Informazioni sul documento

Task n.	**4.5**
Tipo	**SS**
Titolo	**REALIZZAZIONE DEL SISTEMA IOT**
Deliverable previsto nel progetto	**DOCUMENTAZIONE INTERNA ED ESTERNA DEL PRODOTTO REALIZZATO**
Partecipante/i	**ENGINFO CONSULTING SRL**

Sommario

Scopo del documento 3
Struttura del documento 3
Acronimi e termini chiave 3
1 Realizzazione della rete IoT nello spazio espositivo 4
1.1 Componenti utilizzati 4
1.2 I dispositivi Bluetooth 4
1.3 La comunicazione tra i dispositivi e gli algoritmi di gestione dei dati di navigazione 8
2 Realizzazione del portale server (museumsmartview.com) 13
2.1 L'organizzazione del portale 14
2.2 Il frontoffice del portale 15
2.3 Il backoffice del portale 17
2.4 Il database del portale 19
2.5 Le API Rest del portale 26
3 Panoramica funzioni operative del portale 30
3.1 Parametri di configurazione 30
3.2 Inizio visita 34
4 Conclusioni 36

Scopo del documento

Nel presente documento sono descritte le soluzioni implementate per il progetto Museum Smart View relativamente alla realizzazione di una rete IoT per il rilevamento dei percorsi effettuati dai visitatori di un sito espositivo e del portale server di gestione della rete IoT e dei dati da essa provenienti.

Struttura del documento

Il documento è organizzato in quattro capitoli due dei quali, in particolari, descrivono la realizzazione della rete IoT ed il portale server.

Acronimi e termini chiave

ACRONIMO	DESCRIZIONE
SPES	Spazio Espositivo
BROKER	Un broker di messaggi è un software che consente ad applicazioni, sistemi e servizi di comunicare tra loro e scambiare informazioni
BEACON	Dispositivo di trasmissione dati attraverso una rete senza fili
GATEWAY	Dispositivo di ricezione dati
IOT	Internet Of Things (Internet delle Cose)

1 Realizzazione della rete IoT nello spazio espositivo

A seguito della fase di analisi e progettazione si è passato alla realizzazione dell'infrastruttura IoT necessaria alla localizzazione indoor per gli spazi espositivi. A seguire la descrizione delle tecnologie e degli apparati utilizzati durante la fase di implementazione presso i nostri laboratori.

1.1 Componenti utilizzati

a. **Gateway**: Il Gateway è un dispositivo che svolge un ruolo fondamentale nel sistema IoT proposto per il sito museale. È responsabile di raccogliere i dati trasmessi dai dispositivi Beacon BLE, presenti nei badge o braccialetti indossati dai visitatori, e di inoltrarli al Server MQTT tramite la connessione WiFi. Il Gateway fa da intermediario di comunicazione tra i dispositivi Beacon e il Server MQTT, consentendo la trasmissione efficiente dei dati e garantendo una copertura estesa all'interno del museo.

b. **Beacon**: Il Beacon è un piccolo trasmettitore hardware Bluetooth Low Energy (BLE) che trasmette il proprio identificativo univoco a dispositivi elettronici, come i Gateway BLE-WiFi, che utilizzano lo stesso protocollo per la ricezione. I dispositivi Beacon sono integrati nei badge o braccialetti indossati dai visitatori. Il loro compito principale è quello di fornire informazioni sulla posizione dei visitatori, in modo che il Server possa elaborare i dati e offrire indicazioni di navigazione e contenuti personalizzati.

c. **Server MQTT**: Il Server MQTT (Message Queuing Telemetry Transport) è un broker di messaggistica leggero che agisce come punto centrale nel sistema di comunicazione tra i dispositivi Beacon e il Portale. Esso riceve i messaggi inviati dai Gateway BLE-WiFi e li distribuisce ai client sottoscritti a specifici "topic". Il Server MQTT è responsabile della gestione e dell'instradamento dei messaggi tra i dispositivi Beacon e il Portale, consentendo una comunicazione bidirezionale tra i componenti del sistema IoT.

d. **Portale**: Il Portale è il cuore del sistema proposto per il sito museale. Esso riceve i dati di navigazione dai Gateway BLE-WiFi tramite il Server MQTT, elabora le informazioni provenienti dai Beacon e alimenta il database con le informazioni necessarie per fini statistici, analitici e di assistenza alla navigazione del visitatore. Il Portale gestisce la localizzazione precisa dei visitatori utilizzando la tecnica di triangolazione basata sui dati raccolti dai Gateway. Inoltre, il Portale fornisce indicazioni di navigazione, contenuti multimediali e informazioni personalizzate ai visitatori tramite apposita applicazione o attraverso altri mezzi, migliorando l'esperienza complessiva della visita al museo.

1.2 I dispositivi Bluetooth

Durante la fase di realizzazione della rete IoT e del Portale Server sono stati selezionati i componenti più adatti. Sono valutate sul campo le tecnologie più efficienti e precise per la localizzazione dei visitatori, il trasferimento dei dati e l'interazione con il Portale.

Selezione dei componenti Beacon BLE: Sono stati testati diversi modelli di Beacon BLE disponibili sul mercato per identificare quelli che offrivano il migliore rapporto tra precisione, durata della batteria e dimensioni ridotte. Sono stati, quindi, selezionati i dispositivi Beacon più adatti per essere integrati nei badge o braccialetti da fornire ai visitatori.

Selezione dei Gateway BLE-WiFi: È stata fatta un'analisi delle opzioni di Gateway BLE-WiFi per determinare quelli che garantivano un'ampia copertura all'interno del museo e una connettività stabile

con il Server MQTT. La scelta dei Gateway ha preso in considerazione anche le capacità di gestire la comunicazione con i dispositivi Beacon e l'efficienza energetica.

Selezione del Server MQTT: Per garantire un'elevata affidabilità e una trasmissione efficiente dei dati, sono stati esaminati diversi broker MQTT. È stata scelta la soluzione più adatta che potesse gestire il flusso di messaggi provenienti dai Gateway e inoltrarli al Portale in modo veloce e sicuro.

Realizzazione del Portale: Il Portale è stato realizzato in base alle soluzioni digitali applicabili ad un sito museale. Sono state considerate le funzionalità richieste, come la gestione delle posizioni dei visitatori, la generazione di contenuti personalizzati e la raccolta di dati statistici. È stata scelta un'architettura IOT a tre strati per garantire la scalabilità e l'efficienza del sistema.

Test e ottimizzazione: Dopo l'implementazione del sistema, sono stati condotti test per verificare l'accuratezza della localizzazione dei visitatori e la corretta trasmissione dei dati tra i componenti. Sono state apportate eventuali ottimizzazioni per migliorare le prestazioni e l'esperienza complessiva del sistema.

L'obiettivo dello studio è stato quello di realizzare un sistema IoT affidabile, preciso ed efficiente che potesse fornire informazioni dettagliate e personalizzate ai visitatori durante la visita al museo. Grazie alla scelta oculata dei componenti e alla progettazione attentamente curata del sistema, si è raggiunta una soluzione tecnologicamente avanzata e in grado di migliorare notevolmente l'esperienza di visita e la gestione del sito museale.

Per la gestione del sistema IoT proposto, è stato utilizzato il **Gateway GL-S10**. Questo dispositivo si è dimostrato altamente affidabile in diverse esperienze e si è rivelato particolarmente adatto alle esigenze del progetto. Un aspetto fondamentale per il successo del sistema era la compatibilità con lo standard MQTT, necessario per ottenere i risultati desiderati e descritti in dettaglio in seguito.

Il Gateway GL-S10 è stato scelto per diversi motivi. Innanzitutto, offre una connettività stabile e affidabile, garantendo una comunicazione fluida e continua tra i dispositivi Beacon e il Server MQTT. La sua compatibilità con lo standard MQTT è un requisito cruciale, poiché il protocollo di messaggistica leggero è essenziale per la trasmissione efficiente dei dati tra i componenti del sistema.

Un altro fattore decisivo nella scelta del GL-S10 è la sua flessibilità e facilità di configurazione. Questo gateway è stato adattato alle specifiche esigenze del progetto e configurato in modo da consentire la raccolta e l'inoltro accurato dei dati provenienti dai dispositivi Beacon al Server MQTT.

Inoltre, la robusta costruzione e le dimensioni compatte del Gateway GL-S10 lo rendono adatto all'installazione in diverse aree del sito museale senza occupare troppo spazio o interferire con l'estetica dell'ambiente.

Complessivamente, la scelta del Gateway GL-S10 ha contribuito al successo del sistema IoT nel sito museale, garantendo una gestione efficiente dei dati, una comunicazione affidabile e un'esperienza di visita ottimizzata per i visitatori. Grazie alla sua affidabilità e compatibilità con lo standard MQTT, il GL-S10 ha svolto un ruolo chiave nel consentire la localizzazione precisa dei visitatori e il fornire informazioni personalizzate durante la visita al museo

La scelta dei dispositivi **Beacon Keyfob Tag** prodotti e distribuiti dalla **BlueUp** è stata motivata dalla necessità di garantire una localizzazione affidabile e precisa dei visitatori all'interno del sito museale. La maggiore accuratezza delle informazioni e la precisione del segnale offerte da questi beacon li hanno resi la scelta ideale per implementare i criteri di triangolazione descritti precedentemente.

La precisione dei dispositivi Beacon è fondamentale per il funzionamento del sistema di triangolazione, che si basa sulla misurazione delle distanze dai tre Gateway BLE-WiFi posizionati in ogni ambiente del museo. Solo attraverso una misurazione accurata delle distanze dai beacon, il Server di Portale può calcolare con precisione la posizione esatta di ciascun visitatore all'interno del sito espositivo.

I beacon Keyfob Tag di BlueUp sono stati selezionati dopo un'attenta valutazione di diverse opzioni disponibili sul mercato. La loro maggiore accuratezza e precisione del segnale hanno dimostrato di essere cruciali per ottenere risultati affidabili e consentire una navigazione efficace all'interno del museo.

L'utilizzo di dispositivi Beacon di alta qualità ha garantito una migliore esperienza di visita per i visitatori, fornendo indicazioni di navigazione precise, contenuti personalizzati e informazioni dettagliate sulle opere esposte. Inoltre, l'affidabilità dei beacon ha contribuito a ridurre al minimo possibili errori o imprecisioni nella localizzazione dei visitatori, migliorando ulteriormente l'efficienza e l'accuratezza del sistema IoT implementato nel sito museale.

Il Keyfob Tag è un beacon Bluetooth Low Energy alimentato a batteria, simile a un telecomando. Grazie all'hardware di seconda generazione basato sul SoC nRF52832, è stato possibile migliorare il consumo energetico, rendendolo più efficiente dal punto di vista energetico.

Questo dispositivo supporta contemporaneamente più tecnologie, tra cui iBeacon e Eddystone, offrendo un pieno supporto con GATT (Generic Attribute Profile) e EID (Ephemeral ID). Inoltre, supporta la modalità di emulazione di Quuppa, una tecnologia di localizzazione ad alta precisione. Il supporto per il nuovo pacchetto Safety garantisce un livello aggiuntivo di sicurezza nelle comunicazioni.

Il Keyfob Tag è un dispositivo versatile, adatto a molteplici applicazioni. Può essere utilizzato come badge identificativo per il personale o per tracciare e localizzare beni all'interno di un ambiente. Grazie all'accelerometro interno, il Keyfob Tag può essere configurato per entrare automaticamente in modalità a basso consumo quando non è in uso, permettendo di estendere la durata della batteria e di garantire un funzionamento continuo per un periodo più lungo.

La flessibilità e le caratteristiche avanzate del Keyfob Tag lo rendono una scelta ideale per il sistema IoT proposto per il sito museale. La maggiore efficienza energetica, la precisione del segnale e la capacità di supportare diverse tecnologie lo rendono perfetto per fornire una localizzazione affidabile e precisa dei visitatori, migliorando l'esperienza di visita e permettendo al sistema di fornire indicazioni di navigazione personalizzate e contenuti rilevanti in tempo reale.

Il funzionamento del sistema composto da Gateway MQTT e rete di beacon è descritto nella figura in basso:

I gateway vengono installati, per garantirne la triangolazione, in numero di tre in una stanza; in funzione del segnale inviato ai gateway, questi inviano la comunicazione ad un server MQTT impostato negli stessi.

1.3 La comunicazione tra i dispositivi e gli algoritmi di gestione dei dati di navigazione

Un server MQTT (Message Queuing Telemetry Transport) è un broker di messaggistica che facilita la comunicazione tra dispositivi utilizzando il protocollo MQTT. MQTT è un protocollo di messaggistica leggero e flessibile, progettato per dispositivi a risorse limitate e reti con banda ridotta, come quelle utilizzate nelle applicazioni IoT (Internet of Things).

Il server MQTT funge da intermediario tra i vari dispositivi client, consentendo loro di pubblicare (invio) e sottoscrivere (ricevere) messaggi su specifici "argomenti" (topics) di interesse. I messaggi inviati dai client possono essere letti da altri dispositivi che sono interessati allo stesso argomento. In questo modo, i dispositivi possono scambiarsi informazioni in modo efficiente e scalabile.

Un server MQTT gestisce anche la logica di abbonamento e pubblicazione dei messaggi, garantendo che i dati siano consegnati ai client appropriati. Questo tipo di architettura rende MQTT un'opzione popolare per applicazioni IoT, dove il risparmio energetico, la leggerezza del protocollo e l'affidabilità della comunicazione sono fondamentali.

Per quanto riguarda il server MQTT da utilizzare, è stata dimensionata una Macchina Virtuale Cloud con sistema operativo centOS e con il sistema MQTT mosquitto.

Mosquitto è un popolare server MQTT open-source e gratuito. È stato sviluppato dall'Eclipse Foundation ed è scritto principalmente in linguaggio C. Mosquitto fornisce un'implementazione leggera e scalabile del protocollo MQTT, rendendolo adatto per applicazioni IoT e altre situazioni in cui la leggerezza e l'efficienza sono cruciali.

Le caratteristiche principali di Mosquitto includono:

Broker MQTT: Mosquitto agisce come broker MQTT, consentendo ai dispositivi client di comunicare tra loro attraverso il protocollo MQTT.

- Supporto di QoS (Quality of Service): Mosquitto supporta tre livelli di QoS, ovvero QoS 0 (al più una volta), QoS 1 (almeno una volta) e QoS 2 (esattamente una volta). Questo consente ai client di stabilire il livello di affidabilità richiesto per la consegna dei messaggi.
- Sicurezza: Mosquitto offre supporto per l'autenticazione degli utenti e la crittografia SSL/TLS per proteggere la comunicazione tra client e broker.
- Sottoscrizione e pubblicazione: Mosquitto gestisce la logica di sottoscrizione e pubblicazione dei messaggi, garantendo che i dati vengano consegnati solo ai client interessati.
- Supporto multi-piattaforma: Mosquitto è disponibile per diverse piattaforme, compresi Linux, Windows, macOS e molti altri.

Grazie alla sua affidabilità, flessibilità e facilità di utilizzo, Mosquitto è ampiamente utilizzato nella comunità IoT e in molti altri contesti di comunicazione tra dispositivi.

Sul server dimensionato, quindi, lanciando il comando **mosquitto_sub -h 74.234.172.30 -v -t '#'** possiamo vedere l'elenco dei messaggi che vengono inviati dal gateway al server MQTT, che vengono inviati mediante messaggio json

{
"mac": "F2DA14E4ED4E",
"rssi": -87,
"ad": "0201061AFF4C000215ACFD065EC3C011E39BBE1A514932AC01000558DFC2",
"ts": 1690531411936
}

Nella stringa JSON, i campi hanno i seguenti significati:

"mac": Indirizzo MAC del dispositivo Bluetooth.

"rssi": Valore RSSI (Received Signal Strength Indicator) del segnale del dispositivo.

"ad": Dati dell'annuncio del dispositivo.

"ts": Timestamp (data e ora) in formato UNIX del momento in cui è stato ricevuto il messaggio.

Questi valori devono essere raccolti e processati da un client MQTT che ha il compito di leggerli ed instradarli ad un algoritmo che si interessa successivamente della localizzazione, svolta in base al segnale RSSI che, triangolato, fornisce la posizione del beacon all'interno del polo museale.

L'elaborazione dei valori ricevuti dai beacon è svolta tramite un servizio Node-RED installato sullo stesso server dove è i esecuzione il servizio MQTT.

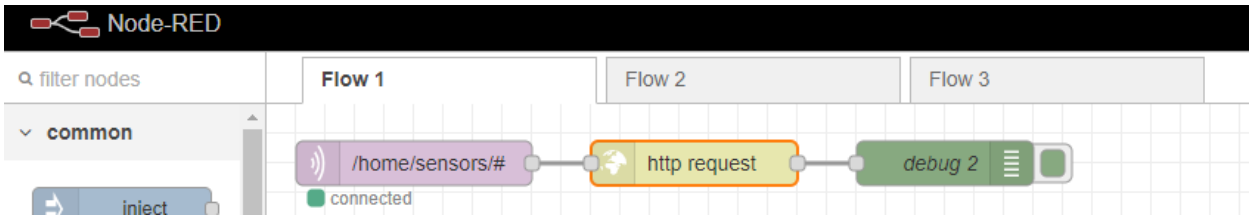
Node-RED è un ambiente di sviluppo visuale open-source per l'Internet delle cose (IoT) sviluppato dalla IBM Emerging Technology. Si basa sul motore JavaScript Node.js ed è utilizzato per creare flussi di lavoro (flow) in modo visuale per connettere dispositivi, servizi e applicazioni IoT in maniera semplice ed efficiente.

Le caratteristiche principali di Node-RED includono:

- Interfaccia visuale: Node-RED offre un'interfaccia grafica basata su browser che permette agli utenti di trascinare e collegare insieme nodi rappresentanti funzioni o servizi. Questa interfaccia visuale semplifica la progettazione e l'implementazione di flussi di dati e automatizzazioni.
- Nodi predefiniti: Node-RED include una vasta gamma di nodi predefiniti che coprono diverse funzionalità, come l'accesso ai dati da sensori, l'invio di notifiche, la gestione delle richieste web, l'interazione con database, l'elaborazione dei dati, ecc. Questi nodi possono essere estesi tramite moduli aggiuntivi.
- Ampio supporto di protocolli e servizi: Node-RED integra nativamente il supporto per MQTT, HTTP, WebSocket, CoAP e altri protocolli comuni utilizzati nelle applicazioni IoT. Inoltre, è possibile utilizzare nodi di terze parti per estendere ulteriormente le funzionalità di Node-RED e connettersi a servizi cloud e dispositivi IoT.
- Flussi condivisibili: I flussi creati in Node-RED possono essere facilmente esportati e condivisi con altre persone, semplificando la collaborazione e il riuso di soluzioni.
- Community attiva: Node-RED gode di una community attiva e di un ecosistema di sviluppatori che contribuiscono con nuovi nodi, soluzioni e supporto.

Node-RED è utilizzato in una vasta gamma di applicazioni IoT, dall'automazione domestica e industriale alla gestione dei dati e alla creazione di dashboard interattive. La sua natura visuale e la facilità di utilizzo lo rendono una scelta popolare per sviluppatori e appassionati di IoT.

Nel caso specifico, abbiamo sviluppato un flusso che legge dal server MQTT i valori inviati dai gateway e li invia come richiesta http ad un algoritmo di processo.



L'algoritmo esegue due operazioni fondamentali:

a. Salva il file json proveniente dal gateway
b. Processa questo file e scrive nelle tabelle di lavoro, se e solo se riceve almeno tre valori che posizionano il beacon nella stanza.

L'algoritmo è sviluppato in PHP e si basa su un ciclo di lettura del file passato dal gateway al server MQTT

```
foreach ($arrBeaconRilevati as $macBeacon => $numeroGatewayRilievo) {
   // Verifica se il numero di gateway rilevati per il beacon è maggiore o uguale a 3
   if (count($numeroGatewayRilievo) >= 3) {
      echo $orario . " - " . $macBeacon . " - " . count($numeroGatewayRilievo) . " - " .
reset($numeroGatewayRilievo)->idAmbienteEspositivo . "<br>";

      // Verifica se il beacon è già presente nell'array delle visite ambienti
      if (isset($arrVisiteAmbienti[$arrBeaconVisita[$macBeacon]])) {
         // Verifica se l'ambiente dell'attuale visita è diverso dall'ambiente del gateway rilevato
         if ($arrVisiteAmbienti[$arrBeaconVisita[$macBeacon]]->idAmbiente !=
reset($numeroGatewayRilievo)->idAmbienteEspositivo) {
            // Calcola data e ora di uscita e i secondi di permanenza dell'attuale visita
            $arrVisiteAmbienti[$arrBeaconVisita[$macBeacon]]->dataOraUscita = $orario;
            $arrVisiteAmbienti[$arrBeaconVisita[$macBeacon]]->secondiPermanenza =
strtotime($arrVisiteAmbienti[$arrBeaconVisita[$macBeacon]]->dataOraUscita) -
strtotime($arrVisiteAmbienti[$arrBeaconVisita[$macBeacon]]->dataOraAccesso);

            // Aggiorna i dati dell'attuale visita nel database
            $msvVisiteAmbientiDb->update($arrVisiteAmbienti[$arrBeaconVisita[$macBeacon]]);

            // Rimuovi l'attuale visita dall'array delle visite ambienti
            unset($arrVisiteAmbienti[$arrBeaconVisita[$macBeacon]]);

            // Crea una nuova visita ambienti con l'ambiente del gateway rilevato
            $msvVisiteAmbienti = new MsvVisiteAmbienti();

            $msvVisiteAmbienti->idVisita = $arrBeaconVisita[$macBeacon];
            $msvVisiteAmbienti->idSpes = $arrVisite[$msvVisiteAmbienti->idVisita]->idSpes;
            $msvVisiteAmbienti->idAmbiente = reset($numeroGatewayRilievo)->idAmbienteEspositivo;
            $msvVisiteAmbienti->dataOraAccesso = $orario;
            $msvVisiteAmbienti->secondiPermanenza = 0;

            // Inserisci la nuova visita ambienti nel database e aggiungi all'array delle visite ambienti
            $idVisitaAmbienti = $msvVisiteAmbientiDb->insert($msvVisiteAmbienti);
            $msvVisiteAmbienti->id = $idVisitaAmbienti['id'];
            $arrVisiteAmbienti[$arrBeaconVisita[$macBeacon]] = $msvVisiteAmbienti;
         }
```

```
    } else {
      // Il beacon non è presente nell'array delle visite ambienti, crea una nuova visita
      $msvVisiteAmbienti = new MsvVisiteAmbienti();

      $msvVisiteAmbienti->idVisita = $arrBeaconVisita[$macBeacon];
      $msvVisiteAmbienti->idSpes = $arrVisite[$msvVisiteAmbienti->idVisita]->idSpes;
      $msvVisiteAmbienti->idAmbiente = reset($numeroGatewayRilievo)->idAmbienteEspositivo;
      $msvVisiteAmbienti->dataOraAccesso = $orario;
      $msvVisiteAmbienti->secondiPermanenza = 0;

      // Inserisci la nuova visita ambienti nel database e aggiungi all'array delle visite ambienti
      $idVisitaAmbienti = $msvVisiteAmbientiDb->insert($msvVisiteAmbienti);
      $msvVisiteAmbienti->id = $idVisitaAmbienti['id'];
      $arrVisiteAmbienti[$arrBeaconVisita[$macBeacon]] = $msvVisiteAmbienti;
    }
  }
}
```

In questo modo, viene popolata la tabella msvVisiteAmbienti, con i record popolati in questo modo, che identificano univocamente la visita (mediante associazione di beacon fatta in precedenza)

id	id_spes	id_visita	id_ambiente	data_ora_accesso	data_ora_uscita	secondi_permanenza
3	11	13	11	2023-06-01 11:30:00	2023-06-01 11:31:00	60
4	11	14	11	2023-06-01 11:30:00	2023-06-01 11:31:00	60
5	12	14	12	2023-06-01 11:31:00	2023-06-01 11:34:00	180
6	11	13	12	2023-06-01 11:31:00	2023-06-01 11:38:00	420
7	12	14	11	2023-06-01 11:34:00	2023-06-01 11:43:00	540
8	11	13	11	2023-06-01 11:38:00	2023-06-01 11:51:00	780
9	12	14	12	2023-06-01 11:43:00	2023-06-01 12:07:00	1440
10	11	13	12	2023-06-01 11:51:00	2023-06-01 12:10:32	1172
11	12	14	11	2023-06-01 12:07:00	2023-06-01 12:19:03	723

2 Realizzazione del portale server (museumsmartview.com)

Il portale server, che è stato realizzato, rappresenta un fondamentale contenitore di soluzioni digitali, che apre le porte alla Digital Transformation di uno spazio espositivo. Agisce come il collettore principale e il centro di controllo delle diverse soluzioni distribuite all'interno del contesto espositivo.

Attraverso il backoffice, sarà possibile parametrizzare e monitorare le soluzioni, tra cui un'importante implementazione riguardante la rete IoT di localizzazione indoor. Questa funzionalità consente una gestione efficiente delle risorse e un'esperienza utente ottimizzata.

Il frontoffice, invece, funge da autentica vetrina delle soluzioni, permettendo la realizzazione della trasformazione digitale sia per musei che per qualsiasi altro spazio espositivo. Una gamma di soluzioni è a disposizione, adattandosi al livello di digitalizzazione iniziale e agli obiettivi che si desidera raggiungere.

La suite di strumenti forniti dal progetto MSV offre una vera e propria "cassetta degli attrezzi", che include la creazione di spazi virtuali e virtual tour immersivi (utilizzando tecnologie di realtà aumentata AR e realtà virtuale VR). Inoltre, permette di sviluppare app personalizzate per arricchire le visite, sfruttando il potenziale dell'interattività per facilitare l'apprendimento e coinvolgere i visitatori.

L'approccio gamification è un elemento chiave per stimolare l'interazione e rendere il percorso espositivo coinvolgente ed entusiasmante. Con l'elaborazione strategica di dati acquisiti sia nello spazio espositivo che online (sfruttando tecnologie come l'internet of things IoT, data analysis e web analysis), si può ottenere una comprensione approfondita delle dinamiche dei visitatori, consentendo un miglioramento continuo dell'esperienza offerta.

Inoltre, le attività di social media marketing e content strategy permettono di promuovere e diffondere la fruizione delle soluzioni digitali, aumentando la visibilità e l'interesse del pubblico.

Tutte queste soluzioni digitali prendono forma e si sviluppano attorno a un concept fondamentale: esaltare le caratteristiche uniche dello spazio espositivo attraverso una narrazione organica, basata su una strategia di storytelling. Questa narrazione avvincente funge da collante, permettendo di coordinare l'insieme dei servizi proposti in modo sinergico e coinvolgente. Il risultato è una trasformazione digitale completa ed entusiasmante, che arricchisce l'esperienza dei visitatori e valorizza le peculiarità dello spazio espositivo.

Le tecnologie utilizzate per il portale

Dal punto di vista dell'infrastruttura Hardware e Software per la realizzazione del portale ci si orienta su soluzioni open source leader di mercato e che siano affidabili e scalabili. In quest'ottica anche ospitare il portale su un'infrastruttura cloud rispetto a quella on-premise soddisfa sicuramente le ultime due aspettative, ma non solo:

- Flessibilità: Il cloud computing offre una maggiore flessibilità rispetto all'on-premise in termini di dimensionamento e di utilizzo delle risorse. Gli utenti possono facilmente scalare le risorse su e giù in base alle proprie esigenze, senza dover investire in hardware o software costosi.
- Costi inferiori: L'utilizzo del cloud computing può essere più economico dell'on-premise poiché i provider di servizi cloud offrono una tariffazione basata sull'uso effettivo delle risorse. Inoltre, il cloud computing elimina la necessità di acquistare, gestire e mantenere l'hardware e il software necessari per l'infrastruttura IT.
- Aggiornamenti automatici: Nel cloud computing, gli aggiornamenti del software vengono eseguiti automaticamente dai provider di servizi cloud, eliminando la necessità di aggiornamenti manuali o di gestione del software in loco.
- Disponibilità elevata: Il cloud computing offre un'alta disponibilità delle applicazioni e dei dati, grazie alla presenza di più copie dei dati e alle tecnologie di replicazione dei dati che riducono i rischi di perdita di dati o di interruzione del servizio.

- Sicurezza migliorata: Molti provider di servizi cloud offrono una sicurezza migliore rispetto all'on-premise grazie a controlli di sicurezza avanzati, monitoraggio costante, accesso limitato ai dati e capacità di gestione delle violazioni di sicurezza.
- Accessibilità remota: Il cloud computing consente l'accesso remoto ai dati e alle applicazioni, rendendo più facile il lavoro da casa o in mobilità.

Questi sono solo alcuni esempi dei vantaggi del cloud computing rispetto all'on-premise. Tuttavia, va tenuto presente che la scelta tra l'adozione di una soluzione cloud o on-premise può comunque dipendere dalle esigenze specifiche di ogni organizzazione o sito espositivo in questo caso.

Per la scelta degli strumenti di sviluppo ci si è orientati su linguaggi di programmazione scalabili e basati su uno sviluppo orientato agli oggetti (Object Oriented Programming) come PHP. Scrivere un'applicazione con un linguaggio orientato agli oggetti significa suddividerla in piccole parti ciascuna con un compito specifico. Questo semplifica lo sviluppo, il testing e la manutenzione dell'applicazione. Inoltre, questo consente di riutilizzare il codice già scritto in diverse parti dell'applicazione, risparmiando tempo e migliorando l'efficienza.

Un'altra caratteristica importante dell'approccio orientato agli oggetti è l'incapsulamento, che protegge i dati aumentando la sicurezza dell'applicazione. Inoltre, l'ereditarietà permette di semplificare la creazione di codice specializzato e migliorando la manutenzione e la modifica dell'applicazione.

In sintesi, l'utilizzo di un linguaggio di programmazione orientato agli oggetti semplifica lo sviluppo, migliora l'efficienza e la manutenzione dell'applicazione e aumenta la flessibilità del codice.

La scelta degli ambienti di runtime va su Apache http mentre la parte di persistenza dei dati la scelta ricade su Oracle MySQL Server per le sue caratteristiche che lo rendono uno dei database open source di maggiore utilizzo, ma anche per una serie di caratteristiche come ad esempio:

- Affidabilità e sicurezza: I database come MySQL Server sono progettati per garantire l'affidabilità e la sicurezza dei dati. Ciò significa che i dati sono protetti da accessi non autorizzati e che sono garantiti livelli di disponibilità e durabilità elevati.
- Scalabilità: I database come MySQL Server sono progettati per gestire grandi quantità di dati e un gran numero di richieste di accesso contemporaneamente. Ciò significa che è possibile aumentare le prestazioni del database in base alle esigenze dell'applicazione.
- Velocità: MySQL Server è noto per le sue prestazioni elevate e la velocità di accesso ai dati. Ciò rende il database ideale per applicazioni in cui la velocità di accesso ai dati è critica.
- Funzionalità avanzate: MySQL Server offre una vasta gamma di funzionalità avanzate, come l'indicizzazione dei dati, l'ottimizzazione delle query, la gestione delle transazioni e la replicazione dei dati, che possono migliorare le prestazioni e la scalabilità del database.
- Comunità e supporto: MySQL Server è un database open source con una vasta comunità di utenti e sviluppatori attivi. Ciò significa che è possibile trovare molte risorse, documentazione e supporto online per risolvere problemi e migliorare le prestazioni del database.

2.1 L'organizzazione del portale

Il portale è sarà quindi organizzato in un frontoffice ed un backoffice.

Il frontoffice sarà fruibile dai visitatori che potranno visionare le tecnologie che il progetto mette a disposizione per la digitalizzazione dello spazio espositivo.

Il backoffice sarà accessibile dagli amministratori del portale e, tramite accesso riservato con utenza e password, dagli addetti degli spazi espositivi che potranno effettuare le attività di configurazione e gestione dei propri parametri operativi.

2.2 Il frontoffice del portale

Il frontoffice del portale permetterà a chiunque acceda al sito museale di navigare liberamente senza necessità di effettuare alcun accesso in modo da permettere una semplice ed immediata esperienza utente.

A fronte dello studio effettuato, confrontandoci anche con risorse interne esperte di marketing e comunicazione, abbiamo attualizzato la grafica e l'albero di navigazione del portale rendendolo più *user-friendly* con l'obiettivo di garantire all'utente una più semplice fruizione dei contenuti, riscrivendo anche l'albero di navigazione in ottica SEO.

Il frontoffice sarà costituito da una serie di pagine vetrina/informative relative al sito museale e da un sistema di navigazione che porterà il visitatore alle pagine di contenuto sulle tecnologie che possono essere messe a disposizione degli spazi espositivi.

Il CMS scelto per garantire all'utente una migliore esperienza è Wordpress; il portale risponde all'indirizzo https://www.museumsmartview.com/

Le caratteristiche principali di WordPress includono:

- Facilità d'uso: WordPress è noto per la sua interfaccia utente intuitiva e facile da usare. Non è richiesta alcuna conoscenza di programmazione per gestire i contenuti e le pagine del sito.
- Personalizzazione: WordPress offre una vasta gamma di temi (themes) e plugin che consentono di personalizzare l'aspetto e le funzionalità del sito web. I temi definiscono l'aspetto grafico del sito, mentre i plugin aggiungono funzionalità extra.
- Gestione dei contenuti: WordPress consente di creare, modificare e pubblicare facilmente contenuti come articoli, pagine, immagini, video e altro ancora. L'editor di contenuti integrato è semplice da utilizzare e simile a un editor di testo.
- Community attiva: WordPress ha una vasta e attiva community di sviluppatori, designer e utenti che contribuiscono con nuovi temi, plugin, risorse e supporto. Ciò rende WordPress una piattaforma sempre in continua espansione e aggiornamento.
- SEO-friendly: WordPress è progettato per essere amico dei motori di ricerca, il che significa che è possibile ottimizzare facilmente il sito per migliorare la sua visibilità nei risultati dei motori di ricerca.
- Sicurezza: WordPress è impegnato a garantire la sicurezza dei siti web creati con la piattaforma. Tuttavia, è importante mantenere il sistema e i plugin sempre aggiornati per prevenire potenziali vulnerabilità.

Le pagine vetrina/informative saranno così organizzate:

- **Home**
  Pagina di accesso al sito organizzata prevalentemente con messaggi di benvenuto e link alle varie sezioni del sito. Conterrà video esplicativi e sottosezioni da cui poter facilmente accedere alle funzionalità del portale, oltre al link per l'area privata.

- **La mission**
  In questa sezione si ha l possiblità di spiegare le sue proposte ed I suoi obiettivi a lungo termine, o comunque le linee guida che ne ispirano le attività ed I contenuti espositivi.

- **La storia**
  Questa sezione permetterà al sito museale di raccontare la sua storia così come di mostrare la struttura con contributi fotografici.

- **Smart View – Soluzioni Digitali**

  In queste pagine si illustrerà ai visitatori I vantaggi di cui questi potranno godere, in termini di esperienza di visita sia fisica che virtuale, grazie alle tecnologie come la Realtà Aumentata e la

Localizzazione.
Potrà ancora descrivere I sistemi di monitoraggio che contribuiscono alla cura ed alla salvaguardia delle opere e dei visitatori.
Saranno inoltre illustrate le app di supporto alla visita e come ottenerle.

Questa sezione è il vero cuore del portale, e dovrà spiegare nei minimi dettagli come funzionano le varie tecnologie a supporto, in modo che il visitatore possa essere ampiamente informato circa le varie caratteristiche dei servizi.

Di grande e fondamentale importanza è poi una sezione dedicata al percorso da seguire per diventare un museum "smart", cioè le modalità di contatto e di consulenza per arrivare ad implementare le tecnologie nel museo che ne fa richiesta.

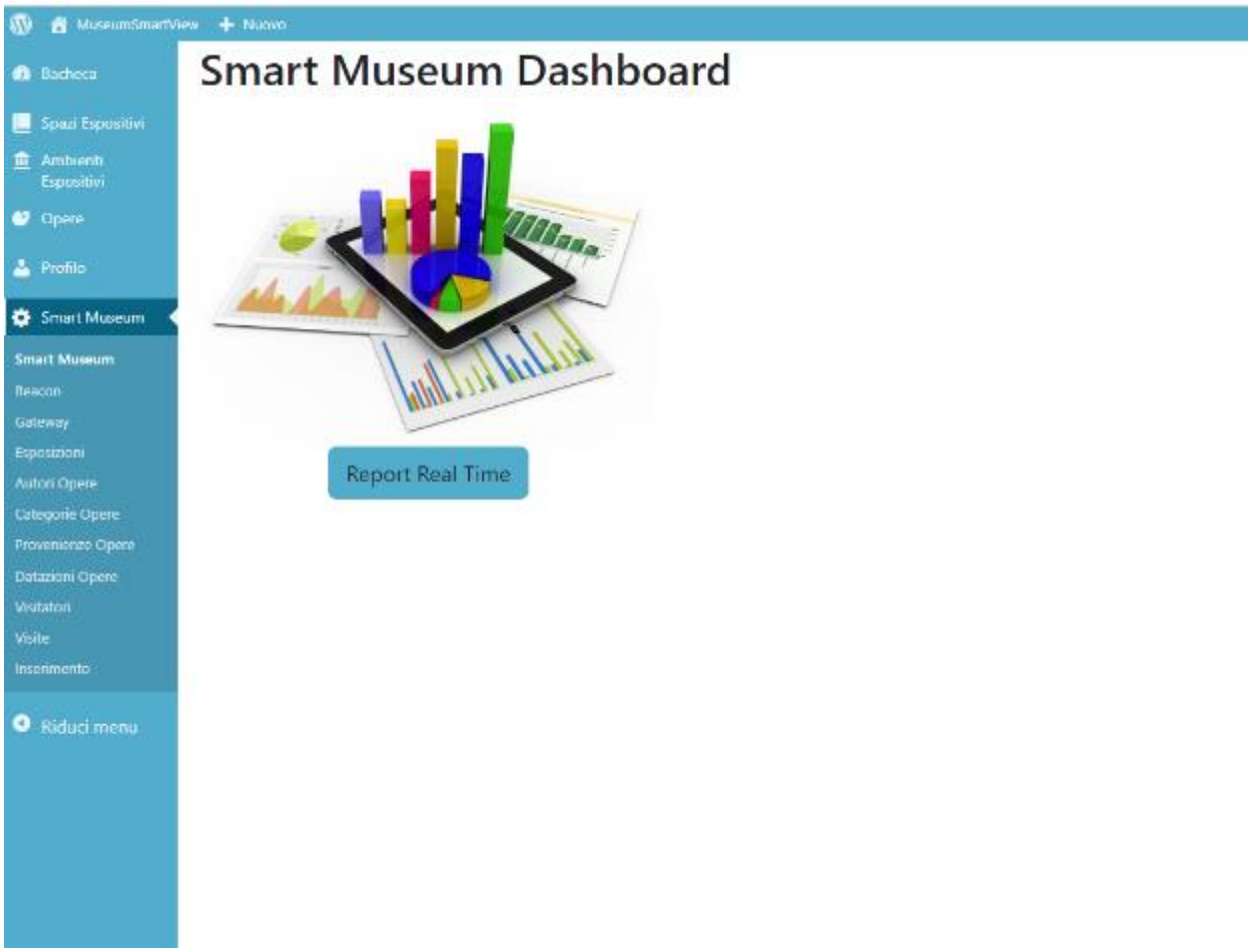
- **Notizie ed eventi**
  Sezione per la pubblicazione di Notizie ed eventi relativi al sito museale.
- **Gli Spazi Espositivi**
  La sperimentazione del progetto Museum Smart View verrà stata condotta in diversi musei e Spazi Espositivi, che in questa sezione verranno presentati al pubblico.

  Per ognuno di loro sarà presenta una breve scheda con foto e storia.
- **Contatti**
  Sezione tipo con tutti i riferimenti di contatto come mail elettronica, pagine social, telefoni, etc.
- **Area Riservata**
  Sezione dalla quale lo spes autorizzato potrà accedere

2.3 Il backoffice del portale

Il backoffice permette l'accesso ad una figura di amministratore per la gestione del portale stesso e ad una o più figure di operatori addette alla manutenzione dei contenuti. In particolare, oltre ai contenuti delle opere esposte nel sito museale, si potranno manutenere l'insieme delle tabelle di codifica degli ambienti espositivi nell'ambito del sito museale così come le codifiche delle tabelle relative ai Gateway ed ai Beacon Ble.

Il backoffice è stato sviluppato mediante programmazione e sviluppo di un apposito plugin wordpress chiamato **SmartMuseum**



Per quanto riguarda la definizione dei ruoli, si è proceduto come da progetto

Ruolo Amministratore: elenco delle tabelle / funzionalità a cui si è autorizzato:

- Tabella **Utenti**
  Lista degli utenti di manutenzione che possono accedere al backoffice;
- Tutte le altre tabelle / funzionalità previste dal ruolo Operatore.

Ruolo Operatore: elenco delle tabelle / funzionalità a cui si è autorizzati:

- Tabella **Opere**
  Contiene la lista delle opere che saranno visionabili dai visitatori;

In particolare le opere potranno essere correlate con la tabella degli Ambienti_Espositivi per selezionare l'ambiente nel quale l'opera si trova esposta.
I contenuti saranno di tipo multimediale come Foto, Video, Audio, etc.

- Tabelle **Autori**, **Categorie**, **Datazioni, Provenienze**
  Queste sono tabelle sono correlate alla tabella delle opere;
- Tabella **Ambienti_Espositivi**
  In questa tabella sono censiti tutti gli spazi adibiti all'esposizione delle opere nell'ambito del sito museale;
- Tabella **Esposizioni**
  La tabella contiene le associazioni tra opere ed ambienti espositivi così da gestire l'evento in cui un'opera sia spostata tra diversi ambienti espositivi;
- Tabella **Visitatori**
  La tabella contiene la lista degli utenti che visitano il sito museale, sia che si tratti di una visita in presenza oppure virtuale attraverso il portale web;
- Tabella **Visitatori_Profilo**
  La tabelle contiene I possibili profili dei visitatori a cui questi dichiarano di appartenere. Es: Appassionato d'arte, Studente, Accademico, etc.
- Tabella **Ble_Gateway**
  La tabella contiene l'anagrafica di tutti I Gateway Ble.
- Tabella **Ble_Beacon**
  La tabella contiene l'anagrafica di tutti I Beacon che vengono forniti ai visitatori (come badge o braccialetti) al fine di monitorare la localizzazione durante la vista.
- Tabella **Ambienti_Espositivi_Ble_Gateway**
  La tabella contiene tutte le associazioni tra gli ambienti espositivi ed I gateway ble. Tramite queste associazioni sarà possibile, rilevando il beacon assegnato all'utente, stabilire il percorso che questo sta effettuando.
- Archivio **Visite**
  Contiene la lista delle visite effettuate da tutti I visitatori che visitano il museo. La visita potrà essere stata in presenza o virtuale.
  L'archivio è alimentato automaticamente.
- Archivio **Visite_Ambienti**
  Contiene la lista degli ambienti espositivi in cui il visitatore si sofferma;
  L'archivio è alimentato automaticamente.
- Archivio **Visite_Opere**
  Contiene la lista delle opere presso le quali il visitatore si sofferma;
  L'archivio è alimentato automaticamente.

2.4 Il database del portale

Per la persistenza dei dati è stato confermato l'utilizzo di MySQL così come individuato in fase di analisi e progettazione dell'architettura.

MySQL è un sistema di gestione di database relazionale (RDBMS) open-source, sviluppato da Oracle Corporation. È uno dei database più popolari al mondo e viene utilizzato ampiamente per gestire dati strutturati in applicazioni web, software aziendali, siti web dinamici, e molte altre applicazioni.

Ecco alcuni punti chiave riguardanti MySQL:

Struttura dei dati: MySQL organizza i dati in tabelle con righe e colonne. Ogni tabella ha uno schema che definisce i nomi delle colonne e i tipi di dati che possono essere memorizzati in esse. È possibile creare relazioni tra le tabelle mediante chiavi primarie e chiavi esterne.

Linguaggio di interrogazione: Per comunicare con il database, MySQL utilizza il linguaggio SQL (Structured Query Language). SQL consente di eseguire operazioni come l'inserimento, l'aggiornamento, la selezione e l'eliminazione di dati dal database.

Supporto di transazioni: MySQL supporta il concetto di transazioni, che permette di eseguire un gruppo di operazioni in modo atomico, cioè tutte le operazioni sono eseguite con successo o nessuna di esse viene eseguita. Questo garantisce l'integrità dei dati in caso di errori o interruzioni.

Scalabilità: MySQL può essere utilizzato su sistemi sia di piccole dimensioni che di grandi dimensioni. È altamente scalabile e può gestire un gran numero di utenti e dati. Può essere distribuito in configurazioni di database cluster per gestire carichi di lavoro più pesanti.

Velocità e prestazioni: MySQL è noto per le sue prestazioni elevate e tempi di risposta rapidi. Le operazioni di lettura e scrittura sono ottimizzate per garantire una buona velocità di esecuzione delle query.

Community e supporto: Essendo un progetto open-source, MySQL ha una vasta community di sviluppatori e utenti che contribuiscono al suo sviluppo e lo supportano. Esistono numerosi forum, documentazione e risorse online per ottenere aiuto e informazioni riguardanti MySQL.

Versioni: MySQL è disponibile in diverse versioni, tra cui MySQL Community Edition (versione gratuita), MySQL Enterprise Edition (versione a pagamento con funzionalità avanzate e supporto tecnico), e altre distribuzioni personalizzate fornite da terze parti.

In sintesi, MySQL è un potente sistema di gestione di database che offre affidabilità, velocità e flessibilità. Grazie alla sua popolarità e alla vasta community di supporto, è una scelta popolare per molti sviluppatori e organizzazioni che cercano un RDBMS affidabile per gestire i loro dati.

A seguire la rappresentazione SQL del database, così come impiegato nell'ambiente di produzione realizzato:

```
-- ----------------------------
-- Table structure for msv_ambienti_espositivi
-- ----------------------------
DROP TABLE IF EXISTS `msv_ambienti_espositivi`;
CREATE TABLE `msv_ambienti_espositivi`  (
  `id_ambiente` int(11) NOT NULL AUTO_INCREMENT,
  `id_spes` int(11) NOT NULL,
  `denominazione` varchar(128) CHARACTER SET latin1 COLLATE latin1_general_ci NOT NULL,
  `piano` varchar(1) CHARACTER SET latin1 COLLATE latin1_general_ci NULL DEFAULT NULL,
  `stanza` varchar(45) CHARACTER SET latin1 COLLATE latin1_general_ci NULL DEFAULT NULL,
```

```
  `ubicazione` varchar(128) CHARACTER SET latin1 COLLATE latin1_general_ci NULL DEFAULT NULL,
  PRIMARY KEY (`id_ambiente`) USING BTREE,
  INDEX `fkSpes`(`id_spes`) USING BTREE,
  CONSTRAINT `fkSpes` FOREIGN KEY (`id_spes`) REFERENCES `msv_spes` (`id_spes`) ON DELETE NO ACTION ON UPDATE NO ACTION
) ENGINE = InnoDB AUTO_INCREMENT = 18 CHARACTER SET = utf8 COLLATE = utf8_general_ci ROW_FORMAT = Dynamic;

-- ----------------------------
-- Table structure for msv_ambienti_espositivi_ble_gateway
-- ----------------------------
DROP TABLE IF EXISTS `msv_ambienti_espositivi_ble_gateway`;
CREATE TABLE `msv_ambienti_espositivi_ble_gateway`  (
  `id_ambiente_espositivo` int(11) NOT NULL,
  `id_gateway` int(11) NOT NULL,
  `rssi_max` int(10) NULL DEFAULT NULL,
  PRIMARY KEY (`id_ambiente_espositivo`, `id_gateway`) USING BTREE,
  UNIQUE INDEX `idx_gateway`(`id_gateway`) USING BTREE
) ENGINE = InnoDB CHARACTER SET = utf8 COLLATE = utf8_general_ci ROW_FORMAT = Dynamic;

-- ----------------------------
-- Table structure for msv_ble_beacon
-- ----------------------------
DROP TABLE IF EXISTS `msv_ble_beacon`;
CREATE TABLE `msv_ble_beacon`  (
  `id_beacon` int(11) NOT NULL AUTO_INCREMENT,
  `id_spes` int(11) NOT NULL,
  `mac_address` varchar(50) CHARACTER SET utf8 COLLATE utf8_general_ci NULL DEFAULT NULL,
  `uuid` varchar(128) CHARACTER SET latin1 COLLATE latin1_general_ci NOT NULL,
  `minor` int(11) NOT NULL,
  `major` int(11) NOT NULL,
  `battery` double NOT NULL,
  `note` varchar(255) CHARACTER SET latin1 COLLATE latin1_general_ci NULL DEFAULT NULL,
  PRIMARY KEY (`id_beacon`) USING BTREE,
  INDEX `id_spes`(`id_spes`) USING BTREE,
  CONSTRAINT `msv_ble_beacon_ibfk_1` FOREIGN KEY (`id_spes`) REFERENCES `msv_spes` (`id_spes`) ON DELETE RESTRICT ON UPDATE RESTRICT
) ENGINE = InnoDB AUTO_INCREMENT = 16 CHARACTER SET = utf8 COLLATE = utf8_general_ci ROW_FORMAT = Dynamic;

-- ----------------------------
-- Table structure for msv_ble_gateway
-- ----------------------------
DROP TABLE IF EXISTS `msv_ble_gateway`;
CREATE TABLE `msv_ble_gateway`  (
```

```
  `id_gateway` int(11) NOT NULL AUTO_INCREMENT,
  `id_spes` int(11) NOT NULL,
  `descrizione` varchar(128) CHARACTER SET latin1 COLLATE latin1_general_ci NOT NULL,
  PRIMARY KEY (`id_gateway`) USING BTREE,
  INDEX `id_spes`(`id_spes`) USING BTREE,
  CONSTRAINT `msv_ble_gateway_ibfk_1` FOREIGN KEY (`id_spes`) REFERENCES `msv_spes` (`id_spes`) ON DELETE RESTRICT ON UPDATE RESTRICT
) ENGINE = InnoDB AUTO_INCREMENT = 19 CHARACTER SET = utf8 COLLATE = utf8_general_ci ROW_FORMAT = Dynamic;

-- ----------------------------
-- Table structure for msv_esposizioni
-- ----------------------------
DROP TABLE IF EXISTS `msv_esposizioni`;
CREATE TABLE `msv_esposizioni`  (
  `id_esposizione` int(11) NOT NULL AUTO_INCREMENT,
  `id_opera` int(11) NOT NULL,
  `id_spes` int(255) NULL DEFAULT NULL,
  `id_ambiente` int(11) NOT NULL,
  `data_inizio` date NOT NULL,
  `data_fine` date NULL DEFAULT NULL,
  `visibile_web` enum('S','N') CHARACTER SET latin1 COLLATE latin1_general_ci NOT NULL DEFAULT 'S',
  PRIMARY KEY (`id_esposizione`) USING BTREE,
  INDEX `idx_opera`(`id_opera`) USING BTREE,
  INDEX `idx_ambiente_espositivo`(`id_ambiente`) USING BTREE,
  CONSTRAINT `msv_esposizioni_ibfk_1` FOREIGN KEY (`id_opera`) REFERENCES `msv_opere` (`id_opera`) ON DELETE RESTRICT ON UPDATE RESTRICT,
  CONSTRAINT `msv_esposizioni_ibfk_2` FOREIGN KEY (`id_ambiente`) REFERENCES `msv_ambienti_espositivi` (`id_ambiente`) ON DELETE RESTRICT ON UPDATE RESTRICT
) ENGINE = InnoDB AUTO_INCREMENT = 4 CHARACTER SET = utf8 COLLATE = utf8_general_ci ROW_FORMAT = Dynamic;

-- ----------------------------
-- Table structure for msv_opere
-- ----------------------------
DROP TABLE IF EXISTS `msv_opere`;
CREATE TABLE `msv_opere`  (
  `id_opera` int(11) NOT NULL AUTO_INCREMENT,
  `id_spes` int(10) NULL DEFAULT NULL,
  `codice_opera` varchar(255) CHARACTER SET latin1 COLLATE latin1_general_ci NULL DEFAULT NULL,
  `nome` varchar(45) CHARACTER SET latin1 COLLATE latin1_general_ci NOT NULL,
  `descrizione` text CHARACTER SET latin1 COLLATE latin1_general_ci NOT NULL,
  `id_autore` int(11) NULL DEFAULT NULL,
  `id_datazione` int(11) NULL DEFAULT NULL,
  `id_provenienza` int(11) NULL DEFAULT NULL,
```

```
  `id_categoria` int(11) NULL DEFAULT NULL,
  `materiali` varchar(128) CHARACTER SET latin1 COLLATE latin1_general_ci NULL DEFAULT NULL,
  `tecnica` varchar(128) CHARACTER SET latin1 COLLATE latin1_general_ci NULL DEFAULT NULL,
  `misure` varchar(128) CHARACTER SET latin1 COLLATE latin1_general_ci NULL DEFAULT NULL,
  `condizione_giuridica` varchar(128) CHARACTER SET latin1 COLLATE latin1_general_ci NULL DEFAULT
NULL,
  `storia_aneddoti` text CHARACTER SET latin1 COLLATE latin1_general_ci NULL,
  `bibliografia` text CHARACTER SET latin1 COLLATE latin1_general_ci NULL,
  PRIMARY KEY (`id_opera`) USING BTREE,
  INDEX `idx_autori`(`id_autore`) USING BTREE,
  INDEX `idx_datazione`(`id_datazione`) USING BTREE,
  INDEX `idx_provenienza`(`id_provenienza`) USING BTREE,
  INDEX `idx_categoria`(`id_categoria`) USING BTREE,
  INDEX `fk_opera_spes`(`id_spes`) USING BTREE,
  CONSTRAINT `fk_opera_autore` FOREIGN KEY (`id_autore`) REFERENCES `msv_opere_autori`
(`id_autore`) ON DELETE RESTRICT ON UPDATE RESTRICT,
  CONSTRAINT `fk_opera_categorie` FOREIGN KEY (`id_categoria`) REFERENCES `msv_opere_categorie`
(`id_categoria`) ON DELETE RESTRICT ON UPDATE RESTRICT,
  CONSTRAINT `fk_opera_datazioni` FOREIGN KEY (`id_datazione`) REFERENCES `msv_opere_datazioni`
(`id_datazione`) ON DELETE RESTRICT ON UPDATE RESTRICT,
  CONSTRAINT `fk_opera_provenienze` FOREIGN KEY (`id_provenienza`) REFERENCES
`msv_opere_provenienze` (`id_provenienza`) ON DELETE RESTRICT ON UPDATE RESTRICT,
  CONSTRAINT `fk_opera_spes` FOREIGN KEY (`id_spes`) REFERENCES `msv_spes` (`id_spes`) ON DELETE
RESTRICT ON UPDATE RESTRICT
) ENGINE = InnoDB AUTO_INCREMENT = 18 CHARACTER SET = utf8 COLLATE = utf8_general_ci
ROW_FORMAT = Dynamic;

-- ----------------------------
-- Table structure for msv_opere_autori
-- ----------------------------
DROP TABLE IF EXISTS `msv_opere_autori`;
CREATE TABLE `msv_opere_autori`  (
  `id_autore` int(11) NOT NULL AUTO_INCREMENT,
  `nome` varchar(128) CHARACTER SET latin1 COLLATE latin1_general_ci NOT NULL,
  `data_nascita` date NOT NULL,
  `luogo_nascita` varchar(128) CHARACTER SET latin1 COLLATE latin1_general_ci NOT NULL,
  `data_morte` date NULL DEFAULT NULL,
  `luogo_morte` varchar(128) CHARACTER SET latin1 COLLATE latin1_general_ci NULL DEFAULT NULL,
  PRIMARY KEY (`id_autore`) USING BTREE
) ENGINE = InnoDB AUTO_INCREMENT = 8 CHARACTER SET = latin1 COLLATE = latin1_general_ci
ROW_FORMAT = Dynamic;

-- ----------------------------
-- Table structure for msv_opere_categorie
-- ----------------------------
```

```
DROP TABLE IF EXISTS `msv_opere_categorie`;
CREATE TABLE `msv_opere_categorie`  (
  `id_categoria` int(11) NOT NULL AUTO_INCREMENT,
  `descrizione` varchar(128) CHARACTER SET latin1 COLLATE latin1_general_ci NOT NULL,
  PRIMARY KEY (`id_categoria`) USING BTREE
) ENGINE = InnoDB AUTO_INCREMENT = 3 CHARACTER SET = latin1 COLLATE = latin1_general_ci
ROW_FORMAT = Dynamic;

-- ----------------------------
-- Table structure for msv_opere_datazioni
-- ----------------------------
DROP TABLE IF EXISTS `msv_opere_datazioni`;
CREATE TABLE `msv_opere_datazioni`  (
  `id_datazione` int(11) NOT NULL AUTO_INCREMENT,
  `descrizione` varchar(128) CHARACTER SET latin1 COLLATE latin1_general_ci NOT NULL,
  `ordinamento` int(11) NOT NULL DEFAULT 0,
  PRIMARY KEY (`id_datazione`) USING BTREE
) ENGINE = InnoDB AUTO_INCREMENT = 4 CHARACTER SET = latin1 COLLATE = latin1_general_ci
ROW_FORMAT = Dynamic;

-- ----------------------------
-- Table structure for msv_opere_provenienze
-- ----------------------------
DROP TABLE IF EXISTS `msv_opere_provenienze`;
CREATE TABLE `msv_opere_provenienze`  (
  `id_provenienza` int(11) NOT NULL AUTO_INCREMENT,
  `descrizione` varchar(128) CHARACTER SET latin1 COLLATE latin1_general_ci NOT NULL,
  PRIMARY KEY (`id_provenienza`) USING BTREE
) ENGINE = InnoDB AUTO_INCREMENT = 6 CHARACTER SET = latin1 COLLATE = latin1_general_ci
ROW_FORMAT = Dynamic;

-- ----------------------------
-- Table structure for msv_spes
-- ----------------------------
DROP TABLE IF EXISTS `msv_spes`;
CREATE TABLE `msv_spes`  (
  `id_spes` int(11) NOT NULL AUTO_INCREMENT,
  `codice_spes` varchar(255) CHARACTER SET latin1 COLLATE latin1_general_ci NULL DEFAULT NULL,
  `denominazione` varchar(128) CHARACTER SET latin1 COLLATE latin1_general_ci NOT NULL,
  `indirizzo` varchar(100) CHARACTER SET latin1 COLLATE latin1_general_ci NULL DEFAULT NULL,
  `citta` varchar(45) CHARACTER SET latin1 COLLATE latin1_general_ci NULL DEFAULT NULL,
  PRIMARY KEY (`id_spes`) USING BTREE
) ENGINE = InnoDB AUTO_INCREMENT = 15 CHARACTER SET = utf8 COLLATE = utf8_general_ci
ROW_FORMAT = Dynamic;
```

```
-- ----------------------------
-- Table structure for msv_visitatori
-- ----------------------------
DROP TABLE IF EXISTS `msv_visitatori`;
CREATE TABLE `msv_visitatori` (
  `id_visitatore` int(11) NOT NULL AUTO_INCREMENT,
  `ip_visitatore_virtuale` varchar(45) CHARACTER SET latin1 COLLATE latin1_general_ci NULL DEFAULT NULL,
  `email` varchar(512) CHARACTER SET latin1 COLLATE latin1_general_ci NULL DEFAULT NULL,
  `sesso` enum('M','F') CHARACTER SET latin1 COLLATE latin1_general_ci NULL DEFAULT NULL,
  `titolo_studio` varchar(255) CHARACTER SET latin1 COLLATE latin1_general_ci NULL DEFAULT NULL,
  `data_registrazione` date NULL DEFAULT NULL,
  `eta` int(11) NULL DEFAULT NULL,
  `origine` enum('Museo','Web') CHARACTER SET latin1 COLLATE latin1_general_ci NOT NULL,
  `nazione` varchar(2) CHARACTER SET latin1 COLLATE latin1_general_ci NULL DEFAULT NULL,
  PRIMARY KEY (`id_visitatore`) USING BTREE
) ENGINE = InnoDB AUTO_INCREMENT = 84 CHARACTER SET = utf8 COLLATE = utf8_general_ci ROW_FORMAT = Dynamic;

-- ----------------------------
-- Table structure for msv_visite
-- ----------------------------
DROP TABLE IF EXISTS `msv_visite`;
CREATE TABLE `msv_visite` (
  `id_visita` int(11) NOT NULL AUTO_INCREMENT,
  `id_spes` int(11) NOT NULL,
  `id_visitatore` int(11) NOT NULL,
  `data_riferimento` date NOT NULL,
  `data_ora_inizio_visita` datetime(0) NOT NULL,
  `eta` int(3) NULL DEFAULT NULL,
  `data_ora_fine_visita` datetime(0) NULL DEFAULT NULL,
  `secondi_visita` int(11) NULL DEFAULT NULL,
  `id_beacon` varchar(45) CHARACTER SET latin1 COLLATE latin1_general_ci NULL DEFAULT NULL,
  `visita_web` enum('S','N') CHARACTER SET latin1 COLLATE latin1_general_ci NOT NULL,
  PRIMARY KEY (`id_visita`) USING BTREE,
  INDEX `idx_visitatori`(`id_visitatore`) USING BTREE,
  INDEX `id_visita`(`id_visita`) USING BTREE,
  INDEX `id_spes`(`id_spes`) USING BTREE,
  CONSTRAINT `msv_visite_ibfk_1` FOREIGN KEY (`id_spes`) REFERENCES `msv_spes` (`id_spes`) ON DELETE RESTRICT ON UPDATE RESTRICT,
  CONSTRAINT `msv_visite_ibfk_2` FOREIGN KEY (`id_visitatore`) REFERENCES `msv_visitatori` (`id_visitatore`) ON DELETE RESTRICT ON UPDATE RESTRICT
) ENGINE = InnoDB AUTO_INCREMENT = 89 CHARACTER SET = utf8 COLLATE = utf8_general_ci ROW_FORMAT = Dynamic;
```

```
-- ----------------------------
-- Table structure for msv_visite_ambienti
-- ----------------------------
DROP TABLE IF EXISTS `msv_visite_ambienti`;
CREATE TABLE `msv_visite_ambienti`  (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `id_spes` int(11) NOT NULL,
  `id_visita` int(11) NOT NULL,
  `id_ambiente` int(11) NOT NULL,
  `data_ora_accesso` datetime(0) NOT NULL,
  `data_ora_uscita` datetime(0) NULL DEFAULT NULL,
  `secondi_permanenza` int(11) NOT NULL,
  PRIMARY KEY (`id`) USING BTREE,
  INDEX `idx_id_visitatore`(`id_visita`) USING BTREE,
  INDEX `idx_id_ambiente`(`id_ambiente`) USING BTREE,
  INDEX `id_spes`(`id_spes`) USING BTREE,
  CONSTRAINT `msv_visite_ambienti_ibfk_1` FOREIGN KEY (`id_spes`) REFERENCES `msv_spes` (`id_spes`) ON DELETE RESTRICT ON UPDATE RESTRICT,
  CONSTRAINT `msv_visite_ambienti_ibfk_2` FOREIGN KEY (`id_ambiente`) REFERENCES `msv_ambienti_espositivi` (`id_ambiente`) ON DELETE RESTRICT ON UPDATE RESTRICT,
  CONSTRAINT `msv_visite_ambienti_ibfk_3` FOREIGN KEY (`id_visita`) REFERENCES `msv_visite` (`id_visita`) ON DELETE RESTRICT ON UPDATE RESTRICT
) ENGINE = InnoDB AUTO_INCREMENT = 39 CHARACTER SET = utf8 COLLATE = utf8_general_ci ROW_FORMAT = Dynamic;

-- ----------------------------
-- Table structure for msv_visite_opere
-- ----------------------------
DROP TABLE IF EXISTS `msv_visite_opere`;
CREATE TABLE `msv_visite_opere`  (
  `id_visite_opere` int(255) NOT NULL AUTO_INCREMENT,
  `id_visita` int(11) NOT NULL,
  `id_spes` int(10) NULL DEFAULT NULL,
  `id_ambiente` int(11) NOT NULL,
  `id_opera` int(11) NOT NULL,
  `data_ora_accesso` datetime(0) NOT NULL,
  `data_ora_uscita` datetime(0) NULL DEFAULT NULL,
  `secondi_permanenza` int(11) NOT NULL DEFAULT 0,
  PRIMARY KEY (`id_visite_opere`) USING BTREE,
  INDEX `idx_id_visitatore`(`id_visita`) USING BTREE,
  INDEX `idx_id_ambiente`(`id_ambiente`) USING BTREE,
  INDEX `idx_opera`(`id_opera`) USING BTREE,
  CONSTRAINT `msv_visite_opere_ibfk_1` FOREIGN KEY (`id_visita`) REFERENCES `msv_visite` (`id_visita`) ON DELETE RESTRICT ON UPDATE RESTRICT,
```

CONSTRAINT `msv_visite_opere_ibfk_2` FOREIGN KEY (`id_ambiente`) REFERENCES `msv_ambienti_espositivi` (`id_ambiente`) ON DELETE RESTRICT ON UPDATE RESTRICT,
CONSTRAINT `msv_visite_opere_ibfk_3` FOREIGN KEY (`id_opera`) REFERENCES `msv_opere` (`id_opera`) ON DELETE RESTRICT ON UPDATE RESTRICT
) ENGINE = InnoDB AUTO_INCREMENT = 10 CHARACTER SET = utf8 COLLATE = utf8_general_ci ROW_FORMAT = Dynamic;

2.5 Le API Rest del portale

Al fine di offrire servizi ad applicazioni esterne al portale, ma comunque integrate nella soluzione MSV, sono state realizzate un insieme di API Rest che permettono di inviare al portale l'informazione di inizio/fine visita da parte di un visitatore. Inoltre, tramite le API, è possibile notificare che l'utente si è soffermato presso una determinata opera all'interno dello spazio espositivo.

A seguire le specifiche delle API Rest realizzate.

Login

La procedura di login viene effettuata inserendo username e apikey; le username ed apikey sono fisse ed invariabili per il singolo spes.

Ad ogni chiamata, il sistema restituisce un token che dura un'ora e che viene refreshato ad ogni chiamata; l'ultimo token restituito è il token valido.

Se la chiamata non va a buon fine, il sistema restituisce un errore (success=0)

INPUT	{
	"username": "testSpes",
	"secret ": "123456"
	}

JSON OUTPUT	{
	"success": 1,
	"data": {
	"token": "62287cc9c9bccef85285b83f1e87ebd4325a18c5193f9767efe3032d4314eb9a67",
	"indirizzoIp": "195.60.190.240",
	"dataRichiesta": "2023-10-03 19:33:35",
	"dataScadenza": "2023-11-03 19:33:35",
	}
	}

addVisita

Mediante questa chiamata, si invia al sistema l'inizio di una visita da parte di un visitatore; il sistema scrive un record che identifica il visitatore (identificato dalla mail) ed uno che identifica la sua visita.

Nel caso in cui la mail sia già esistente, il sistema aggiorna le informazioni del visitatore; l'id del beacon, in particolare, è l'id del singolo beacon che viene consegnato al visitatore, necessario per verificare l'ingresso e la permanenza negli ambienti.

Il token viene passato come *Bearer Token*

I campi sono tutti obbligatori.

Viene restituito un record Json con success=0 in caso di fallimento della chiamata, oppure con success=1 che riporta l'id del visitatore e l'id della visita

INPUT	{
	"codiceSpes":"Spes001",
	"email":"museum@test.it",
	"sesso":"M" ,
	"titoloStudio":"Laurea",
	"dataRegistrazione":"2023-03-27 11:00:00",
	"eta":20,
	"origine":"biglietto",
	"nazione":"Italia",
	"dataRiferimento":"2023-03-27",
	"dataOraInizioVisita":"2023-03-27 11:00:00",
	"idBeacon":1
	}

JSON OUTPUT	{
	"success": 1,
	"data": {
	"idVisitatore": 74,
	"idVisita": 12,
	}
	}

endVisita

Mediante questa chiamata, si invia al sistema la fine di una visita da parte di un visitatore; il sistema calcola i secondi di visita

Il token viene passato come *Bearer Token*

I campi sono tutti obbligatori.

Viene restituito un record Json con success=0 in caso di fallimento della chiamata, oppure con success=1 che riporta l'id del visitatore, l'id della visita e i secondi di permanenza

INPUT	{
	"idVisita":12 ,
	"dataOraFineVisita":"2023-03-27 12:00:00"
	}

JSON OUTPUT (esempio)	{
	"success": 1,
	"data": {
	"idVisitatore": 74,
	"idVisita": 12,
	“secondi”: 1254
	}
	}

addVisitaOpera

Mediante questa chiamata, il sistema associa ad una visita esistente la visita di una specifica opera.

Le opere vanno catalogate da backEnd e per ognuna sarà assegnato un codice opera univoco.

Il token viene passato come *Bearer Token*

I campi sono tutti obbligatori.

Viene restituito un record Json con success=0 in caso di fallimento della chiamata, oppure con success=1 che riporta l’id della visita dell’opera specifica

INPUT	{
	"idVisita":1,
	"codiceOpera":”Op001”,
	"dataOraAccesso":"2023-03-27 11:00:00"
	}

JSON OUTPUT	{
	"success": 1,
	"data": {
	"idVisitaOpera": 123,
	}
	}

endVisitaOpera

Mediante questa chiamata, si invia al sistema la fine di una visita di un opera da parte di un visitatore; il sistema calcola i secondi di visita

Il token viene passato come *Bearer Token*

I campi sono tutti obbligatori.

Viene restituito un record Json con success=0 in caso di fallimento della chiamata, oppure con success=1 che riporta l'id della visitaOpera ed i secondi di permanenza

INPUT	{
	"idVisitaOpera":123 ,
	"dataOraFineVisita":"2023-03-27 14:00:00"
	}

JSON OUTPUT (esempio)	{
	"success": 1,
	"data": {
	"idVisitaOpera":123 ,
	"secondi": 1254
	}
	}

3 Panoramica funzioni operative del portale

Ogni visitatore sarà rilevato all'interno del percorso, nel quale è presente la rete IoT, grazie ad un beacon che gli verrà fornito all'inizio della sua visita da un addetto dello spazio espositivo.

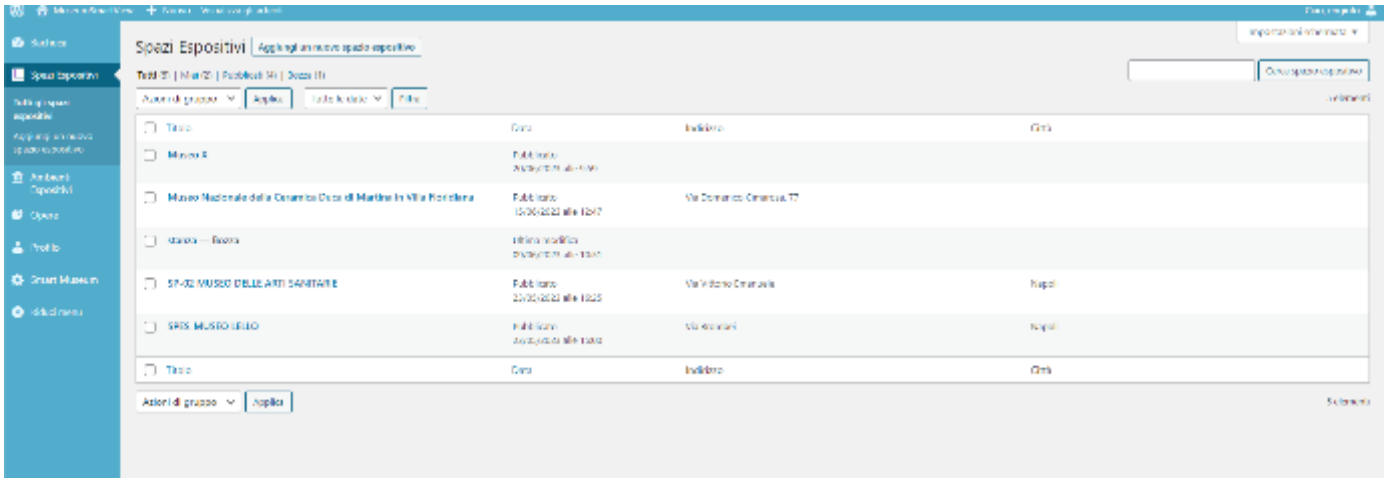
In questa fase, il beacon, che sarà dotato di un identificativo univoco, sarà associato al visitatore.
Questa associazione potrà avvenire attraverso due vie: la prima a carico di un'apposita funzione integrata nel backoffice del portale server; la seconda attraverso l'App di gestione dei contenuti con la quale il visitatore si registrerà prima di avviare la sua visita.

A seguire una serie di specifiche che descrivono le procedure disponibili dal backoffice del portale server per la gestione delle configurazioni e la registrazione dell'inizio visita per un visitatore.
Dal backoffice del portale, a seguito di login con utenza e password, gli addetti dei siti museali potranno configurare i dispositivi e gli ambienti all'interno dei quali i visitatori potranno spostarsi.

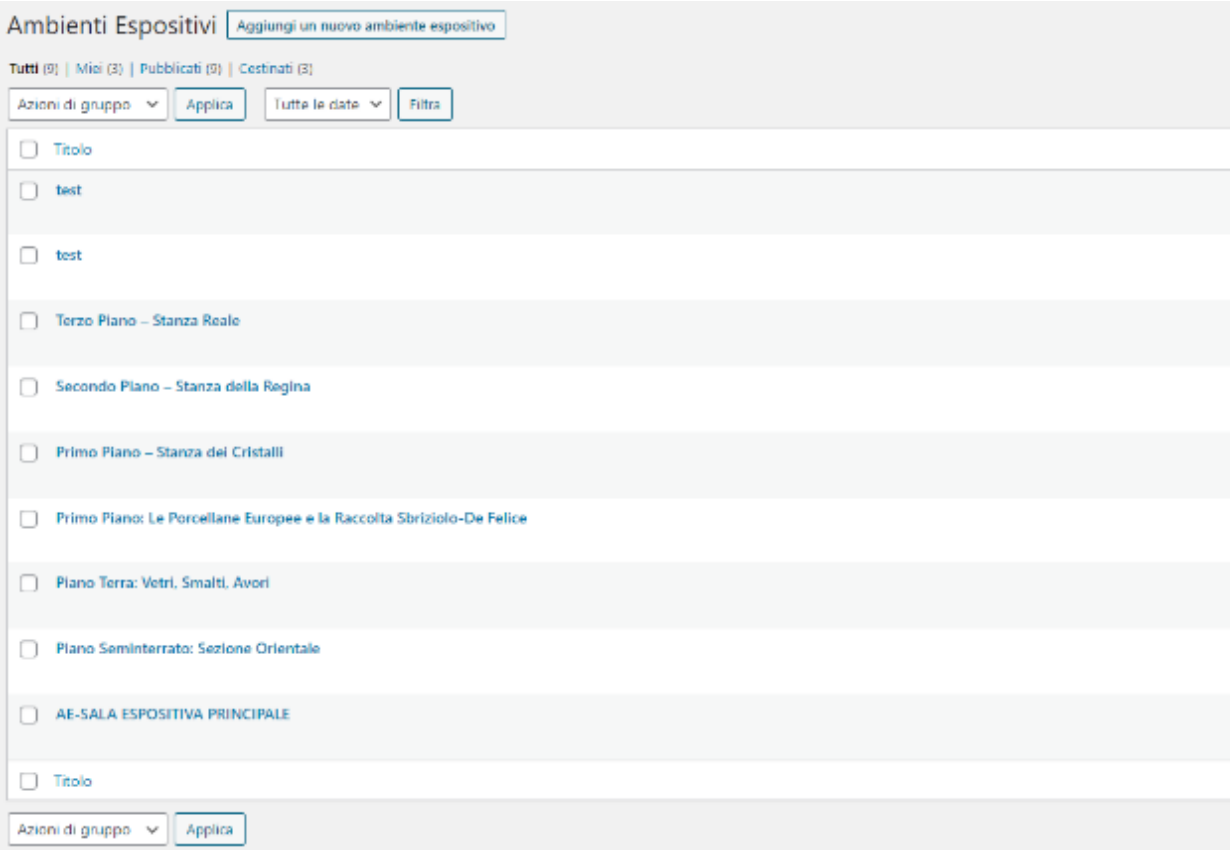
3.1 Parametri di configurazione

In questa sezione è possibile aggiungere, cancellare e modificare:

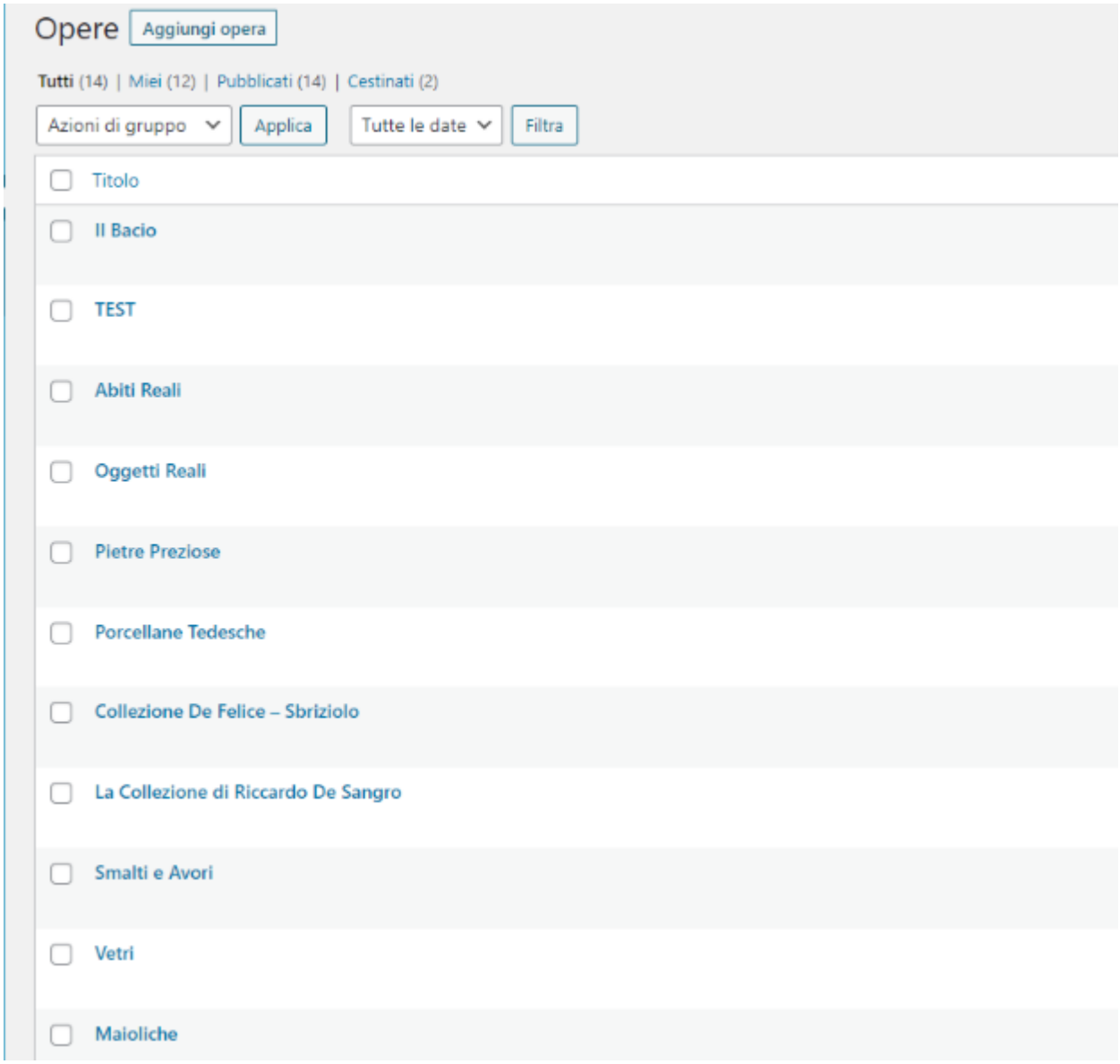
Spazi espositivi



Ambienti Espositivi



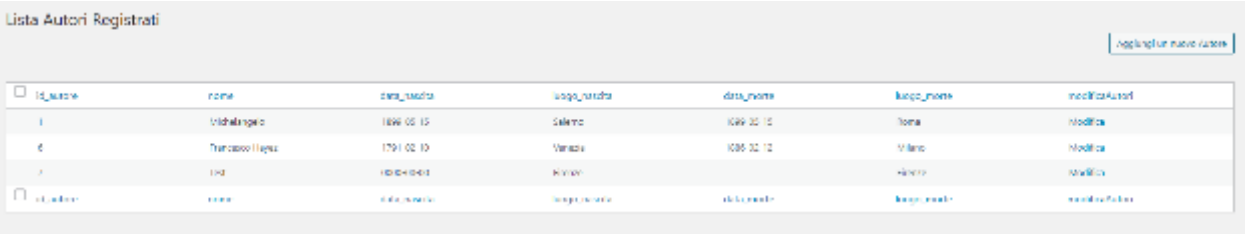
Opere


Opere
Aggiungi opera
Tutti (14) | Miei (12) | Pubblicati (14) | Cestinati (2)
Azioni di gruppo
Applica
Tutte le date
Filtra
Titolo
Il Bacio
TEST
Abiti Reali
Oggetti Reali
Pietre Preziose
Porcellane Tedesche
Collezione De Felice – Sbriziolo
La Collezione di Riccardo De Sangro
Smalti e Avori
Vetri
Maioliche

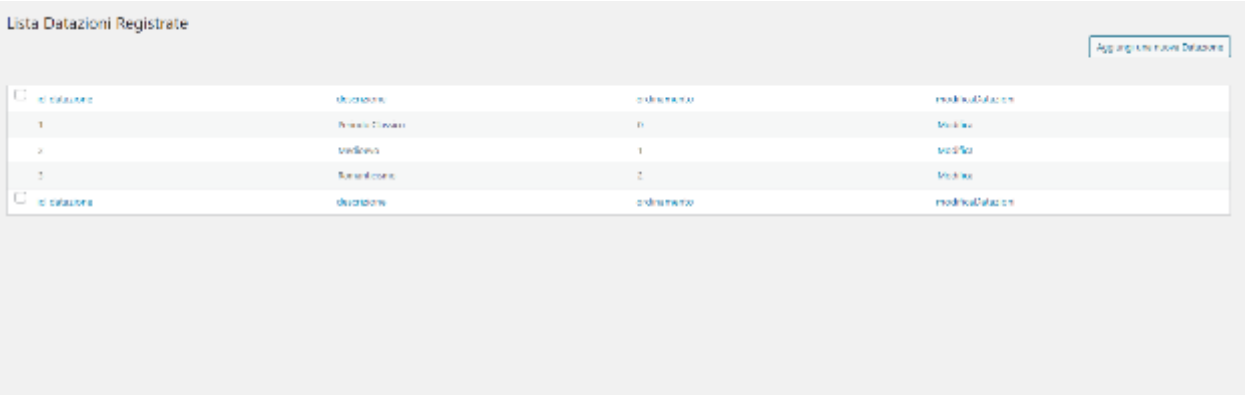
Beacon

Gateway

Autori

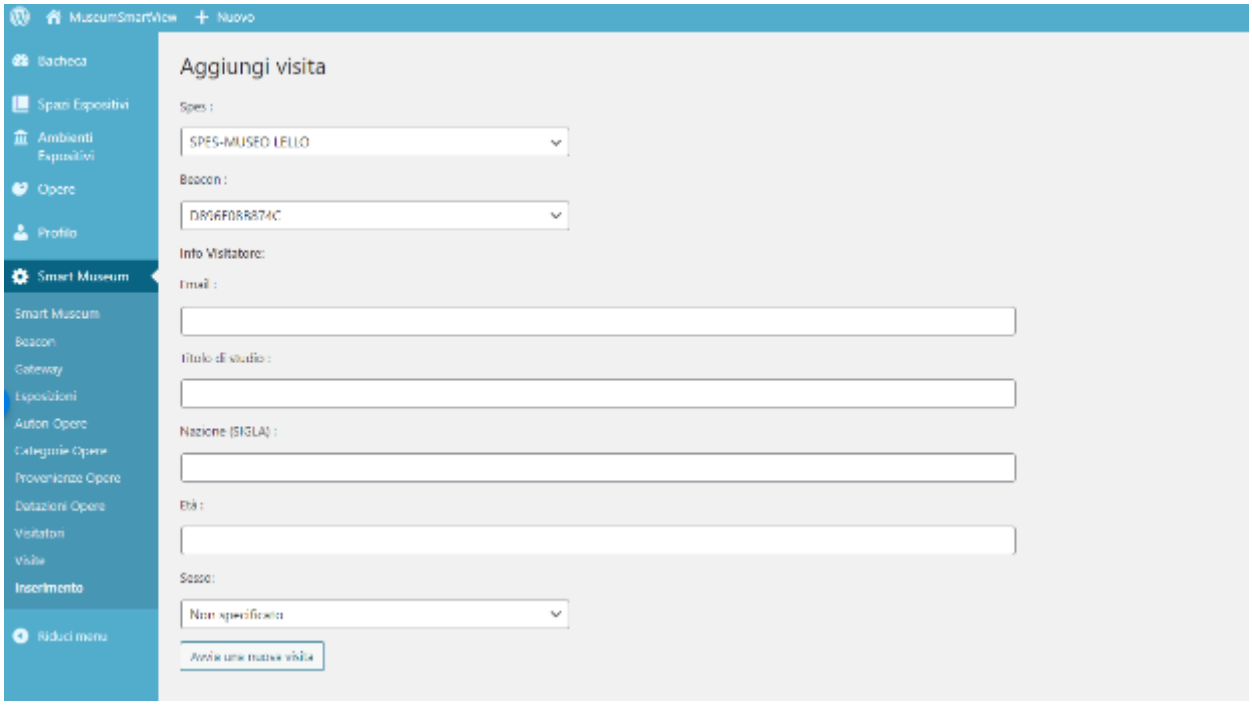


Attributi delle opere (categoria, provenienza, datazione)

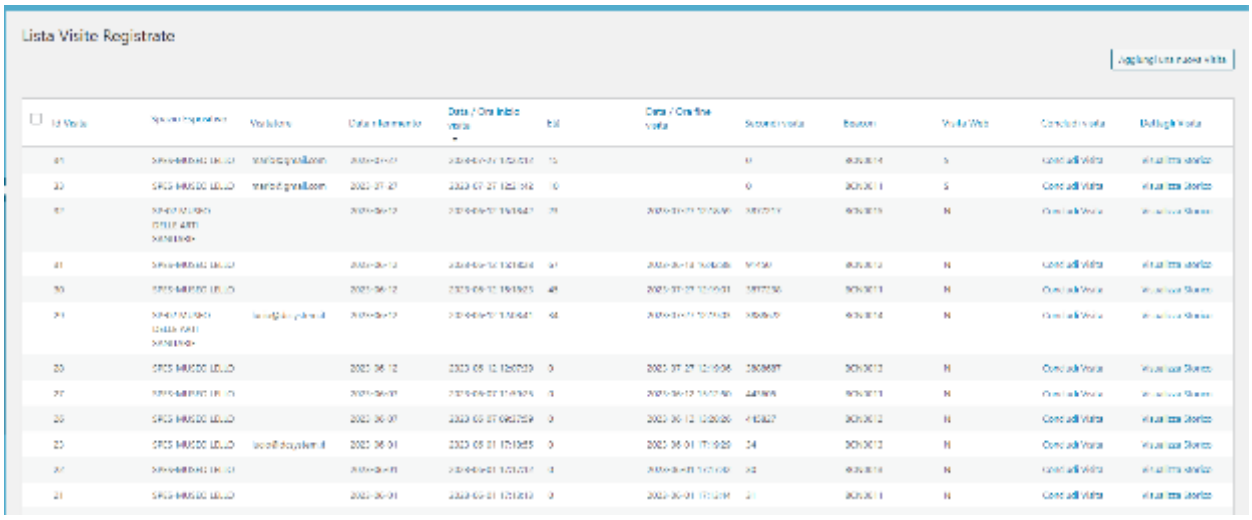


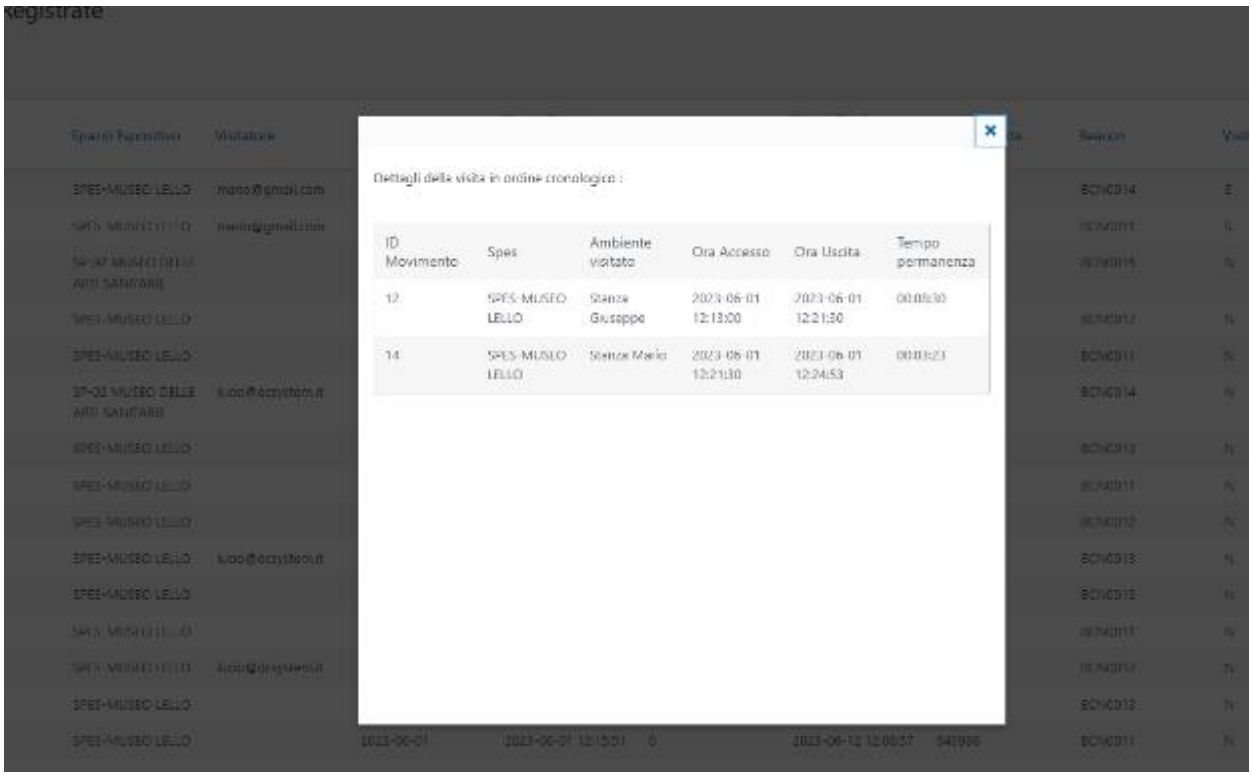
3.2 Inizio visita

La visita viene inserita mediante un apposito pannello, dove semplicemente si associa allo SPES un BEACON; il sistema in autonomia rileva quel beacon mediante flusso IoT e associa il visitatore alla stanza specifica.



Successivamente, il sistema, mediante il flusso IoT, visualizza le visite effettuate con lo storico delle varie stanze visitate e del tempo di permanenza.




Dettagli della visita in ordine cronologico :
ID Movimento
Spes
Ambiente visitato
Ora Accesso
Ora Uscita
Tempo permanenza
12
SPES-MUSEO LELLO
Stanza Giuseppe
2023-06-01 12:13:00
2023-06-01 12:21:30
14
SPES-MUSEO LELLO
Stanza Mario
2023-06-01 12:21:30
2023-06-01 12:24:53

4 Conclusioni

In conclusione, la realizzazione della rete IoT per la navigazione indoor, già dai test di laboratorio ha dimostrato di rappresentare un passo avanti significativo nell'ottimizzazione dell'esperienza degli utenti all'interno di spazi espositivi.

Attraverso la sinergia tra tecnologie all'avanguardia e la potenza dell'Internet delle Cose (IoT), possiamo essere in grado di offrire un’esperienza di visita altamente efficiente e che arricchisce la fruizione degli spazi interni.

fine del documento