**Obiettivi del sistema**

Il sistema è stato progettato al fine di semplificare la ricerca dei locali usando e comparando i risultati di diversi servizi come google, yelp e foursquare che a causa delle stringhe immesse dagli utenti producono risultati diversi e incompleti.  
Per questo motivo l’applicazione si pone di identificare i tag e l’opportuna identificazione geografica per ottimizzare i risultati ed includere più locali possibili.  
Inoltre essendo un servizio principalmente di consultazione che serve quando una persona si trova per strada.  
Il modello delle recensioni invece è interno al sistema.

**Design Goals**

**Adaptability**: Il sistema ottimizza le ricerche memorizzando i luoghi , dove ad ognuno di essi è associato una lista di locali. Se una ricerca è già stata effettuata da un utente in un lasso di tempo relativamente breve i tempi di risposta si ridurranno alla sola geolocalizzazione del luogo e alla lettura della lista dei locali.

**Usability** : Essendo un applicativo mobile deve essere semplice da comprendere e veloce da utilizzare quindi non sono presenti molte opzioni o form lunghi da compilare dato che non rispecchiano i principi di usabilità in ambito mobile.

In particolar modo c’è da tener conto del principio di iterazione con una solo mano , in cui si immagine che l’utente utilizza l’app con il pollice , ciò significa che la posizione dei widget deve essere strategica.

Un altro aspetto importante riguarda l’input, dove appunto il numero di tocchi deve essere ridotto al minimo , nel nostro specifico caso si riduce alla selezione delle categorie e al input della ricerca.

Eventualmente fossero necessari caricamenti , l’utente deve essere necessariamente informato dello status di avanzamento.

**Reliability :** Risulta fondamentale la capacità di evitare errori , per questo motivo l’esecuzione della ricerca con le informazioni relative ai locali deve essere vista come un’unità **atomica** che quindi premette solo la possibilità di riuscita o fallimento. Inoltre il servizio essendo un servizio web deve essere in grado di esaudire tutte le richieste dei client.

**Performance :** Il sistema ha come obiettivi di fornire risposte rapide e corrette agli utenti , tenendo presente della capacità di adattamento precedentemente descritta , le performance variano anche dalla “conoscenza” dell’app per questo motivo le performance hanno una dipendenza dai servizi REST usati.

L’obiettivo medio è di fornire una risposta in meno di 3 secondi. (Eventualmente mettere un sottopunto per la precisione del gps , anche se secondo me è inutile)

**Supportability :** Il concetto di supportability varia tra client e server , nel primo caso si riferisce esclusivamente al supporto tra le varie versioni di android , in particolar modo si cerca di ricoprire almeno il 90% dei dispositivi android invece per quanto riguarda il lato server il concetto di portabilità è intrinseco in java anche se rimane legato a tomcat

**-Maintainability :** (Sottopunto di supportability) Per semplificare il riadattamento a modifiche dovute alla

tecnologia o al cambiamento dei servizi rest usati il software sarà suddiviso in componenti e moduli   
 assestanti che interagiscono tra di loro al fine del corretto funzionamento generale.

**Definizioni,Acronimi e Abbreviazioni**

|  |  |
| --- | --- |
| Sigla/Termine | Definizione |
|  |  |

**Riferimenti**

Il progetto essendo un **Greenfield Project** significa chelo sviluppo comincia da zero, non esiste nessun sistema a priori e i requisiti sono ottenuti dall’utente finale e dal cliente.

Nonostante non esista un applicazione del genere, esistono servizi integrati in altre app o siti web che permettono di effettuare ricerche dei locali.

Come ad esempio **maps** o **tripadvisor** che permettono di cercare un locale o di ottenere una lista di punti d’interesse in un determinato luogo, per questo motivo questi due sistemi sono stati presi come punto di riferimento.

Per la stesura del seguente documento sono stati presi come punti di riferimento i seguenti testi:

- B. Bruegge, A.H. Dutoit, Object Oriented Software Engineering, Using UML, Patterns and Java Prentice Hall

-http://elearning.informatica.unisa.it/el-platform/mod/folder/view.php?id=9164 , System Design Documentation

**2-PROPOSED SOFTWARE ARCHITECTURE**

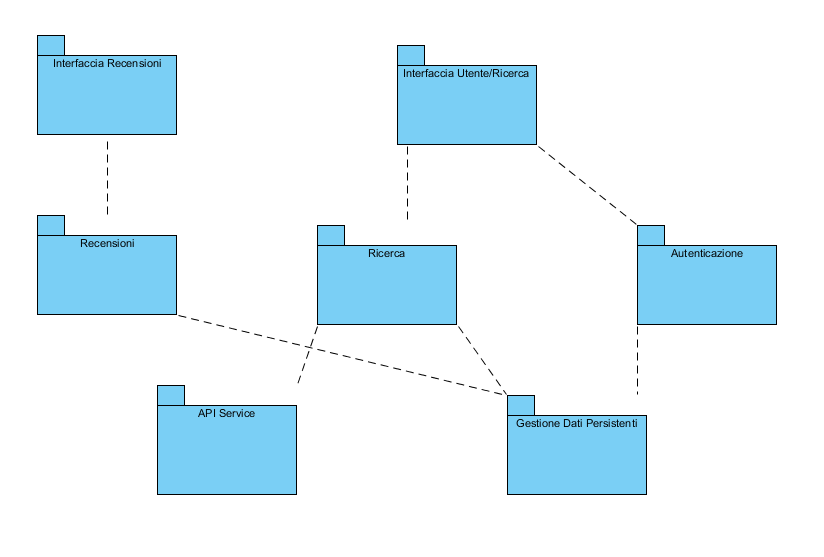
2.1 Panoramica

L’architettura scelta per il sistema è ibrida in quando sfrutta un modello client-server per quando riguarda le richieste inviate ed elaborate dal server , ma il server viene suddiviso in 3 layer Model-View-Control.

Il lato client si occupa di eseguire le richieste e decodificare e mostrare opportunamente le risposte nell’applicazione.

Il lato server presenta invece la logica e la conoscenza del dominio applicativo , il concetto di view diventa più generico e si occupa solo della formattazione dei dati in risposta alle richieste del client.

2.2 Decomposizione in sottosistemi

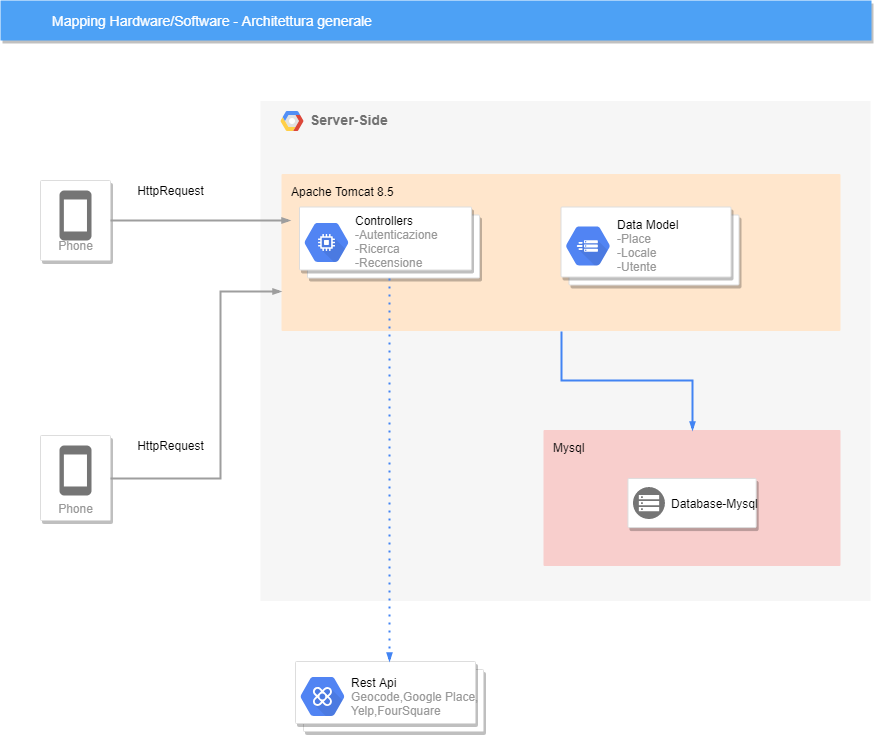


|  |
| --- |
| **Interface Layer** |
| Interfaccia recensioni | Raggruppa tutte le interfacce utente per la stesura e lettura di una recensione |
| Interfaccia Utente/Ricerca | Raggruppa tutte le interfacce che raggruppano i tipi di ricerca e il profilo utente |

|  |  |
| --- | --- |
| **Application Logic Layer** |  |
| Recensioni | * Permette la modifica di una recensione * Permette di aggiungere una recensione |
| Ricerca | * Permette di effettuare una ricerca dato in input un nome di un locale * Permette di effettuare una ricerca dato un luogo e delle categorie |
| Autenticazione | * Permette la registrazione al servizio (non richiede iterazione con l’utente) * Permette l’autenticazione al servizio (non richiede iterazione con l’utente) |

|  |
| --- |
| **Data Model Layer** |
| Api Service | * Si occupa del recupero dati dai webservice * Si occupa della geolocalizzazione del luogo |
| Gestione data persistenti | * Gestisce il luogo * Gestisce il locale e le recensioni * Gestisce l’utente |

2.3 Mapping Hardware/Software



Il sistema software è un ibrido tra client-server e mvc.

È possibile notare che la parte di **Presentation Layer** è spostata sul lato client che si occupa esclusivamente di inviare richieste ed elaborare graficamente ed opportunamente i dati ricevuti.

Il modello di client utilizzato è Fat-Client dato che il client è responsabile di gestire i dati e le connessioni con il server.

La comunicazione tra Client e Server avviene tramite protocollo HTTP (Hypertext Transfer Protocol), un protocollo di trasferimento di ipertesti utilizzato per trasmettere l’interazione tra Client e Server attraverso un meccanismo di request/response.

Il web-server scelto è Tomcat dove appunto è presente la parte logica dell’applicazione e la gestione dei dati , sulla stessa macchina risiede anche il database MySQL con la quale interagisce mediante JDBC.

Al di fuori del sistema sono presenti servizi necessari per l’identificazione del luogo e la ricerca dei locali , si tratta di end-point messi a disposizione da varie società che si collocano nel ambito dei webservice che permetteno **l’interoperabilità**  tra diversi elaboratori in un contesto distribuito.

Il modello Model-View-Control è presente nell’organizzazione del progetto server-side dove appunto ogni sottosistema implementa determinate operazioni in maniera indipendente, il layer di View si occupa solo di organizzare la risposta da inviare. Il sottosistema del model opera con i dati sfruttando la comunicazione con mysql e i controller valutano le richieste proveniente dall’esterno ed elaborano i dati da notificare alla view sfruttando l’iterazione con le api esterne e il sottosistema di data-model.

Le richieste inviate dai client , sul lato server sono di tipo stateful , ciò significa che viene mantenuta una sessione per ogni utente correttamente autenticato.

2.4 Persistent Data Manager

//Da integrare con le altri parti!