Mini-agent: citire Gmail si planificare Google Calendar (proiect laborator)

Wolff Andreas-Erhard (Informatica, anul 3, grupa 3)

1. Descriere generala

Agentul asculta mailbox-ul (polling la un interval de timp) si pentru fiecare e-mail nou ruleaza un model LLM (Gemini Lite) pentru a extrage structuri de tip calendar (Pydantic Structured Output). Daca LLM returneaza cel putin un `event`, scriptul construieste un obiect de tip eveniment Google Calendar si il insereaza in calendarul **primary**.

Componente principale:

- `app.py` - orchestratorul: autentificare OAuth, pornire polling, apel catre LLM si inserare eveniment in Calendar.
- `gmail.py` - functii pentru Gmail API: autentificare, citire mesaj, parsing body (strip HTML), mecanism de polling.
- `structure.py` - modele Pydantic care definesc schema JSON acceptata de LLM si folosita intern.

2. Fluxul de executie

1. `main()` din `app.py` verifica/realizeaza autentificarea Google (foloseste `credentials.json` si produce `token.json`).
2. Se initiaza obiectele serviciilor Google: Gmail (`service`) si Calendar (`cal`).
3. Se porneste generatorul `gmail.start_polling(service)` - acesta yield-uieste detalii pentru fiecare e-mail nou detectat.
4. Pentru fiecare e-mail nou: se trimit instructiunile de sistem + continutul complet al e-mail-ului catre modelul Gemini folosind `client.models.generate_content(...)` cu `response_mime_type='application/json'` si `response_schema=structure.ExtractedCalendarInfo`.
5. Raspunsul LLM este mapat la modelul Pydantic `ExtractedCalendarInfo`. Daca `events` nu este gol: se ia primul event, se converteste `date_time` in `datetime` Python, se calculeaza durata implicita (1h) si se construieste payload-ul pentru Calendar API.
6. Se insereaza evenimentul cu `cal.events().insert(calendarId='primary', body=event).execute()`.

3. Structura datelor (Pydantic)

`CalendarEvent` (acasa in `structure.py`) contine:

- `title` (str) - titlu/subject eveniment
- `date_time` (str) - data/ora in ISO 8601 preferabil (ex: `2025-11-25T15:00:00Z`); daca e ambiguu, poate ramane text uman

- `timezone` (str) - fusul orar; daca lipseste, agentul presupune GMT+2 (conform instructiunilor din model)
- `location` (Optional[str]) - locatie fizica sau link de video conferinta
- `summary` (Optional[str]) - descriere scurta
- `attendees` (list[str]) - lista de emailuri/nume ale participantilor

`ExtractedCalendarInfo` contine un `events` (lista posibil goala). Agentul **creeaza eveniment doar daca** `len(events) > 0`.

4. Autentificare & permisiuni

- Fisiere necesare: `credentials.json` (Google OAuth client), variabila de mediu `GEMINI_API_KEY` in `.env`.
- Scopes folosite in `gmail.py`:
  - `https://www.googleapis.com/auth/gmail.readonly` - citire e-mailuri
  - `https://www.googleapis.com/auth/calendar` - creare evenimente
- `token.json` este creat automat dupa prima autentificare. Daca se schimba scope-urile, stergem `token.json` si ne reautentificam.

5. Comportament important

- Agentul este **idempotent la nivel de polling**: foloseste `LAST_PROCESSED_MAIL_ID` intern pentru a nu reprocesa acelasi mail.
- Data/ora extrase de LLM trebuie sa fie in format ISO; `app.py` transforma `Z` in `+00:00` pentru `datetime.fromisoformat(...)`.
- Durata evenimentului este implicita 1 ora (se poate ajusta calculul). Daca LLM extrage explicit `end_time`, ar trebui modificata logica.
- Daca evenimentul are `attendees`, acesta este transmis direct catre Calendar API.

6. Functii cheie din cod

- `gmail.get_message_body(payload)` - extrage recursiv `text/plain` si face fallback pe `text/html`. Normalieaza padding base64 si decodeaza, apoi curata HTML cu BeautifulSoup.
- `gmail.get_latest_email(service)` - obtine ultimul mesaj din INBOX, returneaza id + detalii structurate (subject, from, date, body).
- `gmail.start_polling(service)` - loop infinit cu `time.sleep(POLLING_INTERVAL_SECONDS)`; `yield` cand gaseste e-mail nou.
- In `app.py`: apelul `client.models.generate_content(...)` cu `response_schema=structure.ExtractedCalendarInfo` spune modelului sa raspunda strict in JSON valid pentru Pydantic.

7. Limitari cunoscute si riscuri

- Detectarea evenimentelor se bazeaza pe LLM: exista fals pozitive/negative, daca extindem aplicatia, prompt-engineering-ul ar putea reduce erori.
- Data/ora ambigue sau mentionari multiple pot conduce la extrageri gresite.

- Polling constant (interval implicit 10-30s) poate consuma resurse si atinge rate limits; recomandat ar fi sa folosim webhooks (Push Notifications via Gmail API, Watch API) pentru productie sau un proiect mai "serios".

8. Testare & rulare

- Verifica fisierul `README.md` din GitHub Repo sau arhiva proiectului.