



UNIVERSITÀ
DEGLI STUDI
FIRENZE

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

TITOLO IN ITALIANO

TITLE IN ENGLISH

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	6
1.4.1	Microservizi	7
1.4.2	Swagger (API)	7
1.4.3	Database (relazionale + documentale)	9
1.5	Front-end	10
2	Applicazione mobile	13
2.1	Framework Ionic	16
2.1.1	NgRx gestore stato	21
2.2	Social login	22
2.3	Navigazione impianto	23
2.4	Geolocalizzazione	25
2.4.1	QRcode	26
2.4.2	GPS	27
2.4.3	iBeacon	28
2.5	Chatbot	30
2.5.1	Brainstorming	31
2.5.2	implementazione	32
3	Backoffice	35
3.1	Framework Angular	35
3.2	Material Design	35
3.3	Firebase	35
3.3.1	Pro/Contro servizi cloud	35
3.3.2	Real Time Database	35
3.4	CI Jenkins	35
4	IoT	37
4.1	Bluetooth Low Energy	37
4.1.1	Gateway (attivo)	37
4.1.2	Smartphone (attivo/passivo)	37
4.1.3	iBeacon Box (passivo)	37
4.1.4	iBeacon Wearable (passivo)	37
4.2	Conta persone	37

4.3 Telecamere densita' 37

1

ESERVANT

1.1 PANORAMICA

eServant 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.

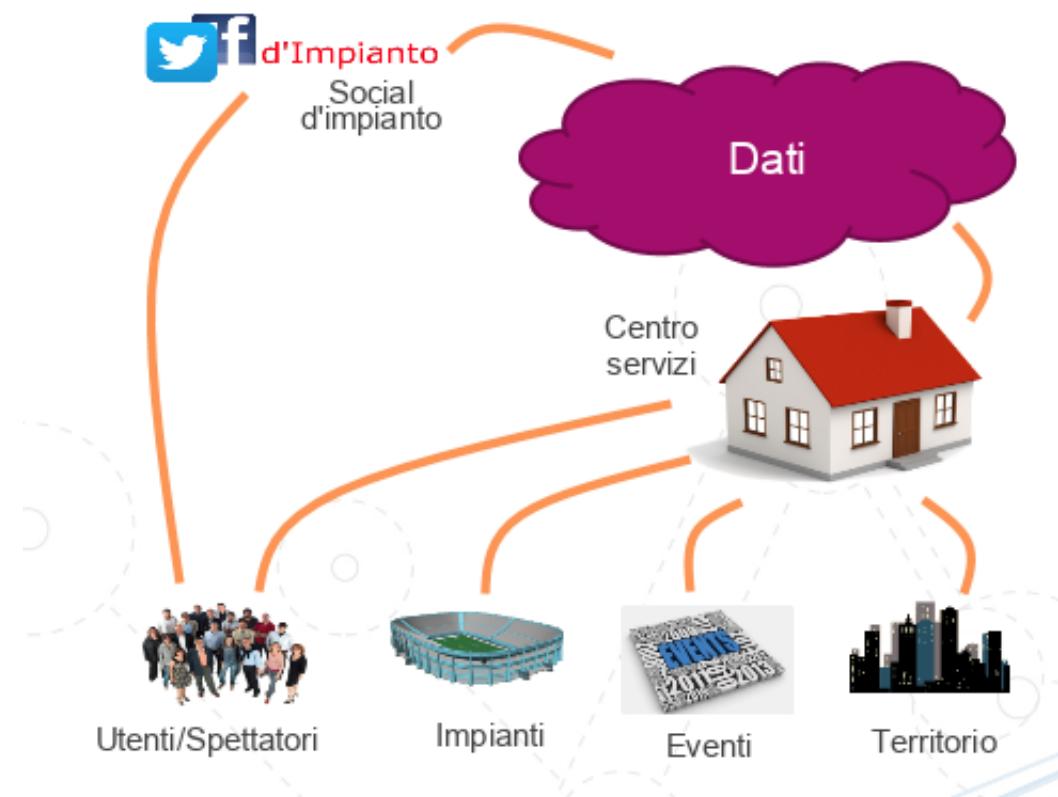
Si tratta quindi di un insieme di processi, che sono gestiti in modo assolutamente innovativo, sino a creare un processo evoluto, costituito dalla piattaforma eServant, che viene industrializzato in una struttura hardware e software altamente replicabile.

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.

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 misure e 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. 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 E' 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.



1.2 AZIENDE COINVOLTE

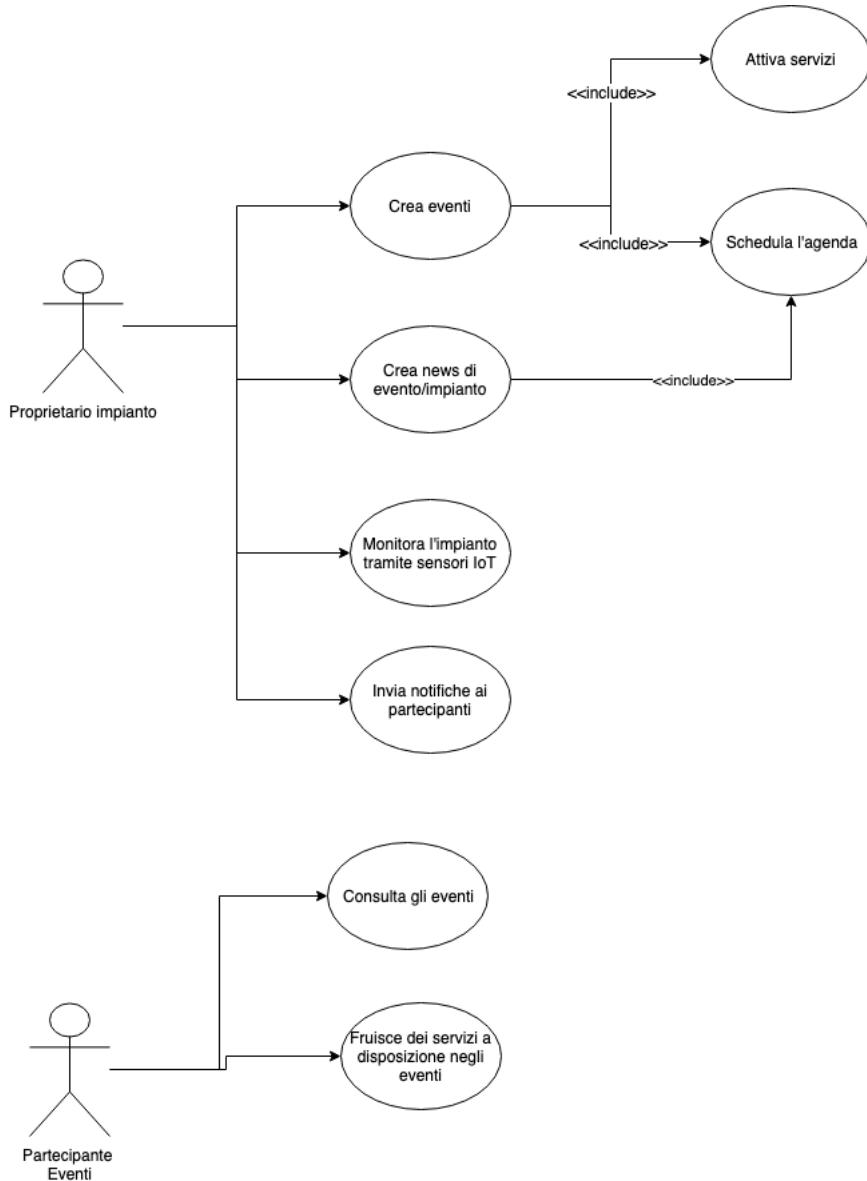
Quid Informatica spa Come capofila di progetto, Quid Informatica spa si è occupata di sviluppare mettere a disposizione l'infrastruttura tecnologica per fruire i moduli sviluppati dai vari partner. Sokom srl Sokom srl si è occupata dei cablaggi e disposizione dei vari access point necessari per fare 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. Sviluppando inoltre un modulo di mappe, insieme alla tecnologia precedente, per consigliare ai partecipanti eventuali percorsi alternativi in caso di affollamento. Si aggiunge al partner MICC anche il 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



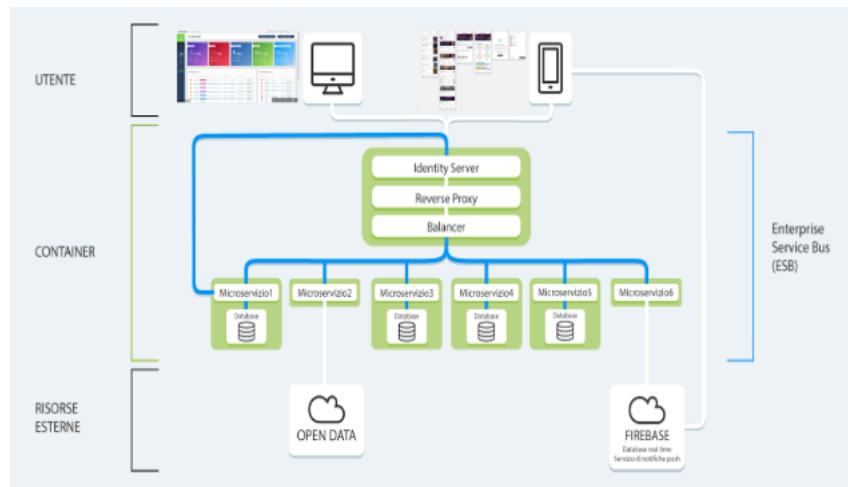
1.4 BACK-END

OSGI (Open Service Gateway Initiative) 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



1.4.2 Swagger (API)

Swagger 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 3 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 uno strumento 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 la varie applicazioni dei partner.

1.4.3 Database (*relazionale + documentale*)

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

PostgreSQL è noto come il più avanzato sistema di gestione di dati open-source disponibile sul mercato. SI tratta di un database di tipo 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. Altamente portatile (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 relazione. Tutta la gestione massiva dei dati viene invece gestita attraverso il database non-SQL MongoDB.

Hibernate (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 esonera lo sviluppatore dal lavoro inherente la persistenza dei dati. Fortunatamente Hibernate gestisce una persistenza trasparente per "Plain Old Java Object".

MongoDB è un Database management System, 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 possiede caratteristiche redeterminate può aggiungere qualsiasi numero di campi con una qualsiasi lunghezza. Nei doc-oriented 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) 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.

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 alla 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 e poi di Angular JS.

Angular da un lato esalta e potenzia l'approccio dichiarativo dell'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: 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 desktop;

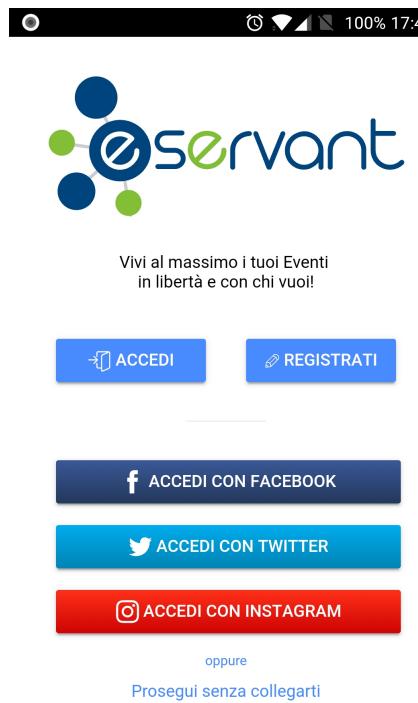
modularità: l'architettura di Angular 2 è altamente modulare e favorisce la scrittura di applicazioni modulari e la maggior parte dei componenti del framework è opzionale ed è possibile sostituirli con altri di terze parti o sviluppati in proprio;

prestazioni; un'attenzione particolare è stata posta sulle prestazioni del framework ed alla 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 disegno delle interfacce sia su mobile (e quindi per gli utenti/spettatori) che di tipo desktop (e quindi dedicate alla centrale di controllo e gestione).

2

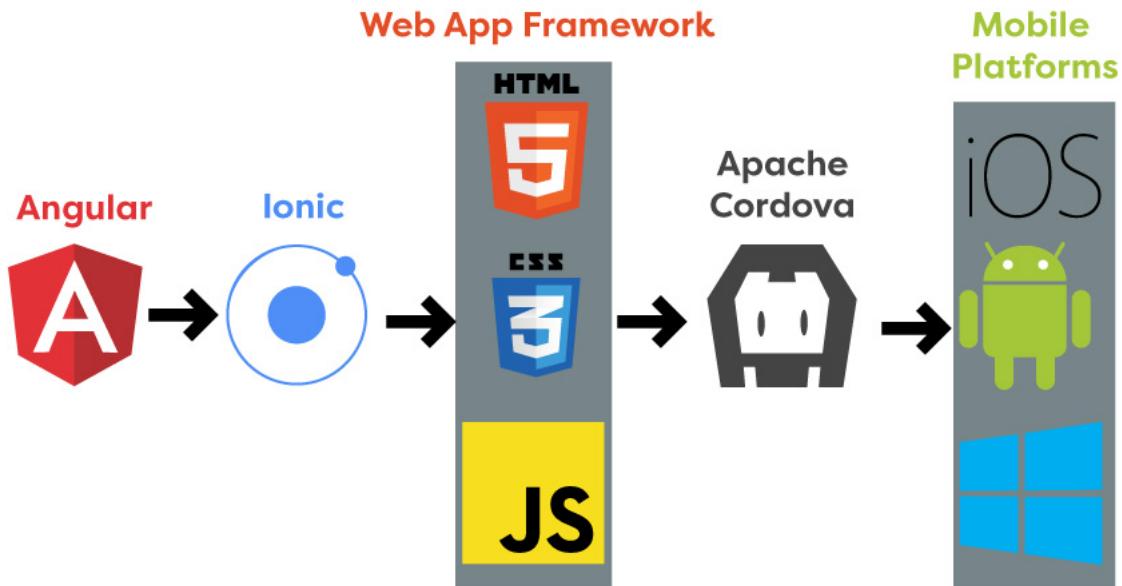
APPLICAZIONE MOBILE



L'interfaccia della piattaforma eSERVANT verso gli utenti finali è come da progetto una APP per mobile. Per questo insieme a Sintra Consulting s.r.l. siamo andati a disegnare 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. Tutte le slide sono state montate e condivise tra tutti i partner del progetto all'indirizzo <https://share.proto.io/U2B8EV/>, anche se per motivi di praticità non sono state sviluppate tutte le articolazioni della stessa, ma solo le funzioni di maggior impatto e complessità. Il prototipo di APP è quindi stato creato andando a realizzare slide sui cinque 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



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. Svilizzare 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 Nodejs. 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.

Installazione Ionic tramite CLI Linux

```
npm install -g ionic
```

Elementi positivi emersi sviluppando in Ionic:

- essendo tra i più famosi web framework, la **velocità di sviluppo** è un aspetto caratterizzante di Ionic
- **semplice da capire**
- supporta il **live reload** permettendo così al developer di vedere immediatamente le modifiche direttamente sul dispositivo mobile.

Elementi negativi emersi:

- l'accesso alle funzioni native tramite Cordova è **limitato**. Non è assolutamente paragonabile alle API native dell'OS.abilità e mappe;

- **instabilità** nell'accesso nativo. Generalmente le librerie Cordova non sono allineate con le ultime API rilasciate da Android/IOS.

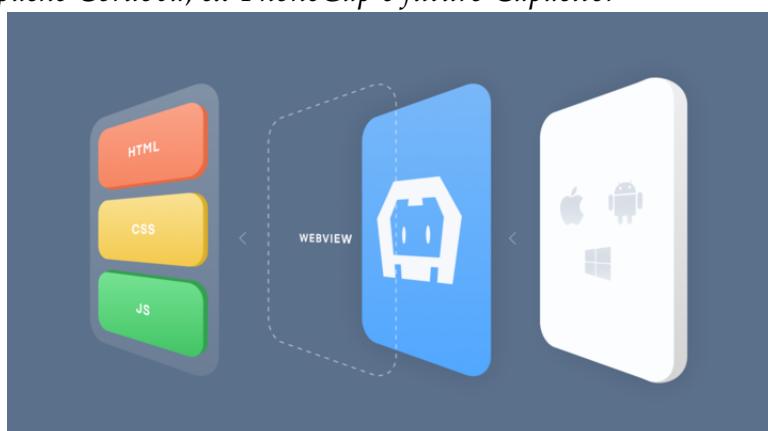
La struttura di Ionic è costituita da Angular, Apache Cordova (ex-PhoneGap) e SASS. Ciò consente la creazione di applicazioni mobili ricche di funzionalità che utilizzano esclusivamente tecnologie web.

Angular

Angular è un framework web open source creato da Google ed è il successore di AngularJS (primissima versione). Angular permette ai developer di costruire applicazione fruite nel web, su piattaforme mobile o desktop. Angular capisce solo TypeScript dato che il framework è stato proprio scritto in Typescript.

Cordova

Apache Cordova, ex-PhoneGap e futuro Capacitor



Apache Cordova è un framework di sviluppo di applicazioni mobili che può essere utilizzato per creare app mobili multipiattaforma usando HTML5 e puro JavaScript. Per multipiattaforma, intendiamo che il codice di applicazione può essere scritto una volta utilizzando HTML5 e JavaScript e può essere eseguito su diversi OS; come Android, iOS o Windows Mobile.

Riporto qua un frammento di codice che permette di aprire la camera nativa del dispositivo mobile.

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

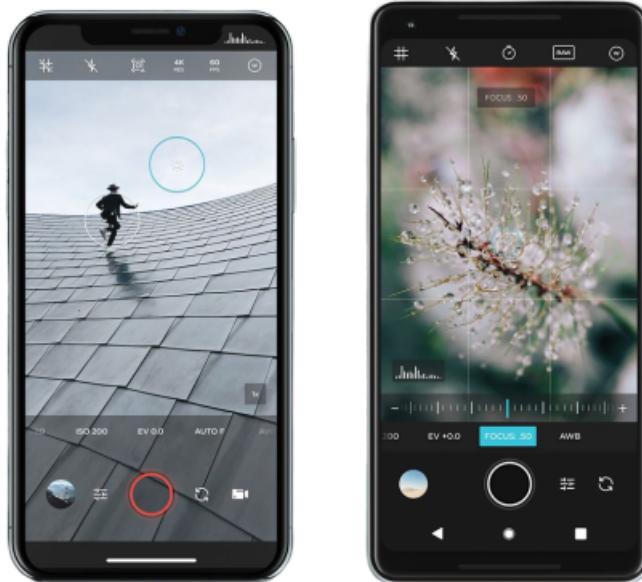
```
import { Camera, CameraOptions } from '@ionic-native/camera/
ngx';
```

```
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 16 linee di codice siamo riusciti a raggiungere il seguente risultato, sia su IOS che su Android:



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 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 tuo codice e genererà l'output CSS che un browser può comprendere.

Vediamo alcune funzioni nel dettaglio:

Variabili

```
// SASS
$font-stack: Helvetica, sans-serif
$primary-color: #333

body
  font: 100% $font-stack
  color: $primary-color
```

VS

```
// CSS
body {
  font: 100% Helvetica, sans-serif;
  color: #333;
}
```

Mixin

```
// SASS
@mixin transform($property)
  -webkit-transform: $property
  -ms-transform: $property
  transform: $property
```

```
.box
  @include transform(rotate(30deg))
```

VS

```
// CSS
.box {
  -webkit-transform: rotate(30deg);
  -ms-transform: rotate(30deg);
  transform: rotate(30deg);
}
```

Loop

```
// SASS
$class-slug: for !default

@for $i from 1 through 4
  .#${$class-slug}-#{$i}
    width: 60px + $i
```

VS

```
// CSS
.for-1 {
  width: 61px;
}

.for-2 {
  width: 62px;
}

.for-3 {
  width: 63px;
}

.for-4 {
  width: 64px;
}
```

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.

- **Unica sorgente di verità:** significa che l'intero stato dell'applicazione è memorizzato nella stessa struttura dati.
- **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).
- **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, 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.

Figura 1: NgRx flow

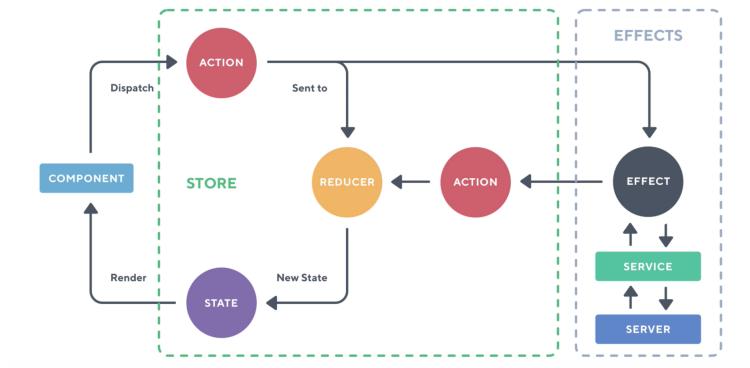
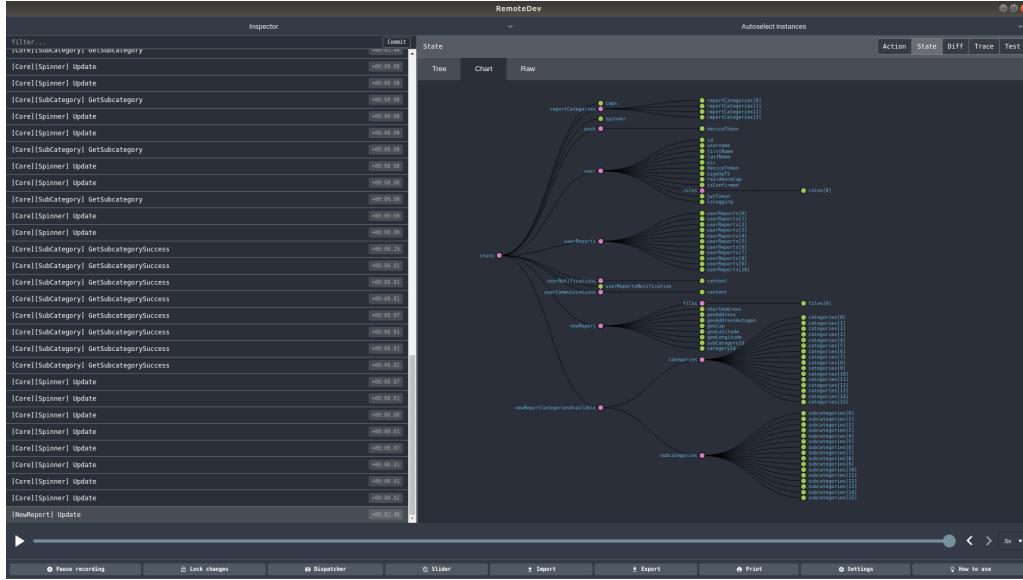


Figura 2: Without NgRx



Figura 3: Case study su eServant



2.2 SOCIAL LOGIN

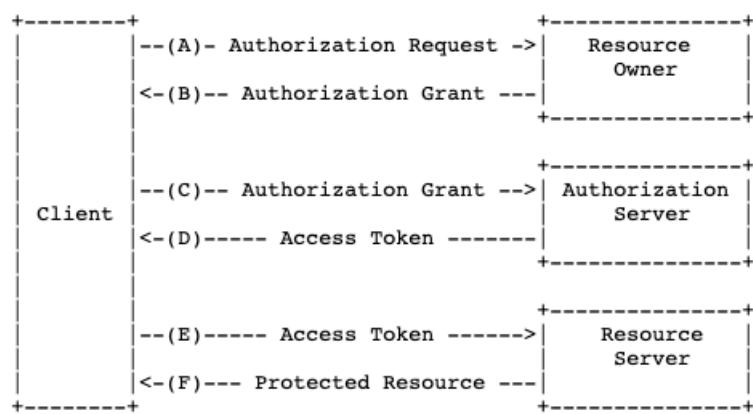
Al primo accesso all'APP eServant è necessario dare il consenso per almeno uno dei seguenti social:

- 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 OAuth 2.0 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 e obsoleta il protocollo OAuth 1.0 descritto in RFC 5849.

OAuth 2.0 flow



Attori

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 di proprietario di 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

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 due 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

Visto che questa feature di eServant è un software standalone, l'integrazione all'interno dell'app Ionic è stata fatta tramite iframe. Frammento di codice:

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

Figura 4: Selezione POI



Figura 5: Avvio navigazione verso il POI



2.4 GEOLOCALIZZAZIONE

Come ho 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 mutarsi 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 tecnologie 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

Risultato finale:



2.4.1 QRcode



I codici QR sono stati creati nel 1994 da Denso Wave, 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:

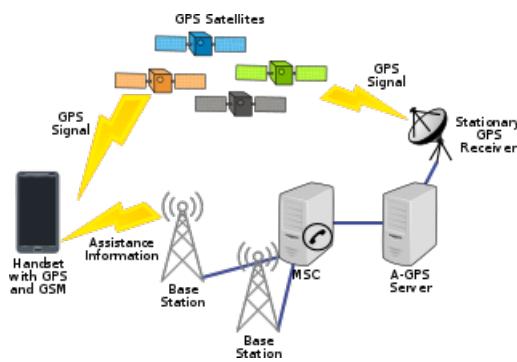
1. Scansione QRcode tramite l'app di eServant
2. Decodifica del QRcode e ottenimento del POI ID
3. Invio del POI ID, insieme all' ID dell'utente, al server
4. Il server, una volta ottenute le coordinate del POI, provvede ad aggiornare l'ultima posizione dell'utente

```
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);
});
```

2.4.2 GPS



```
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



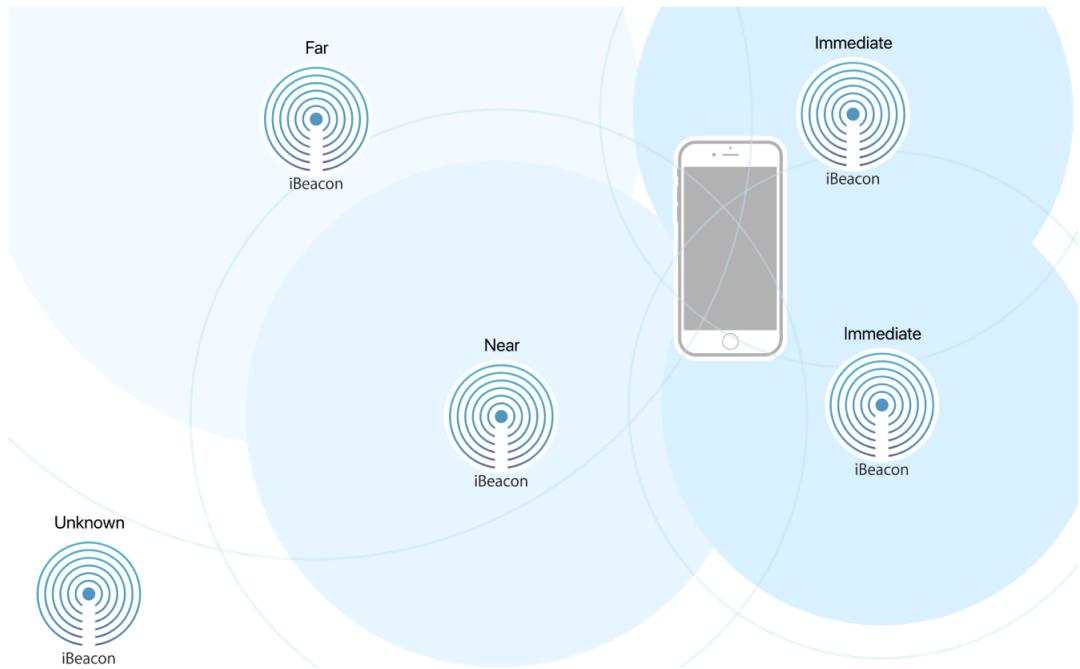
iBeacon è una tecnologia basata su Bluetooth Low Energy. A partire dal 2013 iBeacon è integrato in Apple iOS 7. Il primo progetto pilota è stato lanciato nei negozi Apple nel Dicembre 2013.

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.

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



Frammento di codice Ionic:

```
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:

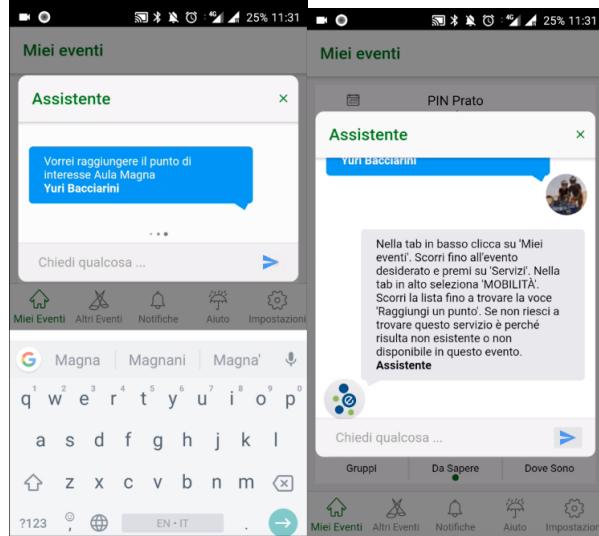
- Intent: rappresenta l'azione presente nella frase elaborata, è generalmente un verbo.
- Slot: rappresenta uno o più attributi che si legano all'intent. In molti casi l'intent è lo stesso, cambiano solo i valori degli slot.

Vediamo nel caso specifico di eServant.

Vorrei raggiungere il punto di interesse Aula Magna

Analisi tramite NLP:

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

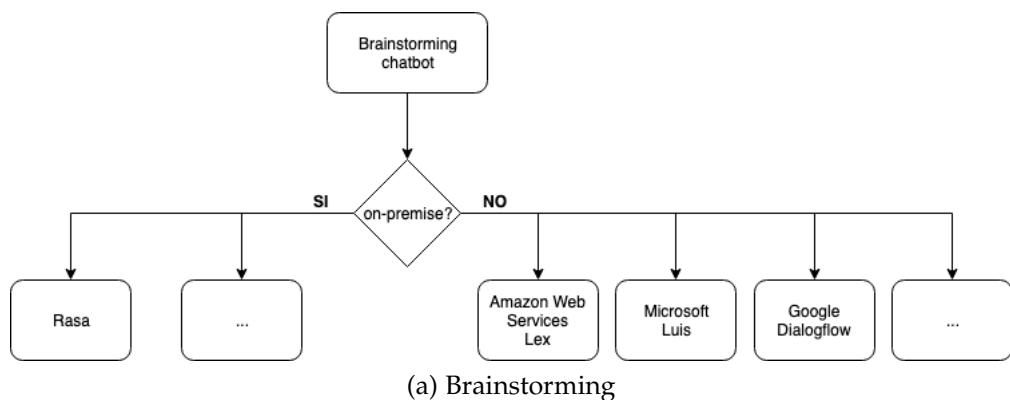


(a) Domanda

(b) Risposta

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

2.5.1 Brainstorming



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

Vediamo nel dettaglio le differenze principali tra le due strade:
Cloud

pro

- servizio pronto all'uso
- semplice da configurare, allenare, mantenere
- riceve aggiornamento continuo
- scalabilità

contro

- costo non basso (generalmente per singola chiamata)
- privacy, i dati vengono trasmetti sui server del cloud provider
- non portatile

In-house

pro

- privacy, i dati non escono dalla workstation
- controllo del provider infrastrutturale

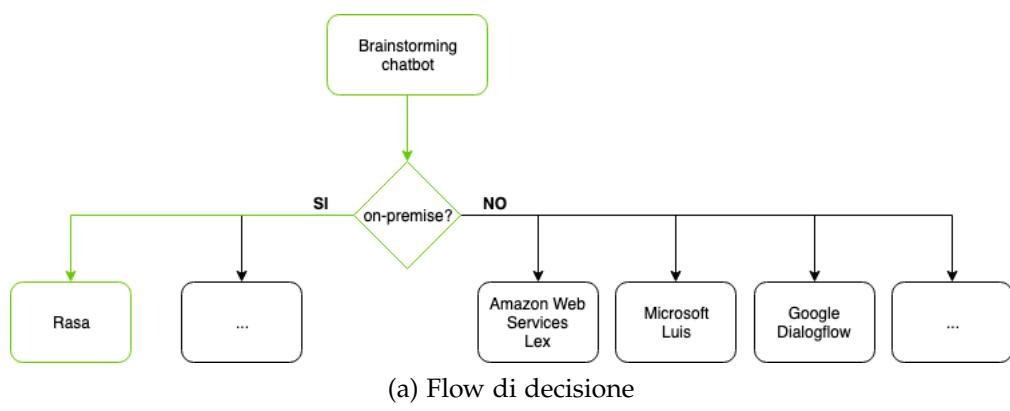
contro

- tempi di apprendimento e sviluppo maggiori
- mantenimento

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

Empty thesis model

3.1 FRAMEWORK ANGULAR

3.2 MATERIAL DESIGN

3.3 FIREBASE

3.3.1 *Pro/Contro servizi cloud*

3.3.2 *Real Time Database*

3.4 CI JENKINS

4

IOT

Empty thesis model

4.1 BLUETOOTH LOW ENERGY

4.1.1 *Gateway (attivo)*

4.1.2 *Smartphone (attivo/passivo)*

4.1.3 *iBeacon Box (passivo)*

4.1.4 *iBeacon Wearable (passivo)*

4.2 CONTA PERSONE

4.3 TELECAMERE DENSITA'