

---

# System Design Documentation

Progetto Smart Office - IoT and 3D Intelligent Systems

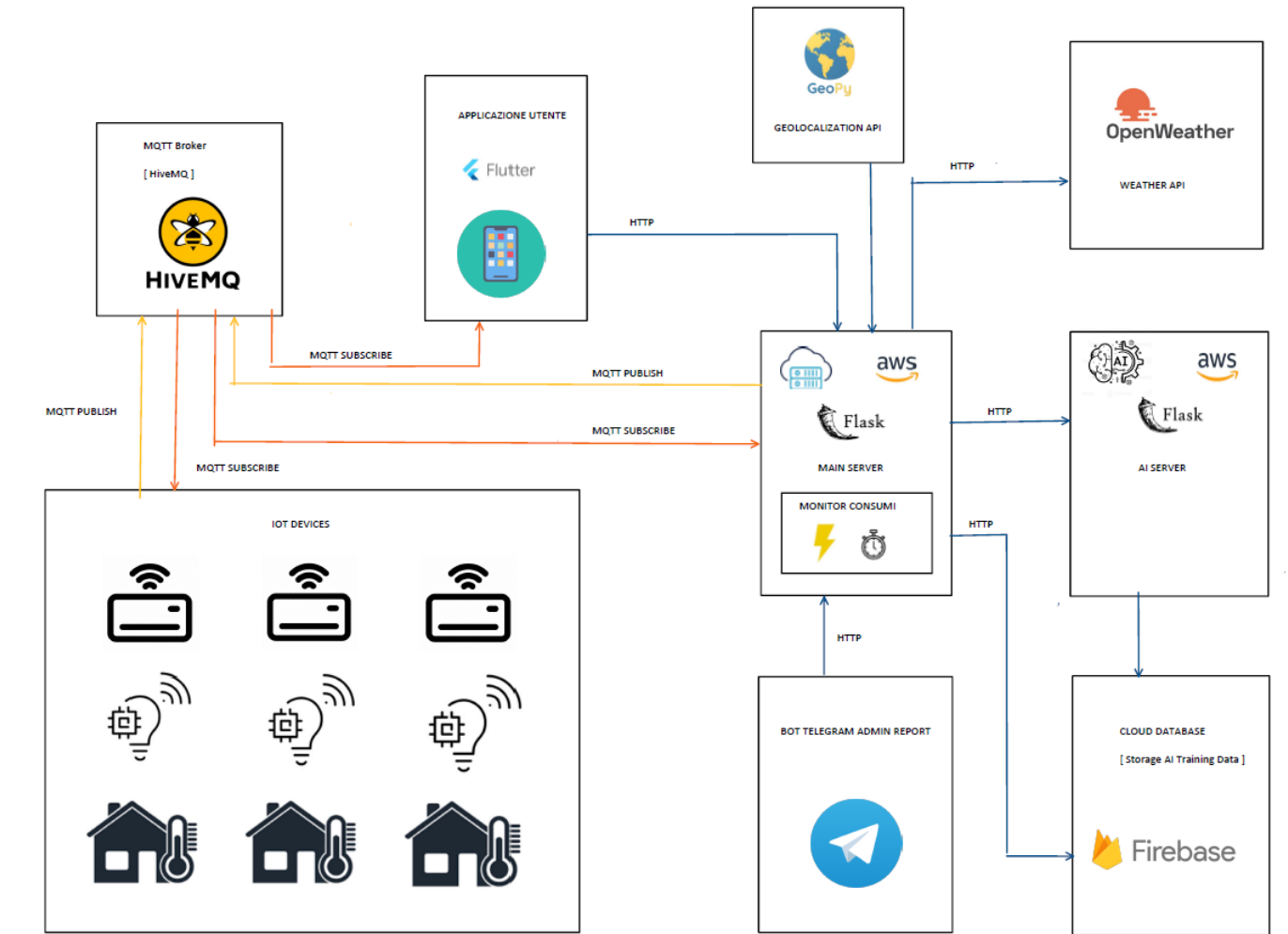
Macellaro Vincenzo

Arfaoui Bilel

Felicia Puzone



## Panoramica




## AWS Setup

Istanze EC2 (**Amazon Elastic Compute Cloud**) create mediante la piattaforma di cloud computing Amazon Web Services (AWS) di Amazon.

## Modulo AI

Istanza **t2.micro**:

- 1 vCPU Intel Xeon scalabile, fino a 3.3 GHz

- 
- 1 GiB RAM
  - 30 GB EBS (Amazon Elastic Block Store)
  - Ubuntu 22.04 LTS (Jammy Jellyfish), Image Build 2023-01-15
  - IP elastico

Esecuzione del **Server Flask** attraverso il linguaggio di programmazione **Python** (versione 3.7.16), librerie utilizzate:

- Flask (ver. 2.2.2)
- Numpy (ver. 1.9.15)
- TensorFlow (ver. 2.5.0)
- Keras (ver. 2.11.0)
- H5py (ver. 3.1.0)

## Modulo Server Principale

Istanza **t2.micro**:

- 1 vCPU Intel Xeon scalabile, fino a 3.3 GHz
- 1 GiB RAM
- 30 GB EBS (Amazon Elastic Block Store)
- Windows Server 2022 English Full base
- IP elastico

L'esecuzione dei programmi **Server Flask**, **Server AI Locale**, **Bot Telegram Monitor di Consumi** avvengono attraverso il linguaggio di programmazione **Python**.

Librerie utilizzate **Server Flask**, **Server AI Locale**:

- Flask (ver. 2.2.2)

Librerie utilizzate dal **Bot Telegram**:

- Telebot

Librerie utilizzate all'interno del **Server Flask**:

- Flask-Admin
- Flask-Login

- Requests
- Geopy
- Firebase
- SQLAlchemy
- Plotly
- Pandas

Librerie utilizzate all'interno del **Monitor Consumi**:

- Paho Mqtt
- Schedule

## Device, Bridge, MQTT

Il dispositivo comunica in seriale con un bridge Python, il quale effettua la connessione MQTT e gestisce il flusso di informazioni.

Il broker MQTT utilizzato è Hive MQ. La connessione è stata ottenuta con la seguente configurazione:

```
Port = 1883
Server = broker.hivemq.com
```

Ogni stanza è dotata di un building ID e di un room ID, necessari alla creazione dei topic MQTT grazie ai quali essa comunica.

### Last Will

Quando il bridge si disconnette invia un messaggio di testamento con payload 'Bad' al topic `smartoffice/building_*/room_*/health`.

```
smartoffice/building_*/room_*/health
```

QoS: 1

retain: True

```
smartoffice/building_*/room_*/status
```

QoS: 1

retain: True

```
smartoffice/building_%s/room_%s/actuators/color
```

QoS: 1

retain: True

```
smartoffice/building_%s/room_%s/actuators/brightness
```

QoS: 1

retain: True

```
smartoffice/building_%s/room_%s/actuators/temperature
```

QoS: 1

retain: True

```
smartoffice/building_%s/room_%s/sensors/light_sensor
```

QoS: 0

retain: False

```
smartoffice/building_%s/room_%s/sensors/noise_sensor
```

QoS: 0

retain: False

## REST API

Il sistema presenta due Endpoint locati su due diversi server. Il primo è l'interfaccia presente sul server principale, il secondo sul server dedicato all'hosting degli algoritmi di AI.

### REST API Server Principale

Le route del server principale gestiscono la comunicazione tra server e i vari client. Possono gestire sia richieste attraverso **web-form** (libreria *flask-login*), sia richieste inviate da applicazione **Android/iOS** e da **bot Telegram**. L'accettazione di questi due tipi di richieste differisce per tipologia elaborazione, e quindi è stato adoperato il parametro **Content-ID** presente nell'header http per discernere la tipologia di client. I client di tipo applicazione presentano Content-ID del tipo <<#-APP>>, mentre i client di tipo browser hanno questo parametro nullo. Ciò permette di:

1. inviare tipi di risposte differenti (JSON per l'app, application/x-www-form-urlencoded per il web, gestite da libreria).
2. Gestire l'autenticazione del client App attraverso Token di autenticazione (generati con [URLSafeTimedSerializer](#)).

### Tipi di CLIENT

- Web browser
- Mobile App
- Telegram Bot

### REST API Server AI

Le route del server AI gestiscono sia la predizione dei parametri relativi alla personalizzazione dell'ufficio, sia l'aggiornamento autonomo del sistema.

- `/getUserColor` è utilizzata per la predizione del colore dell'illuminazione; riceve il JSON con i dati della sessione dal server Principale ed effettua l'inferenza sulla rete `color_model`;
- `/getUserLight` è utilizzata per la predizione dell'intensità di luce; riceve il JSON con i dati della sessione dal server Principale ed effettua l'inferenza sulla rete `light_model`;
- `/getUserTemp` è utilizzata per la predizione della temperatura dell'ufficio; riceve il JSON con i dati della sessione dal server Principale ed effettua l'inferenza sulla rete `temp_model`;
- `/systemUpdate` è utilizzata per effettuare l'update del sistema all'inizio del nuovo mese solare; contatta il DB Firebase mediante una richiesta GET e scarica tutte le sessioni relative al mese precedente, utilizzandole poi per l'aggiornamento delle reti neurali.

## Digital Twin

Il **digital twin** è modellato dal dispositivo IoT installato in ogni stanza registrata.

I suoi parametri sono:

1. Light sensor
2. Noise sensor

3. Led color actuator
4. Led brightness actuator
5. Temperature actuator

Essi vengono aggiornati in tempo reale dal server attraverso i dati acquisiti dai sensori ed i dati selezionati dagli utenti per gli attuatori attraverso interfaccia web o applicazione mobile; ciò corrisponde alla cosiddetta **sessione**.

Il sistema presenta una pagina di visualizzazione dello stato del digital twin in tempo reale.

## Storicizzazione dati

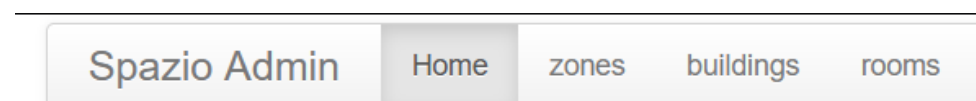
Alla chiusura di una sessione, il server Principale collassa i dati acquisiti in una tupla, che viene poi salvata in un **DB NoSQL Firebase**, hostato su cloud.

## Re-Training AI (Continuous Learning)

I dati storicizzati saranno utilizzati per il **training continuo** delle reti neurali, che terranno aggiornati i dati di profilazione utente in base alle sue preferenze nelle impostazioni della stanza.

## Dashboards

Il server principale gestisce l'elaborazione e visualizzazione in HTML di grafici contenenti lo storico e le statistiche dei dati del digital twin accumulati nel tempo. Essi sono visibili degli admin e dai super-admin, e sono raggruppati per **stanza**, per **edificio** e per **zona**.



Ciò che è possibile visualizzare è:

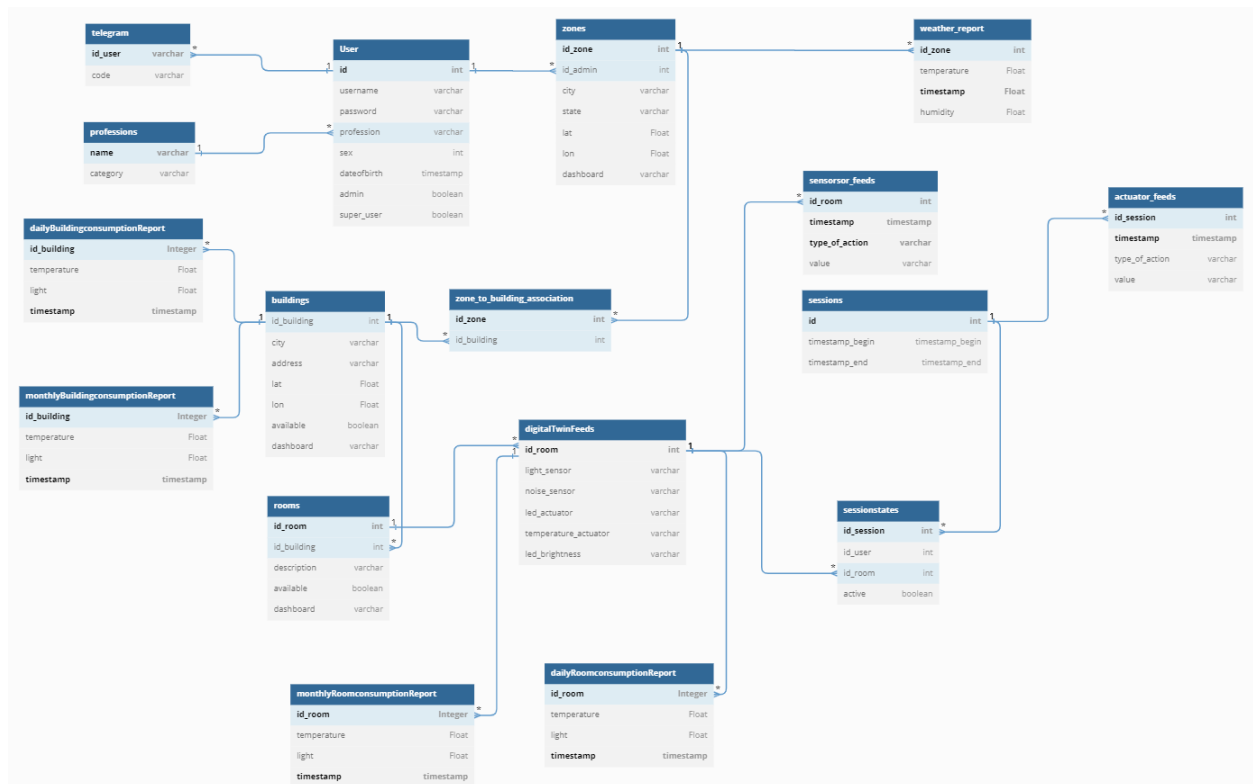
Per ogni sezione:

- Storico del sensore di luce
- Storico dei colori impostati del LED
- Storico delle intensità del LED
- Storico del riscaldamento
- Storico dei consumi giornalieri
- Storico dei consumi mensili

Per la dashboard *zone* inoltre ci sono in aggiunta:

- Storico temperatura meteo
- Storico umidità meteo

## Database



## Monitor consumi

Il monitor dei consumi ha lo scopo di evitare che i consumi energetici di una determinata zona superino una threshold stabilita in base alle esigenze. L'idea è quella di decrementare in tempo reale la stima dei consumi di ora in ora in modo equamente distribuito tra gli utenti di una zona. Decrementando i consumi seguendo questa strategia, l'esperienza utente di ciascuno non verrà compromessa in modo drastico, poiché le variazioni rispetto ai parametri impostati dall'utente saranno minime se il numero di utenti è elevato.

Ad esempio, un utente potrebbe essere indifferente ad una variazione di temperatura di 0,5 C° - 1 C° rispetto ai valori da lui impostati. Ma questa variazione, se applicata a 100 utenti, permette un risparmio energetico per una zona notevole.

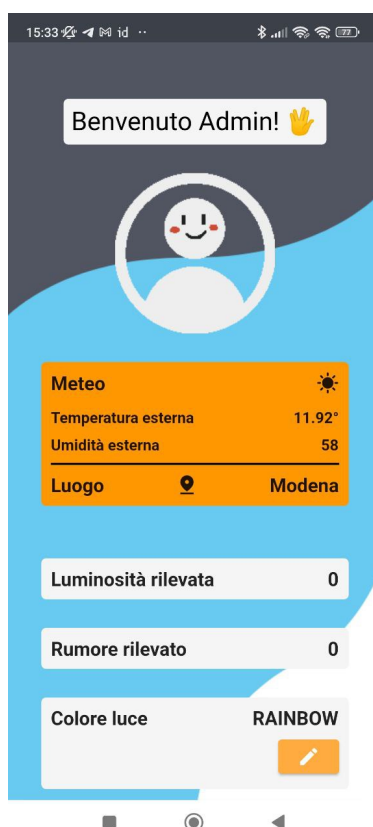


## Dati Meteo e Geolocalizzazione

I dati meteo vengono acquisiti attraverso il servizio OpenWeather, fornendo i dati di latitudine e longitudine acquisiti dal servizio GeoPy.

## Applicazione Utente

Ogni nuovo utente può registrarsi al sistema tramite applicazione mobile Android/iOS



## Bot Telegram

Il bot serve agli admin per visualizzare il report dei consumi mensili dei suoi edifici su una zona. L'admin si autentica tramite Telegram key visibile nella sezione "zona admin".

## Casi d'uso

### Utente

Cliente utilizzatore del servizio

1. Registrazione
2. Login
3. Occupazione stanza
4. Modifica colore luce
5. Modifica luminosità luce
6. Modifica temperatura
7. Visualizzazione dati sensore luce
8. Visualizzazione dati sensore rumore
9. Rilascio stanza
10. Logout

### Admin

Persona che ha messo a disposizione un tot numero di stanze.

1. Registrazione, modifica e rimozione edificio
2. Creazione ed eliminazione zone
3. Visualizzazione dashboard zone
4. Visualizzazione dashboard edifici
5. Visualizzazione dashboard stanze
6. Visualizzazione telegram key

## 7. Visualizzazione report telegram

### Super-Admin

1. Assegnazione ruolo admin
2. Assegnazione ruolo super-admin
3. Visualizzazione tabella utenti
4. Creazione professioni

### Tempo

1. Monitor consumi
2. Storizzazione dati
3. Re-training dati AI

In dettaglio:

#### 1.1 Registrazione

- L'utente apre l'app e clicca "Registrati"
- L'app invia una richiesta GET al server sulla route /professions e visualizza la lista di professioni nella apposita sezione
- L'utente inserisce Username, password, professione e data di nascita nei rispettivi form e clicca "Registrati"
- L'app invia una richiesta POST sulla route /register con i dati inseriti.
- Il server invia un messaggio di risposta con l'outcome della richiesta

#### 1.2 Login

- L'utente apre l'app e inserisce username e password nei relativi form e clicca "Login"
- L'app invia una richiesta POST sulla route /login con i dati inseriti.
- Il server invia una risposta

- Se l'utente si è loggato con successo: { logged\_in: True }
  - **Caso 1:** l'utente non ha una stanza occupata

```
{ logged_in: True,
outcome: Login,
buildings: {"id_building": self.id_building,
            "city": self.city,
            "lat":self.lat,
            "lon":self.lon,"address":self.address }}
```

Viene visualizzata la schermata Mappa e popolata con i dati della lista buildings.

- **Caso 2:** l'utente ha una stanza occupata

```
{ logged_in: True,
outcome: "Active",
digitalTwin:
    {"room_color": self.led_actuator,
     "room_temperature": int(self.temperature_actuator),
     "room_brightness": self.led_brightness},
id_edificio: id_building,
id_room: id_room,
username: username,
weather: {temperature: temperature,
          humidity: humidity }}
```

Viene visualizzata la schermata Home. La schermata home è popolata con i dati reperiti dal digital twin relativo alla sessione attiva e dai topic MQTT dei sensori, reperiti attraverso id\_edificio e id\_room.

- Se l'utente non si è loggato con successo: { logged\_in: False }  
Viene visualizza il messaggio "WRONG-CREDENTIALS"
- Se lo status code è diverso da 200

Viene visualizza il messaggio “ERRORE-RETE”

### 1.3 Occupazione stanza

- Dalla schermata Mappa, l'utente seleziona una stanza attraverso i pin situati sulla mappa. Dalla finestra pop-up clicca su “Prenota”.
- L'app invia al server una richiesta POST sulla route /selectRoom contenente l'ID della stanza selezionata
- Il server controlla se nel DB sono già presenti dati meteorologici acquisiti nelle ultime 3 ore, altrimenti interroga il servizio OpenWeather tramite GET attraverso il link  
[http://api.openweathermap.org/data/2.5/weather?lat="+lat+"&lon="+lon+"&APPID=API\\_KEY&units=metric](http://api.openweathermap.org/data/2.5/weather?lat=), con parametri latitudine e longitudine del luogo (precedentemente salvati attraverso GeoPy)
- Il server principale interroga il server AI con richiesta POST sulle route /getUserTemp, /getUserLight, /getUserColor per ottenere i parametri ottimali suggeriti per il tipo di utente specifico, elaborati attraverso le tre reti neurali.
- In caso di mancata raggiungibilità del server AI, il server principale manderà una richiesta al server AI locale, precisamente alla route /AI.
- Il server principale restituisce al client una risposta

```
{ logged_in: True,
  outcome: "Active",
  digitalTwin:
    {"room_color": self.led_actuator,
     "room_temperature": int(self.temperature_actuator),
     "room_brightness": self.led_brightness},
  id_edificio: id_building,
  id_room: id_room,
  username: username,
  weather: {temperature: temperature,
            humidity: humidity }}
```

### 1.4 Modifica colore luce

- L'utente clicca sull'icona di modifica alla riga Colore luce

- L'app visualizza la schermata ColorChanger
- L'utente seleziona dalla griglia il colore desiderato
- L'app invia al server una richiesta POST sulla route /update nella forma

```
{ "room_color": led_actuator,  
  "room_temperature": temperature),  
  "room_brightness": led_brightness }
```

- Il server controlla se i valori sono diversi dai precedenti, in tal caso aggiorna lo stato del digital twin e pubblica il nuovo valore sul topic MQTT `smartoffice/building_%s/room_%s/actuators/color`

### 1.5 Modifica luminosità luce

- L'utente clicca sull'icona di modifica alla riga Intensità luce
- L'app visualizza la schermata BrightnessChanger
- L'utente seleziona dalla griglia il livello di luminosità desiderato
- L'app invia al server una richiesta POST sulla route /update nella forma

```
{ "room_color": led_actuator,  
  "room_temperature": temperature),  
  "room_brightness": led_brightness }
```

- Il server controlla se i valori sono diversi dai precedenti, in tal caso aggiorna lo stato del digital twin e pubblica il nuovo valore sul topic MQTT `smartoffice/building_%s/room_%s/actuators/brightness`

### 1.5 Modifica temperatura

- L'utente clicca sull'icona di modifica alla riga Temperatura
- L'app visualizza la schermata TemperatureChanger
- L'utente seleziona dalla comboBox il livello di temperatura desiderato
- L'app invia al server una richiesta POST sulla route /update nella forma

```
{ "room_color": led_actuator,  
  "room_temperature": temperature),
```

```
"room_brightness": led_brightness }
```

- Il server controlla se i valori sono diversi dai precedenti, in tal caso aggiorna lo stato del digital twin e pubblica il nuovo valore sul topic MQTT `smartoffice/building_%s/room_%s/actuators/temperature`

### 1.6 Visualizzazione dati sensore di luce e di rumore

- L'app alla visualizzazione della schermata Home effettua il subscribe ai topic MQTT grazie alla libreria [mqtt\\_client](#).

```
smartoffice/building_%s/room_%s/sensors/light_sensor
```

```
smartoffice/building_%s/room_%s/sensors/noise_sensor
```

All'interno delle rispettive sezioni verranno visualizzati in tempo reale.

### 1.8 Rilascio stanza

- L'utente dalla schermata Home clicca sul bottone "Lascia stanza"
- L'app invia una richiesta POST sulla route `/freeRoom`
- Il server termina la sessione del Digital Twin e restituisce

```
{ logged_in: True,
  outcome: Login,
  buildings: {"id_building": self.id_building,
             "city": self.city,
             "lat":self.lat,
             "lon":self.lon,"address":self.address }}
```

- L'app visualizza la schermata Mappa la schermata Mappa e popolata con i dati della lista buildings.

### 1.9 Logout

- L'utente dalla schermata Home o dalla schermata Mappa clicca su "Logout"
- L'app invia una richiesta POST sulla route `/logout`
- Il server termina la sessione dell'utente
- L'app visualizza la schermata Login

## 2.1 Registrazione, modifica e rimozione edificio

- L'utente con permessi Admin dalla schermata di gestione edifici, potrà creare e modificare edifici tramite un form che richiederà città, stato, via e numero.
- Una volta inviati i dati del form, il server tramite API GeoPY, controllerà la validità dell'indirizzo.
- Se l'indirizzo non è valido il server principale avviserà l'utente.
- Se l'indirizzo è valido, il server controlla se la città esiste nel DB, se sì lo assegnerà all'edificio creato/modifica, altrimenti lo creerà e per poi assegnarlo all'edificio.
- L'utente con permessi Admin sulla schermata Admin può eliminare gli edifici, una volta mandata la richiesta, il server controllerà che non ci siano sessioni attive nell'edificio.
- Se l'edificio ha sessioni attive assegnate, il server principale annullerà la procedura, mandando un avviso all'utente.
- Se l'edificio è libero, il server eliminerà a cascata le stanze assegnate e poi l'edificio stesso.

## 2.2 Creazione ed eliminazione zone

- L'utente con permessi di Admin, dalla schermata di gestione delle zone, potrà creare una nuova zona inserendo il nome della città e nazione nel form.
- Una volta inviati i dati del form, il server ricaverà tramite API "GeoPy", latitudine e longitudine della zona.
- Una volta ricavati i dati della zona, il server li salverà nel DB.
- L'utente con permessi di Admin, potrà eliminare le zone dalla schermata di gestione, una volta mandata la richiesta di eliminazione, il server controllerà se sono presenti edifici assegnati alla zona, se sì, il server annullerà la procedura avvisando l'utente dell'esito negativo.

## 2.3 Visualizzazione dashboard zone

- Gli utenti con permessi Admin possono visualizzare la dashboard di una zona, tramite la schermata di gestione zone.
- La dashboard contiene grafici del sensore di luce, dati meteo, andamento degli attuatori e consumi dei palazzi presenti nella zona durante il corso del mese attuale.
- Il grafico dei consumi è organizzato per palazzi della zona.

## 2.4 Visualizzazione dashboard edifici

- Gli utenti con permessi Admin possono visualizzare la dashboard di un edificio, tramite la schermata di gestione edifici.



- La dashboard contiene grafici del sensore di luce, andamento degli attuatori e consumi avvenuti durante il corso del mese attuale.
- Il grafico dei consumi è separato in intensità LED e riscaldamento.

### 2.5 Visualizzazione dashboard stanze

- Gli utenti con permessi Admin possono visualizzare la dashboard di una stanza, tramite la schermata di gestione stanze.
- La dashboard contiene grafici del sensore di luce, andamento degli attuatori e consumi della stanza avvenuti durante il corso del mese attuale.
- Il grafico dei consumi è separato in intensità LED e riscaldamento.

### 2.6 Visualizzazione Telegram key

- L'utente con permessi Admin, può accedere alla schermata chiamata "telegram", dove risiede il codice a 6 cifre assegnato.

### 2.7 Visualizzazione report Telegram

- L'utente tramite applicazione Telegram può contattare il bot **@unimore\_smartoffice\_bot** per ottenere report dei consumi degli ultimi 30 giorni
- Inizialmente l'utente dovrà identificarsi tramite comando /auth, il bot chiederà il codice a 6 cifre. Il bot prende in input il codice e chiederà tramite richiesta POST alla route /botauth del server principale, la validità del codice.
- Una volta autenticati, l'utente potrà richiedere il report dei consumi tramite comando /receivelastreport
- In caso di mancata autenticazione, sessione scaduta o codice errato, il bot telegram chiederà all'utente di identificarsi con un codice valido.

### 3.1 Assegnazione ruolo di admin

- Gli utenti con permessi di super-admin possono nominare altri utenti come admin.
- Una volta assegnato il ruolo admin, una chiave a 6 cifre per il bot telegram verrà generata.
- Gli utenti con permessi di super-user possono revocare il ruolo di Admin agli utenti e di conseguenza il codice a 6 cifre per il bot telegram.
- L'account iniziale denominato 'Admin' non potrà essere declassato.

### 3.2 Assegnazione ruolo di super-admin

- Gli utenti con permessi di super-admin possono nominare altri utenti come super-admin, l'assegnamento del super-admin comporta anche l'assegnamento del ruolo di Admin.
- Una volta assegnato il ruolo super-admin, una chiave a 6 cifre per il bot telegram verrà generata.
- Gli utenti con permessi di super-user possono revocare il ruolo di Admin agli utenti e di conseguenza il codice a 6 cifre per il bot telegram.
- L'account iniziale denominato 'Admin' non potrà essere declassato.

### 3.3 Visualizzazione tabella utenti

- Gli utenti con permessi di Admin possono visualizzare i dati del proprio account.
- Gli utenti con permessi di super-Admin possono visualizzare i dati di tutti gli account presenti nel DB.

### 3.4 Creazione professioni

- Gli utenti con permessi di Admin, possono visualizzare la lista di professioni esistenti nel DB.
- Gli utenti con permessi di super Admin, possono creare e modificare le professioni all'interno del DB, tramite form con campi nome e categoria.
- Gli utenti con permessi di super Admin possono eliminare le professioni esistenti nel DB, se esistono utenti con la professione assegnata, la procedura di eliminazione è annullata con messaggio di errore.

## 4.1 Monitor consumi

- Il Monitor di consumi, all'avvio vengono inizializzati i parametri necessari della città da monitorare, viene instaurata la connessione con il broker MQTT "Hive MQ". Inoltre verrà impostato uno scheduler che eseguirà la funzione di controllo a cadenza oraria.
- All'inizio di ogni ora, lo scheduler esegue la funzione di monitoraggio, controlla all'interno del DB se hanno avuto o stanno avendo luogo sessioni del Digital Twin, in caso positivo il Monitor calcolerà i consumi, controllerà che la soglia sia rispettata e salverà i consumi sul DB principale.
- In caso di sfioramento dei consumi, il Monitor ricaverà la quantità dello sfioramento e regolerà i consumi delle sessioni attive, inviando i nuovi dati degli attuatori sul topic `/smartoffice/building_/room_/actuators/#` e aggiornando i dati del Digital Twin. Una

volta regolato il consumo, il monitor calcolerà la soglia da rispettare per la prossima iterazione.

- Dopo 24 iterazioni, la soglia verrà impostata al valore iniziale.

### Test Case

**Temperatura esterna: 10 C°**

**Numero di stanze occupate: 3**

**Temperatura impostata dall'utente: 30 C°**

**Consumo base riscaldamento: 1500 W**

Seguendo la formula del consumo relazione alla differenza di temperatura tra interno ed esterno, ogni stanza consuma in media

$$\begin{aligned}\text{CONSUMO} &= [(T(\text{interna}) - T(\text{esterna})) \cdot 0.03 + 1] \cdot 1500 \text{ W} \\ &= [(30 - 10) \cdot 0.03 + 1] \cdot 1500 \text{ W} = \mathbf{2400 \text{ W}}\end{aligned}$$

Quindi il totale dei consumi/h della zona è 7200 W

Viene impostata una threshold giornaliera di 158.400 W

In ogni ora, il massimo consumo accettabile è  $158.400 / 24 = 6600 \text{ W}$

In ogni ora, quindi, verranno consumati  $7200 - 6600 = \mathbf{600 \text{ W}}$  oltre la soglia consentita.

Il decremento dei consumi ad ogni stanza dovrà essere  $600 / 3 = \mathbf{200 \text{ W}}$  per l'ora successiva al fine di bilanciare.

Applicando la formula inversa, (con il nuovo consumo stimato  $2400 - 200 = \mathbf{2200 \text{ W}}$  per ogni stanza) la nuova temperatura interna settata per ogni stanza sarà

$$T(\text{interna}) = \left[ \frac{2200}{1500} - 1 + 0.03 \cdot T(\text{esterna}) \right] \cdot \frac{1}{0.03}$$

Quindi ogni stanza raggiungerà per opera del monitor una temperatura di **25 C°** interni.

## 4.2 Storicizzazione dati

- Alla chiusura della sessione di un Digital Twin, il server Flask ricaverà i dati necessari per il re-training dell'AI.

- Si prendono i valori degli attuatori con maggiore durata, e i dati riguardanti la sessione, l'utente, la zona e il meteo, formando un JSON:

```
{'user_age','user_sex','user_task','room_id','building_id','date',  
'session_open_time','session_close_time','ext_temp',  
'ext_humidity','ext_light','user_temp','user_color','user_light'}
```

- I dati verranno salvati su un DB non relazionale di Firebase, pronti all'uso.

### 4.3 Re-training AI

- Il server principale, all'inizio del nuovo mese solare, comunica al server AI di effettuare il re-train delle reti neurali inviando una richiesta POST sulla route /systemUpdate
- Il server AI va in manutenzione impostando la variabile locale server\_status a 0, da questo momento in poi, fino alla fine del processo di aggiornamento, le richieste ricevute verranno elaborate dall'AI locale presente sul server principale
- Il server AI scarica dal DB Firebase tutte le sessioni relative al mese solare precedente, sotto forma di lista di JSON, mediante una richiesta GET al DB
- Il server AI analizza e normalizza i dati contenuti nelle sessioni, adattandoli ai formati consentiti dalle reti neurali utilizzate
- Il server AI salva su disco i nuovi valori per la normalizzazione, ottenuti a partire dalle sessioni scaricate creando due file NormValues.json e UniqueValues.json, in un path temporaneo /Data/OpData/Temp
- Il server AI triggera il retrain delle reti, salvando i modelli aggiornati nel path /AI/Temp
- Il server AI aggiorna il sistema sostituendo le precedenti reti ed i precedenti file con quelli appena elaborati
- Il server AI torna disponibile impostando la variabile server\_status ad 1