---

## Manuale dello Script: Blueprint di Automazione Gemini-Gmail-Calendar

### 1. Obiettivo del Progetto

Creare uno script Python **totalmente automatico** (ControllaEmailCreaEvento.py) che viene eseguito tre volte al giorno. Il suo compito è:

1. Connettersi all'account Gmail **x31mint@gmail.com**.
2. Cercare e leggere tutte le email **non lette**.
3. Utilizzare la CLI di Gemini per analizzare il contenuto di ogni email e identificare se contiene un evento, un appuntamento o una scadenza con una data e (opzionalmente) un orario.
4. Se viene trovato un evento, usare Gemini per generare un titolo e una descrizione appropriati.
5. Creare un evento nel **Google Calendar** associato all'account.
6. Contrassegnare l'email come **letta** per evitare di elaborarla nuovamente.

A differenza del progetto YouTube, questo flusso è progettato per operare senza alcuna supervisione o approvazione manuale.

### 2. Configurazione dell'Ambiente

- **Cartella Principale del Progetto:**
  C:\Calendar_Automation
- **File Presenti nella Cartella:**
  - ControllaEmailCreaEvento.py: L'unico script automatico. Si occupa di tutta la logica: autenticazione, lettura email, chiamata a Gemini, creazione dell'evento su Calendar e aggiornamento dello stato dell'email.
  - client_secret.json: **File di Credenziali NUOVO**. È il file segreto generato da Google Cloud Console specificamente per l'account x31mint@gmail.com e per questo progetto. Autorizza lo script a chiedere i permessi.
  - token.json: **File di Autorizzazione**. Verrà creato automaticamente dallo script al primo avvio, dopo aver concesso i permessi all'account x31mint@gmail.com. Contiene l'autorizzazione per accedere a Gmail e Calendar. Non cancellarlo.
  - .env: **File delle Variabili d'Ambiente**. Creato manualmente in questa cartella, conterrà unicamente la chiave API di Gemini.
-

### 3. Dettagli Tecnici, Regole e Metodi da Usare

- **Account Gmail di Riferimento:**
  - Indirizzo: x31mint@gmail.com
  - Dove: Questo è l'account che autorizzerà l'applicazione e su cui lo script opererà.
-
- **Calendario Google di Destinazione:**

    - ID: primary (il calendario principale dell'account autorizzato).
    - Dove: Specificato all'interno dello script ControllaEmailCreaEvento.py durante la chiamata all'API di Calendar.
-
- **Modello AI Utilizzato:**
    - Modello: gemini-1.5-pro (chiamato esclusivamente tramite la CLI gemini).
    - Dove: La chiamata viene eseguita dallo script Python.
-

---

### 3.1 Regole e Metodi Chiave dello Script

Lo script ControllaEmailCreaEvento.py dovrà seguire questa logica e utilizzare i seguenti metodi/librerie Python.

- **Librerie Principali da Importare:**
    1. os e subprocess: Per interagire con il sistema operativo e lanciare la CLI di Gemini.
    2. json: Per interpretare (parsare) l'output JSON restituito da Gemini.
    3. google.oauth2.credentials, google_auth_oauthlib.flow, googleapiclient.discovery: Per l'autenticazione e l'interazione con le API di Google.
    4. dotenv: Per caricare la chiave API dal file .env.
-
- **Fase 1: Autenticazione Google (Metodo: Flow e build)**
    1. **Regola:** Lo script deve prima cercare un file token.json valido.
    2. **Se non esiste:** Deve usare il file client_secret.json per avviare un flusso di autenticazione (InstalledAppFlow). Questo chiederà all'utente di accedere e autorizzare gli ambiti (scopes) per Gmail (gmail.modify) e Calendar (calendar.events).
    3. **Regola:** Le credenziali ottenute devono essere salvate in un nuovo file token.json per le esecuzioni future.
    4. **Metodo:** Creare gli oggetti "servizio" per Gmail e Calendar usando la funzione build('gmail', 'v1', credentials=creds) e build('calendar', 'v3', credentials=creds).
-
- **Fase 2: Lettura delle Email (Metodo: users().messages().list() e get())**
    1. **Regola:** La ricerca deve essere mirata solo alle email non lette.
    2. **Metodo:** Usare service.users().messages().list(userId='me', q='is:unread').execute() per ottenere la lista degli ID dei messaggi.
    3. **Regola:** Per ogni ID, il corpo del messaggio deve essere recuperato e decodificato (spesso è in formato base64).
    4. **Metodo:** Usare service.users().messages().get(userId='me', id=msg_id).execute() per ottenere il contenuto completo dell'email.
-
- **Fase 3: Chiamata alla CLI di Gemini (Metodo: subprocess.run)**

1. **Regola:** Per ogni email, deve essere costruito un prompt preciso che chieda a Gemini di restituire una risposta **esclusivamente in formato JSON**. Questo garantisce che l'output sia prevedibile e facile da elaborare.

**Prompt Esempio da usare nel codice:**
code Python
downloadcontent_copyexpand_less
```
    prompt = f"""
Analizza il testo di questa email e determina se contiene un evento con una data. Rispondi SOLO in formato JSON con i seguenti campi:
- "creare_evento": "si" o "no".
- "titolo": Un titolo breve e descrittivo per l'evento.
- "data": La data in formato AAAA-MM-GG. Se non trovata, metti "null".
- "ora_inizio": L'orario di inizio in formato 24 ore HH:MM. Se non trovato, metti "null".
Testo dell'email:
---
{corpo_email}
---
"""
```

2.
3. **Metodo:** Eseguire la CLI usando subprocess.run(['gemini', prompt], capture_output=True, text=True), recuperando l'output.

- **Fase 4: Creazione Evento su Calendar (Metodo: events().insert())**
    1. **Regola:** Analizzare il JSON ricevuto da Gemini. L'evento deve essere creato **solo se creare_evento è "si" e data non è "null"**.
    2. **Regola:** Se ora_inizio è "null", l'evento deve essere creato come "giornata intera". Se è presente, l'evento deve avere un inizio e una fine (es. durata di un'ora).
    3. **Metodo:** Costruire il "corpo" dell'evento in formato dizionario Python e passarlo a service.events().insert(calendarId='primary', body=event_body).execute().
- **Fase 5: Marcare Email come Letta (Metodo: users().messages().modify())**
    1. **Regola:** Questa azione deve essere eseguita **solo dopo che l'evento è stato creato con successo su Calendar**. È un passaggio cruciale per evitare duplicati.
    2. **Metodo:** Usare service.users().messages().modify(userId='me', id=msg_id, body={'removeLabelIds': ['UNREAD']}).execute().

**4. Flusso di Lavoro Automatico (Esecuzione dello Script)**

1. **Avvio Automatico:** L'Utilità di pianificazione di Windows avvia lo script python C:\Calendar_Automation\ControllaEmailCreaEvento.py all'orario prestabilito (3 volte al giorno).

2. **Autenticazione Silenziosa:** Lo script trova il file token.json, si autentica con Google in background e carica la chiave API dal file .env.
3. **Ciclo di Elaborazione:**
    - Lo script chiede a Gmail la lista delle email non lette.
    - Se non ce ne sono, lo script termina la sua esecuzione.
    - Se ce ne sono, per ogni email, esegue le fasi 3, 4 e 5 descritte sopra.
4. 
5. **Risultato Finale:** Nuovi eventi appaiono nel Google Calendar dell'account x31mint@gmail.com e le email da cui sono stati generati vengono contrassegnate come lette. L'intero processo è invisibile all'utente.