

# Sistemi e Applicazioni Cloud – Laboratorio

## Esercitazione esame 19 dicembre 2025

Tempo consegna: 2h 30m

Si realizzi un'applicazione su Google Cloud Platform per la gestione delle prenotazioni della "Sala Riunioni 42", l'unica sala conferenze dell'azienda "Procrastination Solutions Inc" che può ospitare al massimo 20 persone contemporaneamente.

I dipendenti devono poter prenotare la sala per inutili riunioni improduttive specificando:

1. Chi organizza la riunione (il "Colpevole")
2. La lista di colleghi da trascinare nella follia
3. Quanto tempo verrà sprecato
4. L'orario di inizio della riunione (formato HH:MM)

Il sistema deve poter:

- Bloccare le prenotazioni che superano il limite di 20 vittime... pardon, partecipanti
- Permettere di modificare una prenotazione esistente (magari per aggiungere altre vittime)
- Fornire statistiche cruciali per l'azienda come "Dipendente più improduttivo del mese" e "Manager sabotatore dell'anno"

Si rendano disponibili i seguenti tipi di interfacce di utilizzo:

1. API per inserire e correggere le prenotazioni della sala
2. Pagina Web per la visualizzazione delle prenotazioni da parte dell'HR (Human Remains)

L'applicazione deve essere testata per il deployment su piattaforma GCP utilizzando i servizi visti a lezione:

- App Engine
- Firestore
- Functions

## Parte 1 – Backend API REST

L'applicazione deve esporre le seguenti funzionalità tramite opportune Web API RESTful:

1. Inviando richieste **POST** all'URI `/api/v1/room42/{data}` è possibile registrare una prenotazione per una certa data. Il corpo della richiesta deve contenere:

- **colpevole**: nome del dipendente che organizza (campo obbligatorio)
- **vittime**: lista di nomi dei partecipanti (massimo 20)
- **durata**: durata della riunione in ore (decimale tra 0.5 e 8, multiplo di 0.5)
- **orario\_inizio**: orario di inizio in formato HH:MM

Se per lo stesso colpevole e la stessa data esiste già una prenotazione, l'invio di una nuova richiesta comporta un errore di duplicazione (ergo lo stesso colpevole non può prenotare due riunioni nello stesso giorno anche se ad orari non in conflitto tra loro).

Il sistema deve verificare che non ci siano sovrapposizioni temporali tra le riunioni della stessa giornata.

2. Inviando richieste **PUT** all'URI `/api/v1/room42/{data}` è possibile aggiornare una prenotazione esistente. Il corpo della richiesta deve contenere gli stessi campi del metodo POST. Se non esiste una prenotazione per quel colpevole e quella data, si genera un errore. L'aggiornamento deve rispettare le stesse regole di validazione del POST, inclusa la verifica delle sovrapposizioni temporali.
3. Inviando richieste **GET** allo URI `/api/v1/room42/{data}` l'HR(Human Remains) può ottenere:
  - il numero totale di ore di lavoro sprecate per quella data
  - la lista di tutte le riunioni con colpevoli, vittime e orari di inizio (ordinate per ora di inizio)
4. Inviando richieste **POST** allo URI `/api/v1/panic` è possibile cancellare TUTTE le prenotazioni (feature utile il venerdì pomeriggio)

L'interfaccia di utilizzo delle API deve rispettare rigorosamente il file di specifica *OpenAPI* disponibile insieme alla presente specifica.

Per testare la funzionalità dell'API è possibile far riferimento al seguente URL:

<https://sacvalidator2025.appspot.com/>

## Parte 2 – Web Application

Realizzare due pagine Web per l'HR(Human Remains):

1. Una "Mappa della Follia" che mostra:

- Le riunioni della settimana con un indicatore di gravità (ore x partecipanti)
- La sala odierna evidenziata in rosso se ci sono più di 4 ore di riunioni
- Un contatore "Ore di vita sprecate questo mese"

2. Una pagina di dettaglio "Sala Riunioni 42" che per una data specifica mostra:

- La lista delle riunioni previste con colpevoli e relative vittime
- Il totale di ore perse
- Il "Dannato del Giorno" (chi ha organizzato la riunione più lunga)

## Parte 3 – Statistiche Vitali

Implementare un sistema di statistiche per misurare la mancanza di produttività:

- Una Cloud Function triggerata da ogni nuova/modifica prenotazione che aggiorna:
  - Il contatore mensile per ogni dipendente (ore passate in riunioni)
  - Il contatore di riunioni organizzate per ogni manager
- Un endpoint GET `/api/v1/stats/slackers/mese` che restituisca:
  - Il "Dipendente più Improduttivo del Mese" (più ore in riunioni)
  - La sua "efficienza" calcolata come:  $\text{ore\_riunioni} / \text{ore\_lavoro}$ (\*)
- (\*) Si assume "ore\_lavoro" come una costante di 8 ore al giorno, 5 giorni alla settimana.
- Un endpoint GET `/api/v1/stats/saboteurs/mese` che restituisca:
  - Il "Manager Sabotatore del Mese" (più riunioni organizzate)
  - Il suo "indice di sabotaggio" calcolato come:  $\text{numero\_riunioni} \times \text{numero\_vittime\_totali}$

## Dettagli Implementativi

- Le prenotazioni si salvano in Firestore nella collezione `meeting_crimes`
- Le statistiche mensili in `productivity_nightmare`
- Validazioni:
  - \* Massimo 20 persone totali (colpevole + vittime)
  - \* Durata tra 0.5 e 8 ore, multiple di 0.5 (le riunioni più lunghe sono crimini contro l'umanità)
  - \* Formato orario: HH:MM (24 ore)
  - \* Ora di inizio valida: tra le 08:00 e le 19:00 (orario lavorativo)
  - \* Non possono esserci sovrapposizioni temporali: due riunioni non possono sovrapporsi nella stessa giornata
  - \* Una riunione non può terminare dopo le 20:00 (orario di chiusura della sede)
- Formato date: DD-MM-YYYY per l'UI, YYYY-MM-DD per il DB
- Per il controllo sovrapposizioni: data + orario\_inizio + durata determinano l'intervallo temporale

## Note

## Note

- Se la validazione dell'API con il validatore non è positiva, l'esame è automaticamente insufficiente.
- In caso di validazione positiva, il voto sarà basato su quante parti dell'esame sono state portate a termine e sulla qualità del codice e delle soluzioni proposte.

## Funzioni per gestire le date e gli orari

```
from datetime import datetime
def date_from_str(d):
    # converte 'gg-mm-YYYY' in oggetto datetime
    try:
        return datetime.strptime(d, '%d-%m-%Y')
    except:
        return None

def str_from_date(d):
    # converte oggetto datetime in 'gg-mm-YYYY'
    return d.strftime('%d-%m-%Y')

def time_from_str(t):
    """converte 'HH:MM' in oggetto time"""
    try:
        return datetime.datetime.strptime(t, '%H:%M').time()
    except:
        return None

def calculate_end_time(start_time_str, duration):
    """Calcola l'orario di fine dato l'orario di inizio e la durata"""
    try:
        start_time = time_from_str(start_time_str)

        # Converti in minuti
        start_minutes = start_time.hour * 60 + start_time.minute
        end_minutes = start_minutes + int(duration * 60)

        # Converti di nuovo in time
        end_hour = end_minutes // 60
        end_minute = end_minutes % 60

        return datetime.time(end_hour, end_minute)
    except ValueError:
        return None
```

```

def parse_month(month_str):
    """Converte un mese da MM-YYYY a datetime"""
    try:
        return datetime.datetime.strptime(month_str, '%m-%Y')
    except ValueError:
        return None

```

## Prototipo per gestire le sovrapposizioni

```

# given docs as a list containing all existing meetings for the same day, start_t

new_start = time_from_str(start_time_str)
new_start_minutes = new_start.hour * 60 + new_start.minute
new_end_minutes = new_start_minutes + int(duration * 60)

for doc in docs:
    # if we are in the update of a certain X document, be sure to skip it
    if doc.id == id_of_document_being_updated:
        continue
    existing_start = time_from_str(doc.orario_inizio)
    existing_start_minutes = existing_start.hour * 60 + existing_start.minute
    existing_end_minutes = existing_start_minutes + int(existing_duration * 60)
    if not (new_end_minutes <= existing_start_minutes or existing_end_minutes <= new_start_minutes):
        return True
return False

```

## Calcolo dell'Indice di Sabotaggio

$$IS_{M,m} = \sum_{i=1}^{R_{M,m}} (h_i \times p_i)$$

Con:

- $R_M, m$ : numero di riunioni organizzate dal manager  $M$  nel mese  $m$
- $h_i$ : ore della riunione  $i$
- $p_i$ : partecipanti alla riunione  $i$

Il manager con il massimo  $IS_{M,m}$  vince il titolo di "Sabotatore del Mese".