Katowice, 03.02.2026 r. MIP10

Projekt Programowanie Użytkowe

Gemini Stock Chatbot

Wojciech Kubowicz

Mikołaj Manowski

1. Opis i etapy działania aplikacji

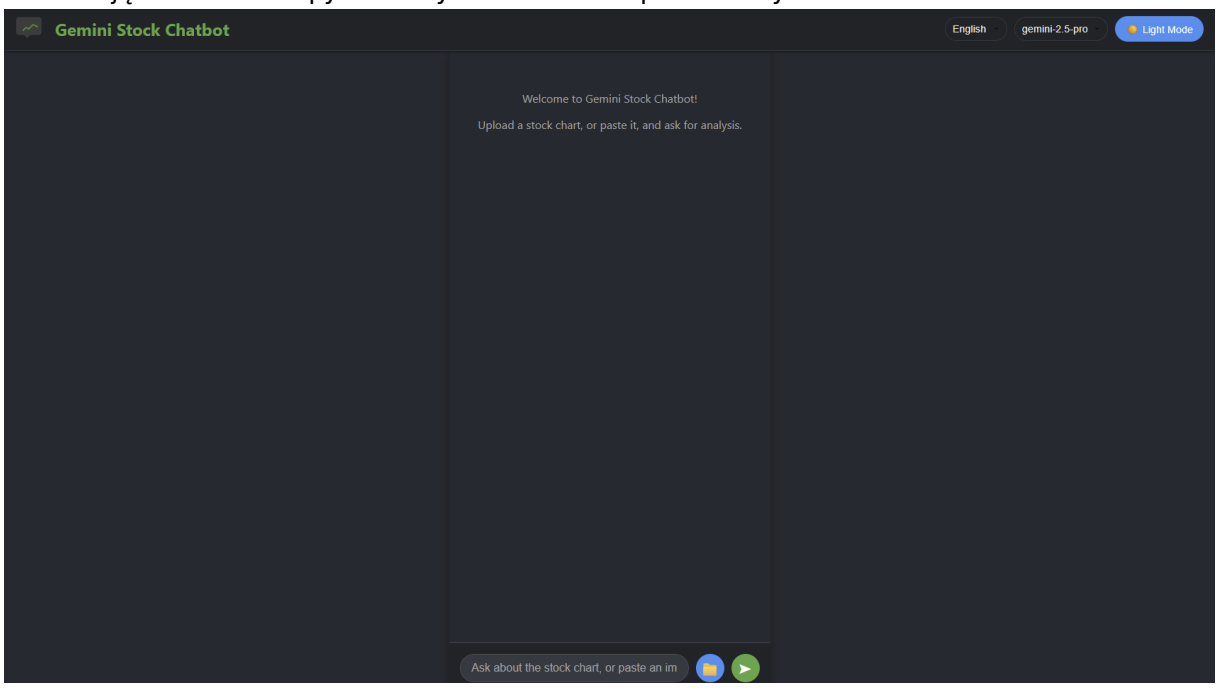
Aplikacja została zaprojektowana jako interaktywne narzędzie finansowe, łączące analizę danych giełdowych z możliwościami generatywnej sztucznej inteligencji. Proces działania programu można podzielić na cztery kluczowe etapy:

1.1. Inicjalizacja środowiska i konfiguracja

W pierwszej fazie system ładuje niezbędne zmienne środowiskowe, w szczególności klucz API Google, co jest krytyczne dla bezpieczeństwa aplikacji. Następuje konfiguracja klienta modelu Gemini (google.generativeai), co pozwala na nawiązanie połączenia z usługą chmurową Google.

1.2. Warstwa prezentacji (Frontend)

Interfejs użytkownika został zbudowany w oparciu o bibliotekę **React**. Umożliwia on intuicyjną interakcję poprzez panel boczny lub centralne pole tekstowe, gdzie użytkownik wprowadza symbol giełdowy (tzw. Ticker, np. NVDA, AAPL). System renderuje historię czatu, oddzielając wizualnie zapytania użytkownika od odpowiedzi systemu.



Zdj. 1.2. Wygląd głównego interfejsu aplikacji

1.3. Pobieranie i przetwarzanie danych (Backend)

Po wprowadzeniu zapytania, skrypt wykorzystuje bibliotekę **yfinance** do pobrania rzeczywistych danych rynkowych. Proces ten obejmuje:

- Pobranie historycznych cen zamknięcia.
- Uzyskanie podstawowych informacji fundamentalnych o spółce.
- Konwersję surowych danych numerycznych na ustrukturyzowany format tekstowy, zrozumiały dla modelu językowego.

1.4. Generowanie analizy (Inference)

Przygotowane dane finansowe są łączone z odpowiednio skonstruowanym poleceniem (promptem) i przesyłane do modelu Gemini Pro. Sztuczna inteligencja przetwarza kontekst numeryczny i generuje opisową analizę, która jest następnie wyświetlana użytkownikowi w czasie rzeczywistym.

```python
@app.post("/ingest")
async def ingest_knowledge(
    text: str = Form(...),
    x_gemini_api_key: str = Header(...)
):
    """Dodaje tekst do bazy wiedzy RAG."""
    try:
        client = genai.Client(api_key=x_gemini_api_key)
        # Generowanie embeddingu dla tekstu
        resp = await asyncio.to_thread(
            client.models.embed_content,
            model="text-embedding-004",
            contents=text
        )

        embedding = resp.embeddings[0].values

        rag_collection.add(
            documents=[text],
            embeddings=[embedding],
            ids=[str(datetime.now().timestamp())]
        )
        return {"status": "ok", "message": "Wiedza dodana do bazy."}
    except Exception as e:
        raise HTTPException(status_code=500, detail=str(e))
```

Zdj. 1.4. Funkcja wykonująca embeding danych

2. Zidentyfikowane problemy i wyzwania techniczne

Podczas analizy kodu źródłowego zidentyfikowano następujące obszary problemowe, które miały wpływ na proces wytwórczy oraz stabilność aplikacji

2.1. Zarządzanie stanem sesji (Session State)

Specyfika biblioteki Streamlit powoduje, że przy każdej interakcji z użytkownikiem cały skrypt jest przeładowywany. Głównym wyzwaniem było zachowanie historii czatu między kolejnymi zapytaniami, co rozwiązano poprzez implementację mechanizmu st.session_state.

2.2. Obsługa błędów zewnętrznych API

Aplikacja jest uzależniona od dwóch zewnętrznych źródeł danych (Yahoo Finance i Google API). Problemem jest ryzyko przekroczenia limitów zapytań (Rate Limits) lub czasowa niedostępność serwerów, co wymagało implementacji bloków zabezpieczających try-except, aby zapobiec awarii programu w przypadku błędu połączenia.

2.3. Walidacja danych wejściowych

Użytkownik może wprowadzić niepoprawny symbol giełdowy. Wyzwaniem było stworzenie mechanizmu, który weryfikuje istnienie spółki przed wysłaniem zapytania do AI, aby uniknąć generowania "halucynacji" na podstawie pustych danych.

3. Harmonogram realizacji projektu "Gemini Stock Chatbot"

2.4. Inicjalizacja projektu i konfiguracja środowiska – 20.10

- Określenie założeń projektowych: stworzenie interaktywnego narzędzia finansowego opartego na AI,
- Inicjalizacja środowiska (pkt 1.1): Instalacja bibliotek, konfiguracja zmiennych środowiskowych,
- Uzyskanie i zabezpieczenie klucza API Google oraz konfiguracja klienta modelu google.generativeai,
- Przygotowanie repozytorium i struktury plików pod projekt.

2.5. Implementacja warstwy Backend i pobierania danych – 13.01

- Pobieranie i przetwarzanie danych (pkt 1.3): Integracja biblioteki yfinance,
- Opracowanie skryptów pobierających historyczne ceny zamknięcia oraz informacje fundamentalne o spółkach,
- Stworzenie mechanizmu konwersji surowych danych numerycznych na format tekstowy zrozumiały dla modelu LLM,
- Wstępne testy pobierania danych dla popularnych tickerów (np. NVDA, AAPL).

2.6. Budowa interfejsu (Frontend) i integracja z AI – 20.01

- Warstwa prezentacji (pkt 1.2): Budowa interfejsu użytkownika (panel boczny, pole tekstowe, renderowanie czatu),
- Generowanie analizy (pkt 1.4): Łączenie danych finansowych z promptem i przesyłanie ich do modelu Gemini Pro,
- Implementacja wyświetlania odpowiedzi generowanych przez AI w czasie rzeczywistym,
- Wizualne oddzielenie zapytań użytkownika od odpowiedzi systemu.

2.7. Rozwiązywanie problemów technicznych i optymalizacja – 27.01

- Zarządzanie stanem sesji (pkt 2.1): Implementacja st.session_state w celu zachowania historii czatu przy przeładowaniu skryptu,
- Obsługa błędów (pkt 2.2): Dodanie bloków try-except dla obsługi limitów zapytań i błędów połączenia z zewnętrznymi API,
- Walidacja danych (pkt 2.3): Wdrożenie mechanizmu weryfikacji symboli giełdowych przed wysłaniem zapytania, aby uniknąć "halucynacji" modelu.

2.8. Prezentacja projektu i jego oddanie – 03.02

- Finalny przegląd kodu i weryfikacja spełnienia założeń MVP (Minimum Viable Product),

- Przygotowanie wniosków końcowych i podsumowania (skalowalność, automatyzacja analizy),
- Prezentacja działającej aplikacji: Demonstracja procesu analizy giełdowej w czasie rzeczywistym,
- Złożenie sprawozdania i oddanie projektu.

3. Podsumowanie

Projekt **Gemini Stock Chatbot** stanowi funkcjonalny przykład integracji klasycznej analityki danych z nowoczesnymi modelami językowymi (LLM).

Kluczowe wnioski:

1. Zastosowanie architektury opartej na Pythonie i Streamlit pozwoliło na szybkie prototypowanie (Rapid Application Development) i stworzenie działającego MVP (Minimum Viable Product),
2. Aplikacja skutecznie automatyzuje proces wstępnej analizy giełdowej, zastępując ręczne przeglądanie tabel syntezą generowaną przez AI,
3. Rozwiązanie jest wysoce skalowalne – kod źródłowy pozwala na łatwe dodawanie nowych wskaźników finansowych bez konieczności przebudowy całego silnika.

Podsumowując, aplikacja spełnia założenia projektowe, dostarczając użytkownikowi przystępnych informacji rynkowych, przy zachowaniu prostoty obsługi i przejrzystości interfejsu.