

گزارش پروژه: مشاور هوشمند کسب‌وکار

پرهام طالبیان

۱۰ فروردین ۱۴۰۴

۱ مرور کلی پروژه

هدف این پروژه توسعه یک مشاور هوشمند کسب‌وکار با استفاده از تکنیک‌های مدرن هوش مصنوعی و مدل‌های زبانی بزرگ (LLM) است. این عامل هوشمند به افراد علاقه‌مند به راه‌اندازی کسب‌وکار کمک می‌کند تا مشاوره‌ای شخصی‌سازی‌شده، دقیق و قابل اجرا دریافت کنند. این مشاور از منابع مختلف داده شامل فایل‌های Word، PDF، تاریخچه گفتگو و جستجوهای وب بهره می‌برد و با ترکیب داده‌های تخصصی و عمومی، پشتیبانی عملیاتی و قابل اعتماد ارائه می‌دهد.

۲ اهداف پروژه

- فراهم‌سازی مشاوره کسب‌وکار با زبان طبیعی با استفاده از LLM.
- کمک به ایجاد و اصلاح برنامه کسب‌وکار.
- استفاده از منابع متنوع داده (اسناد، گفتگو، وب).
- پیگیری اقدامات و ارائه توصیه‌های شخصی‌سازی‌شده.

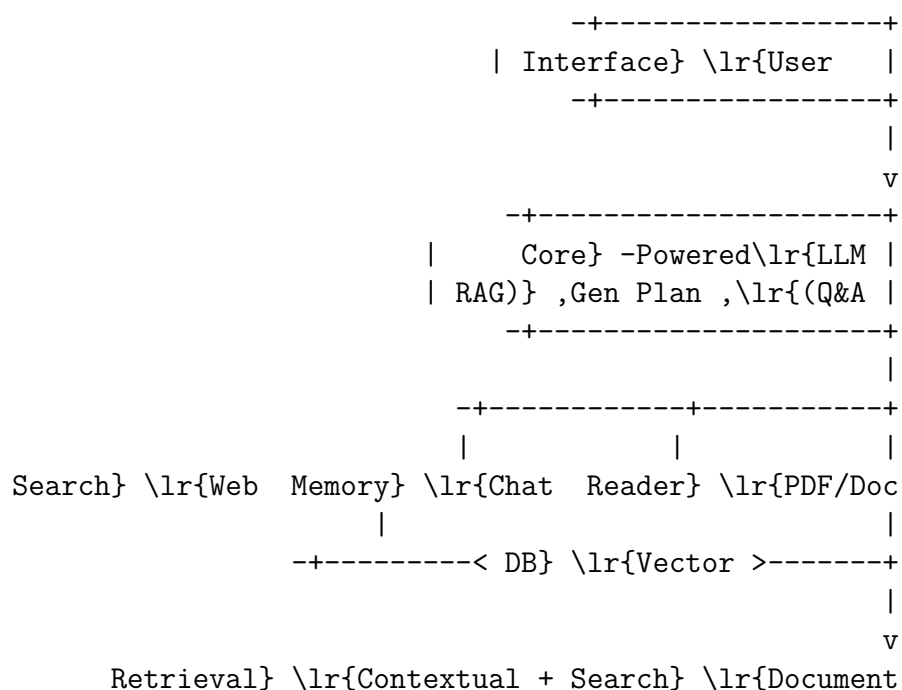
۳ ویژگی‌های کلیدی

- موتور پاسخگویی به سوالات
- سیستم خواندن اسناد (PDF، Word، تاریخچه گفتگو)
- یکپارچه‌سازی جستجوی وب
- تولید و پیگیری برنامه کسب‌وکار
- حافظه عامل و سیستم پیگیری

۴ تکنیک‌ها و معماری هوش مصنوعی

- یکپارچه‌سازی LLM مبتنی بر GPT یا مدل‌های منبع باز
- تولید تقویت‌شده با بازیابی (RAG)
- جستجوی معنایی با استفاده از پایگاه‌داده‌های برداری (مانند FAISS، Chroma)
- معماری چندعاملی (اختیاری)

۱.۴ نمای کلی معماری سیستم



۵ مسیر اجرای پروژه

۱.۵ فاز ۱: طراحی و برنامه‌ریزی

- تعریف دامنه و پرسونای کاربر
- شناسایی ویژگی‌های اصلی
- انتخاب ابزارها و فناوری‌ها

۲.۵ فاز ۲: راه‌اندازی محیط توسعه

- راه‌اندازی محیط پایتون و نصب وابستگی‌ها
- اتصال و تست اولیه LLM

۳.۵ فاز ۳: خواندن داده‌ها و بازیابی اطلاعات

- ساخت ماژول خواندن اسناد PDF و Word
- تولید بردارها و پیاده‌سازی جستجوی معنایی

۴.۵ فاز ۴: موتور پاسخگویی کسب‌وکار

- ساخت سیستم پاسخ‌دهی مبتنی بر بازیابی
- یکپارچه‌سازی حافظه گفتگو

۵.۵ فاز ۵: تولیدکننده برنامه کسب‌وکار

- طراحی قالب‌های ساختارمند برای پرامپت‌ها
- امکان ویرایش و ذخیره‌سازی مرحله‌ای برنامه

۶.۵ فاز ۶: پیگیری و مدیریت وظایف

- استخراج اقدامات از برنامه‌ها
- ارائه یادآوری و پیگیری پیشرفت

۷.۵ فاز ۷: رابط کاربری

- شروع با رابط ساده (Streamlit) یا (Flask)
- افزودن چت‌بات یا رابط گرافیکی پیشرفته در ادامه

۸.۵ فاز ۸: تست و بهینه‌سازی

- تست در حوزه‌های مختلف کسب‌وکار
- بهینه‌سازی پرامپت‌ها و کیفیت پاسخ‌ها

۹.۵ فاز ۹: استقرار

- استقرار در سرور ابری
- ذخیره جلسات و امکان ورود کاربران

۱۰.۵ فاز ۱۰: مستندسازی و ارائه

- تهیه راهنمای کاربر و ویدیو دمو
- مستندسازی کامل و ارائه پروژه

۶ ساختار نمونه پوشه پروژه

```
business_agent_project/  
    app/  
        main.py  
        llm_interface.py  
        document_reader.py
```

```

retrieval_engine.py
business_plan_generator.py
memory_manager.py
    data/
        uploaded_files/
            vector_db/
                prompts/
business_plan_template.txt
requirements.txt
README.md

```

۷ گزارش قسمت به قسمت پروژه

۱.۷ llminterface.py

```

business_plan_generator.py app • document_reader.py app • llm_interface.py X test_llm_interface.py • .env • test_en
app > llm_interface.py > ...
1 from together import Together
2 import os
3 from retrieval_engine import RetrievalEngine
4 # from memory_manager import MemoryManager
5
6 class LLMInterface:
7     def __init__(self, api_key=None, model_name="meta-llama/llama-3.3-70B-Instruct-Turbo"):
8         self.api_key = api_key or os.getenv("cf695d0e05061568bad485126184ca88a09340cd8540ec31deee936b83914c1a")
9         if not self.api_key:
10             raise ValueError("API Key تنظیم نشده است برای Together.")
11
12         self.model = model_name
13         self.client = Together(api_key=self.api_key)
14         self.retrieval_engine = RetrievalEngine()
15         # self.memory = MemoryManager()
16
17     def ask(self, prompt, user_id="default"):
18         # history = self.memory.get_context(user_id)
19         relevant_data = self.retrieval_engine.search(prompt)
20
21         combined_prompt = f"""
22             اطلاعات مرتبط: {relevant_data}
23             پرسش جدید: {prompt}
24             """
25
26         response = self.client.chat.completions.create(
27             model=self.model,
28             messages=[{"role": "user", "content": combined_prompt}],
29             ).choices[0].message.content.strip()
30
31         # self.memory.add_memory(user_id, f"User: {prompt}\nAI: {response}")
32         return response
33

```

۱.۱.۷ ساختار کلاس

۲.۱.۷ سازنده کلاس

متد `__init__` مقادیر اولیه را می‌سازد:

- مقدار `api_key` را دریافت می‌کند. در صورت عدم ارائه، مقدار آن را از متغیر محیطی دریافت می‌کند.
- مدل مورد استفاده را تنظیم کرده و یک نمونه از `Together` ایجاد می‌کند.
- نمونه‌ای از `RetrievalEngine` برای بازیابی اطلاعات مرتبط ایجاد می‌شود.

۳.۱.۷ متد ask

این متد برای ارسال پرسش و دریافت پاسخ از مدل به کار می‌رود:

- ورودی‌ها:

- prompt: سوال کاربر

- user_id: که دیگر استفاده نمی‌شود

- ابتدا از RetrievalEngine برای یافتن اطلاعات مرتبط استفاده می‌شود.
- پرسش همراه با اطلاعات بازیابی‌شده در یک قالب مشخص ترکیب شده و به مدل Llama ۳ ارسال می‌شود.
- پاسخ از API دریافت شده و مقدار نهایی برگشت داده می‌شود.

۴.۱.۷ حذف مدیریت حافظه

در نسخه قبلی این کلاس، یک شیء از نوع MemoryManager برای ذخیره تاریخچه مکالمات استفاده می‌شد، اما در این نسخه:

- مدیریت تاریخچه مکالمات حذف شده است.
- مکالمات گذشته دیگر در ورودی مدل لحاظ نمی‌شوند.

۲.۷ retrieval-engine.py

```
.env test_env_read.py retrieval_engine.py app X memory_manager.py app main.py
retrieval_engine.py > RetrievalEngine > retrieve
from sklearn.neighbors import NearestNeighbors
import numpy as np
import pickle

class RetrievalEngine:
    def __init__(self, n_neighbors=5, algorithm='ball_tree'):
        self.n_neighbors = n_neighbors
        self.algorithm = algorithm
        self.model = None
        self.data = None

    def fit(self, data):
        self.data = np.array(data)
        self.model = NearestNeighbors(n_neighbors=self.n_neighbors, algorithm=self.algorithm)
        self.model.fit(self.data)

    def retrieve(self, query_vector):
        query_vector = np.array(query_vector).reshape([1, -1])
        distances, indices = self.model.kneighbors(query_vector)
        return indices[0], distances[0]

    def save(self, filename="retrieval_model.pkl"):
        with open(filename, "wb") as f:
            pickle.dump((self.model, self.data), f)

    def load(self, filename="retrieval_model.pkl"):
        with open(filename, "rb") as f:
            self.model, self.data = pickle.load(f)

    def search(self, query):
        relevant_data = "این داده‌های مرتبط با پرسش شما هستند."
        return relevant_data
```

۱.۲.۷ ساختار کلاس

۲.۲.۷ سازنده کلاس

متد `init__` وظیفه مقداردهی اولیه را بر عهده دارد:

- مقدار `n_neighbors` تعیین می کند که چند همسایه نزدیک برای هر پرسش جستجو شوند.
- مقدار `algorithm` مشخص می کند که از چه الگوریتمی برای جستجو استفاده شود.
- مدل و داده ها در ابتدا مقدار `None` دارند و هنگام آموزش مقداردهی می شوند.

۳.۲.۷ متد `fit`

این متد مدل `Neighbors Nearest` را با داده های ورودی آموزش می دهد:

- داده های ورودی به یک آرایه `NumPy` تبدیل می شوند.
- مدل `NearestNeighbors` بر اساس مقدار `n_neighbors` و الگوریتم تعیین شده مقداردهی اولیه می شود.
- مدل با داده های ورودی آموزش می بیند.

۴.۲.۷ متد `retrieve`

این متد برای دریافت نزدیک ترین همسایه ها به یک بردار پرسش استفاده می شود:

- ابتدا بردار ورودی به فرمت مناسب تبدیل شده و ابعاد آن تنظیم می شود.
- با استفاده از متد `kneighbors`، نزدیک ترین داده ها و فاصله آن ها از بردار ورودی محاسبه می شوند.
- لیست شاخص های داده های نزدیک و فاصله آن ها بازگردانده می شود.

۵.۲.۷ متدهای ذخیره و بازیابی مدل

- متد `save`، مدل و داده های آن را در یک فایل `pickle` ذخیره می کند.
- متد `load`، فایل ذخیره شده را بارگذاری کرده و مدل را به حالت قبل بازمی گرداند.

۶.۲.۷ متد `search`

این متد برای جستجوی اطلاعات مرتبط با یک پرسش طراحی شده است. در نسخه فعلی، خروجی آن یک پیام ثابت است:

”این داده های مرتبط با پرسش شما هستند.”

۳.۷ `gui-tkinter.py`

```

class LLMInterface:
    def __init__(self, llm):
        self.llm = llm

    def ask_llm(self, prompt):
        response = self.llm.generate(prompt)
        return response

    def ask_llm_with_history(self, prompt, history):
        response = self.llm.generate(prompt + "\n" + history)
        return response

    def ask_llm_with_history_and_settings(self, prompt, history, settings):
        response = self.llm.generate(prompt + "\n" + history + "\n" + settings)
        return response

    def ask_llm_with_history_and_settings_and_release_data(self, prompt, history, settings, release_data):
        response = self.llm.generate(prompt + "\n" + history + "\n" + settings + "\n" + release_data)
        return response

```

شکل ۳: تصویر سوم

```

class LLMInterface:
    def __init__(self, llm):
        self.llm = llm

    def ask_llm(self, prompt):
        response = self.llm.generate(prompt)
        return response

    def ask_llm_with_history(self, prompt, history):
        response = self.llm.generate(prompt + "\n" + history)
        return response

    def ask_llm_with_history_and_settings(self, prompt, history, settings):
        response = self.llm.generate(prompt + "\n" + history + "\n" + settings)
        return response

    def ask_llm_with_history_and_settings_and_release_data(self, prompt, history, settings, release_data):
        response = self.llm.generate(prompt + "\n" + history + "\n" + settings + "\n" + release_data)
        return response

```

شکل ۲: تصویر دوم

```

class LLMInterface:
    def __init__(self, llm):
        self.llm = llm

    def ask_llm(self, prompt):
        response = self.llm.generate(prompt)
        return response

    def ask_llm_with_history(self, prompt, history):
        response = self.llm.generate(prompt + "\n" + history)
        return response

    def ask_llm_with_history_and_settings(self, prompt, history, settings):
        response = self.llm.generate(prompt + "\n" + history + "\n" + settings)
        return response

    def ask_llm_with_history_and_settings_and_release_data(self, prompt, history, settings, release_data):
        response = self.llm.generate(prompt + "\n" + history + "\n" + settings + "\n" + release_data)
        return response

```

شکل ۱: تصویر اول

- مقداردهی اولیه API Key برای ارتباط با مدل LLM.
- ایجاد اشیای مرتبط با کلاس‌های LLMInterface، RetrievalEngine و BusinessPlanGenerator.
- تعریف و نمایش ویجت‌های مختلف مانند فیلد ورودی، دکمه‌ها و ScrolledText برای نمایش پاسخ‌ها.

۱.۳.۷ متد save_settings

این متد مقدار جدید Key API را از ورودی دریافت کرده و ذخیره می‌کند. در صورتی که مقدار جدیدی وارد نشده باشد، پیام هشدار نمایش داده می‌شود.

۲.۳.۷ متد ask_llm

این متد مسئول ارسال پرسش کاربر به مدل زبانی و دریافت پاسخ است:

- بررسی می‌کند که کاربر سوالی وارد کرده باشد.
- ارسال پرسش به LLMInterface و دریافت پاسخ.
- نمایش پاسخ در قسمت مربوطه.
- جستجوی اطلاعات مرتبط با استفاده از RetrievalEngine و نمایش آن.
- تولید طرح کسب‌وکار بر اساس پاسخ دریافتی و نمایش آن.

۳.۳.۷ متد load_file

این متد امکان بارگذاری محتوای فایل را فراهم می‌کند:

- بررسی می‌کند که مسیر فایل وارد شده باشد.
- در صورت ورود مسیر، بررسی می‌شود که فایل از نوع Word PDF یا متن ساده باشد.
- خواندن محتوای فایل با استفاده از DocumentReader و نمایش آن در بخش مربوطه.

۴.۳.۷ متدهای ذخیره خروجی

- download_output: ذخیره پاسخ‌ها و طرح کسب‌وکار در قالب یک فایل متنی.
- download_pdf: تولید خروجی در قالب فایل PDF با استفاده از ReportLab.

```

business_plan_generator.py app • document_reader.py app • llm_interface.py • test_llm_interface.py • .env • test
app > gui_streamlit.py > ...
1 import streamlit as st
2 from llm_interface import LLMInterface
3 from retrieval_engine import RetrievalEngine
4 from business_plan_generator import BusinessPlanGenerator
5 from document_reader import DocumentReader
6 from memory_manager import MemoryManager
7
8 API_KEY = "cf695d0e05061568bad485126184ca88a09340cd8540ec31dee936b83914c1a"
9 llm = LLMInterface(api_key=API_KEY)
10 retrieval_engine = RetrievalEngine()
11 business_plan_generator = BusinessPlanGenerator()
12 doc_reader = DocumentReader()
13 memory_manager = MemoryManager()
14
15 st.title("مشاور هوشمند کسب‌وکار")
16
17 question = st.text_input("سوال خود را وارد کنید")
18
19 if st.button("ارسال"):
20     if question.strip():
21         with st.spinner("در حال پردازش..."):
22             response = llm.ask(question)
23             st.success("پاسخ دریافت شد!")
24             st.write("پاسخ:")
25             st.write(response)
26
27             relevant_data = retrieval_engine.search(question)
28             st.write("اطلاعات مرتبط:")
29             st.write(relevant_data)
30
31             memory_manager.save_conversation(question, response)
32
33             business_plan = business_plan_generator.generate_business_plan(response)
34             st.write("طرح کسب‌وکار:")
35             st.write(business_plan)
36
37     else:
38         st.warning("لطفاً یک سوال وارد کنید")
39
40 uploaded_file = st.file_uploader("بارگذاری فایل (PDF/Word):", type=["pdf", "docx", "txt"])
41 if uploaded_file is not None:
42     if uploaded_file.type == "application/pdf":
43         content = doc_reader.read_pdf(uploaded_file)
44     elif uploaded_file.type == "application/vnd.openxmlformats-officedocument.wordprocessingml.document":
45         content = doc_reader.read_docx(uploaded_file)
46     else:
47         content = doc_reader.read_text_file(uploaded_file)
48
49     st.write("محتوای فایل بارگذاری شده:")
50     st.write(content)
51

```

۱.۴.۷ تعریف متغیرها و کلاس‌ها

در ابتدای برنامه، کلاس‌های زیر مقداردهی اولیه می‌شوند:

- LLMInterface: برای ارتباط با مدل زبانی.
- RetrievalEngine: برای جستجوی اطلاعات مرتبط.
- BusinessPlanGenerator: برای تولید طرح کسب‌وکار.
- DocumentReader: برای خواندن فایل‌های ورودی.
- MemoryManager: برای ذخیره‌سازی مکالمات کاربر.

۲.۴.۷ نمایش رابط کاربری

رابط کاربری با استفاده از Streamlit پیاده‌سازی شده است:

- نمایش عنوان برنامه با دستور `st.title`.
- دریافت پرسش کاربر از طریق `st.text_input`.
- دکمه ارسال (`st.button`) برای دریافت پاسخ از مدل.

۳.۴.۷ متد دریافت پاسخ

پس از فشردن دکمه ارسال، مراحل زیر اجرا می‌شود:

- بررسی می‌شود که کاربر سوالی وارد کرده باشد.
- در صورت ورود سوال، نمایش پیام در حال پردازش (st.spinner).
- ارسال پرسش به مدل زبانی و دریافت پاسخ.
- نمایش پاسخ به کاربر.
- جستجوی اطلاعات مرتبط و نمایش آن.
- ذخیره مکالمه در MemoryManager.
- تولید و نمایش طرح کسب‌وکار.

۴.۴.۷ بارگذاری فایل

امکان بارگذاری فایل از طریق st.file_uploader وجود دارد:

- بررسی نوع فایل (PDF، Word یا متن ساده).
- پردازش محتوای فایل با استفاده از DocumentReader.
- نمایش محتوای فایل در خروجی.

Documnet-reader.py ۵.۷

```
business_plan_generator.py app • document_reader.py app • llm_interface.py
document_reader.py > DocumentReader > read_pdf
import PyPDF2
import docx

class DocumentReader:
    def __init__(self):
        pass

    def read_pdf(self, file_path):

        with open(file_path, "rb") as file:
            reader = PyPDF2.PdfReader(file)
            text = ""
            for page in reader.pages:
                text += page.extract_text()
            return text

    def read_docx(self, file_path):

        doc = docx.Document(file_path)
        text = ""
        for para in doc.paragraphs:
            text += para.text + "\n"
        return text

    def read_text_file(self, file_path):

        with open(file_path, "r", encoding="utf-8") as file:
            return file.read()
```

۱.۵.۷ متد read_pdf

این متد برای خواندن محتوای فایل‌های PDF استفاده می‌شود:

- فایل PDF در حالت خواندن باینری باز می‌شود.
- از کلاس PyPDF2.PdfReader برای پردازش فایل استفاده می‌شود.
- متن تمامی صفحات استخراج و در متغیری ذخیره می‌شود.

۲.۵.۷ متد read_docx

این متد برای پردازش فایل‌های Word با فرمت DOCX طراحی شده است:

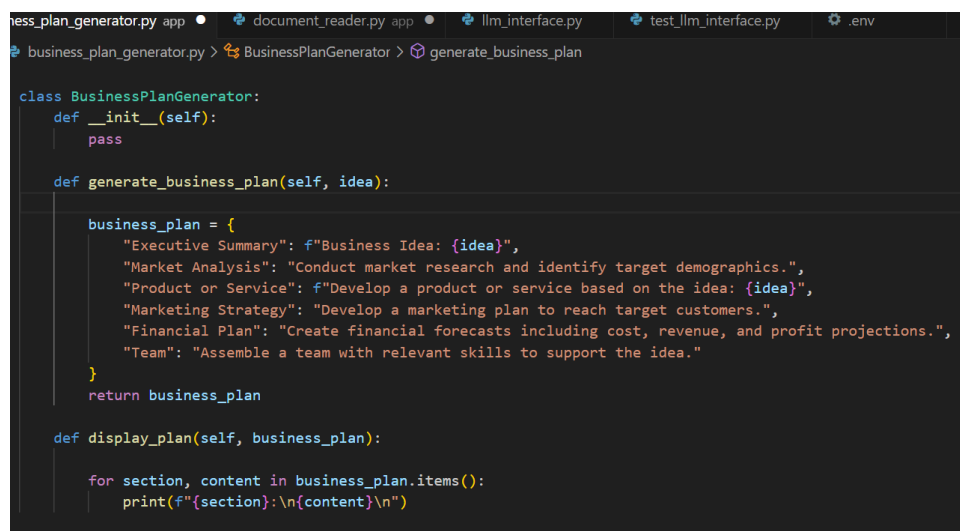
- فایل با استفاده از docx.Document باز می‌شود.
- تمامی پاراگراف‌ها خوانده شده و در قالب یک رشته متنی ترکیب می‌شوند.

۳.۵.۷ متد read_text_file

این متد برای خواندن فایل‌های متنی ساده استفاده می‌شود:

- فایل در حالت خواندن متنی با رمزگذاری UTF-۸ باز می‌شود.
- محتوا به صورت کامل خوانده شده و بازگردانده می‌شود.

۶.۷ business-plan-manager.py



```
class BusinessPlanGenerator:
    def __init__(self):
        pass

    def generate_business_plan(self, idea):

        business_plan = {
            "Executive Summary": f"Business Idea: {idea}",
            "Market Analysis": "Conduct market research and identify target demographics.",
            "Product or Service": f"Develop a product or service based on the idea: {idea}",
            "Marketing Strategy": "Develop a marketing plan to reach target customers.",
            "Financial Plan": "Create financial forecasts including cost, revenue, and profit projections.",
            "Team": "Assemble a team with relevant skills to support the idea."
        }

        return business_plan

    def display_plan(self, business_plan):

        for section, content in business_plan.items():
            print(f"{section}:\n{content}\n")
```

۱.۶.۷ تعریف کلاس

BusinessPlanGenerator شامل دو متد کلیدی است:

۲.۶.۷ متد generate_business_plan

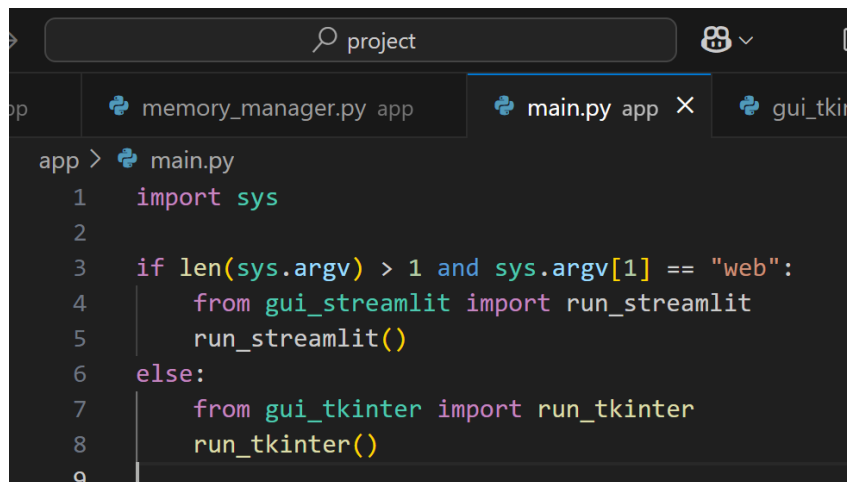
این متد وظیفه‌ی تولید یک طرح کسب‌وکار بر اساس ایده‌ی ورودی را دارد. این طرح شامل بخش‌های زیر است:

- خلاصه اجرایی: معرفی ایده‌ی کسب‌وکار
- تحلیل بازار: تحقیق در مورد بازار و شناسایی مشتریان هدف
- محصول یا خدمات: توسعه‌ی یک محصول یا خدمت بر اساس ایده‌ی ارائه‌شده
- استراتژی بازاریابی: برنامه‌ریزی برای دسترسی به مشتریان هدف
- برنامه‌ی مالی: پیش‌بینی‌های مالی شامل هزینه‌ها، درآمد و سود
- تیم: گردآوری تیمی با مهارت‌های مرتبط برای حمایت از ایده

۳.۶.۷ متد display_plan

این متد وظیفه‌ی نمایش طرح کسب‌وکار تولید شده را بر عهده دارد. این نمایش شامل چاپ تمام بخش‌های طرح به همراه محتوای مربوط به آن‌ها در خروجی است.

۷.۷ main.py



```
project
memory_manager.py app
main.py app X
gui_tkin

app > main.py
1 import sys
2
3 if len(sys.argv) > 1 and sys.argv[1] == "web":
4     from gui_streamlit import run_streamlit
5     run_streamlit()
6 else:
7     from gui_tkinter import run_tkinter
8     run_tkinter()
9
```

۱.۷.۷ ساختار کد

کد با استفاده از ماژول sys بررسی می‌کند که آیا آرگومان "web" به عنوان ورودی خط فرمان ارسال شده است یا خیر.

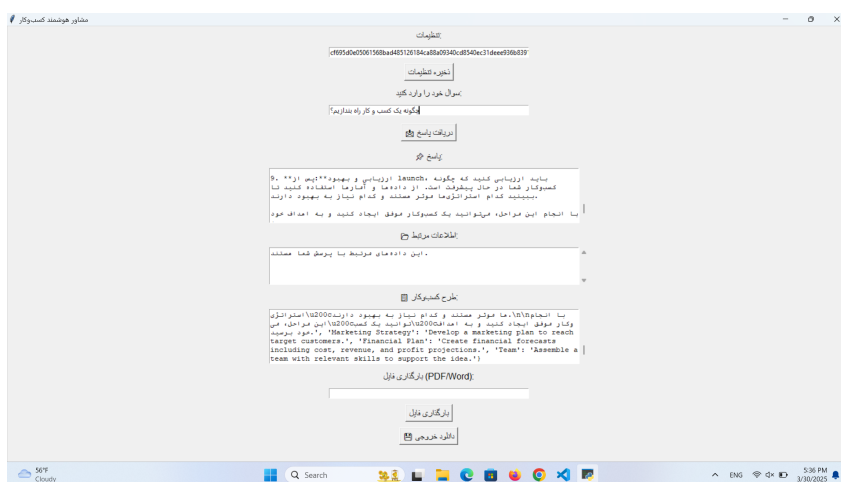
- اگر مقدار ورودی برابر "web" باشد، ماژول gui_streamlit بارگذاری شده و تابع run_streamlit() اجرا می‌شود.
- در غیر این صورت، ماژول gui_tkinter بارگذاری شده و تابع run_tkinter() اجرا می‌شود.

۲.۷.۷ مزایا

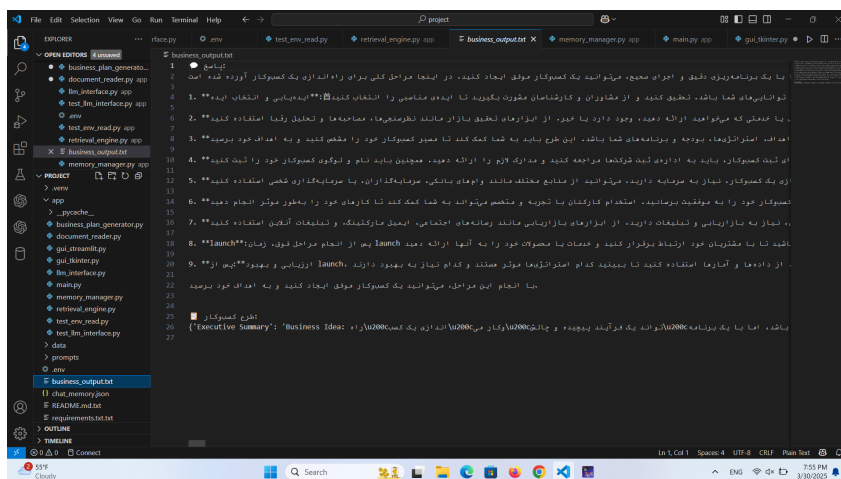
- قابلیت استفاده از دو رابط کاربری مختلف، بسته به نیاز کاربر
- امکان اجرای برنامه به صورت تحت وب با Streamlit
- قابلیت اجرای برنامه به صورت آفلاین با استفاده از Tkinter

۸ ورودی و خروجی

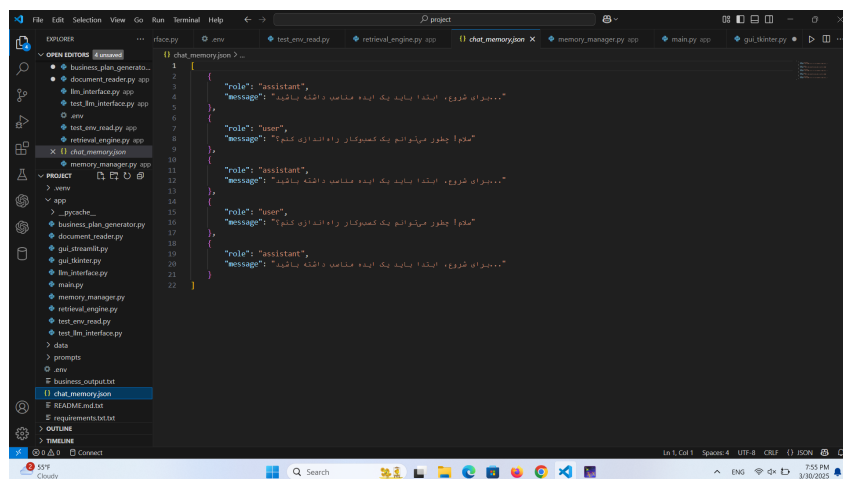
در این قسمت عکس هایی از ورودی و خروجی به همراه فایل ذخیره شده جواب و سوالات ذخیره شده



شکل ۴: نمای کلی برنامه



شکل ۵: فایل خروجی جواب برای دانلود کردن آن



شکل ۶: فایل تاریخچه چت ها