



UNIVERSITÀ
DEGLI STUDI
FIRENZE

Scuola di Scienze Matematiche, Fisiche e Naturali
Corso di Laurea in Informatica

**GESTIONE INTELLIGENTE DEI SERVIZI IN
LUOGHI AFFOLLATI**

**EVENTS IN CROWDED PLACES: A SMART
SERVICE MANAGEMENT**

YURI BACCIARINI

ROSARIO PUGLIESE

Anno Accademico 2018-2019

INDICE

1	eServant	3
1.1	Panoramica	3
1.2	Aziende coinvolte	5
1.3	Casi d'uso	6
1.4	Back-end	7
1.4.1	Microservizi	8
1.4.2	Swagger (API)	9
1.4.3	Database (relazionale + documentale)	10
1.5	Front-end	11
2	Applicazione mobile	13
2.1	Framework Ionic	14
2.1.1	NgRx gestore stato	18
2.2	Social login	21
2.3	Navigazione impianto	21
2.4	Geolocalizzazione	23
2.4.1	QRcode	25
2.4.2	A-GPS	26
2.4.3	iBeacon	27
2.5	Chatbot	29
2.5.1	Brainstorming	31
2.5.2	Implementazione	31
3	Backoffice	33
3.1	Material Design	34
3.2	BEM	35
4	IoT	39
4.1	Bluetooth Low Energy	39
4.1.1	Gateway	39
4.1.2	iBeacon passivi	40
4.1.3	iBeacon Wearable	41
4.1.4	Smartphone	42
4.2	Conta persone	43
5	Sitografia	45
6	Ringraziamenti	47

1

ESERVANT

1.1 PANORAMICA

eServant [1] è un progetto di ricerca cofinanziato con fondi POR-CReO FESR 2014 - 2020 - Bandi per aiuti agli investimenti in ricerca, sviluppo e innovazione RSI. QUID Informatica S.p.A è Capofila di un raggruppamento di Aziende ed Organismi di ricerca finalizzato alla realizzazione del progetto.

Il focus del progetto è la gestione innovativa dei processi collegati ai grandi eventi (musicali, sportivi, del tempo libero o del lavoro) che hanno luogo all'interno di impianti e strutture, sia in termini di comunicazione e controllo, sia in termini di servizi innovativi per gli spettatori/utenti, andando a disegnare un progetto di impianto intelligente, inserito all'interno di una struttura di città intelligente, costruita per un cittadino consapevole, che valorizzi i temi della responsabilità sociale e civile e della sostenibilità.

Si tratta quindi di un insieme di processi, che sono gestiti in modo assolutamente innovativo, sino a creare un processo evoluto, costituito dalla piattaforma eServant, che viene industrializzato in una struttura hardware e software altamente replicabile.

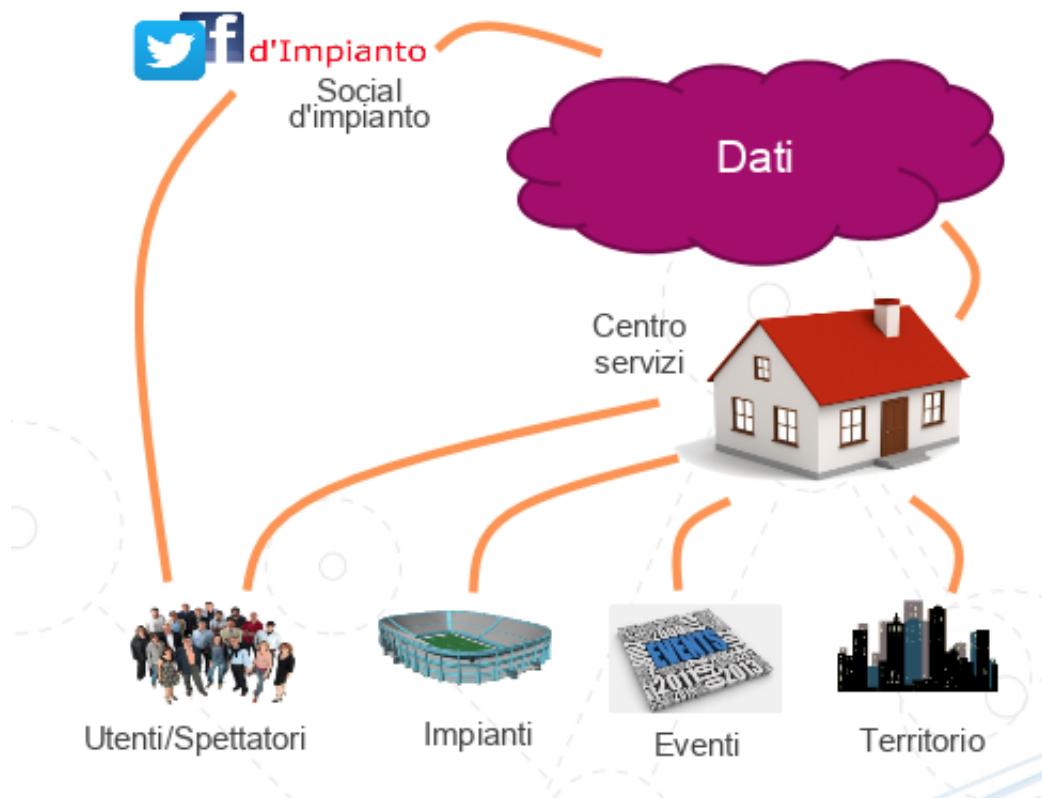
L'obiettivo operativo è quindi quello di progettare un modello di centro-servizi per la produzione, il controllo e la distribuzione di contenuti all'interno di grandi impianti, sportivi e non, in occasione di eventi specifici come mostrato nella Figura 1.

Il concetto di impianto al quale presta i suoi servizi il progetto eServant è molto ampio. Ci sono nel settore dello sport in Italia circa 150.000 impianti sportivi, di tutte le taglie. Ci sono gli impianti per il teatro

(1.162), i centri commerciali, retail park e factory outlet (956), migliaia di punti vendita della GDO, oltre 200 parchi nella sola Toscana, i comuni con decine di migliaia tra centri storici, aree monumentali, aree archeologiche, complessi monumentali e luoghi della cultura, oltre 57.000 plessi scolastici ed un numero incalcolabile di luoghi ove si esprime la socialità [2]. Non però una socialità «generalistica» ed indistinta, come nei social tradizionali tipo Facebook (dove si parla di tutto e di nulla), ma bensì una socialità caratterizzata da tre fattori unici e comuni: tante persone raccolte in un unico spazio all'interno del quale condividono un comune interesse. È questo un tipo di contesto non esplorato, al quale il progetto eServant prova a dare delle risposte e dei servizi mirati.

La sperimentazione del progetto eServant si è svolta presso l'impianto universitario PIN a Prato nel mese di Dicembre 2018.

Figura 1: eServant flow



1.2 AZIENDE COINVOLTE

Quid Informatica spa

Come capofila di progetto, Quid Informatica spa si è occupata di sviluppare e mettere a disposizione l'infrastruttura tecnologica per fruire i moduli sviluppati dai vari partner.

Sokom srl

Sokom srl si è occupata dei cablaggi e della disposizione dei vari access point necessari per fare comunicare i dispositivi IoT posti all'interno dell'impianto.

Magenta srl

Magenta srl si è occupata dei sensori IoT per il conteggio delle persone tramite hardware Raspberry ed in parallelo al recupero di dati da fonti open source, come Lamma e Open Toscana.

MICC

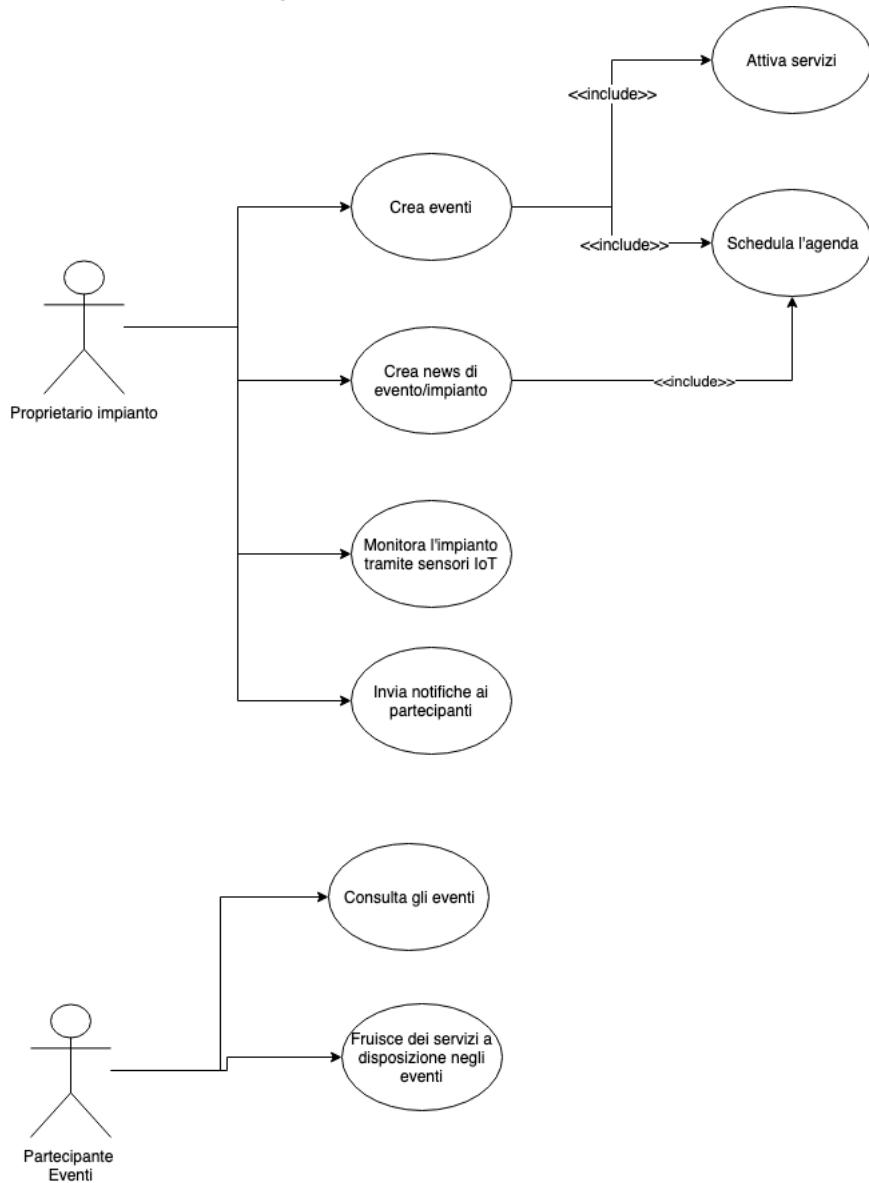
Il MICC invece si è occupato dell'ottenimento della densità di persone in alcuni punti strategici dell'impianto di sperimentazione. Ha inoltre sviluppato un modulo di mappe, il quale, insieme alla tecnologia precedente, permette di consigliare ai partecipanti eventuali percorsi alternativi in caso di affollamento. Infine, il MICC ha ideato anche un motore di affinità tra visitatori grazie al collegamento social degli stessi.

DIISM

Il dipartimento Ingegneria dell'informazione e Scienze Matematiche di Siena è fornitore e configuratore dei dispositivi iBeacon che tramite la tecnologia BLE (bluetooth low energy) ci ha permesso di ottenere la posizione delle persone all'interno dell'impianto di sperimentazione anche in assenza di GPS.

1.3 CASI D'USO

Figura 2: Casi d'uso eServant



La figura 2 descrive le funzioni e servizi offerti dal progetto eServant e percepiti dagli attori che interagiscono col sistema stesso [3].

Gli attori coinvolti nelle funzioni di eServant sono due:

- proprietario impianto;
- partecipante eventi;

Il proprietario dell'impianto gestisce gli eventi nella propria struttura, potendo nello specifico attivare servizi specifici e schedularne l'agenda. Come vediamo dalla figura 2 la funzione per schedulare l'agenda è inclusa anche nella creazione delle news riguardanti l'evento specifico o a carattere generico dell'impianto. Infine il proprietario dell'impianto può monitorare la propria struttura grazie a sensori IoT e mettersi in contatto con i partecipanti tramite notifiche push.

L'attore "Partecipante Eventi" invece, fruisce di tutti quei servizi configurati dal gestore dell'impianto; può quindi consultare gli eventi disponibili ed i relativi dettagli.

1.4 BACK-END

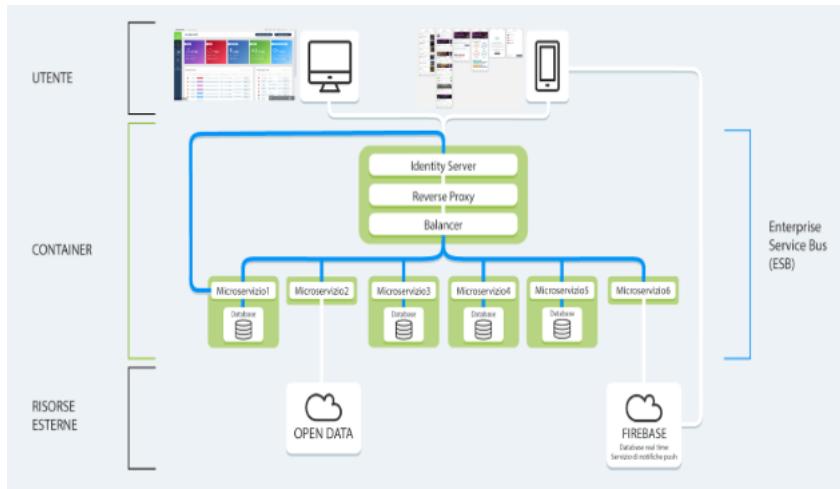
OSGI (Open Service Gateway Initiative)[4] è una specifica che permette di costruire applicazioni modulari a componenti (detti "bundle") e che introduce una programmazione Service Oriented, permettendo una separazione tra interfaccia ed implementazione molto più rigorosa di quella nativa Java. Il nucleo delle specifiche è un framework che definisce la gestione del modello del ciclo di vita del software, i moduli (chiamati bundle), un service registry e un ambiente di esecuzione. L'aumento della complessità in un prodotto software, sia esso embedded, client o server, richiede codice modulare ma anche sistemi che siano estensibili dinamicamente, molto più in applicazioni tipo quelle di eSERVANT. Il framework OSGI implementa un modello a componenti completo e dinamico cioè quello che manca all'ambiente Java. OSGi risolve molti dei problemi legati allo scarso supporto di Java nella modularità e nel dinamismo ed in tal senso OSGI è un framework che permette di fornire:

- un sistema modulare per la piattaforma Java;
- un sistema dinamico, che consente l'installazione, l'avvio, lo stop e la rimozione dei moduli a runtime, senza quindi necessitare di riavvii;
- un sistema orientato ai servizi, i quali possono essere dinamicamente registrati ed utilizzati nella macchina virtuale Java;

Per tutti i motivi indicati OSGI viene adottato all'interno dell'architettura di eSERVANT, come strumento per lo sviluppo delle applicazioni di backend di tutti i partner del progetto.

1.4.1 Microservizi

Figura 3: Architettura



Dalla figura 3 vediamo qual'è il flusso di una richiesta proveniente dall'utente fruitore del backoffice web o dell'applicazione mobile. Tutte le richieste HTTPS vengono intercettate prima di tutto dall'identity server che verifica tramite JWT token se l'utente è autorizzato a ottenere le risorse richieste. Successivamente, le richieste vengono smistate al corretto microservizio specializzato grazie al reverse proxy. Visto che i microservizi sono progettati per avere più repliche, il balancer permette di astrarle e bilanciare correttamente il carico di lavoro.

Ogni microservizio, come da figura 3, se necessario ha internamente un proprio database che nessun'altro oltre a lui può accederci.

L'ultima parte, quella delle risorse esterne, rappresenta tutti servizi fuori dalla rete dei microservizi; "open data" per esempio sono API pubbliche per il recupero di dati pubblici (es. Lamma per il meteo) mentre "Firebase" sono una serie di servizi cloud per push notification e real time database [5].

1.4.2 *Swagger (API)*

Swagger [6] è un insieme di specifiche e di strumenti che mirano a semplificare e standardizzare i processi di documentazione di API per servizi web RESTful, il tutto su di una piattaforma di tipo open-source. Il cuore di Swagger consiste in un file testuale (in formato sia YAML che JSON) dove sono descritte tutte le funzionalità di un'applicazione web e i dettagli di input e output in un formato studiato per essere interpretabile correttamente sia dagli umani che dalle macchine.

I vantaggi di questa standardizzazione sono molti, come una migliore e più condivisibile esplorazione delle funzionalità di un'applicazione, oltre che alla possibilità di generare codice client/server sfruttando direttamente i vincoli definiti nello schema. Infatti, i metadati presenti nel file forniscono informazioni sufficienti sia per generare il componente backend con le rotte http e la validazione degli input, sia la parte client che in modo automatico può adattarsi all'evoluzione delle API del backend. La generazione della parte server è sicuramente vantaggiosa in quanto è possibile concentrarsi direttamente sulla programmazione della logica di business, senza doversi preoccupare di come questa va ad interagire con il sistema al quale si interfaccia. Swagger è sostanzialmente composto da tre moduli:

1. **Swagger Editor:** è lo strumento per iniziare rapidamente nell'attività di progettazione di una nuova API o per la sua modifica in un potente editor che visualizza visivamente la definizione Swagger e fornisce feedback in tempo reale;
2. **Swagger Codegen:** è lo strumento che crea ed abilita le varie applicazioni al consumo delle API utilizzabili su una molteplicità di linguaggi di sviluppo (Java, Javascript Perl, PHP, Python, Ruby, Scala);
3. **Swagger UI:** è lo strumento che permette con un tool visuale il consumo delle API da parte delle varie applicazioni;

Per tutti i motivi indicati Swagger viene adottato all'interno dell'architettura di eSERVANT, come strumento per lo sviluppo delle varie API di interfacciamento verso le applicazioni dei partner.

1.4.3 Database (*relazionale + documentale*)

I due tipi di database usati nel progetto sono stati PostgreSQL e MongoDB, il primo relazionale e il secondo orientato a documenti.

PostgreSQL [7] è noto come il più avanzato sistema di gestione di dati open-source disponibile sul mercato. Si tratta di un DBMS (DataBase Management System), ossia di un vero e proprio insieme di componenti software in grado di permettere la creazione e la manipolazione di un database.

Ma PostgreSQL è anche un ORDBMS (Object Relational DataBase Management System), ossia un sistema che supporta un modello di database object oriented, che implementa oggetti, classi, ereditarietà e permette l'estensione del modello di dati con tipi di dati e metodi personalizzati. È altamente portatile, infatti è disponibile praticamente su tutte le distribuzioni di Linux, Unix, Windows, Mac OS, ecc.

Per tutti i motivi indicati PostgreSQL viene adottato all'interno dell'architettura di eSERVANT come il database di riferimento, utilizzato da tutte le applicazioni che necessitano del supporto di un database di tipo relazionale. Tutta la gestione massiva dei dati viene invece gestita attraverso il database non-SQL MongoDB.

MongoDB [8], invece, è un DBMS orientato alla gestione dei testi che ha come obiettivo quello di strutturare delle basi di dati testuali, in particolare con dati poco strutturati. Questo DBMS eredita il meccanismo di storage dal paradigma doc-oriented che consiste nel memorizzare ogni record come documento che non possiede caratteristiche predeterminate. Nei DB orientati ai documenti, si segue una metodologia differente rispetto al modello relazionale: si accorpano quanto più possibile gli oggetti, creando delle macro entità dal massimo contenuto informativo. Questi oggetti incorporano tutte le notizie di cui necessitano per una determinata semantica. Pertanto MongoDB non possiede uno schema e ogni documento non è strutturato, ha solo due chiavi obbligatorie: id, che serve per identificare univocamente il documento (è comparabile, semanticamente, alla chiave primaria dei database relazionali) e rev, che viene utilizzata per la gestione delle revisioni (ad ogni operazione di modifica infatti la chiave rev viene aggiornata).

Pertanto si può interrogare il DBMS anche per versioni del documento non recenti perché mantiene in memoria tutte le versioni. MongoDB gestisce anche la ridondanza dei dati per rimediare a malfunzionamenti.

Per evitare che lo sviluppatore acceda direttamente al database tramite linguaggio sql per PostgreSQL o tramite i metodi specifici per MongoDB, entra in gioco il framework Hibernate.

Hibernate [9] (talvolta abbreviato in H8) è una piattaforma middleware open source per lo sviluppo di applicazioni Java che fornisce un servizio di Object-relational mapping (ORM), ovvero che gestisce la rappresentazione e il mantenimento su database relazionale di un sistema di oggetti Java. Lo scopo principale di Hibernate è quello di fornire un mapping delle classi Java in tabelle di un database relazionale; sulla base di questo mapping Hibernate gestisce il salvataggio degli oggetti di tali classi su database. Si occupa inoltre del reperimento degli oggetti da database, producendo ed eseguendo automaticamente le query SQL necessarie al recupero delle informazioni e la successiva reistanziazione dell'oggetto precedentemente "ibernato" (mappato su database). Hibernate in pratica esonerà lo sviluppatore dal lavoro inerente la persistenza dei dati. Fortunatamente Hibernate gestisce una persistenza trasparente per "Plain Old Java Object".

1.5 FRONT-END

Lo sviluppo lato app è stato implementato usando il framework di sviluppo applicazioni mobile ibride Ionic. Questo framework utilizza di default l'engine Angular ed aggiunge ad esso una serie di funzioni "speciali" che permettono l'accesso a routine native, es. accensione camera dispositivo. Le funzioni "speciali" sono realizzate dalla libreria Cordova che implementa il ponte tra Javascript e il codice nativo Android/IOS.

Il punto di forza di Ionic è l'essere un framework ibrido. Scrivendo quindi un unico progetto in Javascript è possibile eseguirlo immediatamente su 3 piattaforme diverse: web, Android e IOS.

Angular 2 è un framework Javascript pensato per lo sviluppo di applicazioni di tipo web, sia per mobile che per desktop, basato sul progetto open-source prima di Angular JS e poi di Angular.

Angular da un lato esalta e potenzia l'approccio dichiarativo di HTML nella definizione dell'interfaccia grafica, dall'altro fornisce strumenti per la costruzione di un'architettura modulare e testabile della logica applicativa di un'applicazione. In questo senso Angular fornisce tutto quanto occorre per creare applicazioni moderne che sfruttano le più recenti tecnologie, come ad esempio le Single Page Application, cioè applicazioni le cui risorse vengono caricate dinamicamente su richiesta, senza necessità di ricaricare l'intera pagina, cosa particolarmente utile in una applicazione del tipo eSERVANT.

Le caratteristiche di Angular 2 sono [10]:

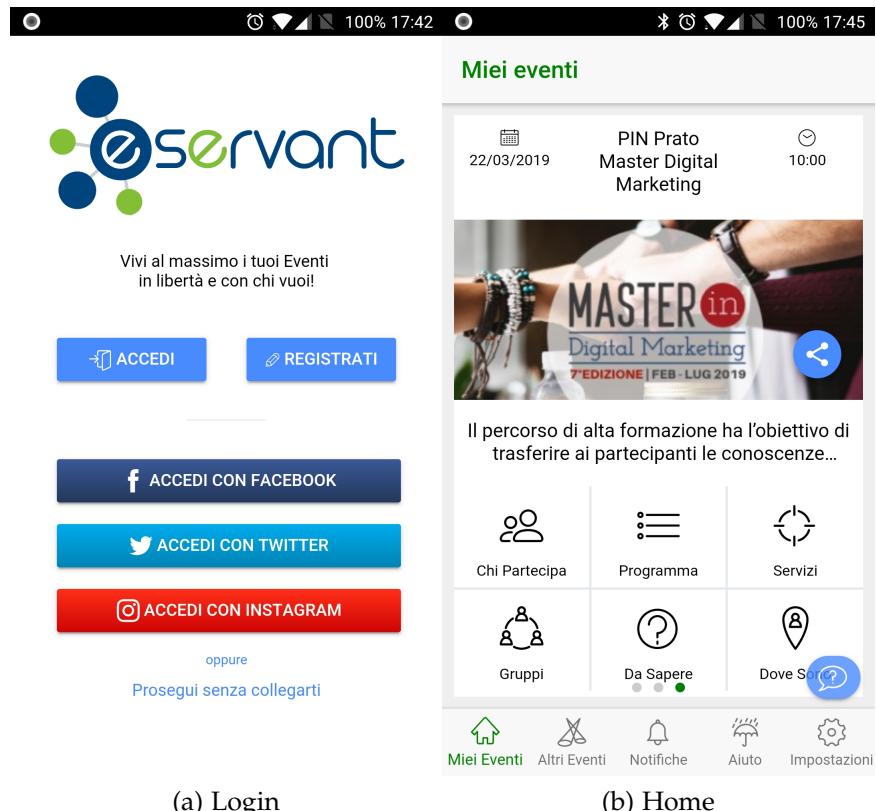
- **mobile first:** uno degli obiettivi del nuovo Angular è di proporsi per lo sviluppo di applicazioni Web sia per l'ambiente desktop che per il mobile. Infatti, Angular 2 supporta di default eventi touch e gesture e promette elevate prestazioni per assicurare una interazione fluida sui dispositivi mobile;
- **riduzione della curva di apprendimento:** Angular 2 ha ottimizzato e semplificato di molto le complessità insite nella realizzazione di interfacce sia per mobile che per desktop;
- **modularità:** l'architettura di Angular 2 è altamente modulare e favorisce la scrittura di applicazioni modulari e la maggior parte dei componenti del framework è opzionale ed è possibile sostituirli con altri di terze parti o sviluppati in proprio;
- **prestazioni:** un'attenzione particolare è stata posta sulle prestazioni del framework e sulla riduzione dei tempi di caricamento e bootstrapping delle applicazioni;

Per tutti i motivi indicati Angular 2 viene adottato all'interno dell'architettura di eSERVANT, come strumento per il progetto delle interfacce sia su mobile (e quindi per gli utenti/spettatori) che su desktop (e quindi dedicate alla centrale di controllo e gestione).

2

APPLICAZIONE MOBILE

Figura 4: Screenshot applicazione mobile eServant



(a) Login

(b) Home

L’interfaccia della piattaforma eSERVANT verso gli utenti finali è come da progetto una APP per mobile. I due screenshot nella figura 4 rappresentano, in ordine, la pagina di atterraggio per gli utenti che non hanno ancora effettuato l’accesso e la home dove si radudano tutti gli eventi seguiti dall’utente.

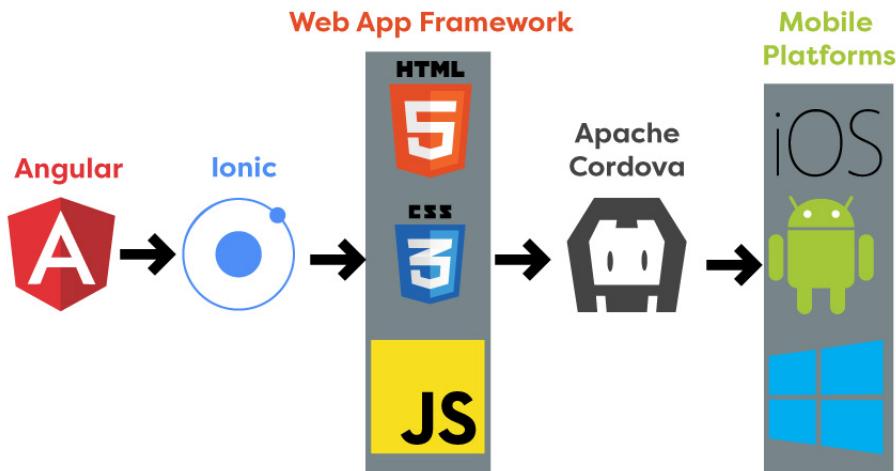
Per questo insieme a Sintra Consulting s.r.l. siamo andati a progettare in modo molto avanzato tutte le interfacce andando a quadrare requisiti di sistema/casi di uso con mockup montati all'interno di una vera e proprio demo dell'APP finale attesa. Tale strumento ci ha permesso di valutare non solo la user experience degli utenti, ma anche di poter fornire agli sviluppatori uno strumento specifico sul quale andare a traghettare la loro attività.

Il prototipo di APP è quindi stato creato andando a realizzare dei mockup sui sei temi di fondo:

- presentazione e riconoscimento;
- gestione evento e socialità;
- gestione delle funzioni di aiuto;
- gestione delle funzioni di mobilità;
- gestione delle funzioni di navigabilità e mappe;
- gestione del proprio profilo;

2.1 FRAMEWORK IONIC

Figura 5: Ionic framework [11]



Ionic è un HTML5 SDK lanciato nella sua prima versione nel 2013 e progettato per essere la base per lo sviluppo di app mobili ibride.

Sviluppare un'app mobile ibrida significa scrivere un unico code base per i due OS più famosi: IOS e Android.

Ionic viene distribuito come pacchetto installabile con npm, il package manager di Node.js. Node.js è un ambiente di run-time JavaScript open-source e multiplattaforma che esegue codice JavaScript all'esterno di un browser, lato server per esempio.

Ionic si installa tramite CLI Linux come di seguito [12]

```
npm install -g ionic
```

La struttura di Ionic è costituita da Angular, Apache Cordova (ex-PhoneGap) e SASS. Ciò consente la creazione di applicazioni mobili ricche di funzionalità che utilizzano esclusivamente tecnologie web come mostrato in figura 5.

Cordova

Apache Cordova, ex-PhoneGap e futuro Capacitor

Figura 6: Cordova [13]



Apache Cordova è un framework di sviluppo di applicazioni mobili che può essere utilizzato per creare app mobili multiplattaforma usando HTML5 e puro JavaScript. Per multiplattaforma intendiamo che il codice di un'applicazione può essere scritto una volta sola utilizzando HTML5 e JavaScript e può essere eseguito su diversi OS, come Android, iOS o Windows Mobile. La figura 6 raffigurara l'astrazione dell'implementazione nativa tramite l'uso di HTML, CSS e JS. Riporto di seguito un frammento di codice che permette di aprire la camera nativa del dispositivo mobile per scattare foto [14].

```
ionic cordova plugin add cordova-plugin-camera
npm install @ionic-native/camera
```

```
import { Camera, CameraOptions } from '@ionic-native/camera/
ngx';

const options: CameraOptions = {
  quality: 100,
  destinationType: this.camera.DestinationType.FILE_URI,
  encodingType: this.camera.EncodingType.JPEG,
  mediaType: this.camera.MediaType.PICTURE
}

constructor(private camera: Camera) { }

openCamera(){
  this.camera.getPicture(options).then((imageData) => {
    // imageData is either a base64 encoded string or a
    // file URI
    // If it's base64 (DATA_URL):
    let base64Image = 'data:image/jpeg;base64,' +
      imageData;
  }, (err) => {
    // Handle error
  });
}
```

Vediamo come in sole 16 linee di codice siamo riusciti ad aprire la camera nativa del sistema operativo per scattare una foto, ovviamente lo stesso codice viene compilato sia su IOS che su Android grazie a Cordova.

SASS

Scrivere CSS (Cascading Style Sheets) è fondamentale per descrivere in modo efficace il modo in cui gli elementi HTML devono essere visualizzati su una pagina Web per definire stili, design, layout e tutto ciò che è necessario per creare un sito Web sbalorditivo. Ma quando inizi a lavorare con siti grandi e complessi, potresti iniziare a chiederti se i CSS potrebbero essere migliori.

In questi casi entra in gioco SASS (Syntactically Awesome Stylesheets). SASS [15] è un pre-processore CSS che consente di utilizzare variabili,

operazioni matematiche, mixin, loop, funzioni, importazioni e altre funzionalità interessanti che rendono la scrittura CSS molto più potente. In un certo senso, si potrebbe pensare a SASS come a un linguaggio di estensione del foglio di stile perché estende le caratteristiche CSS standard introducendo i vantaggi di un linguaggio di programmazione di base. Quindi SASS compilerà il codice e genererà l'output CSS che un browser può comprendere.

Vediamo alcune funzioni SASS in confronto con CSS puro:

Variabili

<pre>// SASS \$font-stack: Helvetica, sans-serif \$primary-color: #333 body font: 100% \$font-stack color: \$primary-color</pre>	<pre>// CSS body { font: 100% Helvetica, sans -serif; color: #333; } .</pre>
---	---

Mixin

<pre>// SASS @mixin transform(\$property) -webkit-transform: \$property -ms-transform: \$property transform: \$property .box @include transform(rotate (30deg))</pre>	<pre>// CSS .box { -webkit-transform: rotate (30deg); -ms-transform: rotate(30 deg); transform: rotate(30deg); } .</pre>
--	---

Loop

<pre>// SASS \$class-slug: for !default @for \$i from 1 through 4 .#\${\$class-slug}-#{\$i} width: 60px + \$i .</pre>	<pre>// CSS .for-1 { width: 61px; } .for-2 { width: 62px; } .for-3 { width: 63px; } .for-4 { width: 64px; }</pre>
--	--

2.1.1 *NgRx gestore stato*

NGRX è un gruppo di librerie per Angular ispirate al pattern Flux sviluppato dal team di Facebook per gestire lo stato delle applicazioni client.

Lo scopo principale di questo pattern è fornire uno stato dell'applicazione prevedibile, basato su tre principi principali [16].

- **Unica sorgente di verità:** significa che l'intero stato dell'applicazione è memorizzato un'unica struttura dati come mostrato in figura 9;
- **Lo stato è read only:** lo stato non viene mai cambiato direttamente, solo tramite *action*. Le *action* descrivono cosa sta succedendo (possiamo vedere le *action* come richieste di ottenimento dati, rimozione, aggiornamento stato). Possiamo notare questa immediata differenza confrontando la figura 7 con la figura 8;
- **I cambiamenti sono fatti solo da funzioni pure:** quando un' *action* viene lanciata, essa verrà intercettata dai *reducer*. Questi *reducer*

(sono solo funzioni pure) ricevono un'azione e lo stato, quindi a seconda dell'azione inviata (di solito con un'istruzione switch), eseguono un'operazione e restituiscono un nuovo oggetto di stato. Lo stato in un'app redux è immutabile! Quindi, quando un *reducer* modifica qualcosa nello stato, restituisce un nuovo oggetto stato;

Possiamo rendersi conto dei tre principi fondamentali sopra citati confrontando la figura 7 con la figura 8.

Figura 7: App con gestore stato NgRx [17]

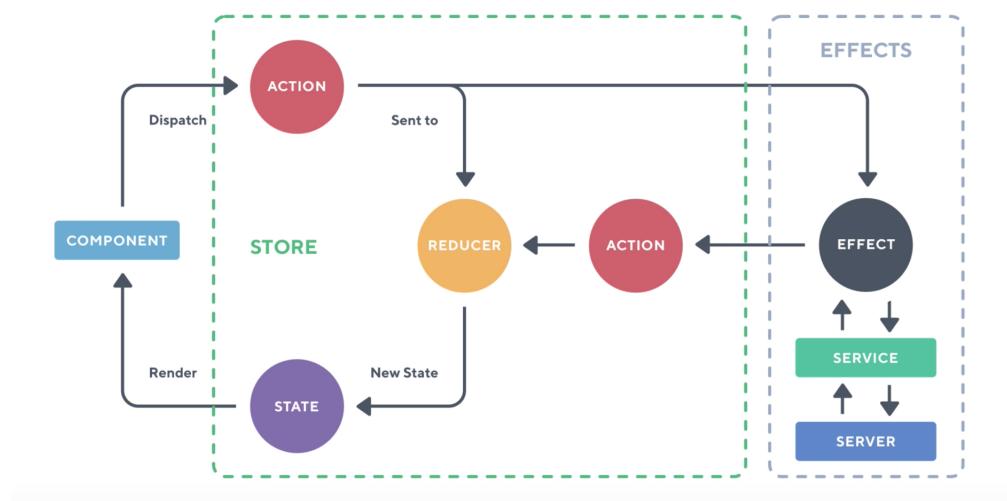
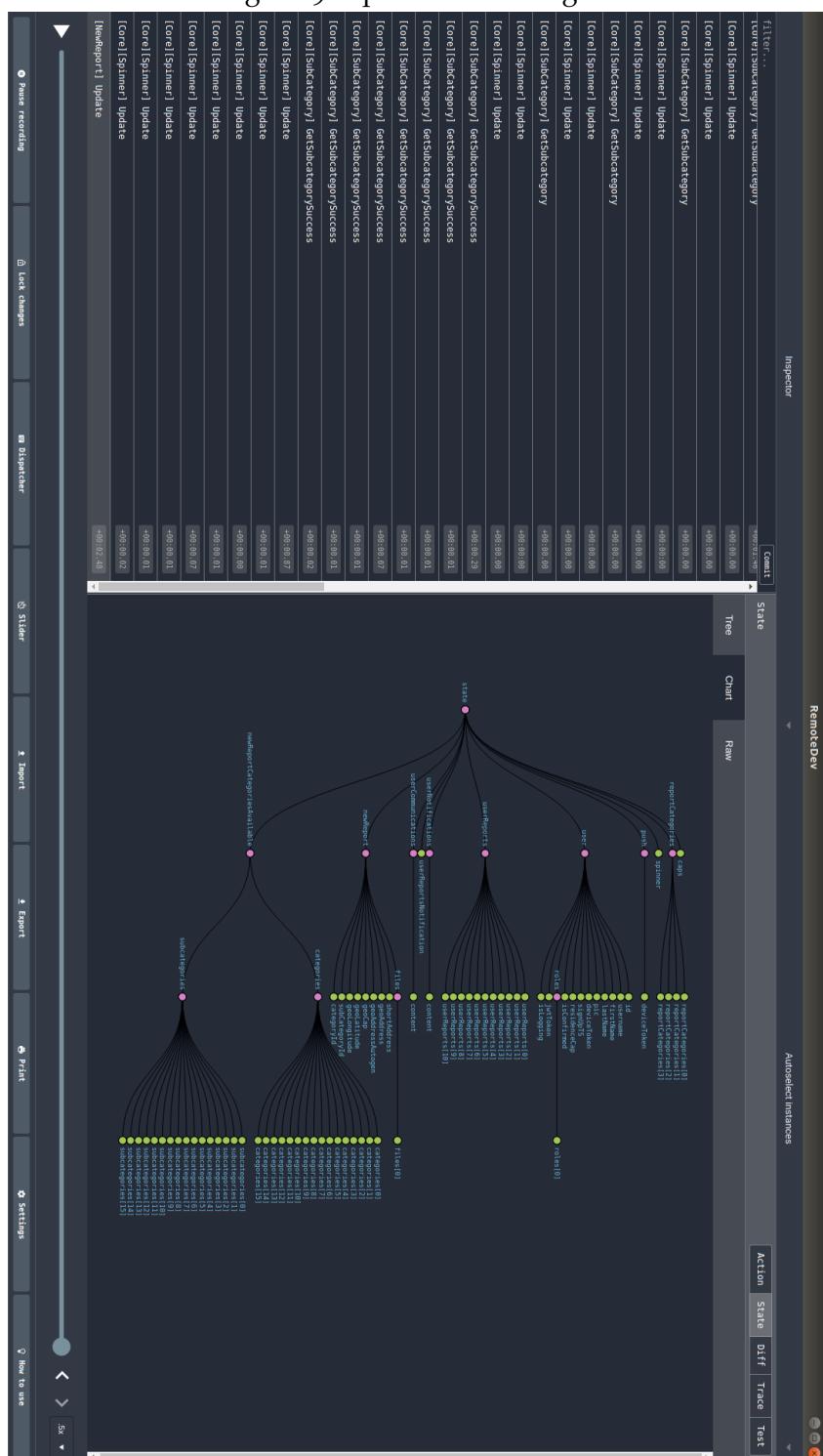


Figura 8: Senza gestore stato NgRx



Figura 9: Ispezione stato NgRx



2.2 SOCIAL LOGIN

Al primo accesso all'APP eServant è necessario dare il consenso per almeno uno dei seguenti social:

- Facebook;
- Instagram;
- Twitter;

Altrimenti viene data la possibilità di fare un accesso privato tramite username e password. L'autenticazione verso i social network sopra citati avviene tramite protocollo OAuth2.0.

Il framework di autorizzazione OAuth2.0 abilita un'applicazione di terze parti a ottenere un accesso limitato a un servizio HTTP, attivo per conto di un proprietario di risorse, orchestrando un'interazione di approvazione tra il proprietario della risorsa e il servizio HTTP. Questa specifica sostituisce il protocollo OAuth 1.0 descritto in RFC 5849.

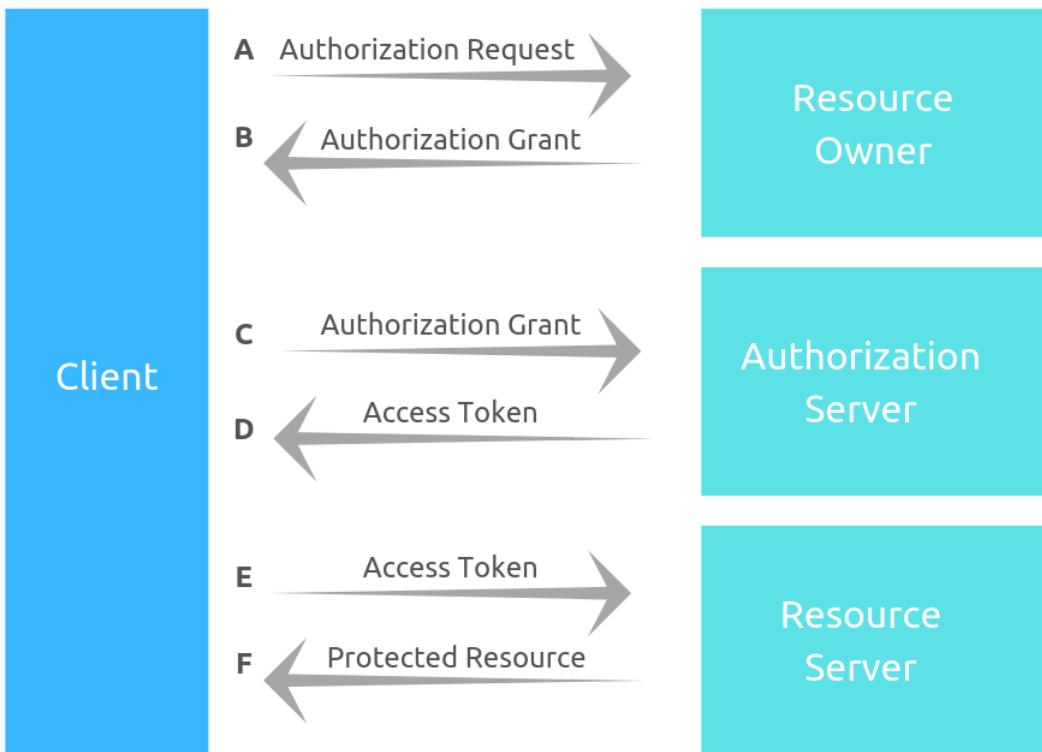
Vediamo quali sono gli **Attori** coinvolti nel protocollo OAuth2.0 mostrato in figura 10 [18].

- *resource owner*: un'entità in grado di approvare l'accesso a una risorsa protetta;
- *authorization server*: il server che invia i token di accesso al client dopo essere stato autorizzato dal resource owner;
- *resource server*: il server che ospita le risorse protette. Il resource Server è in grado di accettare e rispondere alle richieste di risorse protette utilizzando i token di accesso;
- *client*: un'applicazione che fa richieste di risorse protette per conto del proprietario delle risorse e con la sua autorizzazione. Il termine "client" non implica particolari caratteristiche di implementazione (l'applicazione può essere eseguita su un server, un client web o altri dispositivi);

2.3 NAVIGAZIONE IMPIANTO

Grazie alla collaborazione con il MICC di Firenze siamo riusciti ad integrare una cartografia basata su OpenStreetMap per permettere ai

Figura 10: OAuth 2.0 flow [18]



partecipanti di eventi la navigazione verso un determinato POI (point of interest) partendo dall'ultima posizione rilevata del partecipante.

Il nostro compito si è suddiviso in tre parti:

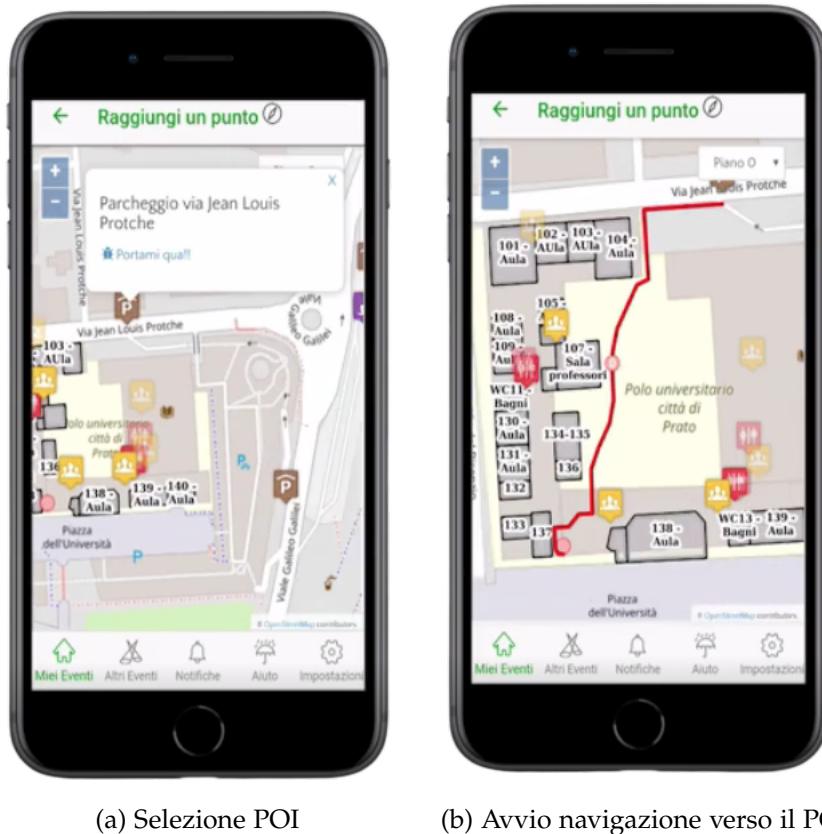
- Creazione livelli da applicare alle mappe OSM per la navigazione interna;
- Integrazione delle mappe sia sull'APP Ionic che sul backoffice web;
- Recupero ultima posizione dell'utente;

L'integrazione all'interno dell'app Ionic è stata fatta tramite iframe. Frammento di codice dell'app Ionic per includere le mappe:

```
<iframe src="https://www.eservant.it/maps"></iframe>
```

Il frammento indicato sopra permette tramite una singola linea di codice di includere le mappe e ottenere il risultato come da screenshot in figura 11.

Figura 11: Mappe nell'app eServant



2.4 GEOLOCALIZZAZIONE

Come abbiamo detto nella sezione precedente, la navigazione all'interno dell'impianto necessita la posizione iniziale dalla quale far iniziare il percorso verso il POI selezionato. Questa posizione ovviamente è un elemento dinamico che deve mutare nel tempo dipendentemente dalla posizione dell'utente all'interno dell'impianto.

Da un primissimo brainstorming è emerso che la tecnologia GPS non è affidabile in impianti indoor ma anche in ambienti outdoor potrebbe fallire; per questo principale motivo è stata creata una gerarchia di tecno-

logie da usare per la Geolocalizzazione dell'utente, dalla più affidabile alla meno affidabile.

In ordine di priorità di affidabilità decrescente, troviamo:

- QRcode manuale;
- GPS;
- iBeacon;

È possibile vedere il risultato della geolocalizzazione nelle mappe nella figura 12.

Figura 12: Localizzazione nelle mappe



2.4.1 QRcode

Figura 13: QRcode



I codici QR (esempio in figura 13) sono stati creati nel 1994 da Denso Wave [19], una filiale giapponese del Gruppo Toyota. L'uso di questa tecnologia è ora gratuito. Il QR Code non è l'unico codice a barre bidimensionale sul mercato, un altro esempio è il codice Data Matrix.

Un codice QR è un codice a barre quadrato bidimensionale che può memorizzare dati codificati. Il più delle volte i dati sono un link a un sito web (URL).

Nel caso di eServant il codice QRcode racchiude un identificativo di un POI.

Vediamo le fasi del processo di geolocalizzazione tramite QRcode:

1. Scansione QRcode tramite l'app di eServant.

Sempre tramite cordova, un esempio di accesso alla camera nativa dell'OS per scansionare codici QRcode [20];

```
import { BarcodeScanner } from '@ionic-native/barcode-scanner/ngx';

constructor(private barcodeScanner: BarcodeScanner) {

    this.barcodeScanner.scan().then(barcodeData => {
        console.log('Barcode data', barcodeData);
    }).catch(err => {
        console.log('Error', err);
    });
}
```

2. Decodifica del QRcode e ottenimento del POI ID;
3. Invio del POI ID, insieme all' ID dell'utente, al server;

4. Il server, una volta ottenute le coordinate del POI, provvede ad aggiornare l'ultima posizione dell'utente;

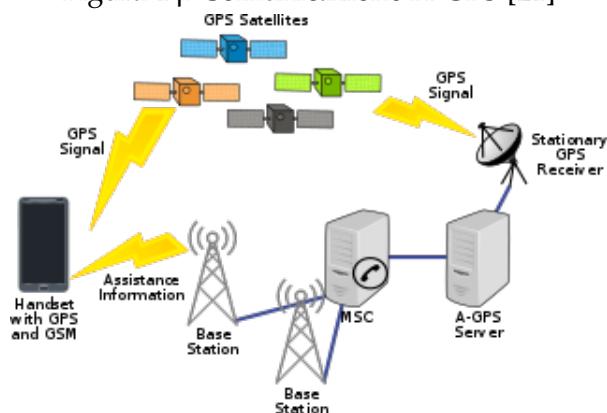
2.4.2 A-GPS

Tramite GPS assistito (A-GPS) la localizzazione del partecipante all'evento è automatica, a patto che il GPS del dispositivo mobile sia abilitato.

La tecnologia GPS assistito permette di acquisire segnale satellitare in maniera più rapida rispetto al GPS tradizionale.

Tramite rete dati cellulare vengono indicate al dispositivo mobile le posizioni previste dei satelliti GPS come mostrato in figura 14. In questo modo, il dispositivo sa dove cercare i satelliti ed è in grado di acquisirne i segnali in pochi secondi, anche in condizioni difficili.

Figura 14: Comunicazione A-GPS [21]



Vediamo un frammento di codice per ottenere in tempo reale la posizione del dispositivo tramite A-GPS [22].

```
import { Geolocation } from '@ionic-native/geolocation/ngx';

constructor(private geolocation: Geolocation) {}

this.geolocation.getCurrentPosition().then((resp) => {
  // resp.coords.latitude
  // resp.coords.longitude
}).catch((error) => {
  console.log('Error getting location', error);
});
```

```

let watch = this.geolocation.watchPosition();
watch.subscribe((data) => {
    // data can be a set of coordinates, or an error (if an
    // error occurred).
    // data.coords.latitude
    // data.coords.longitude
});

```

2.4.3 iBeacon

Figura 15: Dispositivo iBeacon [24]



iBeacon [23] è una tecnologia basata su Bluetooth Low Energy. A partire dal 2013 iBeacon è integrato in Apple iOS 7. Il primo progetto pilota è stato lanciato nei negozi Apple nel Dicembre 2013.

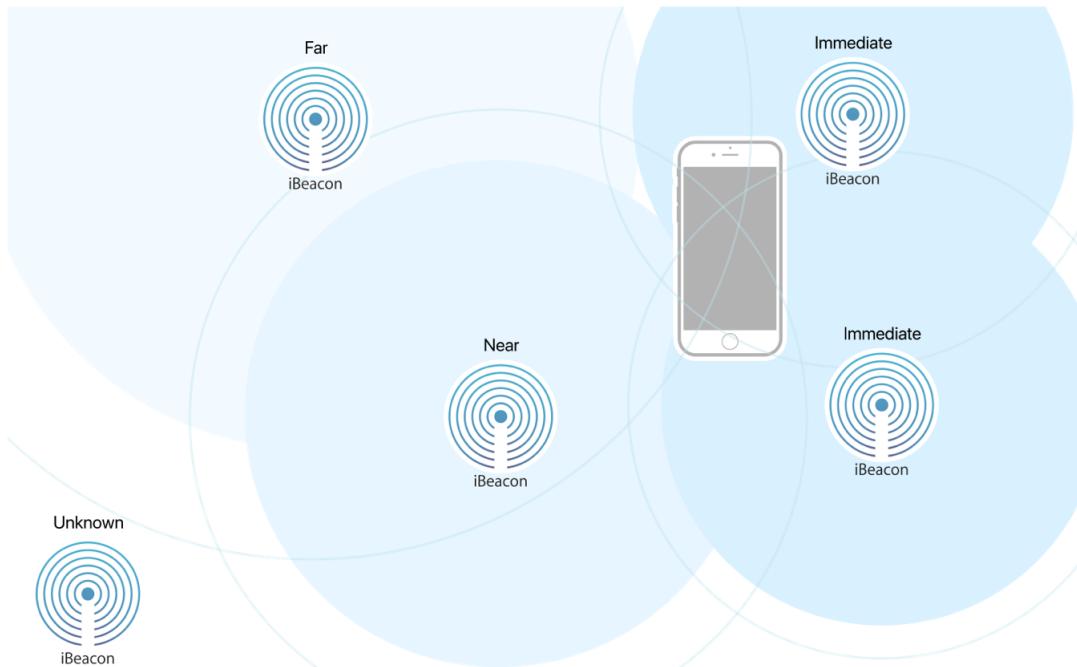
BLE (Bluetooth Low Energy) Bluetooth Low Energy (BLE) è uno standard di trasmissione radio sviluppato dalla community Bluetooth SIG. Le caratteristiche di questo standard sono mirate a soddisfare le esigenze delle moderne applicazioni wireless, come un consumo energetico estremamente basso, una connessione corta che scandisce i tempi, affidabilità e sicurezza. BLE consuma da 10 a 20 volte meno energia (e 1000 volte meno del Wi-Fi), e può trasmettere dati con velocità 50 volte superiore rispetto al Bluetooth classico.

Un esempio di dispositivo iBeacon è mostrato in figura 15. Lato client (Ionic), il plugin cordova IBeacon mette a disposizione due principali metodi che permettono di intercettare due eventi:

- Dispositivo mobile entrato nel raggio dell'ibeacon;
- Dispositivo mobile uscito dal raggio dell'ibeacon;

La figura 16 mostra un dispositivo mobile entrato nel raggio di quattro iBeacon.

Figura 16: Prossimità tramite iBeacon [25]



Frammento di codice Ionic [26]:

```
import { iBeacon } from '@ionic-native/ibeacon/ngx';

constructor(private ibeacon: iBeacon) { }

// create a new delegate and register it with the native layer
let delegate = this.ibeacon.Delegate();

// Subscribe to some of the delegate's event handlers
delegate.didRangeBeaconsInRegion()
  .subscribe(
    data => console.log('didRangeBeaconsInRegion: ', data),
    error => console.error()
  );
delegate.didStartMonitoringForRegion()
  .subscribe(
    data => console.log('didStartMonitoringForRegion: ', data)
    ,
    error => console.error()
  );
```

```

delegate.didEnterRegion()
  .subscribe(
    data => {
      console.log('didEnterRegion: ', data);
    }
  );

let beaconRegion = this.ibeacon.BeaconRegion('deskBeacon',
  'F7826DA6-ASDF-ASDF-8024-BC5B71E0893E');

this.ibeacon.startMonitoringForRegion(beaconRegion)
  .then(
    () => console.log('Native layer received the request to
      monitoring'),
    error => console.error('Native layer failed to begin
      monitoring: ', error)
  );

```

2.5 CHATBOT

Il chatbot agisce come supporto automatico che, tramite una chat integrata nell'applicazione, cerca di rispondere alle domande dell'utilizzatore. Il motore dietro alla chat integrata è un NLP engine (Natural Language Processor). Il suo compito è, per ogni stringa di testo proveniente dalla chat integrata sull'app, cercare di capire il significato semantico della frase.

Introduciamo 2 concetti principali su cui si basa il motore NLP:

- Intent: rappresenta l'azione presente nella frase elaborata, è generalmente un verbo.
- Slot: rappresenta uno o più attributi che si legano all'intent. In molti casi l'intent è lo stesso, cambiano solo i valori degli slot.

Vediamo nel caso specifico di eServant rappresentato dagli screenshot in figura 17.

Vorrei raggiungere il punto di interesse Aula Magna

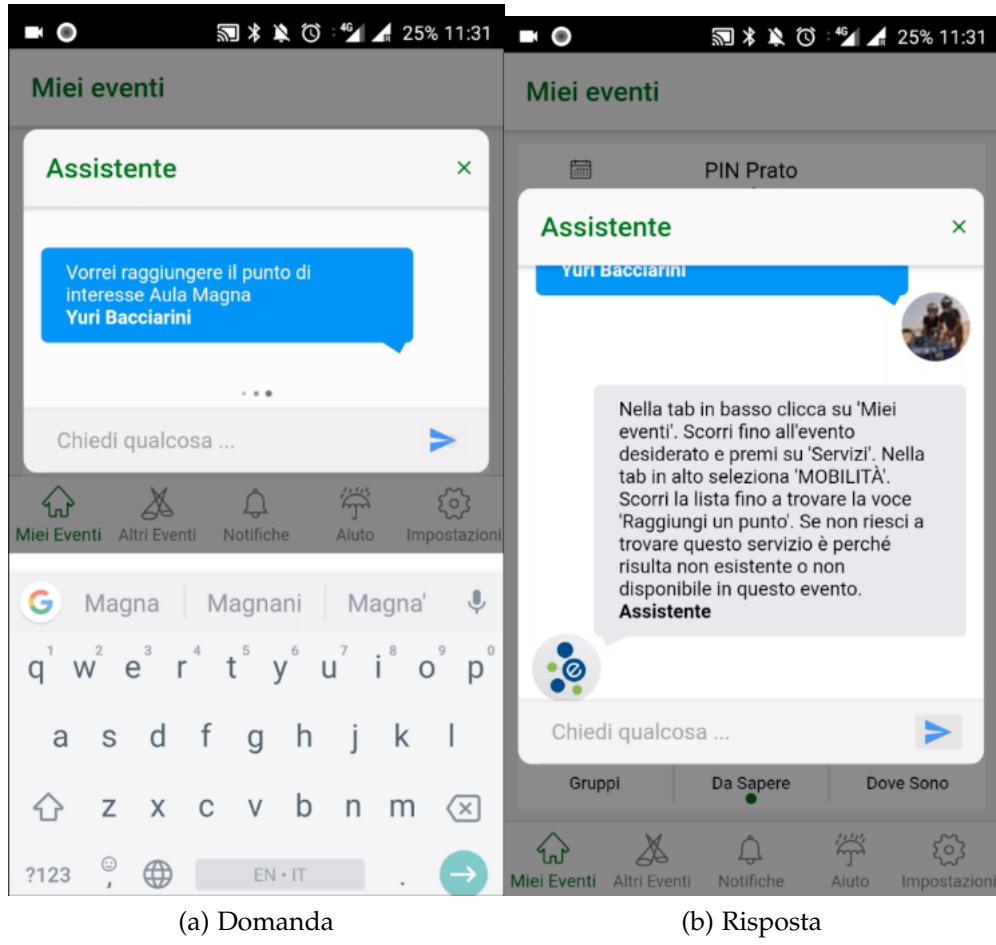
Analisi tramite NLP:

- **intent:** raggiungere il punto di interesse

- slot:

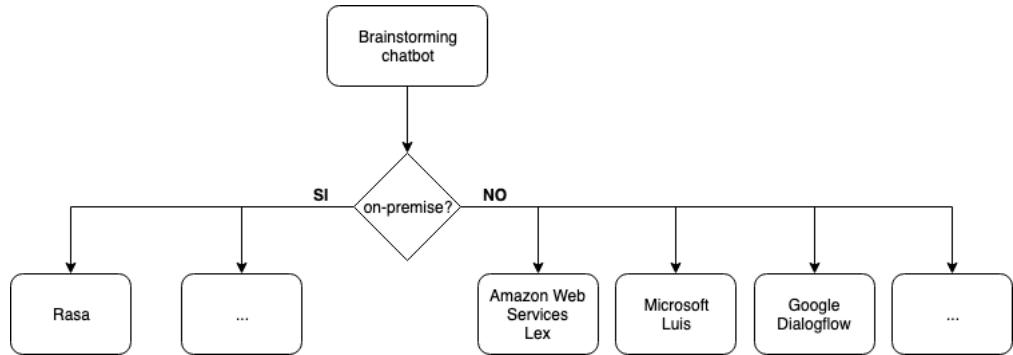
- Tipo slot: POI
- Valore: **Aula Magna**

Figura 17: Chatbot in esecuzione



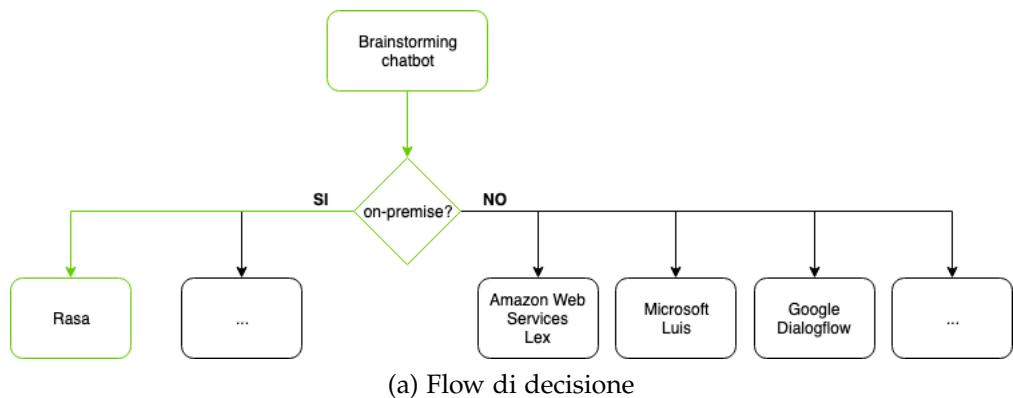
2.5.1 Brainstorming

Figura 18: Brainstorming per l'implementazione del chatbot [27][28][29][30]



Come si nota dal flow decisionale in figura 18, la prima domanda che ci siamo posti è stata se volevamo fare il deploy del chatbot su cloud oppure in-house.

2.5.2 Implementazione



Visto che il progetto eServant racchiude informazioni private degli impianti, abbiamo deciso di usare un motore NLP in-house e non cloud. Date le conoscenze in Quid Informatica del linguaggio di programmazione Python abbiamo deciso di procedere con l'implementazione del chatbot usando il framework Rasa, scritto appunto in Python.

Rasa è un framework open source in Python per la creazione di chatbot basato su machine learning. Invece che basarsi su una cascata di if/else, Rasa risponde usando un modello, creato attraverso librerie di machine learning in Python sulla base di conversazioni campione.

3

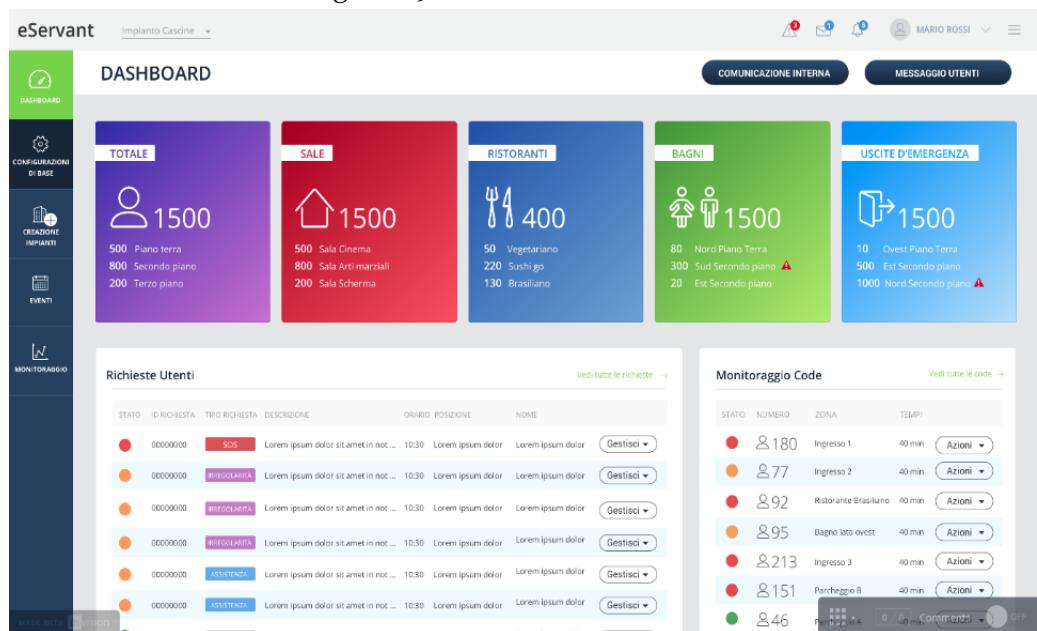
BACKOFFICE

Il Backoffice è l'applicazione web che permette ai gestori degli impianti presenti su eServant di:

- configurare i propri impianti, per ogni impianto gli eventi e per ogni evento i servizi disponibili;
- monitorare real-time l'impianto analizzando i dati provenienti dai sensori IoT;
- monitorare e analizzare offline le prestazioni del sistema/impianto dopo un evento o in un certo periodo;

Vediamo nello screenshot della dashboard in figura 19 le funzioni di monitoraggio dell'impianto.

Figura 19: Screenshot dashboard



Il backoffice di eServant è stato realizzato con il framework Angular usando interamente Material Design come sistema di progettazione dell'esperienza utente. Material Design è un framework open source dell'azienda Google che aiuta i team a creare esperienze digitali di alta qualità in breve tempo.

3.1 MATERIAL DESIGN

Material Design [31] mette a disposizione al team di sviluppo un set di componenti plugAndPlay solo da personalizzare secondo il tema del brand utilizzatore. Il principale vantaggio riscontrato è stata la velocità di sviluppo. Avendo nella maggior parte dei casi dei componenti pronti all'uso, ci ha permesso di concentrarci sulla logica in Typescript invece che sull'esperienza utente (non da sottovalutare).

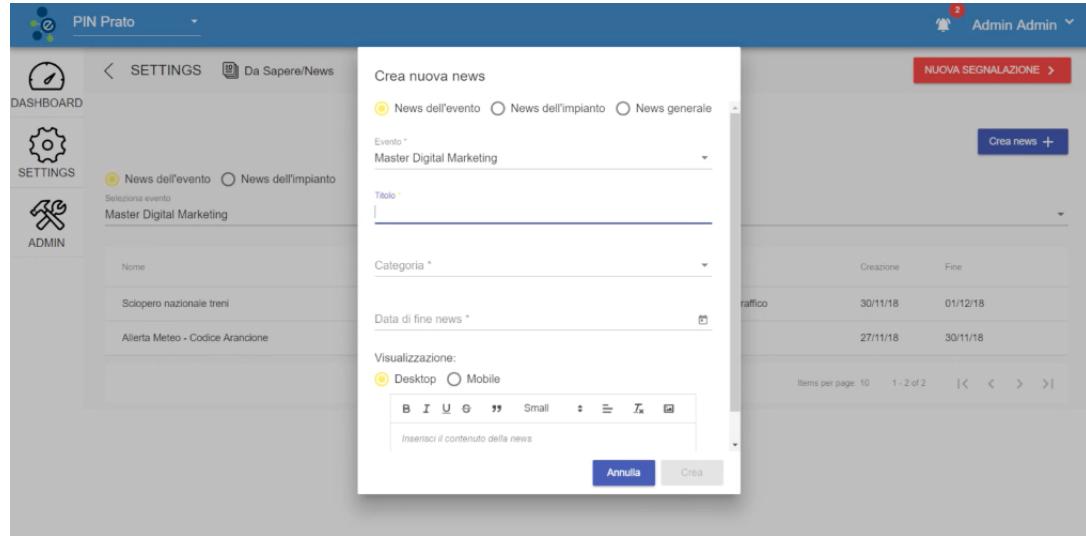
I principali componenti messi a disposizione da Material Design di Google in Angular sono[32]:

- Componenti per i form;
- Navigazione;
- Layout;
- Bottoni;
- Popup;
- Tabelle dati;

Riportiamo un frammento di codice HTML usato nel backoffice di eServant ed il relativo effetto nell'interfaccia utente nella figura 20.

```
<form class="newsform">
  ...
    <mat-form-field class="newsform_field">
      <input matInput placeholder="Titolo">
    </mat-form-field>
  ...
</form>
```

Figura 20: Form creazione nuova news



3.2 BEM

È noto che un'organizzazione corretta dei fogli di stile può aumentare significativamente la velocità di sviluppo, il debug e l'implementazione di nuove feature. Purtroppo, molti codici CSS vengono sviluppati senza alcuna convenzione di struttura o di nome. Ciò porta a una base di codice CSS non mantenibile a lungo termine. Ecco una citazione molto importante dove si afferma che ci sono due principali problemi nell'informatica: l'invalidazione della cache e le convenzioni sui nomi.

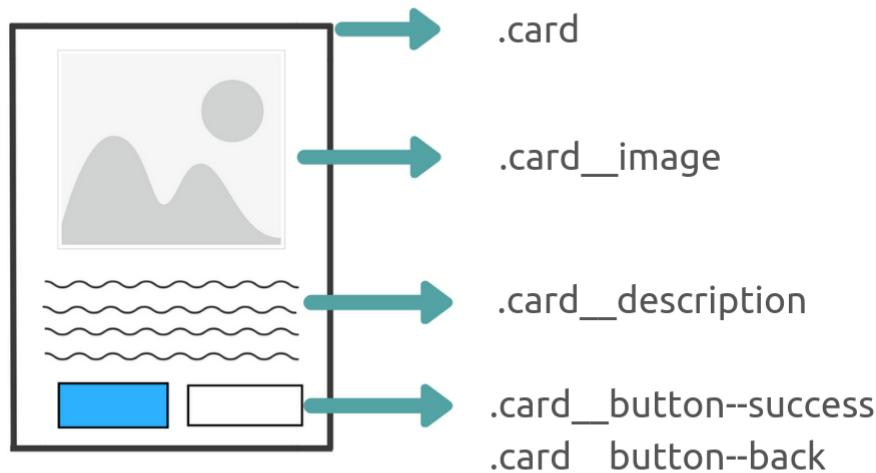
"There are only two hard problems in Computer Science: cache invalidation and naming things" cit. Phil Karlton [33]

L'approccio BEM cerca di garantire a tutti i partecipanti attuali e futuri allo sviluppo di un sito web lavorino con un singolo base code e parlino la stessa lingua.

BEM [34] è l'acronimo di

- Block;
- Element;
- Modifier;

Figura 21: Esempio di adozione BEM [35]



Analizziamo le classi CSS usate in figura 21:

- `.card` è un **block**
- `.card__image` e `.card__descriptor` sono **Element** di `.card`
- `.card__image-success` e `.card__descriptor-back` sono **Modifier** di `.card__image`

Vediamo come è stato usato BEM nel caso specifico dell'header rappresentato in figura 22.

Figura 22: Header



```
<mat-toolbar class="header">
  <mat-toolbar-row>
    <img class="header__logo">
    <mat-select class="header__facilities">
```

```
<mat-option *ngFor="let facility of facilities" [  
  value]="facility.value">  
  {{facility.display}}  
</mat-option>  
</mat-select>  
<mat-icon>alert</mat-icon>  
<div class="settings">  
  <span class="settings__user">Admin</span>  
  <mat-icon class="settings__icon settings__icon--  
    closed">arrow-down</mat-icon>  
</div>  
</mat-toolbar-row>  
</mat-toolbar>
```


4

IOT

Il progetto eServant include al suo interno due principali tecnologie che rientrano nella categoria "Internet Of Things":

- dispositivi BLE per geolocalizzare i partecipanti dove il GPS ha dei grandi limiti
- conta persone per monitorare il flusso nelle aule

Scendendo nel dettaglio, l'impianto PIN di Prato è stato attrezzato con una serie di dispositivi BLE (Bluetooth Low Energy) per intercettare i dispositivi mobili dei partecipanti e con un conta persone situato all'ingresso della biblioteca.

4.1 BLUETOOTH LOW ENERGY

I dispositivi BLE possono operare in modalità attiva o in modalità passiva, rispettivamente esistono le seguenti due modalità:

- ranging;
- advertising;

4.1.1 *Gateway*

I Gateway posizionati negli ingressi principali sono stati realizzati tramite single board computer Raspberry e rivestiti tramite case fissato a muro come in figura 23.

Figura 23: Dispositivo Gateway [36]



Le principali componenti usate sono:

- scheda di rete wireless;
- chip Bluetooth versione 5;
- PoE (Power over Ethernet);

I gateway forniti dall’azienda BlueUp hanno al loro interno già un firmware che all’avvio del sistema operativo raspbian avvia il servizio di ranging. La feature che distingue questi gateway è che sono programmati per chiamare un servizio HTTP custom al quale, tramite metodo POST, vengono inviati la lista degli ibeacon in modalità advertising intercettati.

Queste chiamate HTTP, ricevute da un microservizio specializzato di eServant, registrano che nella data di ricezione sono stati intercettati una serie di ibeacon con una determinata prossimità (BASSA,MEDIA,ALTA).

Visto che i gateway hanno una posizione fissa all’interno dell’impianto, sono stati catalogati nel nostro database con le coordinate di installazione. Queste coordinate rappresenteranno la posizione dei dispositivi mobili (in modalità advertising) quando quest’ultimi vengono intercettati dal gateway attivo.

4.1.2 *iBeacon passivi*

Gli iBeacon passivi sono un’altro dispositivo standalone dotati di batteria e facilmente maneggevoli per geolocalizzare un dispositivo all’interno di

uno spazio indoor. Rispetto ai gateway attivi hanno il vantaggio di essere molto più economici (5 volte meno) e di essere dotati di batterie al litio.

Gli smartphone con eServant installata (in modalità attiva o in background), potranno intercettare questi dispositivi (come in figura 24) ed inviare tramite HTTP al microservizio specializzato l'ID dell'iBeacon rilevato per permettere di geolocalizzare il partecipante nelle coordinate dell'iBeacon storicizzate nel nostro database.

Figura 24: iBeacon passivi in azione



4.1.3 iBeacon Wearable

Figura 25: iBeacon indossabile



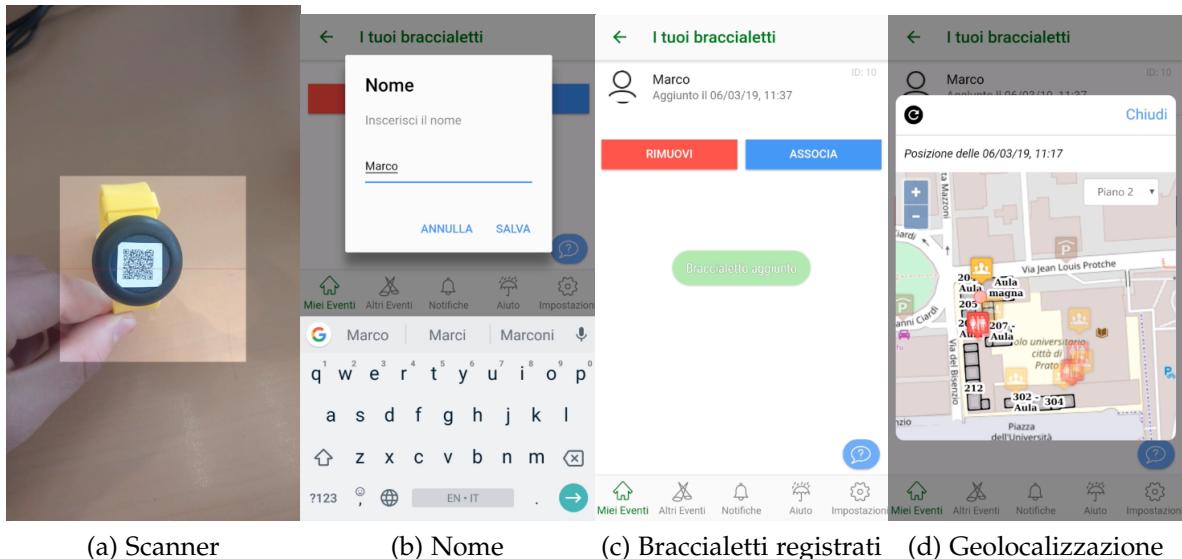
I braccialetti BLE svolgono l'esatto compito degli iBeacon passivi sopra descritti ma vengono utilizzati con una strategia diversa all'interno di

eServant. Il caso d'uso principale è permettere ai genitori di monitorare real-time la posizione dei propri figli all'interno di un impianto affollato. Al genitore, all'ingresso dell'impianto, verranno consegnati un braccialetto (come in figura 25) per ogni figlio.

Nella sezione ricerca un minore dell'app eServant scansionerà il QR-code sopra il braccialetto e da quel momento in poi il suo utente sarà abilitato a monitorare l'ultima posizione rilevata di ogni singolo braccialetto. L'assegnazione dei braccialetti viene rilasciata in maniera spontanea all'uscita dall'impianto oppure tramite un sistema automatico alcune ore dopo la fine dell'evento (nel caso in cui il genitore non riconsegni i braccialetti).

I passi sopra descritti sono riportati in figura 26.

Figura 26: Caso d'uso braccialetti iBeacon



4.1.4 Smartphone

Gli smartphone dei partecipanti agli eventi, tramite l'app di eServant attiva o in background, agiscono sia in modalità advertising per essere intercettati da eventuali gateway sia in modalità attiva (scanning) per intercettare i dispositivi passivi.

4.2 CONTA PERSONE

Figura 27: Single board computer Raspberry [37]



Il conta persone, come i gateway BLE, è stato implementato sempre usando single board computer Raspberry. È dotato inoltre di modulo camera e individua, usando librerie plugAndPlay, le persone che transitano dalla porta della biblioteca dove è stato installato.

Figura 28: Modulo camera per Raspberry [38]



Il conta persone mantiene al suo interno un contatore delle persone attualmente dentro l'aula inviandole con cadenza di 5 minuti ad un microservizio specializzato di eServant che provvede ad aggiornare il dato sul database.

5

SITOGRAFIA

- 1 sito web eServant <http://www.eservant.it>
- 2 link al numero degli impianti (ita e toscana)
- 3 slide ingegneria del software https://stlab.dinfo.unifi.it/vicario/Teaching/SW_Engineering/UseCaseDiagsAndTemplates18.pdf
- 4 OSGi Alliance <http://www.osgi.org>
- 5 What are microservices? <https://microservices.io/>
- 6 Specifiche Swagger <https://swagger.io>
- 7 Specifiche PostgreSQL <https://en.wikipedia.org/wiki/PostgreSQL>
- 8 Specifiche MongoDB <https://en.wikipedia.org/wiki/MongoDB>
- 9 Specifiche Hibernate ORM <https://hibernate.org/orm/>
- 10 <https://codewave.com/insights/mithun-on-switch-to-angular-2-for-modern-applications>
- 11 Immagine da <https://code.tutsplus.com/tutorials/ionic-from-scratch-what-is-ionic-and-how-it-works--cms-29900>
- 12 Tutorial <https://ionicframework.com/docs/installation/cli>
- 13 Immagine da <https://ionicframework.com/enterprise/resources/articles/what-is-apache-cordova>
- 14 Ionic native camera <https://ionicframework.com/docs/native/camera>
- 15 Specifiche SASS <https://sass-lang.com/guide>
- 16 Specifiche NgRx <https://ngrx.io/docs>
- 17 Immagine da <https://blog.imeaginea.com/ngrx-introduction-and-its-basic-structure>

- 18 Protocollo OAauth2.0 <https://tools.ietf.org/html/rfc6749>
- 19 History QRcode <https://www.qrcode.com/en/history/>
- 20 Ionic lettura barcode <https://ionicframework.com/docs/native/barcode-scanner>
- 21 A-GPS specifiche https://en.wikipedia.org/wiki/Assisted_GPS
- 22 Ionic accesso GPS <https://ionicframework.com/docs/native/geolocation>
- 23 iBeacon <http://www.beaconsandwich.com/what-is-ibeacon.html>
- 24 Immagine da <https://www.businessinsider.com/beacons-and-ibeacons-create-a-new-m>
IR=T
- 25 Prossità iBeacon https://developer.apple.com/documentation/corelocation/determining_the_proximity_to_an_ibeacon_device
- 26 Ionic rilevazione iBeacon <https://ionicframework.com/docs/native/ibeacon>
- 27 Specifiche Rasa https://github.com/RasaHQ/rasa_core
- 28 Specifiche AWS lex <https://aws.amazon.com/it/lex/>
- 29 Specifiche Microsoft Luis <https://www.luis.ai/home>
- 30 Specifiche Google Dialogflow <https://dialogflow.com/>
- 31 Material design <https://material.io/design/>
- 32 Material design in Angular <https://material.angular.io/>
- 33 Citazione <https://hilton.org.uk/blog/why-naming-things-is-hard>
- 34 Specifiche BEM <http://getbem.com/naming/>
- 35 Immagine BEM https://medium.com/@dannyhuang_75970/what-is-bem-and-why-you-should-use-it-4a2a2a2a2a2a
- 36 Gateway BlueUp <https://www.blueupbeacons.com/index.php?page=gateway>
- 37 Immagine Raspberry <https://www.raspberrypi.org/>
- 38 Immagine Camera Raspberry <https://www.raspberrypi.org/products/camera-module-v2/>

6

RINGRAZIAMENTI

..