route method (generate_handler(), for example). Domain-specific logic (such as creating prompts, contacting Watson NLU, and producing insights) is carried out by the route. After being returned, a response dictionary is immediately serialized into JSON.

6. Response Lifecycle - When a response is returned by the route, FastAPI serializes it, sets the appropriate HTTP headers (such as content-type and CORS), and then delivers the response back to the client. FastAPI has built-in exception management and will offer a structured error response with status codes if an exception arises at any point.

Milestone 4: Frontend Development

- Activity 4.1: Design Streamlit UI for User Interaction and Input Collection

  This task uses Streamlit to explore the user interface's architectural underpinnings. Using st.set_page_config() and a custom set_background() function, the code first configures the layout and graphics before encoding and injecting a background image using base64-encoded CSS. In order to improve visual engagement, the home page is decorated with frosted glass UI elements utilizing inline HTML/CSS. Additionally, it uses st.session_state.page to build a navigation mechanism that allows for multi-page viewing without requiring page reloads. The NLU Analysis, Q&A, Budget Summary, and Spending Insights capabilities are all accessible through buttons on the main page, each of which updates the page's status.
- Activity 4.2: Display Financial Summaries and Smart Responses

  This activity is executed through dedicated pages (budget-summary, spending-insights, generate, and nlu) that collect structured or free-form JSON input from users via st.text_area(). After selecting "Send," the input is sent to a FastAPI backend (located at http://127.0.0.1:8000) via HTTP POST requests after being parsed using Python's json.loads(). St.json() is used to display the analysis results that the backend provides, such as budget summaries, intelligent responses, or NLU insights. This architecture bridges real-time backend analytics with frontend visualization, making the assistant interactive and responsive.
- Activity 4.3: Add Summary Cards, Recommendations, and Page

  Using reusable CSS classes (like.white-box) to style white-box components improves the user experience. For visual clarity, the container_wrapper(fn) method wraps distinct parts in a styled container with transparency, shadows, and padding. These components display cards for summaries (such as revenue breakdown, cost insights, or AI-generated ideas), which makes the output visually structured and simple to skim. In order to ensure navigability, each feature page also has a "   Back" button that directs users to the homepage.

Milestone 5: Deployment

- Activity 5.1: Prepare the Application for Deployment and Environment Setup

1. Containerize and set up stack in a reproducible environment prior to going live. Initially, establish a specific Python virtual environment and install all dependencies specified in requirements.txt, ranging from the IBM Watsonx SDK to FastAPI, Uvicorn, and Streamlit.

   ```
   python3 -m venv .venv
   source .venv/bin/activate
   pip install -r requirements.txt
   ```

2. In order for the FastAPI service to call Watsonx.ai through the IBM SDK, we next securely provision our Watsonx Granite 3-8b-instruct and NLU credentials as environment variables.

   NLU_KEY, NLU_URL, WATSONX_KEY, WATSONX_URL, WATSONX_MODEL_ID and PROJECT_ID

3. Finally to make sure that model inference functions properly against both the IBM and NLU backends we lastly check the CORS settings in main.py, and run a smoke test by locally executing the endpoints.

- Activity 5.2: Deploy the Chatbot on a Hosting Platform - In one terminal, we start the FastAPI backend with Uvicorn:

  ```
  uvicorn main:app --host 0.0.0.0 --port 8000 –reload
  ```

  This exposes endpoints powered by IBM Granite and NLU models.

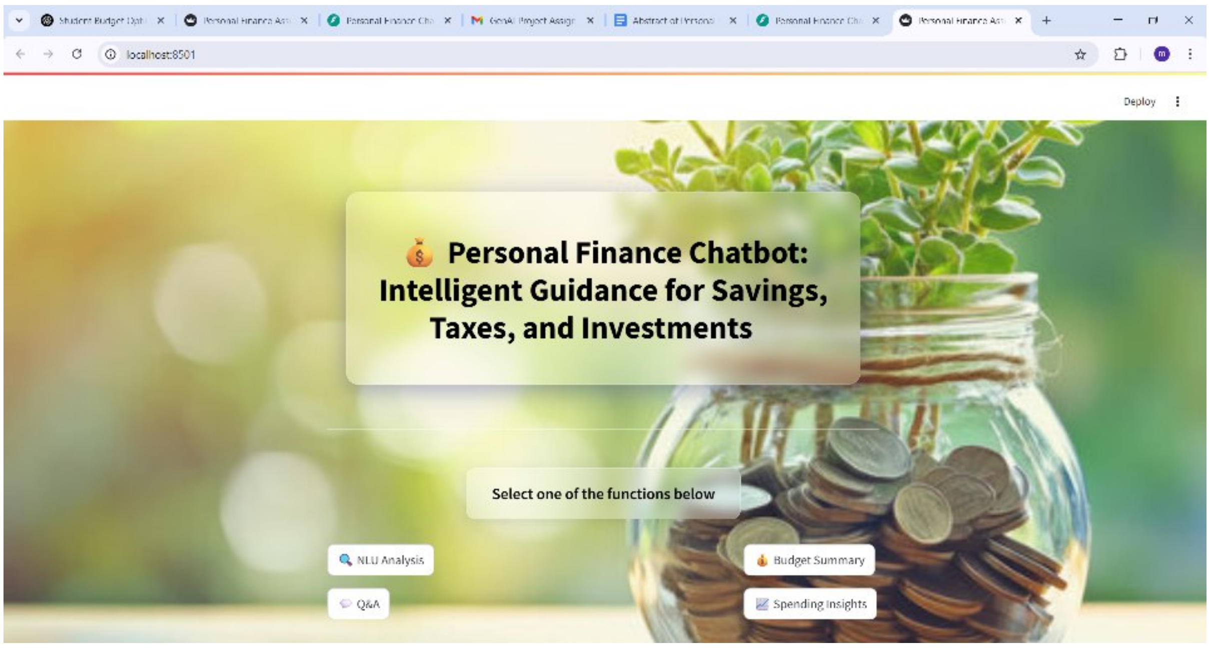
  In a second terminal, we fire up the Streamlit frontend:

  ```
  streamlit run streamlit/app.py
  ```

  Streamlit connects to http://127.0.0.1:8000, enabling users to ask questions, upload data and obtain results.

Exploring website's Web Pages:

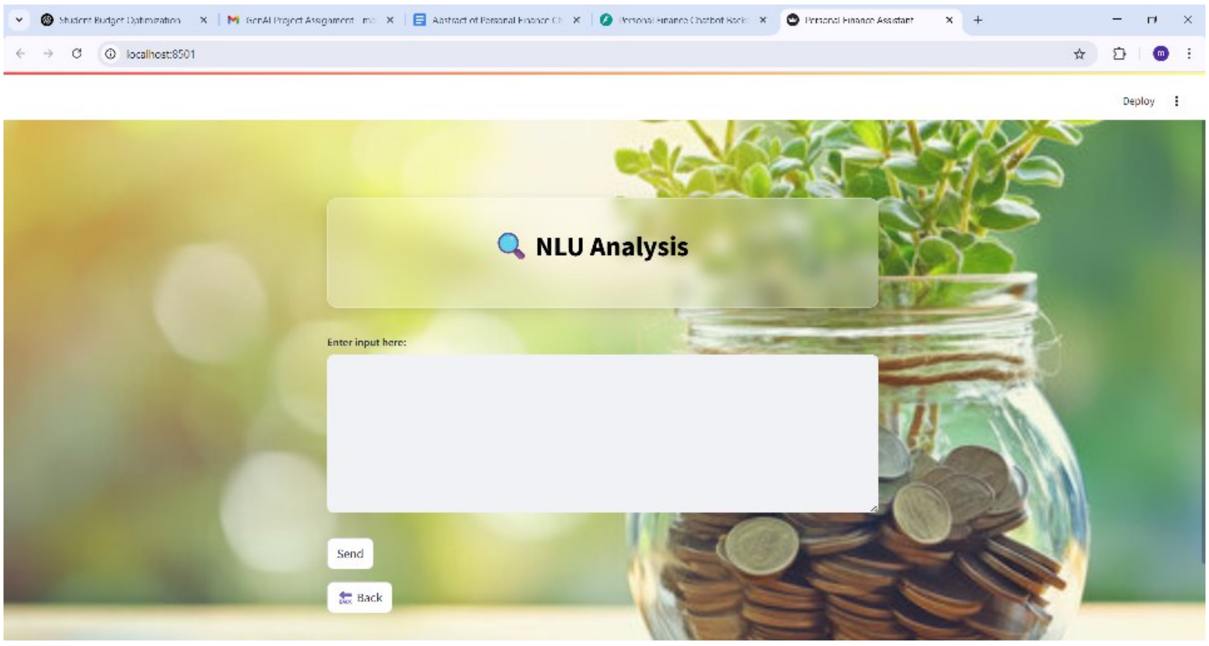
Home Page



💰 Personal Finance Chatbot: Intelligent Guidance for Savings, Taxes, and Investments

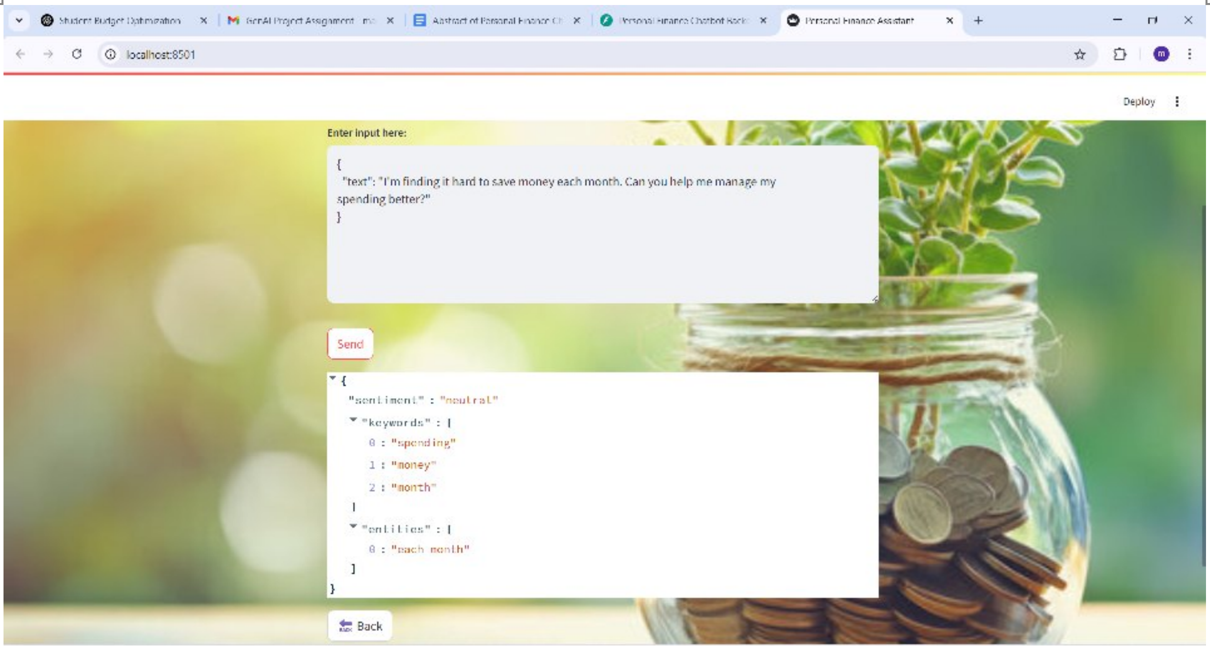
Select one of the functions below

NLU Analysis



🔍 NLU Analysis

Enter input here:

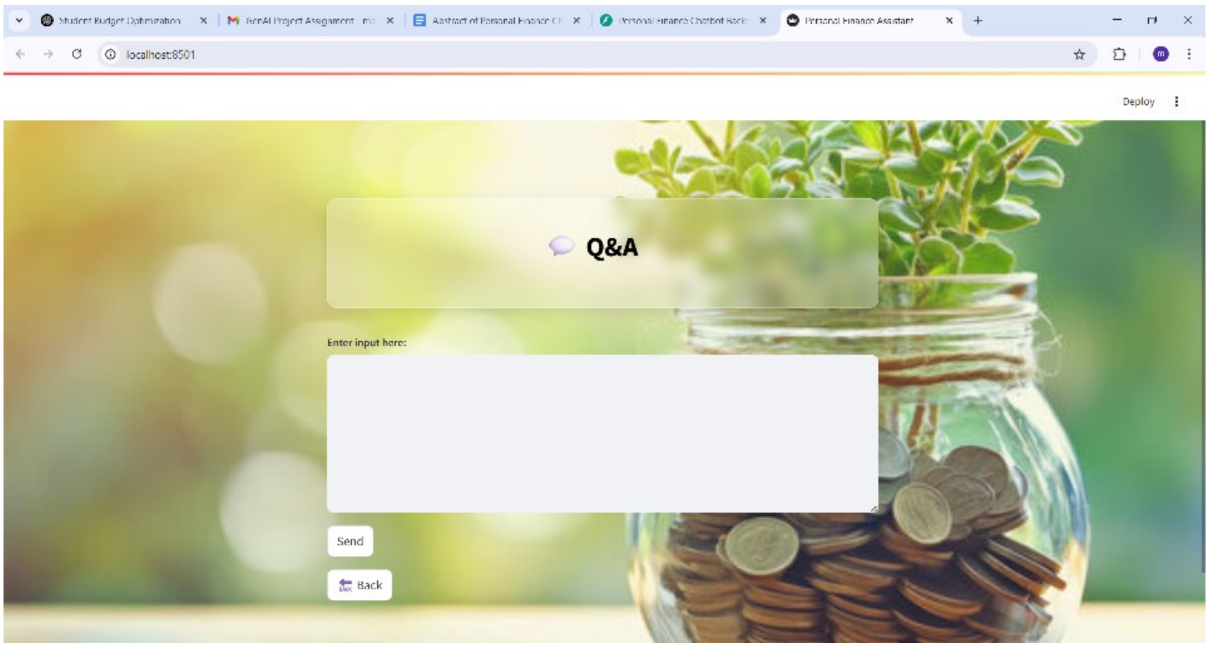


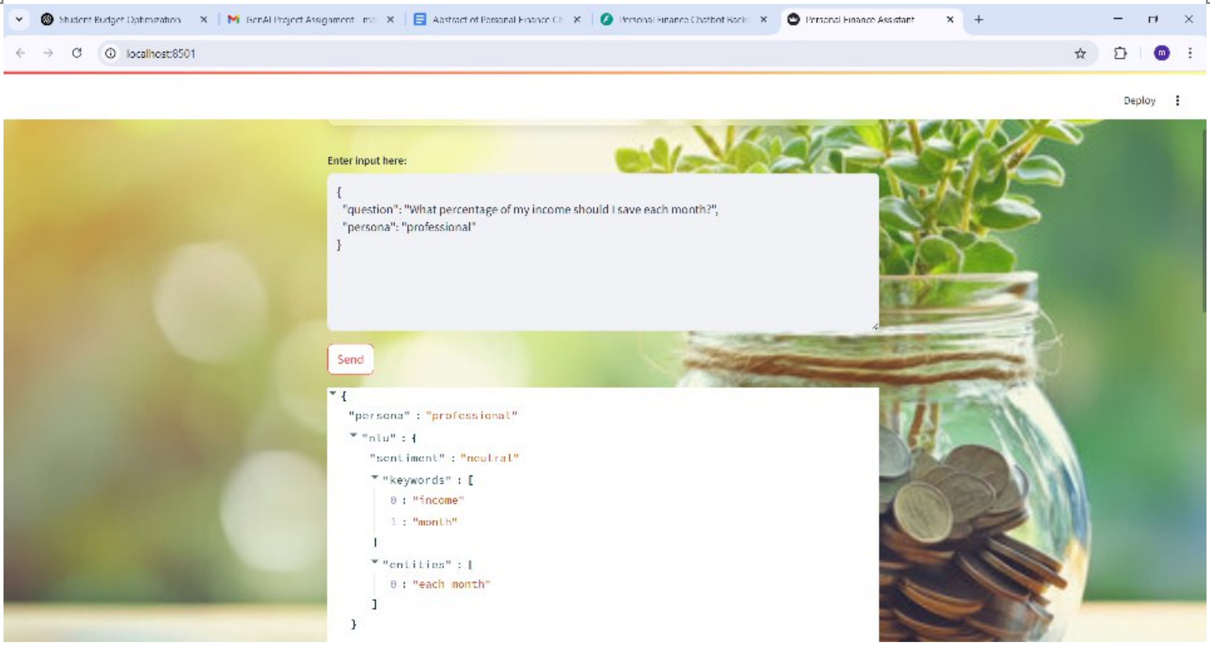
```
{
 "text": "I'm finding it hard to save money each month. Can you help me manage my
spending better?"
}
```

```
{
  "sentiment" : "neutral"
  "keywords" : [
    0 : "spending"
    1 : "money"
    2 : "month"
  ]
  "entities" : [
    0 : "each month"
  ]
}
```

Description: The NLU Analysis page presents a frosted-glass input panel, creating a modern and welcoming feel. Users enter a JSON snippet such as a text asking for help with saving money, then click “Send” to see results. The page displays, for instance, a neutral sentiment alongside keywords like “spending,” “money,” and “month,” and identifies entities such as “each month” in a clear, styled box below. A “Back” button returns to the main menu. This layout emphasizes straightforward input and immediate, readable feedback.

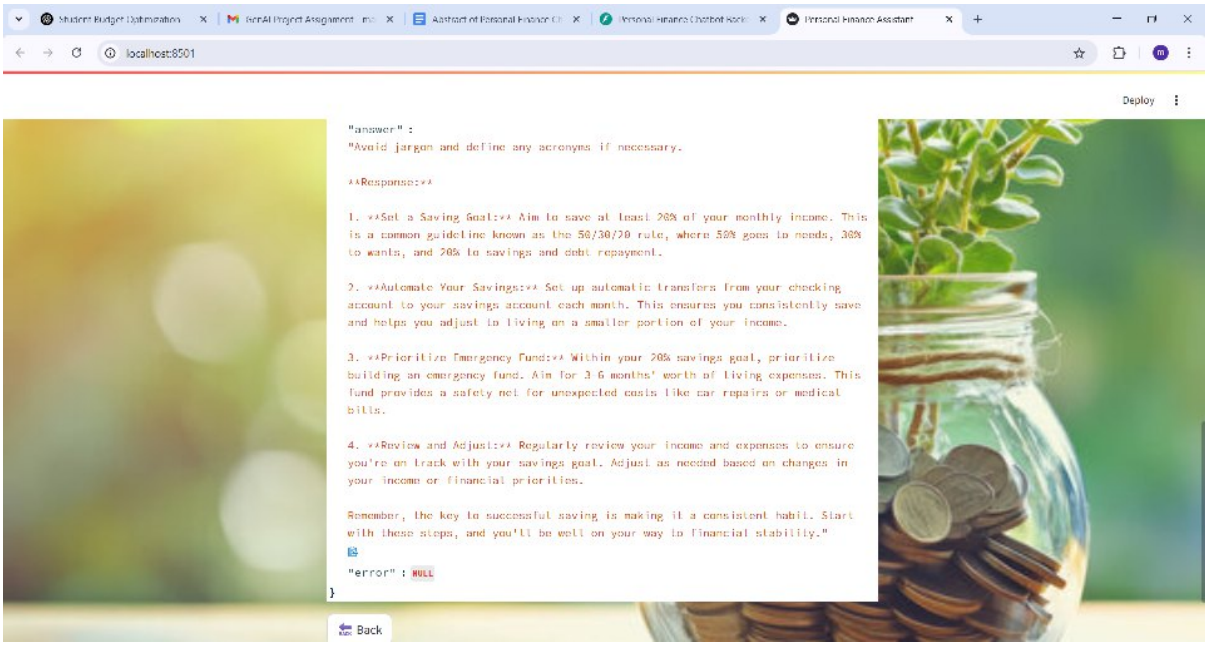
Q&A




Student Budget Optimization
GenAI Project Assignment
Abstract of Personal Finance Ch
Personal Finance Chatbot Back
Personal Finance Assistant
localhost:8501
Deploy
Enter input here:
Send

```
{
 "question": "What percentage of my income should I save each month?",
 "persona": "professional"
}
```

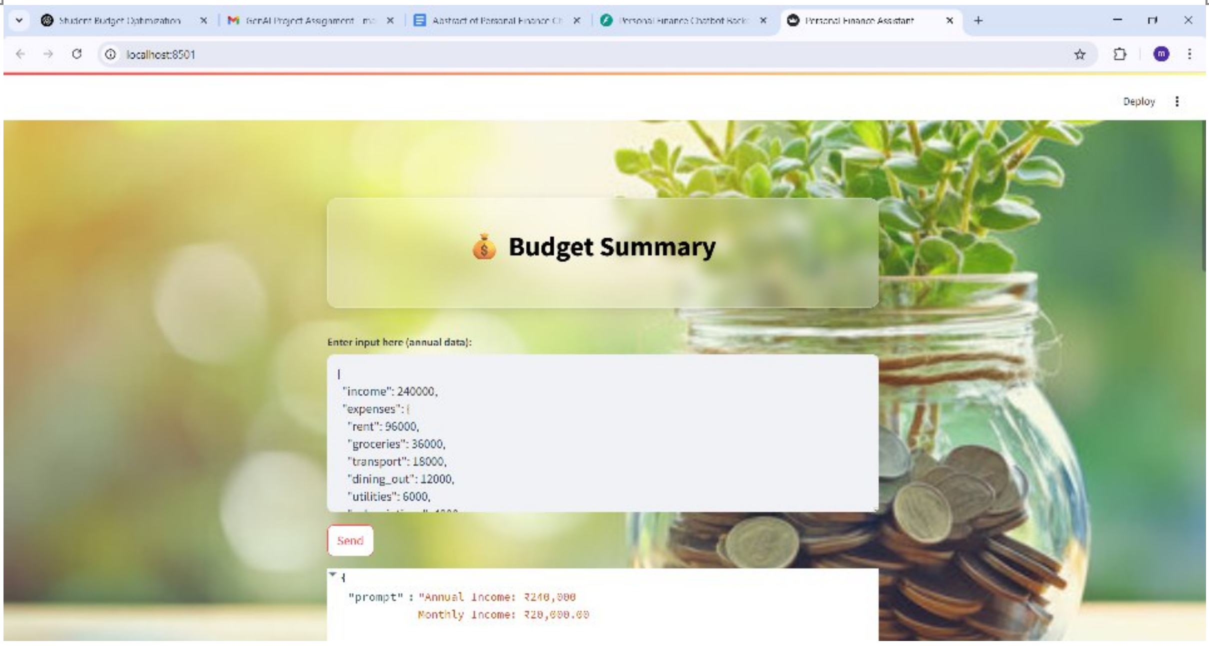
```
{
  "persona" : "professional"
  "nlu" : {
    "sentiment" : "neutral"
    "keywords" : [
      0 : "income"
      1 : "month"
    ]
    "entities" : [
      0 : "each month"
    ]
  }
```


Student Budget Optimization
GenAI Project Assignment
Abstract of Personal Finance Ch
Personal Finance Chatbot Back
Personal Finance Assistant
localhost:8501
Deploy
Back

```
  "answer" :
  "Avoid jargon and define any acronyms if necessary.

  **Response:**

  1. **Set a Saving Goal:** Aim to save at least 20% of your monthly income. This
  is a common guideline known as the 50/30/20 rule, where 50% goes to needs, 30%
  to wants, and 20% to savings and debt repayment.

  2. **Automate Your Savings:** Set up automatic transfers from your checking
  account to your savings account each month. This ensures you consistently save
  and helps you adjust to living on a smaller portion of your income.

  3. **Prioritize Emergency Fund:** Within your 20% savings goal, prioritize
  building an emergency fund. Aim for 3-6 months' worth of living expenses. This
  fund provides a safety net for unexpected costs like car repairs or medical
  bills.

  4. **Review and Adjust:** Regularly review your income and expenses to ensure
  you're on track with your savings goal. Adjust as needed based on changes in
  your income or financial priorities.

  Remember, the key to successful saving is making it a consistent habit. Start
  with these steps, and you'll be well on your way to financial stability."

  "error" : NULL
}
```

Description: The Q&A page displays a frosted-glass input panel, inviting users to enter a JSON object with a "question" and "persona." After clicking "Send," the output appears below in a styled box showing sentiment and keywords alongside a clear, numbered response. For example, it may recommend setting a savings goal of 20% of income, automating transfers, prioritizing an emergency fund, and regularly reviewing progress. A "Back" button returns to the main menu. This layout highlights straightforward input with immediate, actionable feedback, making the advice feel practical and easy to follow.

Budget Summary

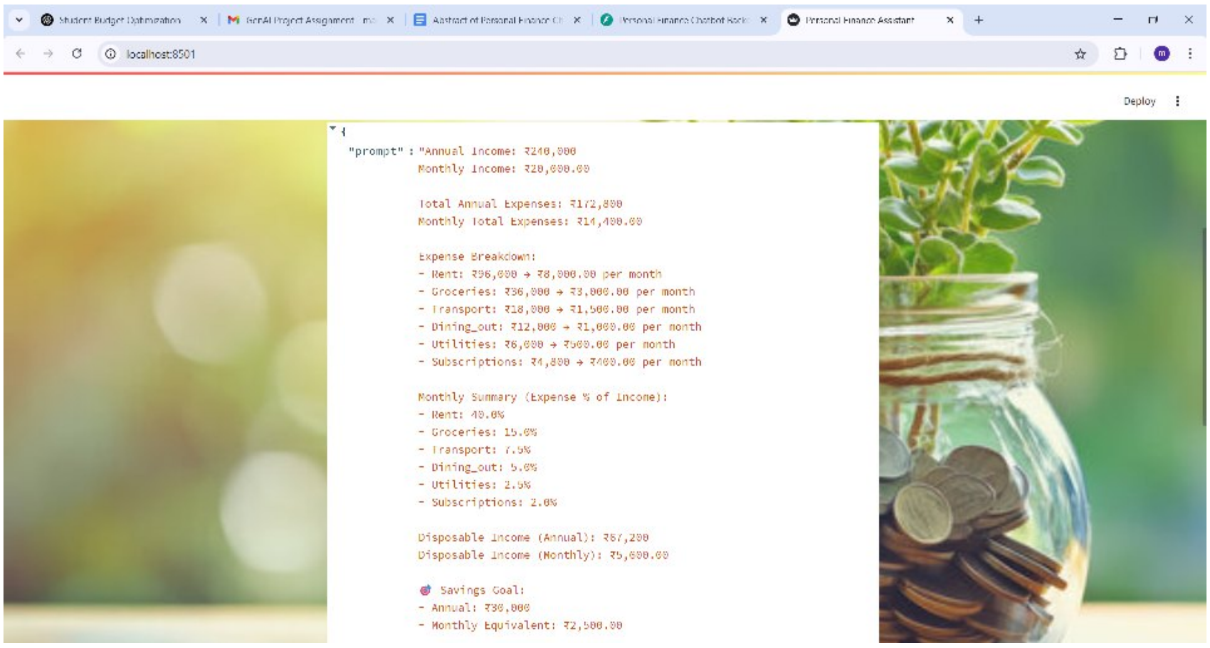


💰 Budget Summary

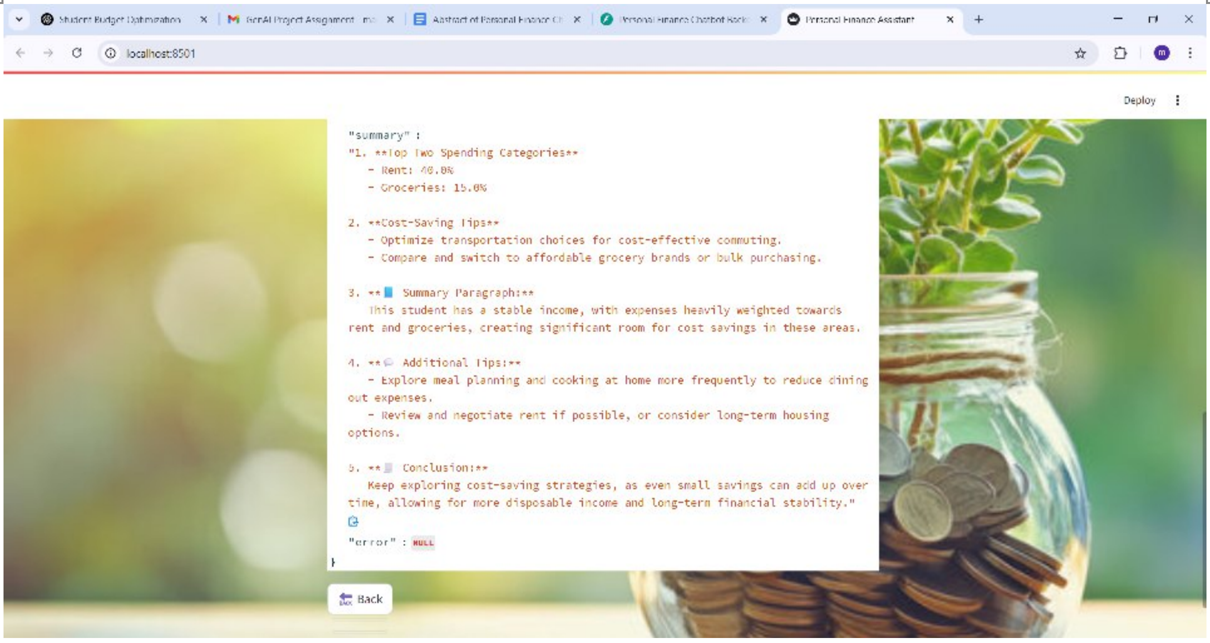
Enter input here (annual data):

```
{
 "income": 240000,
 "expenses": {
  "rent": 96000,
  "groceries": 36000,
  "transport": 18000,
  "dining_out": 12000,
  "utilities": 6000,
```

```
{
  "prompt" : "Annual Income: ₹240,000
             Monthly Income: ₹20,000.00
```



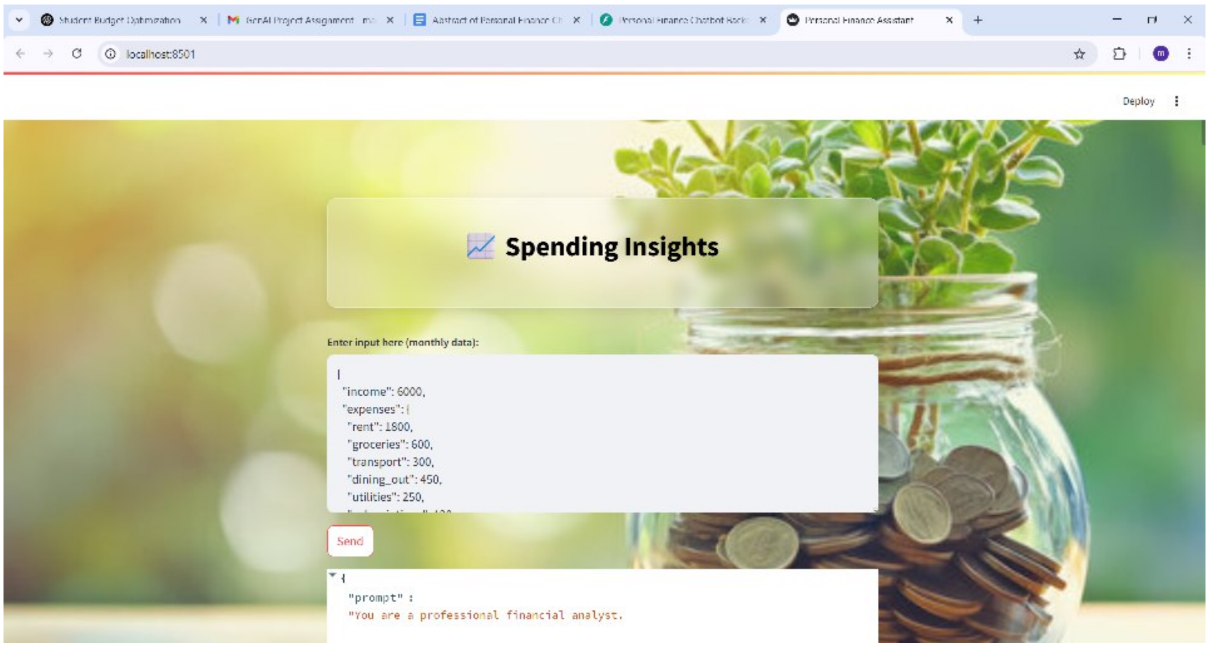
```
{
  "prompt" : "Annual Income: ₹240,000
             Monthly Income: ₹20,000.00

             Total Annual Expenses: ₹172,800
             Monthly Total Expenses: ₹14,400.00

             Expense Breakdown:
             - Rent: ₹96,000 → ₹8,000.00 per month
             - Groceries: ₹36,000 → ₹3,000.00 per month
             - Transport: ₹18,000 → ₹1,500.00 per month
             - Dining_out: ₹12,000 → ₹1,000.00 per month
             - Utilities: ₹6,000 → ₹500.00 per month
             - Subscriptions: ₹4,800 → ₹400.00 per month

             Monthly Summary (Expense % of Income):
             - Rent: 40.0%
             - Groceries: 15.0%
             - Transport: 7.5%
             - Dining_out: 5.0%
             - Utilities: 2.5%
             - Subscriptions: 2.0%

             Disposable Income (Annual): ₹67,200
             Disposable Income (Monthly): ₹5,600.00

             🎯 Savings Goal:
             - Annual: ₹30,000
             - Monthly Equivalent: ₹2,500.00
```



```
"summary" :
"1. **Top Two Spending Categories**
   - Rent: 40.0%
   - Groceries: 15.0%

2. **Cost-Saving Tips**
   - Optimize transportation choices for cost-effective commuting.
   - Compare and switch to affordable grocery brands or bulk purchasing.

3. **📘 Summary Paragraph:**
   This student has a stable income, with expenses heavily weighted towards
rent and groceries, creating significant room for cost savings in these areas.

4. **💬 Additional Tips:**
   - Explore meal planning and cooking at home more frequently to reduce dining
out expenses.
   - Review and negotiate rent if possible, or consider long-term housing
options.

5. **📄 Conclusion:**
   Keep exploring cost-saving strategies, as even small savings can add up over
time, allowing for more disposable income and long-term financial stability."
"error" : NULL
}
```

Description: The Budget Summary page shows a frosted-glass input panel, inviting a JSON object with fields like income, expenses, savings goal, currency symbol, and persona. After clicking "Send," the output appears in a styled box showing a formatted summary: annual income, total expenses with per-category and percentage breakdowns, disposable income, savings goal, and surplus. Below that, a concise section lists top spending categories, cost-saving tips (e.g., optimize transportation or groceries), a brief summary paragraph noting key patterns, additional tips (like meal planning or rent negotiation), and a short conclusion encouraging continued cost-saving strategies. A "Back" button returns to the main menu.

Spending Insights

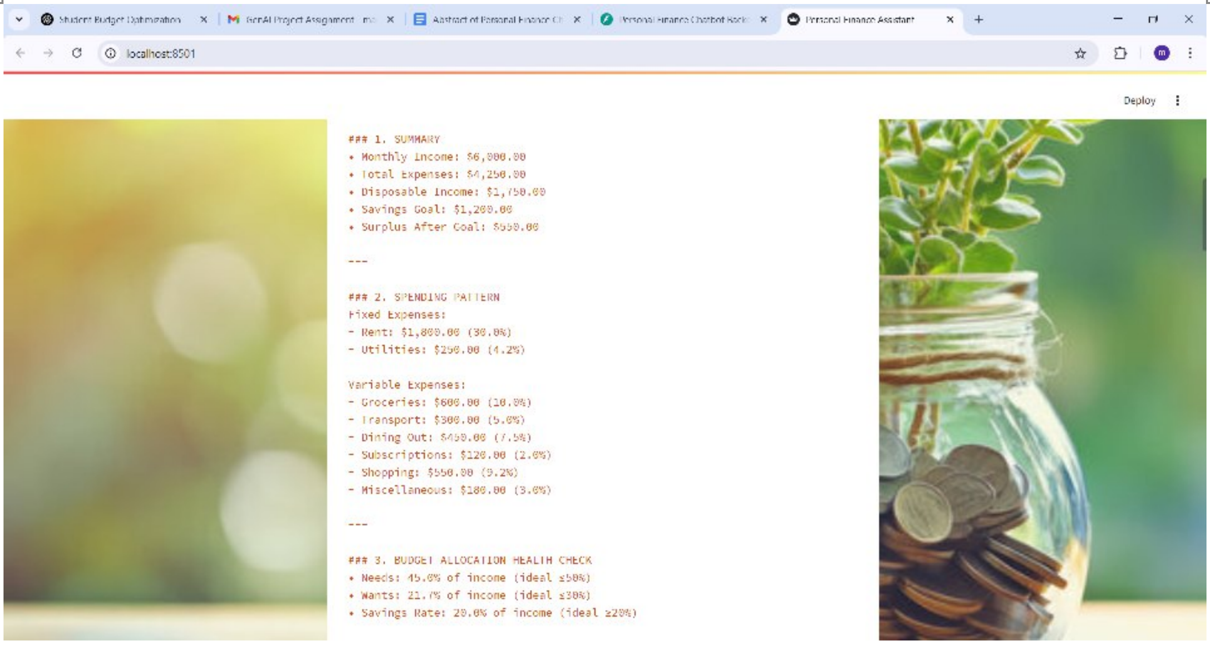


📈 Spending Insights

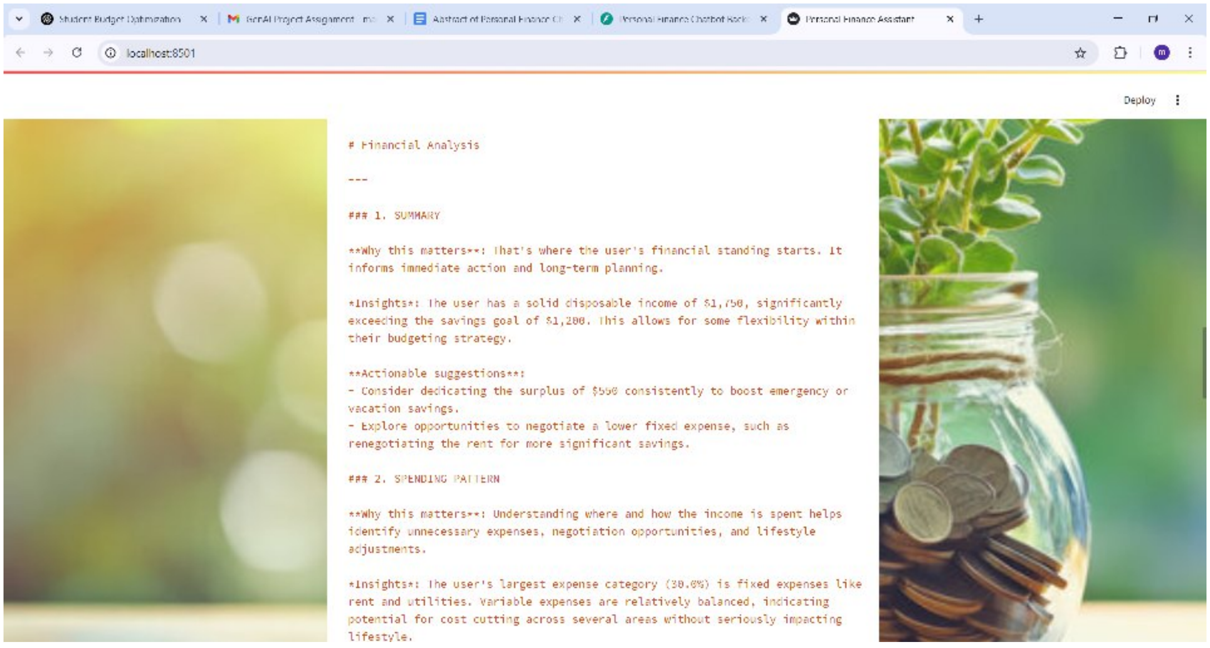
Enter input here (monthly data):

```
{
  "income": 6000,
  "expenses": {
   "rent": 1800,
   "groceries": 600,
   "transport": 300,
   "dining_out": 450,
   "utilities": 250,
```

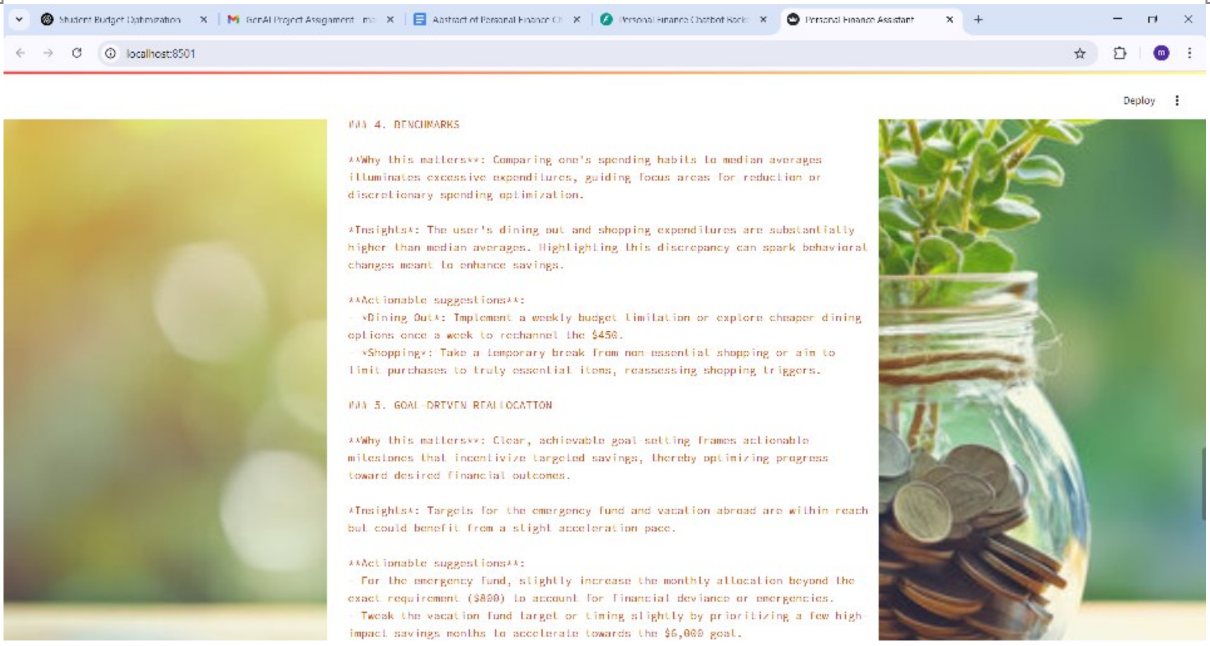
```
{
  "prompt" :
  "You are a professional financial analyst.
```


Student Budget Optimization
GenAI Project Assignment
Abstract of Personal Finance Ch
Personal Finance Chatbot Back
Personal Finance Assistant
localhost:8501
Deploy

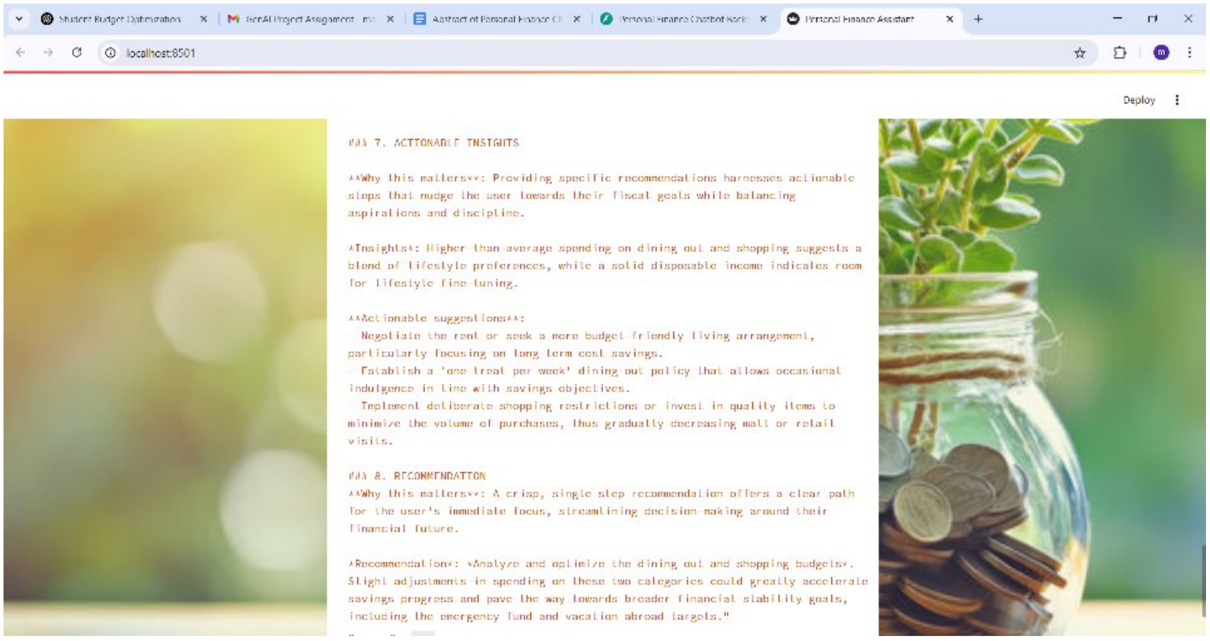
```
### 1. SUMMARY
• Monthly Income: $6,000.00
• Total Expenses: $4,250.00
• Disposable Income: $1,750.00
• Savings Goal: $1,200.00
• Surplus After Goal: $550.00

---

### 2. SPENDING PATTERN
Fixed Expenses:
- Rent: $1,800.00 (30.0%)
- Utilities: $250.00 (4.2%)

Variable Expenses:
- Groceries: $600.00 (10.0%)
- Transport: $300.00 (5.0%)
- Dining Out: $450.00 (7.5%)
- Subscriptions: $120.00 (2.0%)
- Shopping: $550.00 (9.2%)
- Miscellaneous: $180.00 (3.0%)

---

### 3. BUDGET ALLOCATION HEALTH CHECK
• Needs: 45.0% of income (ideal ≤50%)
• Wants: 21.7% of income (ideal ≤30%)
• Savings Rate: 20.0% of income (ideal ≥20%)
```


Student Budget Optimization
GenAI Project Assignment
Abstract of Personal Finance Ch
Personal Finance Chatbot Back
Personal Finance Assistant
localhost:8501
Deploy

```
# Financial Analysis

---

### 1. SUMMARY

**Why this matters**: That's where the user's financial standing starts. It
informs immediate action and long-term planning.

*Insights*: The user has a solid disposable income of $1,750, significantly
exceeding the savings goal of $1,200. This allows for some flexibility within
their budgeting strategy.

**Actionable suggestions**:
- Consider dedicating the surplus of $550 consistently to boost emergency or
vacation savings.
- Explore opportunities to negotiate a lower fixed expense, such as
renegotiating the rent for more significant savings.

### 2. SPENDING PATTERN

**Why this matters**: Understanding where and how the income is spent helps
identify unnecessary expenses, negotiation opportunities, and lifestyle
adjustments.

*Insights*: The user's largest expense category (30.0%) is fixed expenses like
rent and utilities. Variable expenses are relatively balanced, indicating
potential for cost cutting across several areas without seriously impacting
lifestyle.
```


Student Budget Optimization
GenAI Project Assignment
Abstract of Personal Finance Ch
Personal Finance Chatbot Back
Personal Finance Assistant
localhost:8501
Deploy

```
### 4. BENCHMARKS

**Why this matters**: Comparing one's spending habits to median averages
illuminates excessive expenditures, guiding focus areas for reduction or
discretionary spending optimization.

*Insights*: The user's dining out and shopping expenditures are substantially
higher than median averages. Highlighting this discrepancy can spark behavioral
changes meant to enhance savings.

**Actionable suggestions**:
- *Dining Out*: Implement a weekly budget limitation or explore cheaper dining
options once a week to rechannel the $450.
- *Shopping*: Take a temporary break from non-essential shopping or aim to
limit purchases to truly essential items, reassessing shopping triggers.

### 5. GOAL-DRIVEN REALLOCATION

**Why this matters**: Clear, achievable goal-setting frames actionable
milestones that incentivize targeted savings, thereby optimizing progress
toward desired financial outcomes.

*Insights*: Targets for the emergency fund and vacation abroad are within reach
but could benefit from a slight acceleration pace.

**Actionable suggestions**:
- For the emergency fund, slightly increase the monthly allocation beyond the
exact requirement ($800) to account for financial deviance or emergencies.
- Tweak the vacation fund target or timing slightly by prioritizing a few high-
impact savings months to accelerate towards the $6,000 goal.
```


Student Budget Optimization
GenAI Project Assignment
Abstract of Personal Finance Ch
Personal Finance Chatbot Back
Personal Finance Assistant
localhost:8501
Deploy

```
### 7. ACTIONABLE INSIGHTS

**Why this matters**: Providing specific recommendations harnesses actionable
steps that nudge the user towards their fiscal goals while balancing
aspirations and discipline.

*Insights*: Higher-than-average spending on dining out and shopping suggests a
blend of lifestyle preferences, while a solid disposable income indicates room
for lifestyle fine-tuning.

**Actionable suggestions**:
- Negotiate the rent or seek a more budget-friendly living arrangement,
particularly focusing on long-term cost savings.
- Establish a 'one treat per week' dining-out policy that allows occasional
indulgence in line with savings objectives.
- Implement deliberate shopping restrictions or invest in quality items to
minimize the volume of purchases, thus gradually decreasing mall or retail
visits.

### 8. RECOMMENDATION
**Why this matters**: A crisp, single step recommendation offers a clear path
for the user's immediate focus, streamlining decision-making around their
financial future.

*Recommendation*: *Analyze and optimize the dining out and shopping budgets*.
Slight adjustments in spending on these two categories could greatly accelerate
savings progress and pave the way towards broader financial stability goals,
including the emergency fund and vacation abroad targets."
```

Description: This "Spending Insights" feature offers a sleek Streamlit UI with a blurred card header, a JSON input box for monthly financial data, and a "Send" button. On submission, it displays detailed financial summaries, spending patterns, and goal-based advice in a clean, scrollable, red monospaced format. The layout separates input and output for clarity, delivering clear, actionable insights that help users manage expenses, optimize savings, and realign goals effectively.

Conclusion:

Throughout this project, a user-friendly personal finance assistant was developed to offer personalized guidance in plain language and aid users in understanding their income, expenses, and financial objectives. After receiving general inquiries or budget information, the assistant provides concise summaries, spending insights, and suggestions for planning and saving. Users are guided via features like tone analysis, budget summaries, and deeper spending insights by a straightforward web interface. Essentially, the system offers a comprehensive experience that allows users to engage in natural communication, get insightful financial feedback, and feel supported as they progress through their money management journey.

A number of difficulties surfaced during the process. A more straightforward conversational style was chosen over a complete dialogue flow because it was necessary to pay for a subscription in order to enable a specialized dialogue manager with predetermined themes. It took careful language to craft directions that would keep responses on financial topics without veering off topic. Clear prompts were required in order to minimize misunderstanding when handling a variety of user inputs, from well-structured data to informal or partial communications. Mechanisms to gently convey feedback when processing took longer or failed were necessary due to service latency or sporadic interruptions. Prompts and error messages had to be continuously improved in order to maintain the interface's intuitiveness while assuming that users would enter data in the proper format.

Despite these challenges, important knowledge was acquired. It was crucial to stress precise instructions for both input and output in order to make the advise seem approachable and reliable. Writing clear instructions for linguistic intelligence made it clear how important it is to consider the user's point of view when choosing phrases and examples that make the next steps easier to understand. The necessity of straightforward flows and informative signals in the event of faults was emphasized during the design of the navigation and feedback. A more seamless experience resulted from iterative testing of various approaches to presenting feedback or requesting data. Design, communications, and AI capabilities all came together to create a seamless outcome that users will value. Overall, trust was increased in the ability to provide helpful financial advice without overwhelming the user by fusing answer creation and natural language understanding.

Future enhancements can make the assistant even more beneficial. Deeper personalization would be achieved by including fuller, continuous conversations, where previous points are retained and follow-up questions come up over several sessions. Users can quickly view spending patterns or goal progress by incorporating visual components like charts or progress trackers. Reducing human entry may be possible by integrating with actual financial data sources (with express consent) to provide automatic updates and reminders. Long-term participation would be encouraged by introducing customized profiles, where preferences and previous suggestions are retained. Adding accessibility to messaging apps or mobile apps could improve accessibility in day-to-day activities. The assistant will be useful and relevant over time if user feedback is regularly gathered and recommendations are adjusted in response to changing needs.