UNIVERSITÀ DELLA CALABRIA

DIPARTIMENTO DI INGEGNERIA INFORMATICA, MODELLISTICA,
ELETTRONICA E SISTEMISTICA

Corso di Laurea Ingegneria Informatica

Relazione progetto per il corso
di Sistemi Distribuiti e Cloud Computing

MailSpace

Prof. Domenico Talia
Ing. Loris Belcastro

Gabriele Hamil
Mat. 256747

ANNO ACCADEMICO 24/25

Indice

Capitolo 1

Introduzione

1.1 Obiettivi del progetto

Mail Space è un'applicazione web progettata per rivoluzionare la gestione delle email, combinando un'interfaccia intuitiva con le potenzialità dell'**intelligenza artificiale**. Gli obiettivi principali del progetto sono:

- **Gestione sicura degli utenti:** Un sistema di autenticazione robusto, basato su registrazione con conferma email e recupero password, garantisce accessi sicuri e prevenzione di attività non autorizzate.
- **Caricamento flessibile delle email:** Gli utenti possono inserire email manualmente (mittente, oggetto, corpo) o caricare file '.eml', estraendone automaticamente i contenuti.
- **Analisi intelligente con Google Gemini:** Integrato per analizzare il testo delle email, Gemini assegna **tag dinamici** basati sul contenuto, semplificando l'organizzazione e la ricerca. L'analisi, seppur non infallibile, offre una base solida per categorizzare le mail in modo personalizzato.
- **Organizzazione personalizzata:** Creazione di cartelle, modifica dei tag (con aggiornamento retroattivo via Gemini) e filtri avanzati (unione/intersezione di tag) permettono agli utenti di adattare la piattaforma alle proprie esigenze.

- **Ricerca avanzata:** Funzionalità di ricerca semplice (testo libero, case-insensitive) e avanzata (per mittente, oggetto, parole chiave) ottimizzano la reperibilità delle informazioni.
- **Scalabilità e affidabilità:** L'architettura distribuita, basata su servizi cloud come **Amazon EC2** e **RDS**, assicura performance stabili anche con un carico utenti elevato.

Mail Space si propone come uno strumento moderno, in grado di **automatizzare** l'organizzazione delle email attraverso l'AI, mantenendo al contempo flessibilità e controllo nelle mani dell'utente.

1.2 Tecnologie e servizi utilizzati

Per realizzare Mail Space, è stata adottata un'architettura moderna che combina tecnologie lato client e server con servizi cloud avanzati, garantendo **scalabilità**, **sicurezza** e **performance ottimali**. Le scelte tecnologiche sono state guidate dalla necessità di offrire un'esperienza utente fluida e funzionalità intelligenti.

- **Frontend**: Sviluppato con **Angular**, il framework che permette di creare interfacce dinamiche e reattive grazie alla sua architettura modulare. L'uso di componenti, servizi e il two-way data binding semplifica la gestione dello stato dell'applicazione e migliora l'user experience.
- **Backend**: Basato su **Node.js**, ideale per gestire operazioni I/O-intensive in modo asincrono. La sua natura event-driven e il vasto ecosistema di pacchetti (tramite npm) hanno accelerato l'integrazione con API esterne e servizi cloud.
- **Cloud Computing**: L'applicazione è ospitata su **Amazon EC2**, che offre istanze scalabili e configurabili per bilanciare carichi di lavoro variabili senza compromettere tempi di risposta o disponibilità.
- **Autenticazione**: Gestita tramite **Amazon Cognito**, che fornisce un sistema sicuro per registrazione, login (con conferma email) e recupero password, inclusi meccanismi di protezione contro accessi non autorizzati.

- **Database**: **Amazon RDS** con motore **PostgreSQL** assicura una memorizzazione strutturata ed efficiente di email, utenti e metadati. PostgreSQL è stato preferito per il supporto a query complesse, transazioni ACID e la possibilità di estendere le funzionalità con stored procedure.

- **Analisi del testo con Google Gemini**: Integrato per analizzare il contenuto delle email e assegnare **tag** in base al contesto. Gemini sfrutta modelli di linguaggio avanzati per identificare temi ricorrenti, migliorando l'organizzazione e la ricercabilità delle mail. La soluzione offre flessibilità, consentendo aggiornamenti retroattivi dei tag quando l'utente modifica le proprie preferenze.

Tutti i servizi AWS utilizzati (EC2, Cognito, RDS) rispettano rigorosi standard di **sicurezza** (come crittografia end-to-end) e **conformità**, mentre **Google Gemini** aggiunge un layer intelligente per ottimizzare l'interazione utente.

Capitolo 2

Architettura dell'applicazione

2.1 Diagramma e componenti

L'architettura di Mail Space, rappresentata nel diagramma seguente, è progettata per garantire scalabilità, sicurezza e prestazioni ottimali, integrando tecnologie moderne e servizi cloud avanzati.

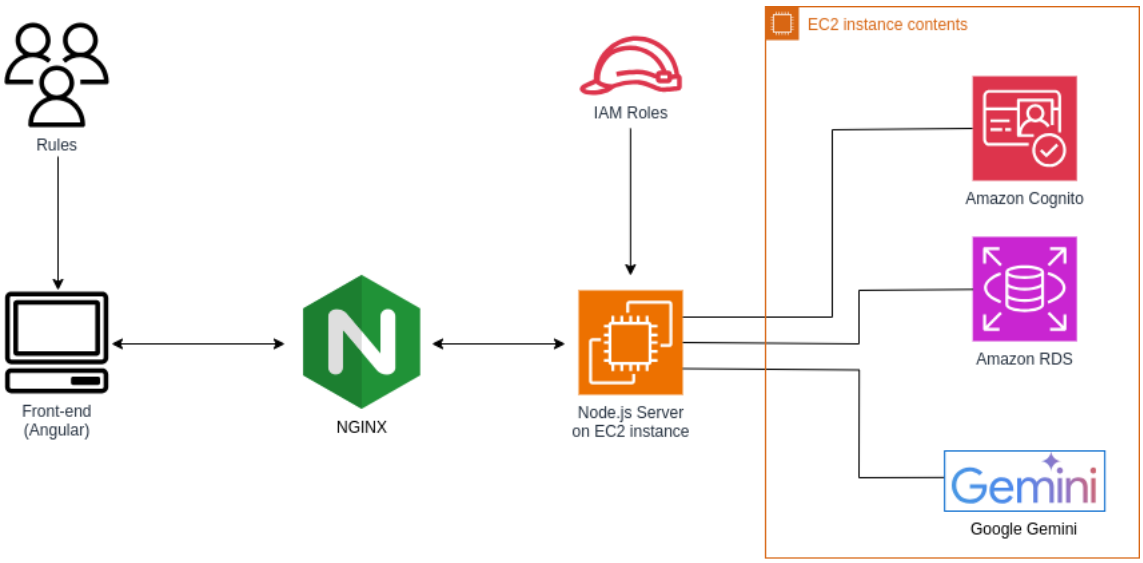

Figura 2.1: Diagramma dell'architettura dell'applicazione

I principali componenti del sistema sono:

- **Utenti**: Interagiscono con l'applicazione attraverso un'interfaccia web responsive, accedendo a tutte le funzionalità di gestione delle email.

- **Frontend (Angular)**: Realizzato con il framework Angular, gestisce l'interfaccia utente e le interazioni client-side, comunicando con il backend tramite API REST.

- **NGINX**: Configurato come reverse proxy, gestisce il routing delle richieste e la distribuzione dei file statici, migliorando sicurezza e prestazioni.

- **Backend (Node.js su EC2)**: Implementato in Node.js e ospitato su Amazon EC2, elabora le richieste degli utenti, gestisce la logica applicativa e coordina l'accesso ai servizi esterni.

- **IAM Roles**: Definiscono le politiche di accesso sicuro ai servizi AWS, garantendo il principio del minimo privilegio per l'istanza EC2.

- **Amazon Cognito**: Fornisce un sistema completo di gestione delle identità, con autenticazione sicura, registrazione utenti e recupero password.

- **Amazon RDS (PostgreSQL)**: Archivia in modo strutturato tutti i dati dell'applicazione, incluse le email, i metadati, le cartelle e le relazioni tra gli elementi.

- **Google Gemini**: Componente chiave per l'analisi intelligente del contenuto delle email. Gemini processa il testo delle mail per:

  - Assegnare automaticamente tag rilevanti basati sul contenuto
  - Supportare l'aggiornamento retroattivo dei tag quando l'utente modifica le proprie categorie
  - Migliorare l'efficacia del sistema di ricerca e filtraggio

L'integrazione sinergica di questi componenti permette a Mail Space di offrire un'esperienza utente completa e personalizzabile, con particolare valore aggiunto derivante dall'analisi avanzata del testo fornita da Google Gemini.

2.2 Descrizione dei flussi operativi

L'architettura di Mail Space prevede una serie di interazioni ben definite tra i componenti, che garantiscono il corretto funzionamento dell'applicazione. Di seguito vengono descritti i principali flussi di comunicazione, con riferimento al diagramma dell'architettura.

1. **Utenti $\longleftrightarrow$ Frontend (Angular)**
   Gli utenti interagiscono con l'interfaccia web attraverso il browser. Le principali operazioni includono:
   - **Richieste in ingresso**: Login, caricamento email, creazione cartelle e ricerche
   - **Risposte in uscita**: Pagine HTML dinamiche e dati JSON aggiornati
2. **Frontend (Angular) $\longleftrightarrow$ NGINX**
   NGINX funge da intermediario tra client e backend:
   - **Richieste API**: Il frontend invia chiamate REST a NGINX
   - **Risposte**: NGINX restituisce file statici o dati dal backend
3. **NGINX $\longleftrightarrow$ Server Node.js (EC2)**
   Le richieste vengono instradate al backend:
   - **Elaborazione**: Node.js processa le richieste e coordina i servizi
   - **Risposte**: Restituisce dati strutturati al frontend
4. **Ruoli IAM $\longrightarrow$ Server Node.js**
   I ruoli IAM garantiscono accesso sicuro ai servizi AWS:
   - **Autorizzazioni**: Permessi granulari per RDS e Cognito
   - **Sicurezza**: Nessuna credenziale statica memorizzata
5. **Server Node.js $\longleftrightarrow$ Servizi AWS**
   - **Amazon Cognito**:
     - Autenticazione utenti e gestione sessioni

    - Verifica credenziali e emissione token JWT
- **Amazon RDS (PostgreSQL)**:
    - **Scrittura**: Metadati email, cartelle utente e associazioni
    - **Lettura**: Ricerche e visualizzazione dati
- **Google Gemini**:
    - **Analisi testo**: Il server invia il contenuto delle email a Gemini
    - **Assegnazione tag**: Riceve e memorizza i tag rilevanti
    - **Aggiornamento dinamico**: Rivalutazione tag su modifica preferenze utente

L'integrazione con Google Gemini rappresenta il fulcro dell'innovazione del sistema, permettendo:

- Un'analisi semantica avanzata del contenuto delle email
- L'assegnazione automatica di tag contestuali
- Un sistema di categorizzazione dinamico e personalizzabile
- La possibilità di aggiornamenti retroattivi delle classificazioni

Tutti i flussi sono progettati per garantire bassa latenza, alta disponibilità e sicurezza dei dati, sfruttando al meglio le potenzialità dell'infrastruttura cloud e dei servizi di intelligenza artificiale.

Capitolo 3

Funzionalità

3.1 Registrazione e autenticazione

Mail Space offre un sistema completo e sicuro per la gestione degli account utente, garantendo un accesso protetto a tutte le funzionalità dell'applicazione.

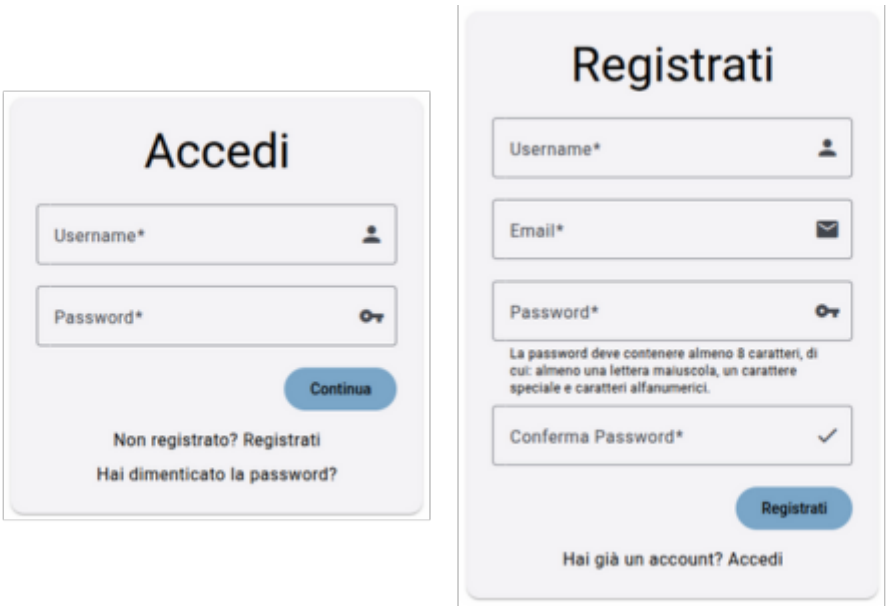

Figura 3.1: Form per effettuare l'accesso e la registrazione

3.1.1 Registrazione degli utenti

Il processo di registrazione è semplice e intuitivo:

Campi richiesti

- **Username**: Identificativo unico per l'accesso
- **Email**: Utilizzata per la verifica e le comunicazioni
- **Password**: Deve includere almeno 8 caratteri, con maiuscole, numeri e caratteri speciali
- **Conferma password**: Previene errori di digitazione

Verifica email:

- Il sistema invia un codice di 6 cifre all'indirizzo fornito
- L'utente deve inserire il codice per completare la registrazione
- L'account diventa attivo solo dopo questa verifica

3.1.2 Autenticazione degli utenti

Il login avviene tramite credenziali username/password, con:

- **Sicurezza avanzata**:
  - Crittografia end-to-end delle credenziali
  - Protezione contro attacchi brute-force
  - Token JWT per le sessioni
- **Gestione sessioni**:
  - Autenticazione persistente (fino a 30 giorni)
  - Logout manuale o automatico dopo inattività

3.1.3 Recupero password

In caso di credenziali dimenticate:

- L'utente può richiedere un reset password
- Viene inviato un codice all'email registrata

- Possibilità di impostare una nuova password sicura

L'intero sistema è implementato utilizzando **Amazon Cognito**, che garantisce:

- Archiviazione sicura delle credenziali
- Scalabilità automatica in base al carico
- Conformità agli standard di sicurezza più recenti

Questa architettura assicura che solo gli utenti autorizzati possano accedere alle proprie email e alle funzionalità di analisi offerte da Google Gemini, mantenendo sempre la massima protezione dei dati personali.

3.2 Gestione delle email

Il sistema offre funzionalità complete per il caricamento, l'analisi intelligente e la gestione organizzata delle email. Ogni aspetto è stato progettato per essere intuitivo.

3.2.1 Caricamento delle Email

L'utente dispone di due modalità flessibili per aggiungere email all'archivio:

- **Inserimento manuale:** attraverso un modulo intuitivo per inserire mittente, oggetto e corpo del messaggio, con validazione in tempo reale dei campi obbligatori.
- **Caricamento da file .eml:** tramite l'upload di file standard (`RFC-5322`), con parsing automatico dei metadati e estrazione del contenuto testuale.

3.2.2 Analisi Intelligente con Google Gemini

Ogni email archiviata viene potenziata dall'intelligenza artificiale di Google Gemini:

- **Assegnazione automatica dei tag:** il servizio esegue un'analisi semantica del contenuto per collegare l'email ai tag più pertinenti tra quelli definiti dall'utente.

- **Adattamento dinamico:** il sistema effettua un aggiornamento retroattivo delle classificazioni ogni volta che l'utente modifica la propria lista di tag, garantendo coerenza a tutto l'archivio.

3.2.3 Visualizzazione e Gestione

L'interfaccia di gestione centrale permette un controllo completo sulle email archiviate:

- **Visualizzazione a griglia:** presenta un'anteprima chiara dei metadati essenziali (mittente e oggetto).
- **Paginazione configurabile:** consente di visualizzare 5, 10 o 25 elementi per pagina per una navigazione ordinata.
- **Gestione in blocco:** offre la possibilità di selezionare più email tramite checkbox per effettuare un'eliminazione multipla.

3.2.4 Filtraggio Avanzato per Tag

Il sistema di ricerca è potenziato da un meccanismo di filtraggio per tag preciso e versatile:

- **Selezione dei tag:** l'utente può cliccare su uno o più tag, visualizzati come "pillole", per attivare il filtro.
- **Modalità "Unione":** mostra tutte le email che corrispondono ad *almeno uno* dei tag selezionati.
- **Modalità "Intersezione":** restringe la ricerca mostrando solo le email che contengono *tutti* i tag selezionati.
- **Controllo intuitivo:** il passaggio tra la modalità "Unione" e "Intersezione" avviene tramite un semplice interruttore (toggle switch).

Questa combinazione di automazione e controllo manuale offre una notevole flessibilità, trasformando la semplice archiviazione in un sistema intelligente che riduce significativamente il tempo necessario per la classificazione e il recupero dei messaggi importanti.

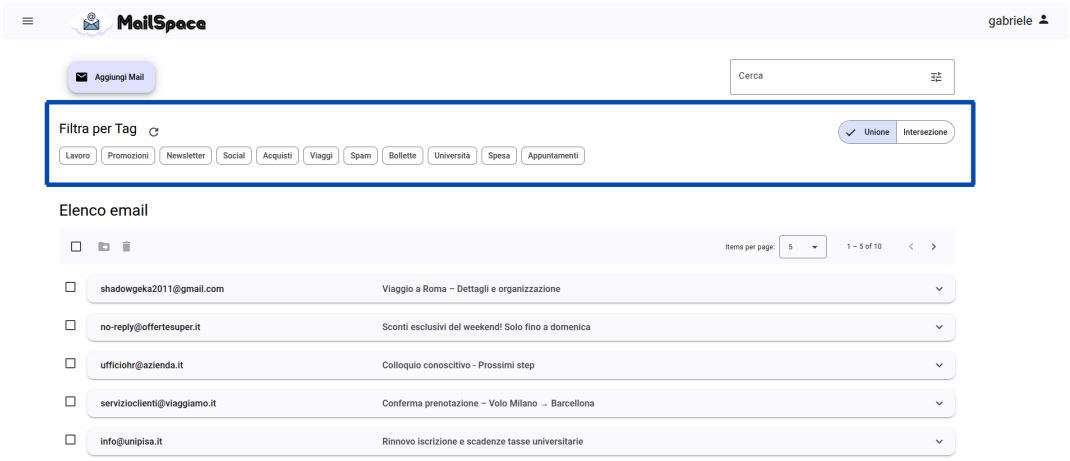

Figura 3.2: Interfaccia di filtraggio con selezione tag e toggle Unione/Intersezione

3.3 Gestione delle cartelle

Mail Space offre un sistema completo per organizzare le email in strutture personalizzate attraverso cartelle.
Le principali funzionalità includono:
Creazione di nuove cartelle:

- L'utente può aggiungere cartelle cliccando sull'icona "+" posizionata nella sezione dedicata
- Una finestra modale permette di:
  - Assegnare un nome descrittivo alla cartella
  - Confermare la creazione, che avviene istantaneamente

Visualizzazione e gestione delle cartelle esistenti:

- Elenco completo di tutte le cartelle create dall'utente
- Per ogni cartella sono disponibili:
  - Nome cliccabile per visualizzare il contenuto
  - Icona di eliminazione (a forma di cestino) per rimuovere la cartella

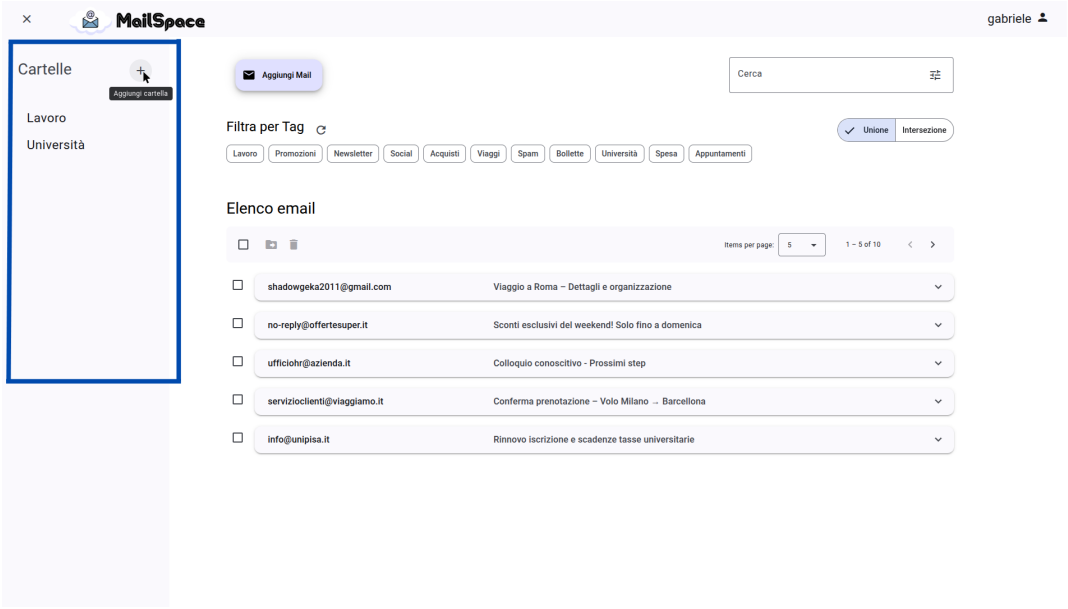

Figura 3.3: Barra laterale per la gestione delle cartelle

- L'eliminazione di una cartella:
  - Non cancella le email contenute
  - Mantiene i messaggi disponibili nella posta generale

3.4 Ricerca avanzata

Mail Space offre un potente sistema di ricerca che combina semplicità d'uso con funzionalità avanzate, permettendo agli utenti di trovare rapidamente qualsiasi email nel proprio archivio. La barra di ricerca, posizionata nell'angolo superiore destro dell'interfaccia, rappresenta il punto di accesso a queste funzionalità.

Modalità di ricerca

Il sistema offre due approcci complementari:

- **Ricerca semplice**
  - Ricerca full-text su tutti i campi (mittente, oggetto, corpo del messaggio)
  - Attivazione tramite pressione del tasto Invio

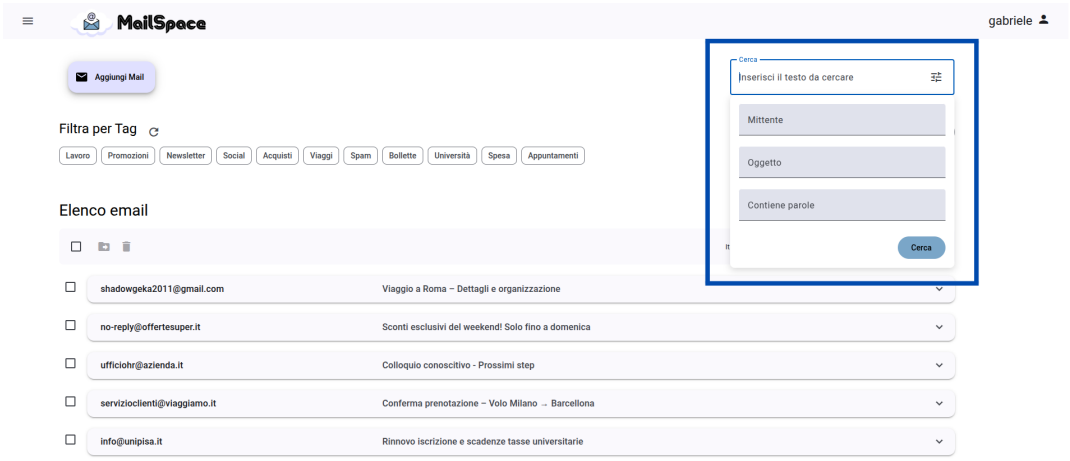

Figura 3.4: Barra di ricerca avanzata con filtri

- **Ricerca avanzata** (accessibile tramite icona 'tune')

  Campi specifici separati:

  - Mittente
  - Oggetto
  - Corpo del messaggio

Caratteristiche tecniche

- **Case insensitivity**: Le ricerche ignorano le differenze tra maiuscole e minuscole
- **Ricerca per sottostringhe**: Restituisce risultati contenenti la sequenza di caratteri inserita

Capitolo 4

Implementazione

4.1 Struttura del codice

4.1.1 Front-end

Il front-end dell'applicazione è stato sviluppato utilizzando Angular, un framework robusto e modulare che facilita la creazione di interfacce utente dinamiche e scalabili. Questa sezione descrive la struttura dei file, le tecnologie utilizzate e le principali funzionalità implementate.

Struttura delle directory

La struttura del codice front-end è stata progettata per garantire la separazione delle responsabilità e facilitare la manutenzione futura. Tutti i file sorgenti si trovano nella directory `/src/app`, che è suddivisa in tre principali sottodirectory:

- `components:` Contiene i componenti che costituiscono l'interfaccia utente dell'applicazione. Ogni componente è organizzato in una propria cartella e include due file principali: un file `.ts` per la logica del componente e un file `.html` per il markup associato. Le cartelle presenti in questa directory includono:
  - `register:` Implementa la logica per la registrazione di un nuovo utente.

  - `confirmation-code`: Gestisce l'inserimento del codice di conferma per attivare un nuovo account.
  - `login`: Implementa la logica per l'autenticazione tramite credenziali.
  - `forgot-pwd`: Gestisce la richiesta di recupero della password da parte dell'utente tramite email.
  - `profile`: Consente la modifica della password e la configurazione dei tag dell'utente.
  - `reset-pwd`: Permette all'utente di impostare una nuova password dalla sezione profilo.
  - `mail-viewer`: Visualizza l'elenco delle email caricate e ne consente la gestione.
  - `mail-uploader`: Consente di caricare nuove email manualmente.
  - `add-folder`: Permette agli utenti di creare nuove cartelle per organizzare le email.
  - `search`: Implementa la funzionalità di ricerca tra le email, con un'opzione avanzata per indicare anche il mittente, l'oggetto e le parole contenute nel testo.

- `models`: Questa directory contiene i modelli dati utilizzati nell'applicazione per rappresentare le entità principali:
  - `email.ts`: Rappresenta la struttura di un'email, inclusi i metadati e i tag associati.
  - `folder.ts`: Rappresenta una cartella personalizzata creata dall'utente.

- `services`: Contiene i servizi che implementano la logica di business e gestiscono la comunicazione con il back-end.
  I principali servizi includono:
  - `auth.service`: Gestisce la registrazione, l'autenticazione e il logout degli utenti.

- `email.service`: Si occupa del caricamento, dell'analisi e della ricerca delle email.
- `folder.service`: Gestisce le operazioni relative alle cartelle utente.
- `labels.service`: Gestisce la creazione, l'eliminazione e il recupero dei tag associati a un utente, utilizzati per classificare le email.
- `filter-state.service`: Mantiene e condivide lo stato dei tag selezionati per la ricerca, facilitando la comunicazione tra i componenti coinvolti nel filtraggio.
- `sidenav.service`: Controlla la visualizzazione della barra laterale (sidenav), che include funzionalità per la gestione e la navigazione tra le cartelle dell'utente.
- `notification.service`: Mostra notifiche personalizzate tramite il componente snack-bar.

Inoltre, nella root della directory `/src/app`, sono presenti i file principali:

- `app.component`: Il componente root che funge da punto di ingresso per l'intera applicazione.
- `app.config`: Contiene la configurazione generale dell'applicazione.
- `app.routes`: Gestisce il routing per la navigazione tra i componenti.

Tecnologie e librerie utilizzate

Per migliorare l'esperienza utente e la produttività nello sviluppo, sono state integrate diverse tecnologie e librerie:

- **Angular Material:** Utilizzato per fornire un'interfaccia utente moderna e reattiva. I componenti predefiniti, come modals, snack-bar, expansion panel e chips, sono stati adattati per soddisfare i requisiti specifici dell'applicazione.
- **RxJS:** Utilizzato per gestire i flussi di dati reattivi, come l'interazione con i servizi e la gestione delle chiamate HTTP.

- **CSS personalizzato:** Sebbene Angular Material offra uno stile di base, è stato applicato del CSS personalizzato per ottimizzare l'estetica dell'applicazione.

Questa struttura e organizzazione hanno reso il front-end modulare e facilmente estensibile, garantendo un'esperienza utente intuitiva e coerente.

4.1.2 Back-end

Il back-end dell'applicazione è stato sviluppato utilizzando Node.js e Express per creare un server scalabile e modulare. L'architettura del codice è stata progettata per separare le responsabilità, rendendo il codice più leggibile e manutenibile. Di seguito, viene descritta la struttura principale del back-end.

Struttura delle directory

La directory principale del server, denominata `node-server`, è organizzata in sottodirectory e file chiave per garantire una chiara divisione delle funzionalità:

- `controllers`: Questa cartella contiene i file che implementano la logica applicativa per ogni funzionalità principale. I controller gestiscono le richieste HTTP e delegano le operazioni specifiche ai servizi:
  - `authController`: Gestisce la registrazione, il login, il logout degli utenti e la gestione del codice di conferma.
  - `emailController`: Gestisce le operazioni relative alle email, come il caricamento, la visualizzazione, l'analisi del contenuto e l'assegnazione di tag tramite Google Gemini.
  - `folderController`: Si occupa della creazione e rimozione delle cartelle personalizzate degli utenti.
  - `labelController`: Si occupa della creazione e rimozione dei tag dell'utente.
- `middlewares`: Contiene i middleware utilizzati per gestire l'autenticazione e altre logiche di intermediazione.

  - `authMiddleware`: Include il metodo `authenticateJWT`, utilizzato per verificare i token JWT nelle richieste HTTP, garantendo che solo gli utenti autenticati possano accedere alle risorse protette.

- `routes`: Questa directory definisce le route per le diverse funzionalità dell'applicazione, collegando le richieste HTTP ai rispettivi controller:
  - `authRoutes`: Contiene le route relative alla registrazione, autenticazione e gestione degli utenti.
  - `emailRoutes`: Definisce le route per operazioni sulle email, come il caricamento, l'analisi e la ricerca.
  - `folderRoutes`: Gestisce le route per le operazioni legate alle cartelle, come creazione ed eliminazione.
  - `labelRoutes`: Contiene le route relative all'accesso ai tag e alle relative operazioni come creazione ed eliminazione.

- `services`: I servizi contengono la logica di business che interagisce con i servizi esterni o con il database. Questa suddivisione consente di mantenere i controller snelli e focalizzati.
  - `geminiService`: Contiene la logica per interagire con l'API di Google Gemini. Questo servizio invia il testo delle email al modello generativo per analizzarne il contenuto ed associare alla mail una serie di tag pertinenti che ne descrivono gli argomenti principali.
  - `rdsService`: Gestisce le operazioni con il database relazionale Postgres, come la memorizzazione e il recupero dei dati relativi a utenti, email, tag e cartelle.

File principali

Oltre alle directory, il back-end include alcuni file fondamentali:

- `app.js`: Punto d'ingresso dell'applicazione, dove viene inizializzato il server Express, configurate le middleware globali (ad esempio, il parsing del JSON) e registrate le route.

- `config.js`: File in cui sono definite le configurazioni per i servizi esterni, come le informazioni per l'API di Google Gemini e i parametri di connessione al database. Questo file centralizza la configurazione per facilitare le modifiche.

- `.env`: File che contiene le variabili d'ambiente e i dati sensibili, come la chiave API per Google Gemini, la stringa di connessione al database e il segreto per la generazione dei token JWT. L'utilizzo di questo file garantisce che le credenziali non siano esposte nel codice sorgente.

Principali tecnologie e librerie utilizzate

- **Express.js**: Framework per la creazione del server web.

- **jsonwebtoken (JWT)**: Utilizzato per gestire l'autenticazione basata su token.

- **Google AI SDK for Node.js (`@google/generative-ai`)**: Libreria ufficiale per interagire con i modelli generativi di Google, inclusa la famiglia di modelli Gemini.

- **dotenv**: Per caricare i valori delle variabili di ambiente dal file `.env`.

- **pg**: Libreria per connettersi e interagire con il database Postgres.

Grazie a questa organizzazione, il codice del back-end è modulare e facilmente estendibile, semplificando sia l'implementazione di nuove funzionalità sia la gestione di quelle esistenti.

4.2 Integrazione con i servizi AWS

4.2.1 Amazon Cognito

Amazon Cognito è stato utilizzato per gestire l'autenticazione e la registrazione degli utenti dell'applicazione. Questo servizio consente di creare e mantenere user pool sicuri per l'autenticazione, semplificando l'integrazione con il back-end e garantendo la scalabilità del sistema.

Configurazione

Per implementare l'autenticazione con Amazon Cognito:

- Ho creato uno **User Pool**, che rappresenta il database degli utenti per l'applicazione. Durante la configurazione, ho abilitato l'autenticazione tramite username e password.
- Successivamente, ho generato un **App Client** all'interno dello User Pool, il quale ha fornito un **Client ID**. Questo Client ID è utilizzato nelle richieste di autenticazione come il signup, la conferma e il login.
- I permessi necessari per utilizzare Cognito sono stati configurati tramite un ruolo IAM, garantendo l'accesso sicuro ai metodi dello User Pool.

Codice del back-end

Nel back-end, il file `authController.js` contiene la logica per l'autenticazione degli utenti. Di seguito sono mostrati i principali metodi utilizzati.

Registrazione (signup)

Questo metodo consente di registrare un nuovo utente nello User Pool:

- Si inviano il nome utente, la password e l'email al metodo `signUp()` di Cognito.
- Cognito genera un codice di conferma inviato via email all'utente.

```
exports.signup = async (req, res) => {
  const { username, password, email } = req.body;
  const params = {
    ClientId: CLIENT_ID,
    Username: username,
    Password: password,
    UserAttributes: [{ Name: 'email', Value: email }],
  };
  try {
    const data = await cognito.signUp(params).promise();
    console.log('Registrazione effettuata con successo!');
    res.json(data);
```

```
  } catch (error) {
    console.log('Registrazione NON effettuata');
    res.status(400).json(error);
  }
};
```

Conferma registrazione (confirm)

Dopo la registrazione, l'utente deve inserire un codice di conferma per completare il processo:

- Il codice di conferma, insieme al nome utente, viene inviato al metodo `confirmSignUp()` di Cognito.
- In caso di successo, la registrazione viene completata.

```
exports.confirm = async (req, res) => {
  const { username, confirmationCode } = req.body;
  const params = {
    ClientId: CLIENT_ID,
    Username: username,
    ConfirmationCode: confirmationCode,
  };
  try {
    const data = await cognito.confirmSignUp(params).promise();
    console.log('Registrazione confermata con successo!');
    res.json(data);
  } catch (error) {
    console.log('Errore, registrazione NON confermata!');
    res.status(400).json(error);
  }
};
```

Login (signin)

Per accedere all'applicazione, l'utente inserisce le sue credenziali:

- Il nome utente e la password vengono inviati a Cognito tramite il metodo `initiateAuth()`.

- In caso di successo, Cognito restituisce un `IdToken` e un `AccessToken`. L'`IdToken` viene decodificato per ottenere informazioni sull'utente, mentre l'`AccessToken` viene utilizzato per l'autenticazione nelle richieste successive.
- Un ulteriore **JWT** firmato viene generato lato server per la comunicazione con il client.

```
exports.signin = async (req, res) => {
  const { username, password } = req.body;

  const params = {
    AuthFlow: 'USER_PASSWORD_AUTH',
    ClientId: CLIENT_ID,
    AuthParameters: {
        USERNAME: username,
        PASSWORD: password,
    },
  };

  try {
    const data = await cognito.initiateAuth(params).promise();

    const idToken = data.AuthenticationResult.IdToken;
    const decodedToken = jwt.decode(idToken);
    const email = decodedToken.email;

    const authToken = data.AuthenticationResult.AccessToken;
    const token = jwt.sign({
        accessToken: authToken,
        username: username,
        email: email
      },
      jwt_secret_key, { expiresIn: '24h' }
    );
    res.json({ token });
    console.log('Accesso avvenuto con successo!');
  } catch (error) {
    console.error('Errore durante il login: ',error)
    res.status(400).json({ error: 'Accesso non riuscito' });
```

```
  }
};
```

Password dimenticata (requestPasswordReset)

Questa funzionalità permette a un utente che ha dimenticato la propria password di avviare il processo di recupero.

- L'utente inserisce il proprio nome utente (o email) nell'interfaccia.
- Il back-end invia questa informazione al metodo `forgotPassword()` di Cognito.
- Cognito si occupa di inviare un codice di recupero all'indirizzo email verificato associato all'account.

```
exports.requestPasswordReset = async (req, res) => {
  const { email } = req.body;
  const params = {
    ClientId: CLIENT_ID,
    Username: email,
  };

  try {
    await cognito.forgotPassword(params).promise();
    console.log('Codice di reset inviato con successo!');
    res.json({ message: 'Codice di reset inviato con successo!'
    });
  } catch (error) {
    console.error('Errore durante l\'invio del codice di reset:
    ', error);
    res.status(400).json({ error: 'Errore durante l\'invio del
   codice di reset' });
  }
};
```

Reset della password (resetPassword)

Una volta ricevuto il codice di recupero, l'utente può impostare una nuova password per completare il processo.

- L'utente fornisce il codice di recupero, il proprio nome utente e la nuova password.
- Questi dati vengono inviati al metodo `confirmForgotPassword()` di Cognito.
- Se il codice è valido e non è scaduto, la password dell'utente viene aggiornata con successo.

```
exports.resetPassword = async (req, res) => {
  const { email, confirmationCode, newPassword } = req.body;
  const params = {
    ClientId: CLIENT_ID,
    Username: email,
    ConfirmationCode: confirmationCode,
    Password: newPassword,
  };

  try {
    await cognito.confirmForgotPassword(params).promise();
    console.log('Password reimpostata con successo!');
    res.json({ message: 'Password reimpostata con successo!' })
  ;
  } catch (error) {
    console.error('Errore durante la reimpostazione della
  password: ', error);
    res.status(400).json({ error: 'Errore durante la
  reimpostazione della password' });
  }
};
```

Cambio della password (changePassword)

Questa funzione è dedicata agli utenti già autenticati che desiderano modificare la propria password dall'interno dell'applicazione.

- L'utente, già loggato, deve fornire la sua password attuale e quella nuova.
- Il back-end, utilizzando il token di accesso dell'utente, invoca il metodo `changePassword()` di Cognito, passando la password vecchia e quella nuova.

- Cognito verifica la correttezza della vecchia password prima di autorizzare la modifica, garantendo che solo il legittimo proprietario dell'account possa effettuare l'operazione.

```
exports.changePassword = async (req, res) => {
  const { oldPassword, newPassword } = req.body;

  const params = {
    AccessToken: req.user.accessToken,
    PreviousPassword: oldPassword,
    ProposedPassword: newPassword
  };

  try {
    await cognito.changePassword(params).promise();
    console.log('Password modificata con successo!');
    res.json({ message: 'Password modificata con successo!' });
  } catch (error) {
    console.error('Errore durante la modifica della password:
    ', error);
    res.status(400).json({ error: 'Errore durante la modifica
    della password' });
  }
};
```

Logout

Il logout invalida il token di accesso dell'utente utilizzando il metodo `globalSignOut()` di Cognito, garantendo che la sessione venga terminata su tutti i dispositivi.

```
exports.logout = async (req, res) => {

  const params = { AccessToken: req.user.accessToken };

  try {
    await cognito.globalSignOut(params).promise();
    console.log('Logout avvenuto con successo!');
    res.json({ message: 'Successfully logged out' });
  } catch (error) {
    console.log('Errore durante il globalSignOut: ', error);
```

```
    res.status(400).json(error);
  }
};
```

4.2.2 Amazon RDS

Amazon RDS (Relational Database Service) è stato utilizzato come soluzione per la gestione dei dati relazionali, sfruttando PostgreSQL come motore di database. La scelta di un servizio gestito come RDS garantisce affidabilità, scalabilità e semplifica le operazioni di manutenzione del database.

Configurazione del database

Il database è stato configurato tramite la console AWS con le seguenti specifiche:

- **Tipo di istanza**: `db.t4g.micro`.
- **CPU e Memoria**: 2 vCPU e 1 GiB RAM.
- **Storage**: 20 GiB.
- **Costo**: Compatibile con il piano Free Tier, con un costo di 0.020 USD/ora al di fuori del Free Tier.

Schema del database

Lo schema è stato progettato e creato utilizzando `pgAdmin 4`. Include le seguenti tabelle principali per la gestione dei dati dell'applicazione.

1. Tabella `emails`

Questa tabella memorizza i dati essenziali delle email, inclusi i metadati e i label (tag) generati dall'analisi del contenuto effettuata tramite il modello Gemini.

```
CREATE TABLE emails (
    id INT PRIMARY KEY,
    user_email VARCHAR(255),
    sender VARCHAR(255),
    subject VARCHAR(255),
```

```
    body TEXT,
    labels TEXT[]
);
```

2. Tabella `folders`

Gestisce le cartelle personalizzabili degli utenti. Ogni utente può creare cartelle con nomi univoci all'interno del proprio account.

```
CREATE TABLE folders (
    id SERIAL PRIMARY KEY,
    user_email VARCHAR(255) NOT NULL,
    name VARCHAR(255) NOT NULL,
    UNIQUE(user_email, name)
);
```

3. Tabella `email_folders`

Implementa la relazione molti-a-molti tra email e cartelle, permettendo che una singola email sia associata a più cartelle e che una cartella contenga più email.

```
CREATE TABLE email_folders (
    folder_id INTEGER NOT NULL,
    email_id INTEGER NOT NULL,
    PRIMARY KEY (email_id, folder_id),
    FOREIGN KEY (email_id) REFERENCES emails(id) ON DELETE
   CASCADE,
    FOREIGN KEY (folder_id) REFERENCES folders(id) ON DELETE
   CASCADE
);
```

Connessione al database

La connessione a RDS è stata configurata nel file `config.js` del back-end, utilizzando il modulo `pg` di Node.js. Per garantire la sicurezza, le credenziali di accesso al database (host, utente, password) sono state esternalizzate e vengono caricate da un file `.env`.

```
require('dotenv').config();
const { Pool } = require('pg');
const pool = new Pool({
```

```
  ssl: {
    rejectUnauthorized: false
  },
  user: process.env.RDS_USERNAME,
  host: process.env.RDS_HOSTNAME,
  database: process.env.RDS_DB_NAME,
  password: process.env.RDS_PASSWORD,
  port: parseInt(process.env.RDS_PORT || '5432'),
});
```

Gestione delle operazioni sul database

Il file `rdsService.js` (o un nome analogo come `db.service.js`) centralizza tutta la logica di interazione con il database, contenendo i metodi per eseguire operazioni CRUD (Create, Read, Update, Delete) sulle tabelle.

Esempio di metodo: `addFolder(userEmail, folderName)` Questo metodo esegue una query SQL per inserire una nuova riga nella tabella `folders`, associandola a un utente specifico.

```
exports.addFolder = async (userEmail, folderName) => {
  const query = `
    INSERT INTO folders (user_email, name)
    VALUES ($1, $2)
    RETURNING id
  `;
  const values = [userEmail, folderName];

  try {
    const result = await pool.query(query, values);
    return result.rows[0].id;
  } catch (error) {
    console.error('Errore nell\'aggiunta della cartella:',
   error);
    throw error;
  }
};
```

Altri metodi per operazioni CRUD

Il servizio di gestione del database espone numerosi metodi, raggruppati per funzionalità, per supportare tutte le operazioni richieste dall'applicazione:

- **Gestione Email:** Metodi per le operazioni di base sulle email, come `insertEmail`, `deleteEmails` e `getUserEmails`.
- **Ricerca:** Funzioni dedicate alla ricerca, che includono una ricerca semplice (`simpleSearch`, `filteredSimpleSearch`) e una avanzata per criteri multipli (`advancedSearch`, `filteredAdvancedSearch`).
- **Gestione Cartelle:** Operazioni per creare, recuperare ed eliminare le cartelle utente (`addFolder`, `getFolders`, `deleteFolder`).
- **Associazione Email-Cartelle:** Metodi per gestire la relazione tra email e cartelle, come `addEmailsToFolder`, `getEmailsFromFolder` e `removeEmailsFromFolder`.
- **Gestione dei Label (Tag):** Un insieme completo di funzioni per la gestione dei tag generati da Gemini. Include metodi per aggiungere e rimuovere tag da singole email (`addLabelsToEmail`, `removeLabelsFromUserEmails`), gestire l'elenco globale dei tag di un utente (`addLabelsToUserLabelsTable`, `removeLabelsFromUserLabelsTable`) e per interrogazioni specifiche (`getLabels`, `getFilteredEmails`, `getDistinctUserLabels`).

4.2.3 Google Gemini

Per l'analisi e la classificazione automatica delle email in base al loro contenuto, è stato impiegato il servizio di intelligenza artificiale Google Gemini. La scelta è ricaduta sul modello `gemini-1.5-flash`, una versione efficiente e veloce, ideale per le esigenze del progetto. L'accesso al servizio è stato ottenuto tramite Google AI Studio, che ha permesso di generare una chiave API gratuita, rendendo la soluzione economicamente vantaggiosa.

Configurazione

La configurazione del servizio è gestita centralmente nel file `config.js`. Qui, viene importata la libreria ufficiale `@google/generative-ai` e viene istanziato

il client `GoogleGenerativeAI`, utilizzando la chiave API memorizzata in modo sicuro come variabile d'ambiente (`process.env.GEMINI_API_KEY`). Questo approccio garantisce che le credenziali non siano esposte direttamente nel codice sorgente.

```
// config.js
const { GoogleGenerativeAI } = require('@google/generative-ai');
const genAI = new GoogleGenerativeAI(process.env.GEMINI_API_KEY);
```

Implementazione del Servizio

Il cuore della logica di classificazione risiede nel file `geminiService.js`, che espone la funzione asincrona `classifyEmail`. Questa funzione accetta due argomenti: il corpo (body) dell'email da analizzare e una lista di etichette (tags) predefinite dall'utente.

Il suo compito è costruire un prompt specifico per il modello Gemini, istruendolo ad analizzare il testo dell'email e a restituire un sottoinsieme delle etichette fornite, formattato come array JSON.

```
// geminiService.js
const { genAI } = require('../config');

exports.classifyEmail = async (body, tags) => {
  const prompt = `
Analizza il seguente testo di una email e restituisci una lista dei tag pertinenti tra quelli forniti.
Testo dell'email:
"${body}"

Tag disponibili: ${tags.join(', ')}

Rispondi con un array JSON contenente solo i tag rilevanti.
`;

  try {
    const model = genAI.getGenerativeModel({ model: 'gemini-1.5-flash' });
```

```
    const result = await model.generateContent(prompt);
    const response = await result.response;
    const text = await response.text();

    // Estrai l'array JSON dalla risposta
    const matches = text.match(/\[.*\]/s);
    const extractedLabels = matches ? JSON.parse(matches[0]) : [];
    console.log('Etichette estratte: ', extractedLabels);
    return extractedLabels;
  } catch (error) {
    console.error('Errore durante l\'analisi con Gemini:', error);
    throw error;
  }
};
```

Il prompt è strutturato per guidare il modello a eseguire un compito di classificazione mirato. Una volta inviata la richiesta, il servizio attende la risposta, la elabora per estrarre l'array JSON di etichette e lo restituisce al chiamante. La gestione degli errori è inclusa per intercettare e registrare eventuali problemi durante la comunicazione con l'API di Gemini, garantendo la robustezza del sistema.

Capitolo 5

Deployment

5.1 Configurazione dell'istanza EC2

La piattaforma scelta per il deployment è Amazon Web Services (AWS). Ho creato una nuova istanza EC2 utilizzando i seguenti parametri:

- **AMI:** Ubuntu Server 24.04 LTS.
- **Tipo di istanza:** t2.micro (1 vCPU, 1 GiB di memoria) al costo di 0.0126 USD/ora.
- **Storage:** 30 GiB di tipo General Purpose SSD (gp3).
- **Accesso remoto:** Per accedere alla macchina è stata generata una coppia di chiavi SSH e il collegamento avviene tramite il comando:

  ```
  ssh -i my_ssh_keys.pem ubuntu@<elastic_ip>
  ```

- **Elastic IP:** È stato associato un Elastic IP per mantenere un indirizzo IP statico anche dopo il riavvio dell'istanza.

Configurazione delle policy IAM

Per consentire alla macchina di accedere ai servizi AWS necessari, è stato definito il ruolo `AmazonRDSFullAccess` per avere completo accesso al database RDS.

Regole del security group

Sono state configurate le regole di sicurezza del **security group** associato all'istanza EC2 per consentire il traffico in entrata sui seguenti servizi e porte:

Inbound rules		
Tipo	Porta	Origine
SSH	22	0.0.0.0/0
HTTP	80	0.0.0.0/0
HTTPS	443	0.0.0.0/0

5.2 Configurazione del server NGINX

Per gestire il traffico HTTP/HTTPS e l'integrazione tra il **front-end** (Angular) e il **back-end** (Node.js), è stato configurato **NGINX** come reverse proxy.

Configurazione di NGINX

La configurazione utilizzata è riportata di seguito:

```
server {
    server_name mail-space.it;

    root /var/www/angular-client/browser;
    index index.html;

    location /auth {
        proxy_pass http://localhost:3000/auth;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection 'upgrade';
        proxy_set_header Host $host;
        proxy_cache_bypass $http_upgrade;
    }
    location /email {
        proxy_pass http://localhost:3000/email;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
```

```
        proxy_set_header Connection 'upgrade';
        proxy_set_header Host $host;
        proxy_cache_bypass $http_upgrade;
    }
    location /folder {
        proxy_pass http://localhost:3000/folder;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection 'upgrade';
        proxy_set_header Host $host;
        proxy_cache_bypass $http_upgrade;
    }

    location / {
        try_files $uri $uri/ /index.html;
    }

    error_page 404 /index.html;

    listen 443 ssl; # managed by Certbot
    ssl_certificate /etc/letsencrypt/live/mail-space.it/
   fullchain.pem; # managed by Certbot
    ssl_certificate_key /etc/letsencrypt/live/mail-space.it/
   privkey.pem; # managed by Certbot
    include /etc/letsencrypt/options-ssl-nginx.conf; # managed
   by Certbot
    ssl_dhparam /etc/letsencrypt/ssl-dhparams.pem; # managed by
    Certbot
}
server {
    if ($host = mail-space.it) {
        return 301 https://$host$request_uri;
    } # managed by Certbot

    listen 80;

    server_name mail-space.it;
    return 404; # managed by Certbot
}
```

Funzionalità principali di NGINX

1. **Hosting del front-end:** Il codice Angular è stato posizionato nella directory `/var/www/angular-client/browser` e configurato come root del server NGINX.

2. **Reverse Proxy:** Le richieste indirizzate ai percorsi `/auth`, `/email` e `/folder` vengono inoltrate al back-end Node.js, eseguito in locale sulla porta `3000`.

3. **HTTPS:** Certificati SSL generati con **Certbot** per garantire una connessione sicura.

Configurazione del dominio

Il dominio `mail-space.it` è stato registrato gratuitamente utilizzando il servizio **register.it**. Una volta ottenuto il dominio, è stato configurato per puntare all'Elastic IP dell'istanza EC2.

Configurazione di HTTPS

Per abilitare HTTPS, ho utilizzato **Certbot** per generare i certificati SSL necessari. Il processo ha incluso:

1. Installazione di Certbot tramite `apt`:

```
sudo apt update
sudo apt install certbot python3-certbot-nginx
```

2. Configurazione automatica dei certificati SSL con il comando:

```
sudo certbot --nginx
```

Capitolo 6

Conclusioni e prospettive

6.1 Requisiti progettuali e Soluzioni Adottate

MailSpace è un'applicazione web completa per la gestione e l'analisi intelligente di email. La progettazione ha seguito i requisiti funzionali e non funzionali, integrando diverse tecnologie cloud per fornire una soluzione robusta e funzionale.

Rispettando il vincolo di utilizzare la piattaforma Amazon Web Services, l'architettura si basa su un'istanza EC2 per l'hosting dell'applicativo back-end (Node.js) e del front-end (Angular), e su Amazon RDS come soluzione di storage gestito per la persistenza dei dati. La gestione degli utenti, requisito implicito per un'applicazione multi-utente, è stata demandata ad Amazon Cognito, che ha permesso di implementare in modo sicuro e affidabile funzionalità di registrazione, autenticazione e recupero password.

Il requisito cardine del progetto, ovvero l'estrazione automatica di informazioni dal contenuto testuale delle email, è stato soddisfatto attraverso l'integrazione di Google Gemini. Al caricamento di una nuova email — sia manualmente che tramite file `.eml` — il servizio analizza il corpo del testo e lo associa ai metadati più pertinenti, rappresentati da tag predefiniti dall'utente. Questa soluzione risponde direttamente alla richiesta di identificare gli argomenti trattati e associarli come metadati.

Le funzionalità di ricerca e organizzazione dei dati sono state pienamente implementate.

L'utente può:

- Ricercare email tramite una ricerca testuale semplice (case-insensitive e per sottostringhe su mittente, oggetto e corpo) o avanzata (per campi specifici).
- Filtrare le email selezionando tag, con una logica commutabile tra *Unione* (mostra le email che contengono almeno uno dei tag selezionati) e *Intersezione* (mostra solo quelle associate a tutti i tag), per un controllo granulare sulla visualizzazione.
- Organizzare le email in cartelle personalizzate, adempiendo a un'ulteriore modalità di raggruppamento.

Infine, l'applicazione è stata arricchita con funzionalità avanzate, come la gestione dinamica dei tag dal profilo utente, la cui modifica avvia una ri-analisi retroattiva di tutte le email archiviate, garantendo che i metadati siano sempre aggiornati e coerenti. In sintesi, MailSpace integra con successo servizi cloud di AWS e Google per fornire una soluzione che non solo risponde a tutti i requisiti progettuali, ma li arricchisce con un'esperienza utente completa e intuitiva.

6.2 Miglioramenti e Sviluppi Futuri

Sebbene MailSpace sia un'applicazione completa che risponde a tutti i requisiti iniziali, esistono numerosi ambiti di miglioramento e sviluppi futuri che potrebbero arricchirne ulteriormente il valore. Tali prospettive possono essere raggruppate in tre aree strategiche principali: esperienza utente, funzionalità tecniche e integrazioni avanzate.

Esperienza Utente (UX)

Miglioramenti focalizzati a rendere l'interazione con la piattaforma più intuitiva, accessibile e gradevole.

- **Interfaccia Mobile:** Sviluppo di un'interfaccia completamente responsive o di una Progressive Web App (PWA) per garantire una piena usabilità da smartphone e tablet.

- **Supporto Multilingua:** Internazionalizzazione dell'interfaccia per supportare utenti di diverse nazionalità, ampliando la potenziale base di utenti.

Funzionalità Tecniche

Evoluzioni dell'architettura e della logica di base per aumentare sicurezza, privacy e performance.

- **Autenticazione a Due Fattori (2FA):** Introduzione del secondo fattore di autenticazione (es. tramite app come Google Authenticator) sfruttando le funzionalità avanzate di AWS Cognito, per innalzare drasticamente il livello di sicurezza degli account.

- **Anonimizzazione dei Dati Sensibili (PII):** Implementazione di un meccanismo di pre-processing che, prima di inviare il testo a servizi esterni come Gemini, identifichi e mascheri le informazioni personali identificabili (PII) quali nomi, indirizzi o numeri di telefono, a tutela della privacy dell'utente.

- **Analisi del Sentiment e Riconoscimento di Entità (NER):** Potenziamento dell'analisi AI per non limitarsi ai tag, ma per estrarre anche il *sentiment* (tono positivo, negativo, neutro) di un'email o riconoscere entità specifiche come date, luoghi e organizzazioni, rendendole a loro volta metadati ricercabili.

Integrazioni Avanzate

Funzionalità che connettono MailSpace all'ecosistema di servizi esterni, trasformandolo in un hub centrale per la gestione delle informazioni.

- **Collegamento Diretto ai Servizi di Posta:** Integrazione con le API di provider come Gmail e Outlook per consentire l'importazione automatica e continua dell'inbox dell'utente, eliminando la necessità del caricamento manuale.

- **Integrazione con i Calendari:** Sfruttando il riconoscimento di entità (date), si potrebbe introdurre un pulsante "Aggiungi al calendario" per creare eventi su Google Calendar o Outlook con un solo click.

L'implementazione di queste funzionalità trasformerebbe MailSpace da un efficace strumento personale a una piattaforma di intelligenza email versatile e integrata, pronta per affrontare anche casi d'uso più complessi e collaborativi.