Università degli Studi di Catania
Dipartimento di Matematica e Informatica
Corso di Laurea in Informatica triennale

Francesco Di Franco

Sistema cloud based per la gestione real-time di eventi geolocalizzati.

Relazione progetto finale

Relatore
Chiar.mo Prof. S. Riccobene

Correlatore
Dott. G. Patanè

Anno Accademico 2015/16

Indice

Capitolo 1

Introduzione

L'elaborato si propone di presentare un progetto aziendale, promosso da Park Smart srl [1], relativo alla gestione real-time di eventi geolocalizzati. Park Smart è una startup che opera nel campo delle smart cities, proponedo un progetto innovativo il cui scopo è semplificare la vita in città, aiutando chi è alla ricerca di un parcheggio per la propria auto.

La soluzione proposta offre un sistema composto da tre componenti fondamentali:

1. Sistema di videoanalisi.
2. Una piattaforma web-services oriented per la gestione degli eventi.
3. Applicazione mobile per l'utente finale.

In questo elaborato verrà trattato il sistema di gestione real-time degli eventi: qunado una telecamera, addetta al monitoraggio di una serie stalli di parcheggio, tramite avanzati algoritmi di videoanalisi, rileva un cambio di stato si occuperà di notificarlo. Tale evento sarà elaborato e notificato all'utente finale in prossimità dell'evento stesso.

Le scelte tecnologiche fatte sono frutto di un accurata analisi comparativa e tutti i moduli software descritti in questo documento sono stati progettati e sviluppati dall'autore. Sotto supervisione dell'azienda ogni passo dello sviluppo ha seguito un iter di validazione composto da due fasi di test.

[1] www.parksmart.it

Capitolo 2

Obiettivi

In questo elaborato verranno descritte le principali tecnologie usate e le valutazioni tecniche che hanno portato alle scelta delle stesse. Alcune scelte come il sevice provider o la specifica tecnologia di Database Managment System (DBMS) sono state influenzate da fattori interni all'azienda motivati non solo da fattori tecnologici ma anche da particolari partnership e/o aggevolazioni di natura economica.

2.1 Problematiche affrontate

La crezione di un sistema real-time basato su tecnologie cloud per la gestione di eventi geolocalozzati necessita di diverse valutazione e in particolare sono stati attenzionati i seguenti punti:

- Scalabilità delle risorse
- Latenza tra il verificarsi dell'evento e la notifica degli utenti
- Sicurezza

Scalabilità delle risorse

L'avanzare delle tecnologie in ambito informatico e delle telecomunicazioni ha apportato diversi cambiamenti nell'ambiente dei sistemi distribuiti e in particolare alla nascita di quello che oggi chiamiamo Cloud. Per cloud oggi si intende la modalità di strutturazione, orchestrazione ed erogazione on-demand di risorse informatiche tramite internet come storage, computing o hardware configurabile. Tali servizi sono completamente ospitati e configurati da provider che su richiesta, tramite

procedure automatizzate, forniscono tali risorse in maniera rapida all'utente che le richiede. I servizi cloud sono divisi in tre categorie fondamentali:

- SaaS (Software as a Service) - Servizi che consentono l'utilizzo di software installati su macchine remote
- DaaS (Data as a Service) - Servizi di storage che consentono la memorizzazione persistente di dati su macchine remote
- HaaS (Hardware as a Service) - Servizi che permettono l'elaborazione di dati in modo distribuito

I vantaggi che derivano dall'utilizzo di tali servizi sono la semplicità e la trasparenza con cui le risorse vengono erogate e/o aggiunte a sistemi già esistenti. Oggi la capacità di un sistema di adeguarsi al carico viene definità scalabilità cioè aumentare o diminuire le risorse in funzione delle necessità.Tale proprietà viene considerata un parametro di qualità determinante in numerosi ecosistemi hardware-software. La distribuzione del carico e la massimizzazione delle prestazioni è un fattore determinante per sistemi software destinati all'utilizzo su vasta scala.

L'architettura proposta fa uso per lo più di servizi SaaS altamente scalabili e il provider scelto è Amazon le cui tecnologie, denominate Amazon Web Services, sono tra le più utilizzate in ambito enterprise.

Latenza tra il verificarsi dell'evento e la notifca degli utenti

Uno degli aspetti fondamentali per migliorare l'esperienza dell'utente è il fattore real-time cioè riuscir a visualizzare in tempo reale lo stato dello stallo di parcheggio interessato e nel contempo rimanere informato se lo stesso cambia di stato. La problematica più rilavante mentre si sta percorrendo un tragitto verso lo stallo interessato è appunto occupazione di quest'ultimo da parte di un'altro automobilista. La notifica del cambio di stato in real-time è un aspetto molto curato che permette alla logica interna dell'applicazione per smartphone di cercare lo stallo libero più vicino a quello interessato. Per contattare l'utente in caso di un eventuale aggiornamento verrà usata la tecnologia "web-socket" che offre un canale di comunicazione full-duplex tra client e server rendendo cosi molto più semplici le comunicazioni dal server verso il client. In tal modo si riesce a diminuire l' I/O riducendo le richieste (API calls) da parte del client e si fa in modo che il client sia contattato solo quando si verifichi un evento.

Sicurezza

Un aspetto che non può essere trascurato è quello della sicurezza e nello specifico le modalità di autenticazione e comunicazione.

Prima di poter cominciare una qualsiasi comunicazione ogni entità dovra essere autenticata e nello specifico le entità che dovranno comunicare e di conseguenza essere autenticate sono due:

- Utenti
- Telecamere

Autenticazione

Formalmente per autenticazione si intende quel processo, durante una comunicazione, tramite il quale un entità verifica che l'altra parte sia realmente chi dichiara di essere. Le due modalità di autenticazione sono profondamente diverse a causa della diversa natura delle entità.

Un utente per usufruire del servizio dovrà registrarsi e di conseguneza cedere alcune sue informazioni personali per essere autenticato. In ausilio a questa procedura verrà utilizzato un sevizio AWS denominato "Cognito" che permette, dopo aver effettuato il login su un qualsiasi identity provider (Facebook, Google, Parksmart stessa), di verificare l'dentità dell'utente e mantenere la consistenza di un dataset personale utile alla sincronizzazione di più dispositivi. Cognito usa un maccanismo che verifica il token fornito dal dispositivo utente con l'identity provider e, dopo un eventuale conferma, inizializza l'identità e la registra in una pool alla quale fanno riferimento delle Access Control List (ACL) che specificano le capabilities della pool. In questo modo si ha la possibilità di gestire diversi gruppi di utenti assegnando o rimuovendo dinamicamente risorse specifiche.

L'autenticazione delle telecamere invece usa un meccanismo basato sulla certificazione digitale "X.509" che permette segrettezza e autenticazione, tali certificati sono denominati "self signed" poichè rilasciati dall'azienda stessa ed hanno validità solo all'interno dell'ecosistema Parksmart. In tal modo ogni entità che vuole inviare un aggiornamento dovrà essere fornita di un certificato valido per completare l'handshake iniziale. Grazie a questo meccanismo si garantisce segretezza nella comunicazione grazie alla crittografia asimmetrica data dallo standard X.509 e si riesce ad autenticare l'entità tramite la verifica del certificato.

Comunicazione

Dopo che il processo di autenticazione è andato a buon fine l'utente potrà richiedere i dati relativi alla zona in prossimità della sua posizione. Il protocollo usato per questo tipo di cominicazioni sarà l' "HTTPS" (HTTP + TLS) che garantisce segretezza e integrità.

Per la comunicazione con le telecamere varrà usato un protocollo molto snello in termini di banda, denominato "MQTT" usa un meccanismo di publish/subscribe e trova molte applicazioni nell "Internet of Things" (IoT).

2.2 Obiettivi raggiunti

Con lo sviluppo di questo progetto sono stati creati due prototipi applicativi completi per la gestione real-time dell'occupazione di dive

Capitolo 3

Rquisiti

Dovrà essere progettato un insieme di componenti software per la gestione real-time dell'occupazione di diversi stalli di parcheggio identificati in deverse aree geografiche. Una serie di stalli di parcheggio sono monitorati da una videocamera connessa a internet e, tramite l'elaborazione dell'immagine, in tempo reale sarà notificato se uno stallo verrà liberato o occupato. Per facilitare la gestione delle coordinate geografiche verrà usata la tecnica di clusteriing geografica denominata "geohash" che suddivide in una griglia l'intero emisfero e permette di identificare un'area geografica con un'arbitraria precisione tramite una sequenza di bit. Sarà possibile reperire, dopo necessaria autenticazione, un insieme di stalli di parcheggio indicando il geohash interessato. Le informazioni che dovranno essere memorizzate per stalli e videocamere sono, oltre ad un id univoco e mnemonico, coordinate GPS e geohash ricavato dalle coordinate. In ogni entità oltre alle informazioni sulla localizzazione saranno presenti le rispettive relazioni relative al monitoraggio: ogni stallo di parcheggio dovrà contenere informazioni sulle videocamere che lo stanno monitorando e ogni videocamera dovrà contenere informazioni sugli stalli di parcheggio che sta monitorando. Tutte queste informazioni dovranno essere memorizzate in maniera persistente in una base dati e a supporto di questa dovrà essere presente un meccanismo di cache per velocizzare le comunicazioni.

Una videocamera quando rileva il cambio di stato di uno stallo di parcheggio, dopo previa autenticazione, dovrà notificare l'evento fornendo l'id dello stallo e il nuovo stato ("free" o "busy").

Ogni utente dopo aver effettuato il login e aver ottenuto un token di sicurezza potrà reperire le informazioni relative a gli stalli di parcheggio contenuti nel geohash corrispondente alla posizione occupata dall'utente. Ad ogni richiesta il client fornirà il token che ha ottenuto tramite il login e la propria posizione GPS dalla quale verrà calcolato

il geohash.

Dopo che la procedura di login sarà andata a buon fine e dopo aver reperito le informazioni relative alla propria posizione il client può creare un collegamento con il server tramite web-socket per ricevere gli aggiornamenti sugli stalli. Sarà previsto pure un handshake tramite web-socket dove verrà verificata la validità del token fornito come primo messaggio.

3.1 Scalabilità delle risorse

L'architettura che sarà progettata dovrà far fronte a molteplici richieste simultaneamente e per facilitare la scalabilità orizzonatale tale architettura dovrà essere per lo più "stateless" cioè nessuna informazione sullo stato dei client dovra essere memorizzata. Le comunicazioni saranno effettuate tutte tramite uno specificato endpoint che si occuperà di ditribuire le richieste in maniera equa al sistema cloud. Per velocizzare la fase di verifica del token di sicurezza potrà essere usata cache condivisa dove saranno memorizzate le ultime credenziali usate dall'utente e la validità del token memorizzato in millisecondi.

3.2 Ottimizzazione dell' I/O sul DB e gestione della cache

Per ridurre l'I/O sul database dovrà essere presente un meccanismo di cache condivisa a tutta l'infrastruttura back end che segua la politica "last recently used" (LRU). Tale politica usata con il meccanismo del "Time to live" (TTL) dovrà ridurre i tempi di elaborazione delle richieste e il numero effettivo di query sul DB. In cache dovranno essere mantenute le suguenti informazioni:

- Singolo stallo di parcheggio e relativo stato.
- Aggregato di stalli di parcheggio con i relativi stati.
- Token forniti dagli utenti e relativi propietari.

tali informazioni dovranno mantenere la consistenza rispetto a quelli memorizzati nel database: ad ogni update da parte di una telecamera il dato dovrà essere aggiornato.

Il TTL dovrà essere modificato in base alle esigenze e adeguarsi al carico della cache: il TTL sarà direttamente proporzionale allo spazio disponibile in cache e valori del TTL oscilleranno tra 6 ore (6 * 3600 * 1000 millisecondi) e 1 secondo (1000 millisecondi).

3.3 Notifica utenti

L'utente dopo aver completato l'handshake tramite web-socket potrà rimanere in ascolto sul canale per eventuali aggiornamenti e nello specifico, grazie al meccanismo dei namespace, potrà essere aggiornato solo degli eventi in prossimità della sua posizione.

Capitolo 4

Tecnologie

Per portare a termine il progetto sono state valutate diverse tecnologie, e in particolare uno degli obbiettivi che ha influenzato l'analisi delle stesse è satto evitare o ridurre al minimo il "provisoning", cioè quel processo in cui un amministartore di sitema configura l'ambiente di lavoro/sviluppo e assegna risorse e permessi a gli utenti. Con le tencologie proposte da AWS questa particolare fase di configuarezione iniziale è molto ridotta e a volte assente. Per questo motivo circa il 90% delle tecnologie sono ospitate da AWS di cui circa la metà completamente propietaria AWS.

4.1 Analisi comparativa

Le tecnlogie che sono state analizzate abbracciano tre categorie di software:

1. Basi di dati
2. Server side framework
3. Web socket

4.1.1 Basi di dati

Tra le tecnologie moderne possiamo trovare diverse soluzioni per la memorizzazione persistente di informazoini in un base dati e molte di queste, rispetto al classico approccio relazionale in cui ogni entità deve avere uno schema ben definito, offrono diversi vantaggi come:

- Struttura "schema less"
- Possibilità di esecuzione su un ambiente distribuito

- Possibilità di referenziare/includere altre entità

Questa categoria di database viene denominata "Not only SQL" (NoSql), e tali sistemi si distunguono appunto dal modello classico per la loro flessibilità, scalabilità e alta resa in termini di presatzioni; requisiti che oggi sono reputati fondamentali per applicazioni enterprise. Le tecnologie di DBMS prese in esame per questo progetto sono due e in particolare verranno analizzati DynamoDb di AWS e MongoDb di MongoDb Inc.

DynamoDb

DynamoDb è una soluzione enterprise completamente sviluppata e ospitata da AWS e questo rappresenta uno dei suoi principali vantaggi, infatti offre la possibilità di gestire le prestazioni e le risorse allocate in modo semplice da un pannello di controllo dedicato. DyanamoDb è schema less il che vuol dire che non esistono limiti sul numero di attributi che un record può avere e indipendentemente dagli altri ognuono può avare un numero arbitrario di attributi ad eccezzione di una chiave che deve essere univoca e presente per ogni record. DynamoDb supporta due tipologie di chiavi

- Partition key (Hash key): tale chiave ha la funzione di una chiave primaria e prende il nome di hash key poichè serve de input ad una funzione hash (interna ad AWS) che calcola la posizione (partizione) dove il record è memorizzato.
- Partition key and sort key (Hash and range key): è una chiave primaria composta da due attributi, la prima è la partition key e la seconda, sort key, viene usata come chiave di ordinamento. Tutti i record con la stessa partition key saranno ordinati secondo la sort key.

Per facilitare le ricerche DynamoDb offre la possibilità di definire per ogni tabella ulteriori indici e in particolare:

- 5 indici globali secondari (global secondary index)
- 5 indici locali secondari (local secondary index)

DynamoDb è un DBMS pensato per essere performante e altamente scalabile perciò non offre alcun tipo di referenziamento con altre tabelle e di conseguenza tutte le operazione ad esse correllate come join o transazioni. Le operazioni che si possono effettuare con DynamoDb sono le seguenti:

- GetItem : data la primary key (Hash) viene reperito il record corrispondente.
- PutItem : inserisce un nuovo record nel database.
- Query : reperisce record secondo indici secondari e offre la possibilità di aggiungere condizioni.
- Scan : offre la possibilità di fare query su attributi non chiave.
- UpdateItem : aggiorna un record esistente.
- DeleteItem : elimina uno specifico record dal database.
- BatchGetItem : reperisce record anche da più tabelle.

MongoDb

MongoDb è un DBMS molto utilizzato e memorizza i dati secondo il modello a documento. Anch'esso è schema less ed è stato pensato per applicazioni in continua evoluzione. Completamente gratuito deve essere ospitato da un provider di terze parti e di conseguenza le prestazioni dipendono dall'ambiente in cui è eseguito. MongoDb offre sia la possibilità di refernziare sia la possibilità di includere altre entità (documenti). Come altri DBMS nella categoria NoSql anche Mongo può essere eseguito su un ambiente distribuito per favorirne prestazioni e scalabilità. Ogni documento in una collezzione è identificato da un id univoco, tale id oltre che essere fondamentale per l'engine di mongo è molto utile per refernziare altri documenti o effettuare query complesse. Diversamente da altri DBMS NoSql offre diverse tipologie di indici:

- Indici secondari: utili a rendere performanati le ricerche
- Indici geospaziali: utili per calcolare distanze tra coordinate geospaziali
- Indici geohaystack: possono essere usatecome gli indici geospaziali ma permettono solo interrogazioni su superfici piane, dando prestazioni più elevate
- Indici testuali: permettono ricerche full-text

MongoDb è una tecnologia molto usata e

DBMS a confronto

Feature	MongoDb	DynamoDb
Hosting	Bare metal o servizi di terze parti	Amazon
Data model	Modello a documetni	Modello chiave-valore Modello a documenti
Licenza	OpenSource	Propietario ma
Chiavi/Indici	Primary key, Hashed index, Geospatial index, Geohystack index, Text index	Hash key, Hash and range, 5 global secondary index, 5 local secondary index
Modalità di accesso	Protocollo propietario	RestAPI (HTTP)
Trigger	Si	Lambda AWS
Mapreduce	Si	Si (EMR)
Gestione testo	RegEx, Substring	Substring

4.1.2 Web services e server side framework

Per favorire l'interoperabilità tra diverse applicazioni, eseguite in diverse piattaforme e/o in ambienti distribuiti come il web, il "world wide web consortium" (W3C) ha delineato un architettura denominata "Web service":

> *A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards.*

Tale architettura, definita come delle linee guida dal W3C, ha portato a quello che oggi è nota come architettura "Representational State Transfer" (REST), introdotta per la prima volta nel 2000 da Roy Fielding. L'architettura REST si basa su un concetto fondamentale che è la risorsa (informazione) identificata da un URI. Ogni sistema software che vuole implementare l'architettura REST deve seguire le seguenti regole:

- Client-server: Netta separazione dei ruoli tra client e server.
- Stateless: Nessuna informazione sullo stato del client deve essere salvata eliminando il vincolo di dover contattare sempre lo stesso server.

- Cacheable: Le risorse inviate dal server dovrebbero essere memorizzabili per un secondo utilizzo.
- Layered system: Sono previsti livelli intermedi come proxy per migliorare la scalabilità e aggiungere livelli di sicurezza.
- Uniform interface: Le interfacce di comunicazione dovrebbero appartenere tutte alla stessa famiglia (Es. HTTP).

Attualmente esistono molti framework che permettono di implementare l'architettura REST e i più noti sono:

- NodeJs (Javascript)
- Laravel (PHP)
- Ruby on rails (Ruby)

In questo elaborato verrano prese in esame solo NodeJs e Laravel.

NodeJS

Anche se javascript è nato come un linguaggio che doveva essere eseguito solo all'interno di un browser con lo sviluppo della versione 8 dell'interprete javascript di Chrome, noto browser di Google, le performance sono notevolmente migliorate al punto che sono nati framework server-side come NodeJs. NodeJs è un "event-driven" framework che grazie alla natura asincrona di javascript riesce ad implementare in maniera soddisfacente il modello "non-blocking I/O". Pensato per costruire applicazioni scalabili riesce a gestire numerose richeste simultaneamente e corredato di "packet manager" gode della più grande libreria di moduli (relativi al framework) open-source del web.

Laravel PHP

Laravel è un framework PHP object oriented che si basa sull'architettura "Model View Controller" permette di ottenere un ottima organizzazione sia logica che strutturale. I due concetti fondamentali di questo framework sono:

- Routes: identifica una risorsa web.
- Controller: specifica un insieme di funzioni, utili ad organizzare il codice.

4.2 Tecnologie selezionate

Per lo sviluppo di questo progetto sono stati selezionati DynamoDb e NodeJs. DynamoDb poichè completamente ospitato e altamente scalabile e performante. NodeJs invece per la sua natura non bloccante e basato sul modello ad eventi. Oltre alle tecnologie appena esamaminate sono state usate anche:

- Elastic beanstalk: servizio AWS, basato su docker (container virtuali), permette scalabilità e "continuos deploy".
- Redis: all-in-memory key-value database, utilizzato come cache per i dati più richiesti.
- AWS IoT: sistema di gestione per IoT basato sul protocollo MQTT.

4.2.1 Elastic beanstalk

Elasitic beanstalk è un servizio SaaS di AWS basato sulla tecnologia introdotta da "Docker Inc." di virtualizzazione denominata "container". Tale tecnologia ottimizza il concetto di virtualizzazione sfruttando lo stesso principio di isolamento delle risorse che viene usato per le macchine virtuali ma utilizza un approccio architetturale molto diverso:

un container è un ambiente virtualizzato isolato per un'applicazione e include tutte le dipendeze necessarie all'esecuzione della stessa ma condivide il kernel con il resto dei container. Questa tecnologia, a differenza delle macchine virtuali, virtualizza solo l'applicazione e permette quindi di sfruttare al meglio la memoria offrendo una scalabilità orizzontale più efficente.

ELastic benastalk integra questa tecnologia con numerevoli feature di contorno:

- Deploy rapido tramite "command line".
- Versioning dell'applicazione trasparente grazie al collegamento del container ad un repository gestito con un control version system (CVS).
- Scalabilità gestita in un paio di click: le istanze virtualizzate possono essere aumentate o diminuite secondo semplici regole.
- Permette lo swap tra container rendendo i deploy trasaprenti dall'esterno riducendo i downtime tra una versione e un'altra.

4.2.2 Redis

Redis è una tecnologia di datatbase NoSql denominato "key/value store", risiede completamente in memoria Ram con persistenza su disco facoltativa. Le principali peculiarità di Redis sono:

- Possibilità di esecuzione in ambiente distribuito
- Ad ogni chiave può essere assegnato un "Time To Live" (TTL), utile se redis viene utilizzato come LRU cache.
- Meccanismo di publish/subscribe utile per la consistenza dei dati in ambienti distribuiti.
- Si possono eseguire transazioni: una sequenza di istruzioni che non varrà interrotta.

Capitolo 5

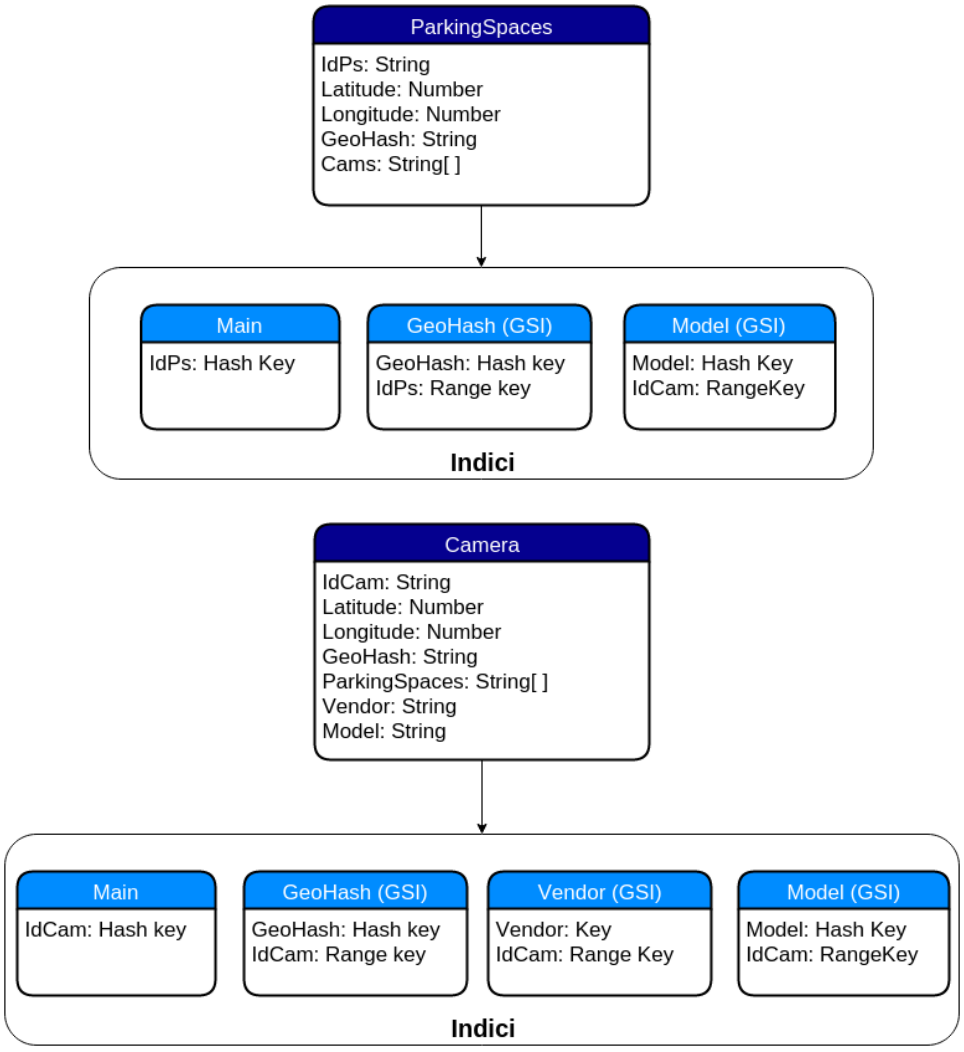
Progettazione software

5.1 Classificazione eventi

Nell'ambito dell'attività di sviluppo del software di videoanalisi del progetto Parksmart sono stati implementati alcuni algoritmi di riconoscimento di particolari eventi, comprendenti lo stato di occupazione di singoli stalli di parcheggio. Il software cosi prodotto viene eseguito su sistemi embedded dedicati al fine di distribuire la computazione dell'attività di riconoscimento per informare l'utente su quale sia lo stallo di parcheggio libero più vicino. Allo stato attuale la classificazione degli eventi comprende esclusivamente la variazione di stato di uno stallo di parcheggio da libero a occupato e viceversa da occupato a libero, ma saranno successivamente introdotti alre categorie di eventi per ampliare il ventaglio di caratteristiche di prodotto sviluppate dal progetto Parksmart.

5.2 Database (DynamoDb AWS)

Le entità che sono memorizzate nel DBMS in maniera persistente sono telecamere e stalli di parcheggio. Oltre a gli attributi definiti dai requisiti sono stati aggiunti alcuni attributi, membri di indici globali secondari, utili all'amministrazione. Data la tecnologia selezionata (DynamoDb) lo schema progettato ha la seguente struttura:



Le operazioni che sono state implementate sono le seguenti:

- Aggiornamento dello stato (Free/Busy) di uno specifico stallo di parcheggio.

- Richiesta di stalli di parcheggio di un geohash: da un geohash in input reperiscono tutti gli stalli di parcheggio che vi appartengono.
- Richiesta di stalli di parcheggio a partire da una posizione : data una posizione arbitraria si calcola il geohash e si reperiscono tutti gli stalli che vi appartengono.
- Richiesta di stalli di parcheggio a partire da un area geografica delimitata da una coppia di coordinate : date due coppie coordinate si calcolano tutti i geohash all'interno dell'area e si reperiscono tutti gli stalli che vi appartengono.
- Richiesta di videocamere a partire da un geohash : da un geohash in input reperiscono tutte le videocamere che vi appartengono.
- Richiesta di videocamere a partire da una posizione : data una posizione arbitraria viene calcolato il geohash e si reperiscono tutti le videocamere che vi appartengono.
- Richiesta di videocamere a partire da un area geografica delimitata da una coppia di coordinate : date due coppie coordinate si calcolano tutti i geohash all'interno dell'area e si reperiscono tutte le videocamere che vi appartengono.
- Richiesta di videocamere per modello : dato uno specifico modello di videocamera vengono reperite tutte le videocamere di quel modello.
- Richiesta di viedocamere per produttore : dato un produttore di videocamere si reperiscono tutte le viedeocamere vendute da quel produttore.

Date le operazioni appena descritte quelle che saranno utilizzate nell'architettura di interesse sono solo quelle relative a gli stalli di parcheggio.

5.3 NodeJs

L'applicazione in nodejs è composta da due componenti fondamentali:

- Moduli: librerie usate all'interno dell'applicazione.
- Routes: identificatori di risorse esposte all'esterno.

I moduli di terze parti che sono stai utilizzati comprendono “Express js” un framework utile per il routing, “ngeohash” libreria utile per codificare e decodificare dei geohash, “AWS Sdk” che contiene tutte le librerie necessarie per comunicare con i servizi AWS (Es. DynamoDb), “node-redis” modulo per eseguire le operazioni in cache e infine “socket.io”+“socket.io-redis” per notificare gli eventi a gli utenti. E’ stato creato un modulo (“psm”) ad-hoc per i nostri scopi che comprende i seguenti file:

1. ParkingSpace.js con le seguenti funzioni “pubbliche”:
    - updateParkingSpace(idPs) : aggiorna lo stato (Free/Busy) di uno specifico stallo di parcheggio.
    - getPsByHash(hashSource): da un geohash in input reperiscono tutti gli stalli di parcheggio che vi appartengono.
    - getPsByPosition(latitude, longitude) : data una posizione arbitraria si calcola il geohash e si reperiscono tutti gli stalli che vi appartengono.
    - getPsByBounds(latHi, latLo, lngHi, lngLo) : date due coppie coordinate si calcolano tutti i geohash all’interno dell’area e si reperiscono tutti gli stalli che vi appartengono.
2. Camera.js con le seguenti funzioni “pubbliche”:
    - getCamsByHash(hashSource) : da un geohash in input reperiscono tutte le videocamere che vi appartengono.
    - getCamsByPosition(latitude, longitude) : data una posizione arbitraria viene calcolato il geohash e si reperiscono tutti le videocamere che vi appartengono.
    - getCamsByBounds(latHi, latLo, lngHi, lngLo) : date due coppie coordinate si calcolano tutti i geohash all’interno dell’area e si reperiscono tutte le videocamere che vi appartengono.
    - getCamsByModel (model): dato uno specifico modello di videocamera vengono reperite tutte le videocamere di quel modello.
    - getCamsByVendor (vendor) : dato un venditore di videocamere si reperiscono tutte le viedeocamere vendute da quel produttore.
3. Websocket.js con le seguenti funzioni “pubbliche”:
    - sendMessage(channel, message) : invia un messaggio in uno specifico namespace.

A seguire un frammento di codice dal file PerkingSpace.js

```
var AWS = require('./AWS_config.js').AWS_config();
var dynamo = require('dynamodb-doc');
var geohash = require('ngeohash');
var docClient = new dynamo.DynamoDB();
var async = require('async');



var asyncWorker = function(hash, returnData){

    this.returnData=returnData;
    this.hash=hash;
    this.getHash = function(callback){


        var params = {};
        params.TableName = 'Parking_space';
        params.IndexName = 'GeoHash';

        params.KeyConditions = [docClient.Condition('GeoHash',
            'EQ', hash)];

        docClient.query(params, function(err, data){

            if(!err && data){
                Array.prototype.push.apply(returnData.Items,
                    data.Items);

            }
            callback(err, true);
        });


    }
}


function ParkingSpace(){



  return {

    ...
    ...

    getPsByPosition: function(lat, lng, callback) {

        if(lat && lng && callback && typeof(callback)==='
            function'){
            var hash = geohash.encode(lat, lng, 7)
```

```
            var hashArray = geohash.neighbors(hash, 7);
            hashArray[hashArray.length]=hash;
            var fnArray = [];
            var returnData = {};
            returnData.Items = [];
            for (var i in hashArray){
                fnArray[i]= (new asyncWorker(hashArray[i],
                    returnData)).getHash;
            }
            async.parallel(fnArray, function(err){

                  callback(null, returnData);

            });
        }
        else
            callback('Invalid params', null);


    },

    ...
  }
}

exports.ParkingSpace= ParkingSpace();
```

Listing 5.1: Code of getPsByPosition

Le URI esposte all'esterno sono le seguenti:

```
HTTP 1.1
GET /ps/getbyposition

Params:
{
  "lat" : Number,
  "lng" : Number
}
```

```
HTTP 1.1
GET /ps/getbyhash

Params:
{
  "hashsource" : String
}
```

```
HTTP 1.1
GET /ps/getbybounds

Params:
{
  "lat_lo" : Number,
  "lng_lo" : Number,
  "lat_hi" : Number,
  "lng_hi" : Number
}
```

```
HTTPs 1.1
GET /ps/update

Params:
{
  "IdPs" : String,
  "Free" : Boolean
}
```

5.4 Redis

5.5 Lambda AWS

Capitolo 6

Architettura

6.1 Architettura 1 (NodeJs, Redis, Socket.io)

6.2 Architettura 2 (AWS API Gateway, AWS IoT, AWS Lambda)

6.3 Architetture a confronto

Capitolo 7

Workflow, testing e tecnologie di supporto

Capitolo 8

Conclusione

Capitolo 9

Sviluppi futuri