



SAPIENZA  
UNIVERSITÀ DI ROMA

# Analisi tecnica di "Fitmaps"

## una piattaforma ideata per la gestione e organizzazione di centri sportivi

Ingegneria dell'informazione, informatica e statistica  
Corso di Laurea in Ingegneria Informatica e Automatica

Candidato

Lorenzo De Santis  
Matricola 1849114

Relatore

Prof. Roberto Beraldi

Anno Accademico 2020/2021

Tesi non ancora discussa

---

**Analisi tecnica di "Fitmaps" una piattaforma ideata per la gestione e organizzazione di centri sportivi**

Tesi di Laurea. Sapienza – Università di Roma

© 2021 Lorenzo De Santis. Tutti i diritti riservati

Questa tesi è stata composta con  $\text{\LaTeX}$  e la classe Sapthesis.

Versione: 14 novembre 2023

Email dell'autore: [desantis.1849114@studenti.uniroma1.it](mailto:desantis.1849114@studenti.uniroma1.it)

# Indice

<b>1</b>	<b>Introduzione</b>	<b>1</b>
1.1	Realtà di interesse e idea di progetto . . . . .	1
1.2	Risultati attesi . . . . .	2
<b>2</b>	<b>Raccolta ed analisi dei requisiti ed esame delle metodologie e tecnologie utilizzate</b>	<b>5</b>
2.1	Specifica dei requisiti attraverso le <i>user stories</i> . . . . .	5
2.2	Illustrazione dei <i>mock-up</i> di interfaccia . . . . .	9
2.3	Autenticazione con servizi esterni mediante <i>Oauth</i> . . . . .	20
2.4	<i>Google Maps</i> API, <i>Place Autocomplete</i> . . . . .	22
2.5	Metodologia utilizzata per la conduzione e divisione del lavoro . . . . .	24
<b>3</b>	<b>Analisi concettuale del sistema</b>	<b>27</b>
3.1	Schema ER e discussione scelte critiche . . . . .	27
3.2	Glossario entità e relazioni . . . . .	29
3.3	Vincoli esterni . . . . .	31
<b>4</b>	<b>Progettazione del sistema</b>	<b>33</b>
4.1	Architettura esterna del sistema . . . . .	33
4.2	Architettura interna del sistema . . . . .	37
4.3	Progettazione logica del Data Layer . . . . .	38
<b>5</b>	<b>Realizzazione del sistema</b>	<b>41</b>
5.1	Data Layer . . . . .	41
5.2	Application Layer . . . . .	45
5.3	Presentation Layer . . . . .	47
<b>6</b>	<b>Validazione e dispiegamento</b>	<b>51</b>
6.1	Testing . . . . .	51
6.2	Installazione dell'applicazione . . . . .	54
6.3	Esecuzione . . . . .	54
	<b>Conclusioni</b>	<b>57</b>



# Capitolo 1

## Introduzione

In questo capitolo descriverò a grandi linee l'idea di progetto, la sua realtà di interesse e i risultati attesi.

### 1.1 Realtà di interesse e idea di progetto

Il progetto dell'applicazione nasce dalla necessità di avere una gestione centralizzata dell'organizzazione di strutture quali palestre e centri sportivi, in quanto molto spesso queste strutture sono carenti nell'organizzazione e gestione dei propri iscritti oltre che in quella dei propri dati interni. Inoltre, in una situazione di pandemia globale, un sistema efficiente e centralizzato può aiutare a prevenire lo scoppio di nuovi focolai all'interno di strutture sportive. Continuiamo illustrando le tipologie di utenti che si interfacciano con l'applicazione. In primo luogo abbiamo i proprietari delle strutture (*manager*), che hanno un controllo completo unicamente della propria organizzazione, poi ci sono i clienti e abbonati al servizio (*client*) che hanno a disposizione diverse interazioni con le strutture, ad eccezione, ovviamente, di quelle sulla loro gestione interna. Infine esiste un amministratore unico (*admin*) che si occupa di controllare che tutto funzioni correttamente e di risolvere gli eventuali problemi.

I clienti dei centri sportivi hanno spesso la necessità e la voglia di cambiare e di non allenarsi sempre nello stesso centro: grazie alla nostra applicazione è possibile avere un abbonamento globale e poi scegliere, giorno per giorno, in quale struttura allenarsi; affinché questa realtà sia realizzabile c'è però bisogno di un accordo fra tutti i gestori per poter avere un unico abbonamento riconosciuto ovunque.

Un obiettivo che ci eravamo prefissati era quello di introdurre una certa dinamicità nell'utilizzo dell'applicazione, non volevamo che fosse un sistema statico e privo di interazione e proprio per questa ragione abbiamo introdotto delle funzionalità, quali recensioni e un piccolo sistema di *instant messages*.

Al giorno d'oggi nessuno compra, o prenota nulla se non prima di aver letto l'opinione di un numero considerevole di persone attraverso le *reviews*, quindi ci siamo chiesti: "perché non applicarlo anche a questo ambiente?". Così è stato introdotto un sistema di valutazioni e recensioni per consentire agli utenti, vista la grande vastità di strutture e manager potenzialmente collegati da questo progetto, di non sentirsi disorientati nello scegliere il luogo del prossimo allenamento; confrontandosi con

altre persone possono capire realmente quale struttura soddisfa le proprie esigenze. Può accadere in qualunque momento che il manager abbia la necessità di informare tutti gli utenti prenotati alla sua struttura di un possibile evento, questo diventa praticamente indispensabile nella situazione di pandemia globale che stiamo vivendo in questi anni, c'è bisogno dunque di un sistema tempestivo che possa raggiungere più persone in poco tempo. Inviare un messaggio sul numero personale di ogni utente non ha sicuramente la stessa efficacia che inviare un messaggio direttamente sulla console di tutti i clienti interessati, senza considerare che potrebbero essere informate persone che non sono proprio entrate in struttura creando inutili allarmi; il manager semplicemente cliccando su un pulsante può mandare un messaggio istantaneo a tutti gli iscritti attualmente al suo centro.

Poter ricercare le strutture più vicine alla propria zona è fondamentale, il metodo più intuitivo per trovare il centro perfetto è integrare il sistema con una mappa digitale (un esempio sono le API offerte da *Google*) che consente di aver un riscontro visivo immediato che renda l'applicazione facilmente utilizzabile da tutte le tipologie di clienti.

A fronte della nuova realtà che siamo stati costretti ad affrontare, a nostro avviso, era necessario differenziare gli utenti che vogliono accedere alla struttura da quelli che realmente lo fanno; abbiamo quindi introdotto un sistema di prenotazioni molto semplice che si basa su tre fasce orarie (mattina, pomeriggio e sera). Per quanto riguarda la conferma dell'ingresso all'interno della struttura, fondamentale per sviluppare un sistema efficiente per la prevenzione della diffusione del virus *Covid-19*, abbiamo ideato un sistema di conferma mediante *token*.

## 1.2 Risultati attesi

Il risultato finale che ci aspettavamo era quello di un' applicazione strutturata su tre tipologie di utenze (*user*, *manager*, *admin*) con altrettante homepages per racchiudere le varie funzionalità. Tutte le pagine che sono accessibili da una tipologia di utente non lo sono dalle altre due: tale realtà si può modellizzare con tre alberi disgiunti, questo vuol dire che non vi sono pagine comuni tra utenze differenti e grazie a ciò non abbiamo falle nella sicurezza.

Il *client* nella sua homepage doveva avere una sezione per la ricerca della struttura e una pagina per la gestione delle sue prenotazioni, mentre la pagina di ricerca del centro doveva avere anche la mappa integrata ed esplorabile di *Google* per una gestione della posizione delle strutture. Un'altra caratteristica era quella di realizzare un sistema di prenotazione basato su tre fasce orarie e con la possibilità di prenotare un posto in qualsiasi giorno, oltre che un sistema di recensioni basato su un punteggio da 1 a 10. Queste ultime due funzionalità sarebbero state accessibili dopo la ricerca, sfruttando le mappe descritte precedentemente; infine, per non dover ricercare sempre uno stesso luogo, avevamo pensato ad una posizione configurabile di default, in modo che tutte le strutture situate all'interno di un determinato raggio venissero mostrate direttamente nell'homepage al log-in del client.

Nell' homepage del *manager* invece dovevano essere mostrate tutte le sue strutture e ovviamente la possibilità di inserirne una nuova sfruttando sempre le API di *Google*

*Maps* , inoltre accanto alla denominazione di ogni centro dovevano esserci le opzioni di invio e visualizzazione dei messaggi, oltre che, naturalmente, la possibilità della rimozione della struttura stessa.

Nella pagina principale dell' admin avremmo avuto la possibilità di rimuovere utenti, strutture e recensioni che non rispettino la norme di comportamento dell'applicazione.





## Capitolo 2

# Raccolta ed analisi dei requisiti ed esame delle metodologie e tecnologie utilizzate

In questo capitolo saranno esposte le specifiche e i requisiti dell'applicazione, arricchendo la trattazione con *mock-up* dell'interfaccia (ovvero una sommatoria rappresentazione di una specifica pagina); saranno infine analizzate le metodologie e tecnologie utilizzate, oltre alle modalità in cui è stato diviso il progetto.

### 2.1 Specifica dei requisiti attraverso le *user stories*

Il punto di partenza per illustrare con precisione il progetto è sicuramente iniziare illustrando le specifiche, che sono in questo caso espresse mediante le *user stories*, le quali verranno definite con precisione nelle prossime sezioni di questo capitolo; una loro classificazione è stata fatta sulla base del tipo di utente a cui sono rivolte.

Cominciamo dall' *unregistered user*, ovvero l'utente che non ha ancora creato alcun tipo di account e si sta interfacciando per la prima volta con l'applicazione.

*As an unregistered user I want to sign up with my e-mail and my data so that I can become a user or a user.*

L'utente deve avere la possibilità di creare un nuovo account di tipo manager o di tipo user.

*As an unregistered user I want to sign in with my nickname so that I can use user's and user's functionalities.*

Nel caso in cui l'utente abbia già effettuato una registrazione e quindi sia in possesso di un account di tipo user o manager, può entrare con le sue credenziali e usufruire delle funzionalità legate alla tipologia di iscrizione.

Continuiamo con lo *user*, l'utente che si è registrato ed ha effettuato il log-in con le

sue credenziali; di seguito sono esposte le funzionalità e servizi dei quali può avvalersi.

*As a user I want to have settings so that I can delete my account.*

Così come deve essere possibile creare un account, così deve, ovviamente, anche essere possibile rimuoversi dal sistema mediante l'accesso ad una specifica pagina di gestione dell'utente.

*As a user I want to have settings so that I can change my e-mail.*

Nella stessa pagina deve anche essere possibile cambiare la propria email, rispettando il requisito che questa non sia già in uso.

*As a user I want to have settings so that I can reset my password with e-mail.*

All'interno della pagina di gestione si può effettuare il cambio della password confermando con l'email e la password precedente.

*As a user I want to have a message board  
so that I can read system's messages.*

Nel momento in cui l'utente ha una prenotazione attiva presso una data struttura, egli potrà vedere i messaggi informativi che il manager ha inviato ai suoi clienti.

*As a user I want to reserve a seat in one of the three time slots  
so that I can go to the gym or pool if there are enough free seats.*

L'utente che accede alla pagina di un centro potrà effettuare una prenotazione, qualora ci siano dei posti disponibili per una delle fasce temporali.

*As a user I want to remove my reservation so that I can let a seat free in the gym/pool.*

L'utente può incontrare qualsiasi imprevisto che gli impedisca di recarsi in struttura ad allenarsi, quindi è indispensabile fornire la possibilità di annullare una prenotazione non ancora confermata.

*As a user I want to have a page so that I can get info about gyms and pools  
within a specific position.*

Prima di prenotare, l'utente deve avere la possibilità di visualizzare tutte le informazioni sulla struttura che gli occorrono, come ad esempio la sua localizzazione, il nome ed il totale dei posti di cui dispone.

*As a user I want to have a page so that I can search in the map registered gyms and pools.*

Avere un'integrazione diretta con il mondo reale è fondamentale per l'utente che vuole confrontare le diverse alternative sulla base delle caratteristiche geografiche, per questo è stata introdotta la possibilità di cercare un indirizzo e poter esplorare tutti i centri che si trovano nelle sue vicinanze.

*As I user I want to set my home position so that I can view gyms and pools near my home.*

Nella pagina di gestione dell'utente è possibile impostare una posizione predefinita in modo da visualizzare le palestre vicine alla sua zona, senza dover effettuare una ricerca manuale.

*As user I want to have a page so that I can view my reservations.*

In un'apposita pagina è possibile visualizzare tutte le prenotazioni che non sono ancora scadute e quindi procedere con un'eventuale conferma di presenza.

*As user I want to have a button so that I can create a review and rating a specific structure.*

Altrettanto fondamentale è la possibilità di condividere la propria esperienza con altri utenti, per questo nella pagina principale della struttura è possibile scrivere una nuova recensione con la relativa valutazione e questa sarà visibile a tutti gli utenti che cercano la struttura in questione.

*As user I want to have a page so that I can view the reviews of a structure.*

Non appena viene pubblicata una recensione di una particolare struttura, questa sarà visualizzabile sulla sua pagina.

*As user I want to view all my reviews so that I can modify or delete some of them.*

Nel momento in cui un utente pubblica una recensione, questa può essere modificata/cancellata unicamente dal suo autore o, nel caso non rispetti la politica del sito, dall'amministratore generale.

*As user I want to insert a token so that I can confirm my participation.*

Nella pagina delle prenotazioni posso inserire il token per confermare la mia presenza in struttura, questo sarà mostrato unicamente all'interno della struttura fisica e viene generato casualmente ogni giorno.

Proseguiamo con il *manager*, è comunque un utente registrato che vuole condividere le sue strutture.

*As a manager I want to create and set data about my gyms and pools so that*

*users can get information about them.*

Ovviamente il manager vuole inserire la sua struttura nel sistema includendo le informazioni principali come il numero di posti, la tipologia del centro e il suo nome.

*As a manager I want to destroy my gyms or pools so that users can't get information about them.*

In qualunque momento è possibile rimuovere la struttura in modo che non sia più accessibile agli utenti.

*As a manager I want to edit my gym's or pools so that users can get update information about them.*

Nel momento in cui alcune informazioni sulla struttura cambiano, il manager può aggiornarle mantenendo la consistenza delle strutture dati del database.

*As a manager I want to set my gym's and/or pool's position so that users can reserve a seat.*

Il proprietario, nel momento in cui registra il suo centro, deve impostare la sua posizione, in questo modo sarà possibile per gli utenti fare una ricerca e trovarlo all'interno della mappa per poi eventualmente prenotare un posto.

*As a manager I want to send message to other users so that I can advertise them if something is wrong.*

Il manager potrebbe avere bisogno di inviare un messaggio urgente ai suoi clienti prenotati, ad esempio per avvisarli di un possibile problema sanitario, quindi attraverso un' apposita pagina sarà possibile notificare ai clienti della struttura tutti gli aggiornamenti opportuni.

*As a manager I want to drop a message from the message-box of my structure so that I can advertise users if the message was wrong.*

Così come è possibile inviare un messaggio, così deve anche essere possibile cancellarlo nel momento in cui il manager ha commesso qualche tipo di errore.

Concludiamo con il ruolo di *admin*, esso si occupa della gestione interna di tutto il sistema e ha il compito di risolvere le inconsistenze ed eventuali errori che si possono creare, oltre che assicurarsi che tutti i membri rispettino le regole di comportamento del sito.

*As an admin I want to have special settings so that I can delete users and managers.*

L'amministratore può dunque eliminare user e manager per le ragioni elencate

in precedenza.

*As an admin I want to have special privileges so that I can see all reservations of all users.*

Per poter controllare il regolare utilizzo del sistema l'admin può ispezionare le prenotazioni e il loro stato.

*As an admin I want to have special privileges so that I can see all gyms/pools of all managers.*

*As an admin I want to have special settings so that I can delete a gym/pool if it would have closed.*

Proprio come con gli utenti, l'amministratore può ispezionare e nel caso cancellare le strutture a seguito di una valida motivazione.

*As an admin I want to have a special setting so that I can delete a review of a structure if it doesn't respect other users.*

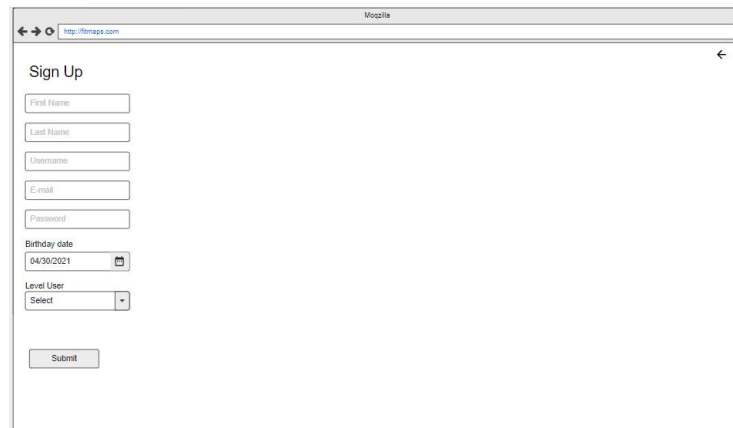
*As an admin I want to have special privileges so that I can drop messages from message-box if it doesn't respect other users.*

Recensioni e messaggi scambiati devono rispettare le regole della piattaforma, se questo viene meno allora l'admin può operare di conseguenza cancellando o modificando tutte le interazioni di manager e client con il sistema.

## 2.2 Illustrazione dei *mock-up* di interfaccia

In questa sezione mostrerò i *mock-up* delle varie pagine e interfacce che compongono la nostra applicazione, per poi presentarne le caratteristiche principali. Logicamente queste interfacce sono delle illustrazioni semplificate e quindi ben lontane dal risultato finale del progetto, ma sono un ottimo mezzo per mostrare a grandi linee il *front end* dell'applicazione.

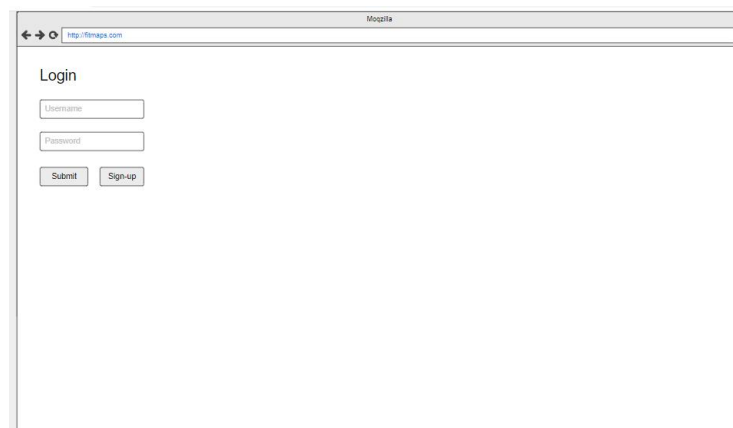
Iniziamo con la pagina relativa alla registrazione (**figura 2.1**), completando opportunamente tutti i campi con le informazioni personali, scegliendo la tipologia di utente e infine cliccando sul bottone di *submit*, verrà effettuato un reindirizzamento sulla pagina di login.



The screenshot shows a web browser window with the address bar displaying "http://ftrmaps.com". The page title is "Sign Up". The form contains the following fields: "First Name", "Last Name", "Username", "E-mail", "Password", "Birthday date" (with a calendar icon), and "Level User" (a dropdown menu currently showing "Select"). A "Submit" button is located at the bottom of the form.

**Figura 2.1.** Sign-up Page.

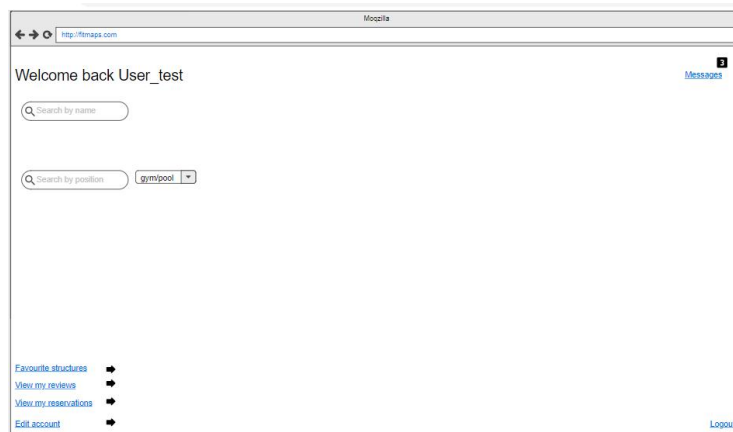
Nella pagina di login (**figura 2.2**) inserendo le credenziali corrette, create in fase di registrazione, si verrà reindirizzati verso l'homepage relativa alla propria tipologia di account.



The screenshot shows a web browser window with the address bar displaying "http://ftrmaps.com". The page title is "Login". The form contains the following fields: "Username" and "Password". Below these fields are two buttons: "Submit" and "Sign-up".

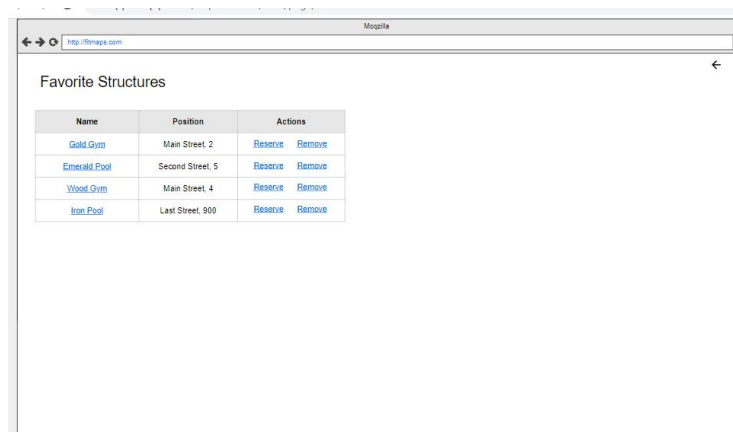
**Figura 2.2.** Log-in Page.

L'utente che ha appena effettuato l'accesso visualizza la pagina della **figura 2.3**, all'interno di questa può cercare la struttura per nome oppure digitandone l'indirizzo; ha la possibilità di visualizzare le sue strutture preferite cliccando su *favourite structures*, leggere le sue recensioni andando su *view my reviews*, controllare le prenotazioni recandosi su *view my reservations* ed infine richiedere la pagina di modifica dell'account cliccando sopra *edit account*.



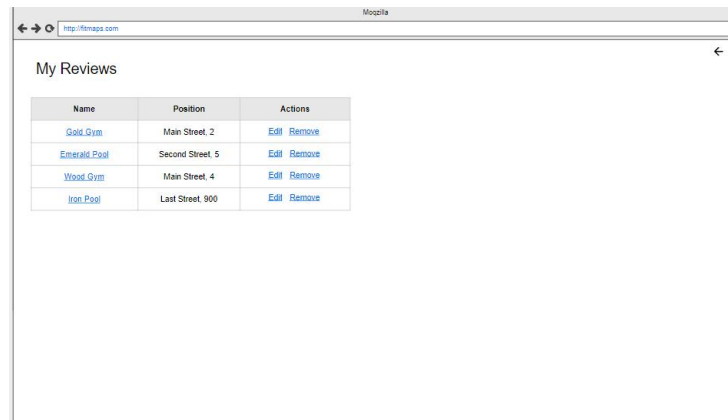
**Figura 2.3.** User's Homepage.

Nella pagina delle strutture preferite (**figura 2.4**) possiamo trovare l'elenco dei centri che sono stati inseriti nel sistema, con la possibilità di effettuare una prenotazione mediante il pulsante *reserve*, oppure rimuovere una specifica struttura dall'elenco cliccando su *remove*.



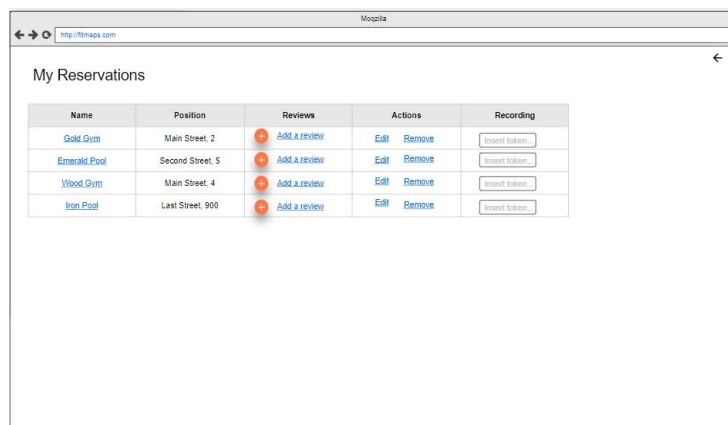
**Figura 2.4.** Favourite structures Page.

Partendo sempre dall'homepage dell'utente (**figura 2.3**), l'utente può visualizzare tutte le recensioni da lui pubblicate nella pagina della **figura 2.5** e, attraverso gli appositi pulsanti *edit* e *remove*, è in grado di modificare o rimuovere facilmente le recensioni che desidera.



**Figura 2.5.** User's reviews Page.

Analogamente a quanto detto precedentemente, l'utente che si trova sulla sua homepage (**figura 2.3**) ha la possibilità di visualizzare la pagina delle sue prenotazioni (**figura 2.6**), qui può modificarle o rimuoverle (attraverso gli specifici tasti *edit* e *remove*) oppure confermarle inserendo il *token* corretto.



**Figura 2.6.** User's reservations Page.

Infine l'utente ha la possibilità di modificare il suo account accedendo alla pagina di modifica dello stesso (**figura 2.7**) a partire dalla sua homepage e cliccando su *edit account*. Qui si possono aggiornare i campi principali (email, password, username e posizione) utilizzando gli appositi *fields* e bottoni; è presente anche la possibilità di cancellare il proprio account dal database.



Old E-mail Old Password Old username  
New E-mail New Password New username  
Change Change Change  
Home Position  
Set Position Delete Account

**Figura 2.7.** Edit account Page.

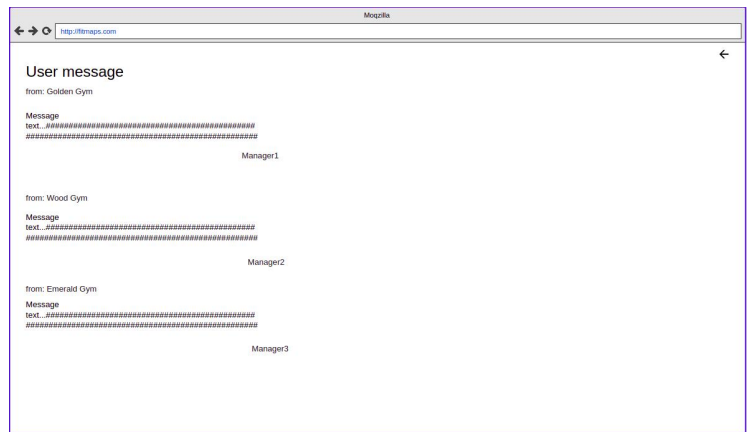
Il requisito principale è sicuramente quello della ricerca delle strutture che l'utente può fare a partire dalla sua homepage (**figura 2.3**) e riempiendo uno dei due campi di ricerca (nome o posizione) con i dati opportuni. Una volta fatto ciò verrà reindirizzato sulla pagina contenente i risultati (**figura 2.8**) dove potrà facilmente aggiungere ai preferiti il centro, accedere alla pagina di prenotazione oppure visualizzare la schermata contenente le informazioni della struttura.

Search by name  
Search by position gym/pool  
Find 4 results...

Name	Position	Actions
Gold Gym	Main Street, 2	Reserve Add to favorites
Emerald Pool	Second Street, 5	Reserve Add to favorites
Wood Gym	Main Street, 4	Reserve Add to favorites
Iron Pool	Last Street, 900	Reserve Add to favorites

**Figura 2.8.** Search Page.

Ci sarà inoltre, nella home dell'utente (**figura 2.3**) il bottone che permette allo *user* di visualizzare i messaggi di notifica; la pagina dei messaggi (**figura 2.9**) contiene tutti i messaggi inviati dai manager delle varie strutture con il testo del messaggio e il mittente.



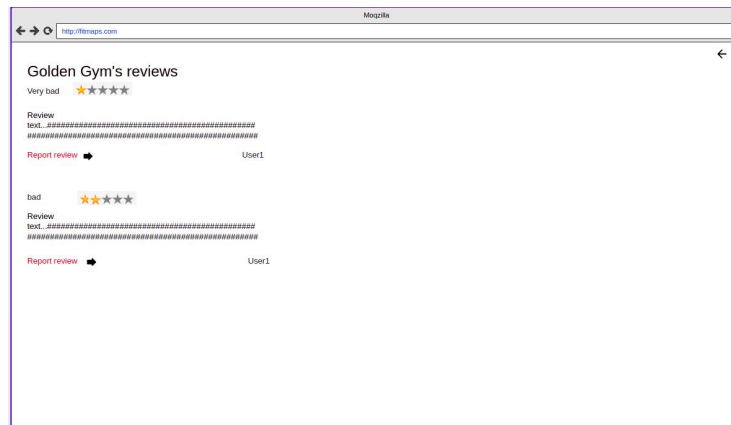
**Figura 2.9.** User messages Page.

Quando lo *user* accede alla pagina di informazioni della struttura (**figura 2.10**) visualizzerà oltre al nome, la posizione, una breve descrizione ed infine il numero di posti totali; potrà inoltre accedere alle recensioni (*view reviews*) o procedere con una prenotazione (*reserve a seat*).

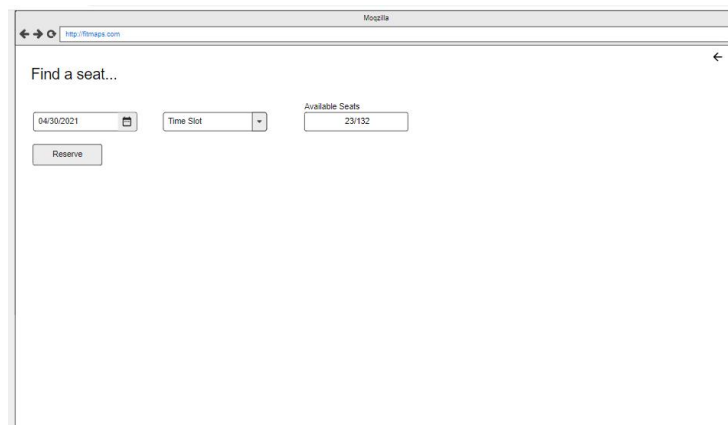


**Figura 2.10.** Structure details Page.

Nella pagina che mostra le recensioni (**figura 2.11**), oltre a leggere il testo delle stesse e visualizzarne il *rating*, è possibile anche effettuare un report nel caso la *review* violi le norme di comportamento del sito.

**Figura 2.11.** Structure reviews Page.

Accedendo alla pagina di prenotazione (**figura 2.12**) l'utente potrà effettuare una prenotazione selezionando la data e la fascia oraria desiderata, qui vedrà anche il numero di posti disponibili in tempo reale; la prenotazione sarà ultimata cliccando su *reserve*.

**Figura 2.12.** Reservation Page.

Nel momento in cui l'utente voglia scrivere una nuova recensione o modificarne una già pubblicata, lo potrà fare attraverso il *form* in **figura 2.13**, aggiornando *review* il titolo, la data della visita, il rating che ritiene più appropriato e completando il tutto con una breve descrizione dell'esperienza che ha avuto; le modifiche verranno apportate una volta cliccato su *save*.

Add or edit review to Golden Gym

Insert rating...

★★★★★

Insert title

05/05/2021

Insert a short review...

Delete Save

**Figura 2.13.** Create or modify review Page.

Se ad effettuare il login è invece il *manager*, visualizzerà la specifica homepage (figura 2.14); in questa pagina il manager può aggiungere una nuova struttura con il pulsante *add a structure*, modificare o rimuovere una struttura già esistente con *edit* o *remove*, mandare un nuovo messaggio (*send a message*) o visualizzare quelli già esistenti (*view messages*). Infine ha la possibilità di consultare la lista di tutti gli utenti (*user list*) che sono prenotati ad una delle sue strutture.

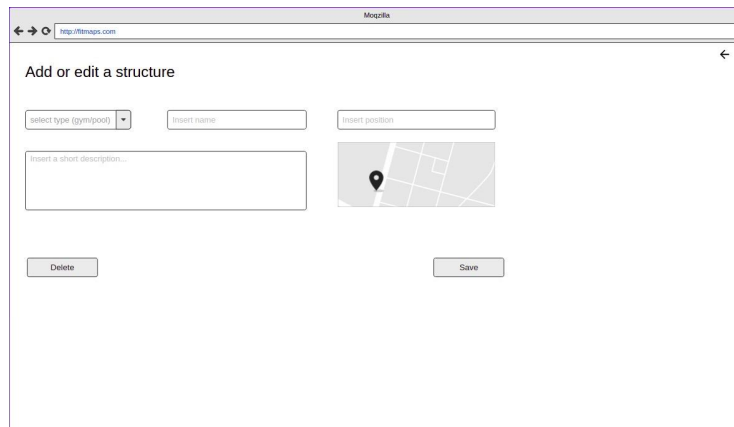
Welcome back, Manager\_test

Name	Position	Actions
Gold Gym	Main Street, 2	<a href="#">Edit</a> <a href="#">Send Message</a> <a href="#">Remove</a> <a href="#">User List</a> <a href="#">Statistics</a> <a href="#">View Messages</a>
Emerald Pool	Second Street, 5	<a href="#">Edit</a> <a href="#">Send Message</a> <a href="#">Remove</a> <a href="#">User List</a> <a href="#">Statistics</a> <a href="#">View Messages</a>
Wood Gym	Main Street, 4	<a href="#">Edit</a> <a href="#">Send Message</a> <a href="#">Remove</a> <a href="#">User List</a> <a href="#">Statistics</a> <a href="#">View Messages</a>
Iron Pool	Last Street, 900	<a href="#">Edit</a> <a href="#">Send Message</a> <a href="#">Remove</a> <a href="#">User List</a> <a href="#">Statistics</a> <a href="#">View Messages</a>

[+ Add a structure](#)

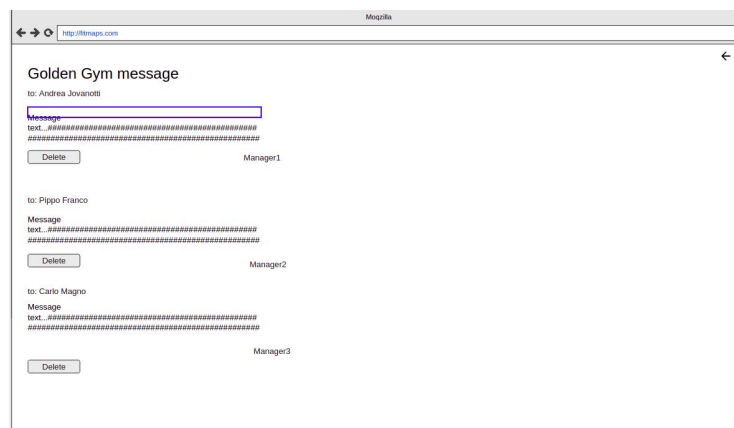
**Figura 2.14.** Manager Homepage.

Nella figura 2.15 possiamo visualizzare la pagina di creazione di una nuova struttura del manager dove può selezionare la tipologia di centro, inserire nome e posizione oltre ad una breve descrizione e concludere il tutto cliccando su *save*.



**Figura 2.15.** Edit or create structure Page.

Per visualizzare i messaggi inviati agli utenti è sufficiente, partendo dalla home del manager (**figura 2.14**) e cliccare su *view messages*. Ora si aprirà la pagina di visualizzazione dei messaggi (**figura 2.16**) dove per ognuno è indicato destinatario e mittente oltre al testo del messaggio. Inoltre è presente il tasto per la cancellazione del messaggio (*delete*).



**Figura 2.16.** Manager messages Page.

Un ulteriore requisito dell'applicazione è quello di permettere al manager di poter inviare messaggi agli utenti prenotati alle proprie strutture. Nella home (**figura 2.14**) possiamo trovare il pulsante *send a message* che ci reindirizza verso la pagina di creazione del messaggio (**figura 2.17**) dove si può selezionare la data, il *time slot* e infine ovviamente il corpo stesso.

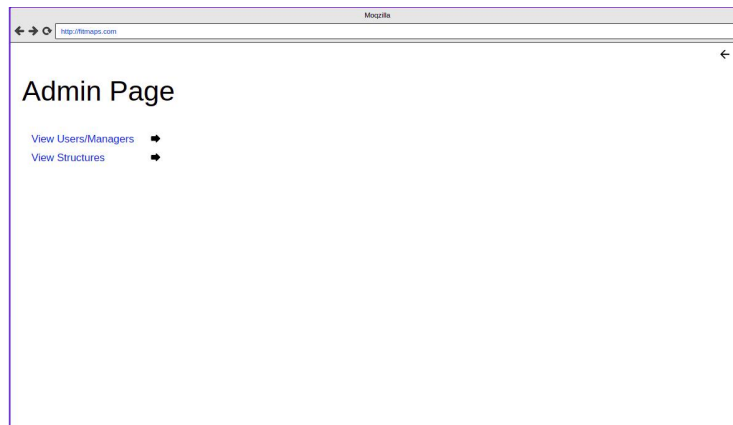
**Figura 2.17.** Create message Page.

Il manager ha la possibilità, a partire dalla sua home (**figura 2.14**), di sfogliare l'elenco di tutti gli utenti che hanno una prenotazione in una delle sue strutture. Grazie alla pagina della *users list* (**figura 2.18**) il manager può vedere quali utenti hanno confermato la prenotazione oppure ha la possibilità di mandare un messaggio diretto al singolo utente.

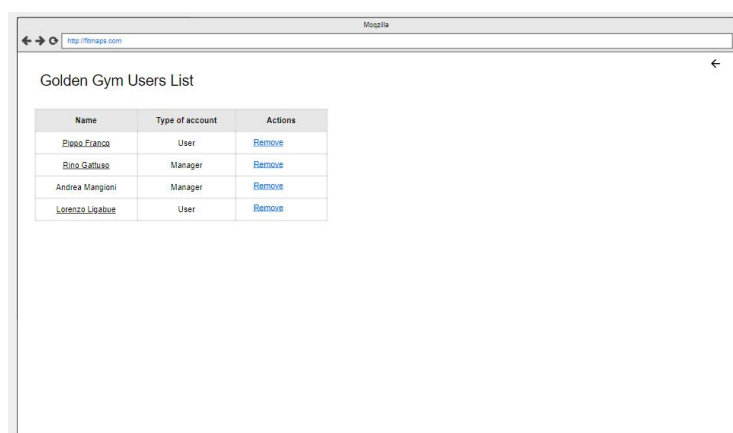
Name	Time slot	Actions	Recorded
Edoardo Franco	morning	<a href="#">send message</a>	yes
Rino Gattuso	afternoon	<a href="#">send message</a>	no
Andrea Mangioni	evening	<a href="#">send message</a>	no
Lorenzo Ligabue	morning	<a href="#">send message</a>	yes

**Figura 2.18.** Manager user list Page.

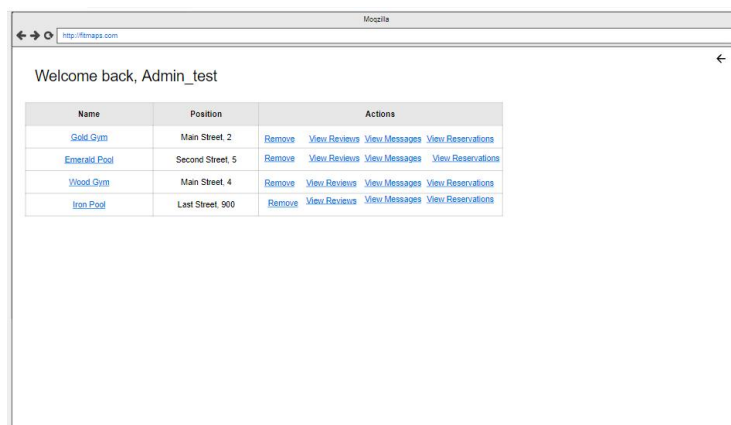
Per quanto riguarda l'*admin*, abbiamo la sua console di gestione in **figura 2.19** dove può visualizzare sia utenti e manager ma anche le strutture presenti nel sistema.

**Figura 2.19.** Admin console Page.

Quando l'amministratore visualizza gli utenti e manager (**figura 2.20**) può decidere, qualora non rispettino il regolamento dell'applicazione, di rimuoverli dal sistema.

**Figura 2.20.** Admin users list Page.

Dalla console l'*admin*, cliccando sull'apposito bottone, può visualizzare tutte le strutture con le informazioni consuete (**figura 2.21**) oltre ad avere anche l'opzione di rimozione (*remove*).



Name	Position	Actions
<a href="#">Gold Gym</a>	Main Street, 2	<a href="#">Remove</a> <a href="#">View Reviews</a> <a href="#">View Messages</a> <a href="#">View Reservations</a>
<a href="#">Emerald Pool</a>	Second Street, 5	<a href="#">Remove</a> <a href="#">View Reviews</a> <a href="#">View Messages</a> <a href="#">View Reservations</a>
<a href="#">Wood Gym</a>	Main Street, 4	<a href="#">Remove</a> <a href="#">View Reviews</a> <a href="#">View Messages</a> <a href="#">View Reservations</a>
<a href="#">Iron Pool</a>	Last Street, 900	<a href="#">Remove</a> <a href="#">View Reviews</a> <a href="#">View Messages</a> <a href="#">View Reservations</a>

**Figura 2.21.** Admin structures list Page.

## 2.3 Autenticazione con servizi esterni mediante *Oauth*

La nostra applicazione oltre ad offrire un sistema di autenticazione interna, gestita interamente all'interno del framework, dispone anche della possibilità di procedere con un' autenticazione esterna mediante *Oauth*.

Questo è un protocollo di delegazione di rete aperto e standard, utilizzato dagli utenti per garantire l'accesso ai loro dati senza condividere con terzi alcuna password. Si può dire che l'*Oauth* fornisce agli utenti un *secure delegated access*[10] alle risorse per conto del proprietario di queste ultime. È stato ideato per funzionare assieme ad *HTTP*, sostanzialmente il servizio concede un *token* al cliente di terze parti; poi quest'ultimo utilizzerà il codice per accedere alle informazioni protette.

Proprio come anticipato all'inizio del paragrafo, l'*Oauth 2.0* non è un protocollo di autenticazione, mentre lo è *OpenID*. Per aiutarci a capire la differenza fra i due ho trovato un commento molto sagace di un utente di *Stack Overflow*: "*OpenID is for humans logging into machines, Oauth is for machines logging into machines on behalf of humans*"[5]. Nella pratica *OpenID* è molto difficile da utilizzare, soprattutto dal punto di vista dello sviluppatore, proprio per questo è stato rilasciato *OpenID Connect*; questo protocollo è una reinvenzione di quello precedente come un *layer* per *Oauth*, rendendo di fatto le due tecnologie complementari in molte implementazioni.[8] Nello specifico noi abbiamo utilizzato questa tecnologia scegliendo come service provider *Google*, per potere usufruire di questo servizio è necessario includere la gemma per le API (*omniauth-google-oauth2* e *omniauth*), oltre a quella già esistente per l'autenticazione interna (*Devise*).

```

31 # gem for authentications
32 gem 'devise'
33 gem 'omniauth', '~>1.9.1'
34 gem 'omniauth-google-oauth2'

```

**Figura 2.22.** Gem for authentications.



Una volta fatto questo è stato necessario creare un' applicazione nell'ambiente di *Google* in modo da poter generare il *client id* e il *client secret*, questi sono conosciuti sia dal provider che dalla nostra applicazione. Per continuare era necessario integrare questo sistema di accesso all'interno di *Devise* inserendo le informazioni all'interno del file di configurazione di quest'ultimo (**figura 2.23**) in modo che anche la funzionalità di *Oauth* fosse controllata dallo stesso.

```
312 #Omniauth google
313 config.omniauth :google_oauth2,
314 end
```

**Figura 2.23.** Configurazione devise, id e secret censurati.

Ora era arrivato il momento di configurare le routes dell'applicazione aggiungendo quella per gestire la richiesta di autenticazione; questo è stato fatto aggiungendo la riga della **figura 2.24** all'interno del file */routes.rb*.

```
6 devise_for :users, controllers: { omniauth_callbacks: 'users/omniauth_callbacks' }
```

**Figura 2.24.** routes per *Oauth*.

Per integrare la funzionalità nel nostro modello dello *user* era necessario aggiornare il file *app/model/user.rb*.

```
8 devise :database_authenticatable, :registerable,
9       :recoverable, :rememberable, :validatable, :omniauthable, omniauth_providers: [:google_oauth2]
```

**Figura 2.25.** Integrazione dell'*Oauth* nel modello dello user.

Per poter gestire la *callback* di *Google* è stato creato il controller specifico che lo faccia (*app/controllers/users/omniauth\_callbacks\_controller.rb*); questa porzione di codice (**figura 2.26**) crea un nuovo utente a partire dalla risposta del provider, per poi effettuare il reindirizzamento. Per fare ciò si serve di un metodo ausiliario della classe *user* (*from\_omniauth()*) implementato in **figura 2.27**, che a partire dal token generato da *Google*, prima controlla se l'utente è presente nel database attraverso una *query* e se questo non è mai stato creato allora lo fa lui.

```
31 def google_oauth2
32   @user = User.from_omniauth(request.env['omniauth.auth'])
33   flash[:notice] = I18n.t 'devise.omniauth_callbacks.success', kind: 'Google'
34   sign_in_and_redirect @user, event: :authentication
35 end
36 end
```

**Figura 2.26.** Metodo nel controller di *callback*.

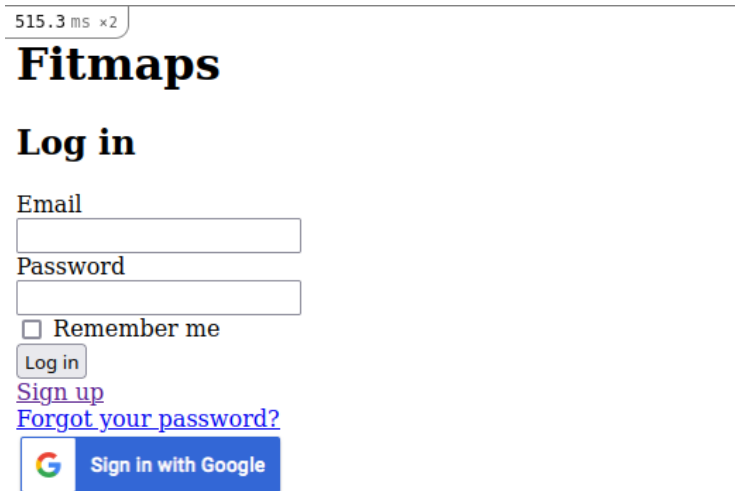
```
8 devise :database_authenticatable, :registerable,
9       :recoverable, :rememberable, :validatable, :omniauthable, omniauth_providers: [:google_oauth2]
```

**Figura 2.27.** Integrazione dell'*Oauth* nel modello dello user.

Infine per poter fornire all'utente un link su cui cliccare è necessario aggiungerlo nell'apposita view (*app/views/users/session/new.html.erb*); inoltre nel nostro progetto ho inserito l'icona caratteristica dell'autenticazione mediante *Google* per rendere il tutto più gradevole (**figura 2.28**). Possiamo osservare il risultato finale in **figura 2.29**.

```
21 <%= if devise_mapping.omniauthable? %>
22 <%= link_to image_tag("google/google.png" , style: 'height:10px;width:10px;'),user_google_oauth2_omniauth_authorize_path %>
23 <% end %>
```

**Figura 2.28.** Codice link per view di autenticazione.



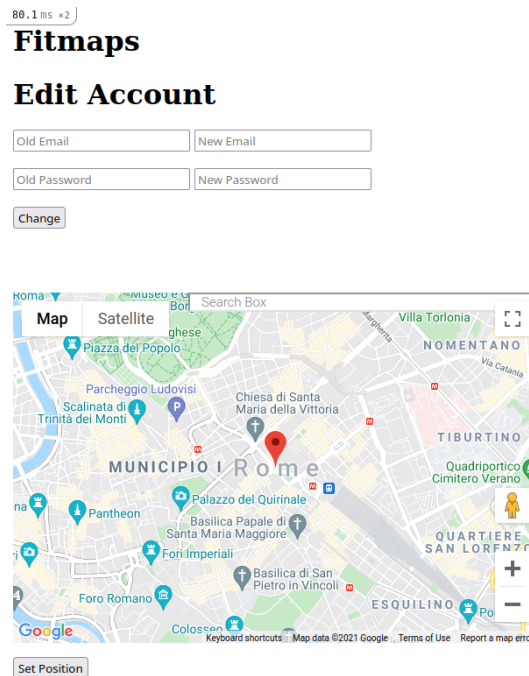
**Figura 2.29.** Esempio autenticazione con *Oauth* di *Google*

## 2.4 *Google Maps API, Place Autocomplete*

Nel corso dello sviluppo dell'applicazione ci è sembrato doveroso integrare le API di *Google Maps*, in particolare la ricerca con autocompletamento. Il *Place Autocomplete* è un servizio che restituisce una predizione in funzione di una richiesta *HTTP*, che specifica una stringa testuale.[2] In particolare ci è stato molto utile per migliorare la fruibilità della ricerca da parte degli utenti (grazie a questa funzionalità infatti è stato possibile visualizzare solo gli elementi pertinenti all'area interessata senza scorrere delle liste infinite e poco pratiche). Un altro valore aggiunto è stata la possibilità di memorizzare la posizione di default dello user in modo che possa vedere facilmente tutte le strutture nei pressi della posizione impostata.

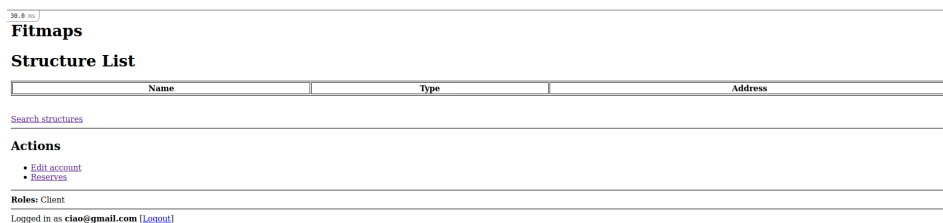
Per integrare queste interfacce per prima cosa abbiamo dovuto importare le librerie predefinite direttamente dalla documentazione di *Google Maps*[3]. Il modello di default (*/app/model/place.rb*), essendo quello predefinito della libreria nativa, non consentiva una grande possibilità di personalizzazione e proprio per questo abbiamo deciso di legare l'entità di default con una nuova creata da noi (*/app/model/structure.rb*) attraverso una relazione con cardinalità 1:1. I metodi principali sono stati inseriti

all'interno del controller (`/app/controllers/places_controller.rb`), mentre il codice *JavaScript* è stato collocato nelle apposite view; in **figura 2.30** è mostrata il *rendering* della view per la modifica dell'account compresa anche la posizione di default.



**Figura 2.30.** Esempio di edit dell'account.

Il suo utilizzo per la ricerca dei centri limitrofi è veramente semplice: inizialmente ci troviamo nell'homepage dell'utente (**figura 2.31**) e non vediamo alcuno risultato, in quanto non è stata effettuata alcuna ricerca e attorno alla posizione di default impostata non ci sono strutture.



**Figura 2.31.** User Homepage.

In seguito recandoci su *Search structures* ci si aprirà la pagina relativa alla ricerca come in **figura 2.32**, riempiendo il campo di ricerca con il luogo desiderato (nel nostro caso sarà Roma), e grazie all'esecuzione del codice *JavaScript* presente all'interno della view, verranno mostrati i suggerimenti dell'autocompletamento (**figura 2.33**).

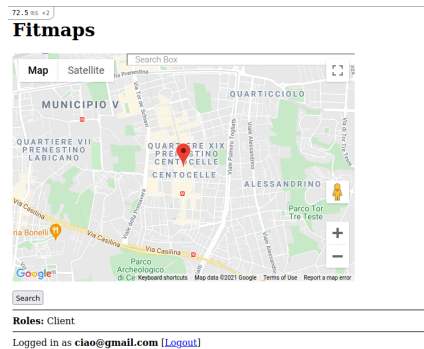


Figura 2.32. Search Page.

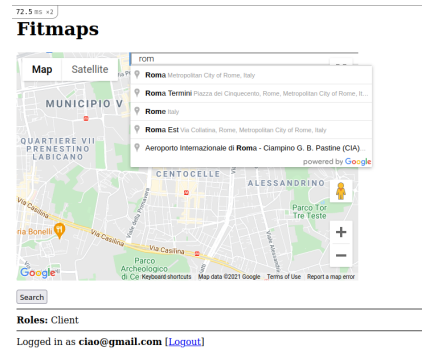


Figura 2.33. Esempio di Autocomplete.

Ora clicchiamo su quello più opportuno e, dopo aver premuto *enter*, verremo reindirizzati alla pagina che conterrà i risultati della nostra ricerca, visibile in **figura 2.34**.

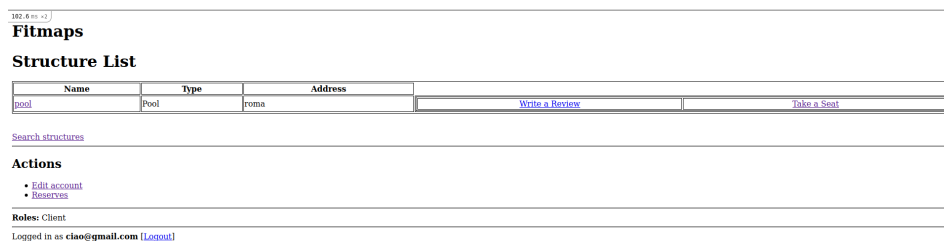


Figura 2.34. Structures List.

## 2.5 Metodologia utilizzata per la conduzione e divisone del lavoro

Sul progetto di *Fitmaps* abbiamo lavorato io ed un mio collega, spinti da un'idea comune che ci ha portato a scegliere questo progetto tra i diversi proposti; per la creazione dell'applicazione abbiamo seguito il libro del corso [1] e le slides dai professori dello stesso [7]. Questo era per entrambi il primo progetto esteso e impegnativo che andasse al di là di qualche riga di codice; inizialmente abbiamo avuto qualche difficoltà nell'organizzare il lavoro equamente in quanto lezioni e impegni differenti non ci consentivano una regolarità nello svolgimento dei compiti individuali. Abbiamo cercato, almeno nella fase iniziale di organizzazione, degli orari comuni per consentire un lavoro in parallelo; quello individuale veniva affiancato da una chiamata video con *Zoom* che ci consentiva di aiutarci reciprocamente. Una volta superati i problemi iniziali circa l'organizzazione e la struttura principale del progetto, abbiamo cominciato a lavorare separatamente concentrandoci ognuno su singoli aspetti; naturalmente nel momento in cui uno dei due avesse avuto bisogno di aiuto o anche di un semplice consiglio, questo avveniva in maniera asincrona. Una ripartizione basata su intere sezioni di codice (quali controller, gemme e modelli), ha permesso di risparmiare una notevole quantità di tempo evitando *bugs* difficili da individuare causati proprio da delle modifiche superficiali del proprio collega: queste

sviste sono determinate da una conoscenza incompleta del codice.

La scelta di utilizzare la specifica *virtual machine*, pubblicata nella cartella di *Google Drive* del corso, è stata motivata dalla sua comodità e facilità di utilizzo. In particolare, avere lo stesso ambiente già configurato, con tutte le versioni coincidenti, ci ha permesso un sereno proseguimento del progetto senza troppe complicazioni. Per quanto riguarda la gestione del codice ci siamo serviti del servizio maggiormente utilizzato per il controllo delle versioni del codice, ovvero *Git*, mentre la cartella è stata condivisa su *GitHub*. Tra i numerosi *IDE* a disposizione io ho optato per *Visual Studio Code* in quanto lo trovo molto intuitivo e funzionale; inoltre, grazie alla possibilità di poter installare tool aggiuntivi, offriva un grande supporto per il completamento del codice. Il nostro *workflow* è stato pensato con una gestione del codice attraverso tre branches, *feature*, *development* e *main*. Il primo veniva aggiornato ad ogni modifica consistente del codice, lo utilizzavamo anche per sincronizzare il lavoro individuale e consentire di revisionare ognuno il codice dell'altro. L'aggiornamento del branch di sviluppo (*development*) veniva effettuato ad ogni nuova funzionalità ultimata; nel momento in cui era disponibile una nuova versione dell'applicazione priva di bugs, questa era pubblicata e inserita nel *main branch*.

Complessivamente sono molto soddisfatto del risultato finale: l'applicazione è stata resa, a seguito di numeri test, molto affidabile e sicura grazie anche ai numerosi controlli introdotti per non consentire agli utenti di effettuare azioni illecite. Mi sarebbe piaciuto curare di più il *front end* con una grafica più accattivante e introdurre nuove funzionalità che potessero rendere il lavoro finale veramente utilizzabile come software professionale. Nel corso dello sviluppo mi sono accorto che il framework di *Ruby on Rails* non mi ha appassionato moltissimo, l'ho trovato poco lineare e personalizzabile. Nel corso dello sviluppo ci siamo imbattuti in un errore molto ostico, causato da un errore del sistema nel nominare una classe, pertanto la sua risoluzione, non essendo abituati a bugs di questo genere, ci ha richiesto una notevole quantità di tempo. Se dovessimo farci un'idea sulla quantità di lavoro prodotto partirei col dire che abbiamo introdotto otto entità per la modellazione e questo ci ha richiesto circa un mese e mezzo di lavoro; ovviamente si tratta solo di una stima, in quanto non è stato possibile dedicare la totalità del nostro tempo a disposizione al progetto. Fare un calcolo della quantità di codice prodotto non è banale, sia per la natura stessa della piattaforma che nasce proprio con l'obiettivo di produrre meno codice possibile e di riutilizzare funzionalità già implementate e anche ampiamente testate; inoltre abbiamo introdotto molte librerie già implementate che non hanno necessitato di grandi modifiche da parte nostra per essere integrate con l'applicazione. Come ribadito più volte la divisione dei compiti è stata molto equa e si è basata su una ripartizione delle user stories, anche se non abbiamo tenuto conto metodicamente del contributo, in termini di tempo dedicato al progetto e codice prodotto, dei vari membri del gruppo.



## Capitolo 3

# Analisi concettuale del sistema

In questo capitolo presenteremo e in seguito analizzeremo lo schema Entità Relazione che modella con precisione la nostra realtà di interesse, allegando naturalmente, oltre allo schema ER, anche il glossario delle entità e delle relazioni.

### 3.1 Schema ER e discussione scelte critiche

Questo schema (**figura 3.1**) è derivato da diverse considerazioni espresse in fase di progettazione, la scelta sicuramente più immediata è stata quella delle entità *Structure*, *Review* e *User*, in quanto fondamentali per la natura stessa dell'applicazione. Per quanto riguarda la modellizzazione dei messaggi (*Message*) ci sono state diverse opinioni e proposte ma la scelta di creare un'entità specifica che alla fine si è dimostrata quella più sensata: il messaggio non poteva infatti essere legato come un attributo dello user oppure della structure, in quanto sarebbe dipeso interamente da questi. Discorso analogo è stato fatto per le *reserve*. Alla fine la decisione è ricaduta su un'entità che fosse relazionata sia con la struttura coinvolta nella prenotazione sia con l'utente che l'aveva creata. La creazione del modello *Place* è stata obbligata dall'integrazione delle API di *Google Maps*, ma fortunatamente abbiamo potuto integrare tale modello con il nostro (*Structure*) mediante una semplice relazione. Sicuramente la scelta più critica è stata quella che riguarda l'entità *Day*; inizialmente non avevamo creato un modello apposito per i giorni ma la gestione delle prenotazioni stava diventando impossibile. Sostanzialmente per poter memorizzare i posti occupati sarebbe stato necessario creare un attributo per ogni giorno e ad ogni mezzanotte traslare tali valori attraverso la chiamata manuale di un'apposita routine. Naturalmente questa scelta non era né la più semplice né tanto meno la più scalabile, e dunque abbiamo optato verso una soluzione decisamente più ragionevole, ovvero la creazione del nuovo modello *Day*. In questo modo l'unica necessità era quella di creare un nuovo oggetto nel momento in cui la prima prenotazione della giornata veniva effettuata e poi memorizzare i posti occupati e quelli confermati per le tre fasce temporali. Grazie a questa accortezza abbiamo potuto gestire praticamente tutte le prenotazioni necessarie per qualunque giorno dell'anno senza modificare gli attributi delle prenotazioni ma semplicemente creando un nuovo oggetto nel momento in cui veniva effettuata la prima prenotazione della giornata.

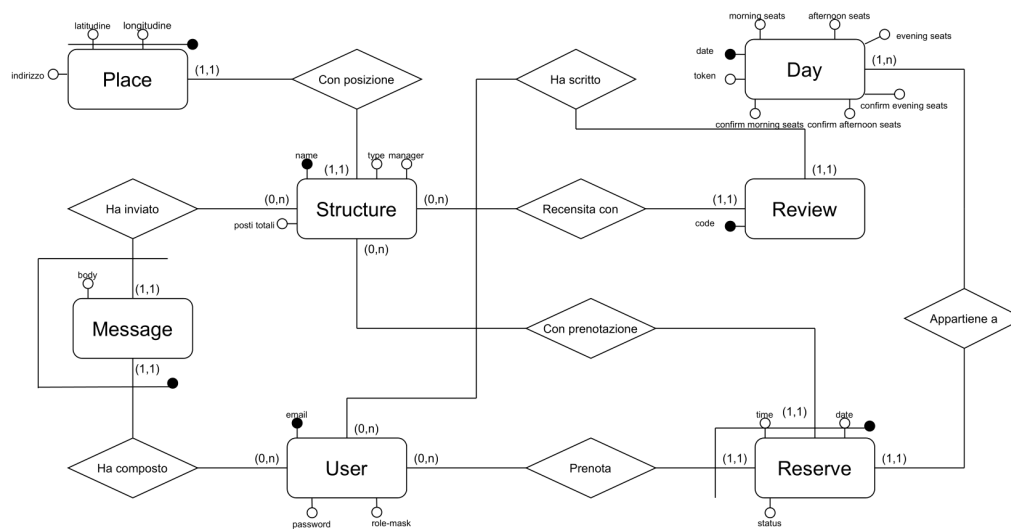


Figura 3.1. Diagramma ER.



## 3.2 Glossario entità e relazioni

Presenterò un semplice glossario per le entità (**tabella 3.1**) e le relazioni (**figura 3.2**) ad integrazioni dello schema ER.

**Tabella 3.1.** Glossario entità.

Entità	Descrizione	Attributi	Identificatori
Structure	Centri sportivi	name, type, manager, posti totali	name
User	Utenti registrati alla piattaforma	email, password, role-mask	email
Reserve	Prenotazioni per le strutture	time, date, status	user(che partecipa a <i>Prenota</i> ), time, date
Message	Messaggi inviati agli utenti	body	user(che partecipa a <i>Ha composto</i> ), structure (che partecipa a <i>Ha inviato</i> )
Place	Luoghi relativi alle strutture	indirizzo, latitudine, longitudine	latitudine, longitudine
Review	Recensioni sulle strutture	code	code
Day	Giorno in cui si svolge una serie di prenotazioni	date, token, morning seats, afternoon seats, evening seats, morning seats confirm, afternoon seats confirm, evening seats confirm	date

Il glossario in **tabella 3.2** non giustifica le cardinalità delle partecipazione delle varie entità alle relazioni. Cominciamo con *Ha inviato* a cui partecipano *Message* e *Structure*. Structure lo fa con cardinalità (0,n) poiché una struttura può inviare un numero arbitrario di messaggi, mentre uno stesso messaggio (con body identico) può essere inviato da una sola struttura. Nella relazione *Ha composto* tra *User* e *Message*, user (in particolare un manager) partecipa con cardinalità (0,n) in quanto

può essere il mittente di molteplici messaggi. In *Con posizione*, *Structure* vi partecipa con cardinalità unitaria poiché una struttura può risiedere in un unico luogo, e al contempo un luogo può ospitare al massimo una struttura (e anche al minimo visto che nel nostro universo di interesse non prendiamo in considerazione i luoghi che non ospitano almeno una struttura). In *Ha scritto* partecipano, *User* con cardinalità arbitraria poiché questo può scrivere o meno quante recensioni voglia; mentre una *Review* può essere scritta unicamente da un utente. Nella relazione *Recensita con*, la struttura può essere recensita molte volte o anche non esserlo per nulla mentre, analogamente al caso precedente, una stessa recensione non può essere attribuita a due centri differenti. Passiamo ora alle relazioni che coinvolgono le *Reserve*, a *Con prenotazione* partecipa *Structure* con cardinalità (0,n) poiché si può prenotare uno spazio in struttura fino a esaurimento dei posti; la prenotazione stessa invece può essere riferita unicamente ad una struttura. La prenotazione riguarda ovviamente oltre ad una struttura anche un utente che *Prenota* un numero arbitrario di posti, ma non più di uno per lo stesso giorno e fascia temporale; ovviamente la stessa prenotazione non può essere riferita a due utenti differenti. Infine l'entità *Day* partecipa ad una sola relazione, ed è quella con *Reserve* (*Appartiene a*), ovviamente nella realtà della nostra applicazione il giorno è di nostro interesse ,solo che è riferito ad almeno una prenotazione e questa può partecipare un' unica volta poiché una stessa istanza di *Reserve* non può riferirsi a due giorni differenti.

Tabella 3.2. Glossario relazioni.

Relazione	Descrizione	Ruoli	Identificatori
Con posizione	Lega la struttura alla sua posizione	Place, Structure	Place
Ha inviato	Denota la relazione che c'è tra il messaggio inviato e la struttura alla quale è riferito	Message, Structure	Structure
Ha composto	Indica quale utente ha composto quale messaggio	User, Message	Message
Ha scritto	Lega la recensione all'utente che l'ha composta	Review, User	Review
Recensita con	Indica la struttura recensita da una specifica review	Structure, Review	Review
Con prenotazione	Denota la struttura oggetto della prenotazione	Structure, Reserve	Reserve
Prenota	Indica l'utente che ha prenotato il posto	User, Reserve	Reserve
Appartiene a	Lega la prenotazione al giorno a cui si riferisce	Reserve, Day	Reserve

### 3.3 Vincoli esterni

Descriveremo tutti i vincoli che non sono esprimibili con lo schema ER, in particolare quelli legati alla gestione interna e fondamentali per la consistenza dell'applicazione.

- L'utente che partecipa alla relazione *Ha composto* deve essere necessariamente un Manager (ovvero avere una role-mask pari a 2) e in particolare, per ogni

istanza di *Ha composto* a cui partecipano il manager  $f$  e il messaggio  $m$ , esiste un'istanza di *Ha inviato* a cui partecipano  $m$  e una *Structure*  $s$ , tale che  $s$  ha come valore dell'attributo *manager* l'email di  $u$ .

- Per ogni istanza di *Structure*  $s$ , la cardinalità delle istanze di *Con prenotazione* a cui partecipa  $s$  non può eccedere il valore dei *posti totali* di  $s$ .
- Per ogni istanza di *Day*  $d$ , la cardinalità delle istanze di *Appartiene a* a cui partecipa  $d$  deve essere la somma del valore dei *morning seats*, *afternoon seats* e *evening seats* di  $d$ .
- Per ogni istanza di *Day*  $d$  il valore dei *confirm morning seats* non può superare quella dei *morning seats*, così come quello dei *confirm afternoon seats* e *confirm evening seats* non può eccedere il valore dei *afternoon seats* e dei *evening seats*.
- Per ogni istanza di *Prenota* a cui partecipano lo *User*  $u$  e la *Reserve*  $s$ ,  $u$  deve essere necessariamente un utente semplice (con una *role-mask* pari a 1).
- Per ogni istanza di *Ha scritto* a cui partecipano lo *User*  $u$  e la *Review*  $r$ ,  $u$  deve essere necessariamente un utente semplice (con una *role-mask* pari a 1).
- Per ogni istanza *Appartiene a* a cui partecipano una *Reserve*  $r$  e un *Day*  $d$  gli attributi *data* di  $r$  e  $d$  devono essere identici.

## Capitolo 4

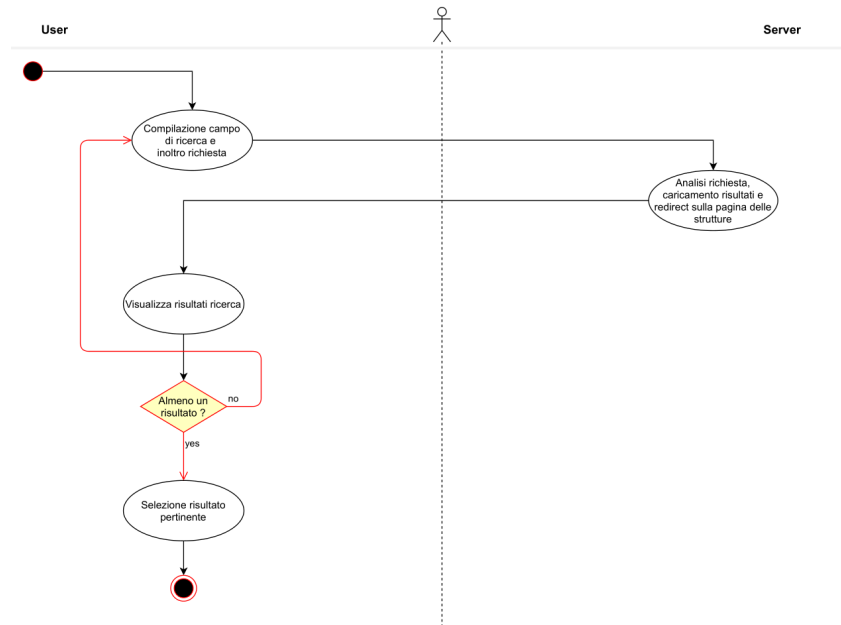
# Progettazione del sistema

In questo capitolo analizzeremo più nello specifico l'architettura dell'applicazione (esterna ed interna), e illustreremo come i vincoli concettuali del sistema vengono realizzati da un punto di vista logico.

### 4.1 Architettura esterna del sistema

Inizieremo con la descrizione della struttura esterna del sistema, arricchendo le descrizioni con diagrammi *UML* delle attività. Si procederà con l'illustrazione delle interazioni dell'utente con l'applicazione, evidenziando le operazioni svolte da *user* e *server*.

In **figura 4.1** ho illustrato lo schema per la ricerca base di una struttura da parte dell'utente. In primo luogo, l'utente inizia compilando l'apposito campo con le informazioni sulla struttura e selezionando il luogo apposito sulla mappa e dopo la risposta del server vengono mostrati i risultati pertinenti. Se c'è almeno un risultato allora il client può selezionare quello più opportuno, altrimenti deve tentare una nuova ricerca.



**Figura 4.1.** Diagramma attività *search structure*.

La prenotazione di un posto è indicata con il flusso in **figura 4.2**: la prima cosa è procedere con la ricerca della struttura (**figura 4.1**) per poi inserire i valori di data e fascia temporale desiderati. Dopo l'analisi del server, se vi è almeno un posto per i parametri indicati, si procede con la richiesta di prenotazione e a seguito la sua conferma.

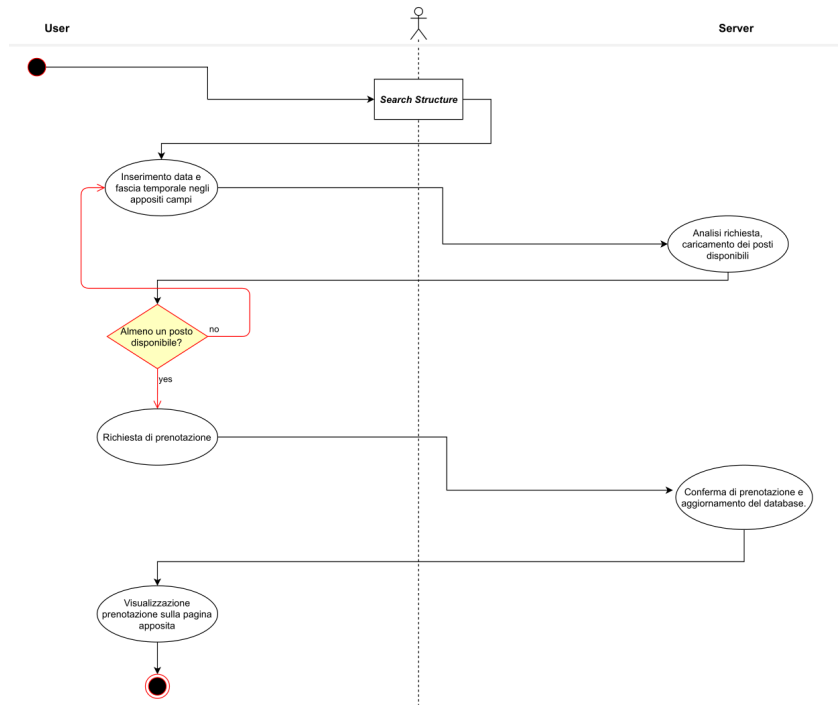


Figura 4.2. Diagramma attività *take a seat*.

Dopo aver visualizzato la pagina di informazioni della struttura è possibile scrivere una nuova recensione e questo viene fatto inoltrando, dopo aver riempito gli appositi campi, la richiesta al server. Se tutto il procedimento si è concluso correttamente, l'utente potrà vedere la propria recensione con le opzioni di modifica ed eliminazione, altrimenti potrà ritentare l'operazione; lo schema risultante è visibile in figura 4.3.

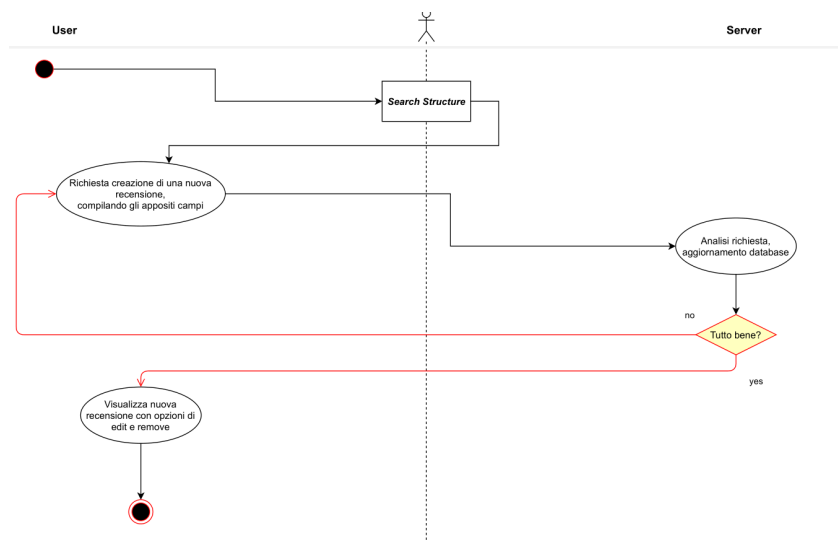
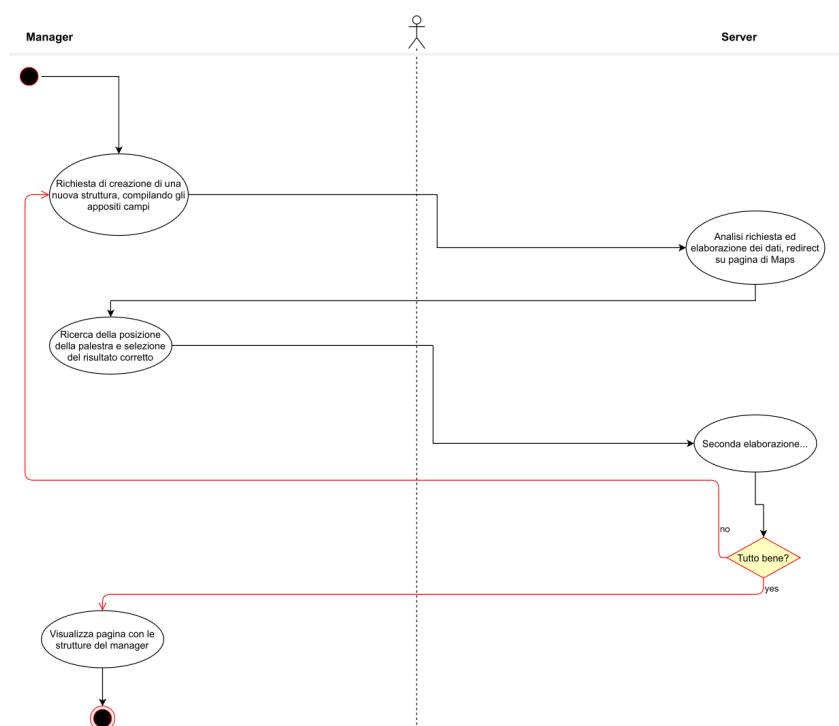


Figura 4.3. Diagramma attività *write a review*.

Proseguiamo con le operazioni che coinvolgono *manager* e *server*. In **figura 4.4** possiamo vedere il flusso di esecuzione della creazione di una nuova struttura. Inizialmente compiliamo gli appositi campi di *type*, *name* e *total seats*, una volta fatto questo il server acquisirà i dati e reindirizzerà il manager alla pagina per l'inserimento della posizione (che avverrà sfruttando le API di *Google Maps*). Se il tutto è andato a buon fine, il manager visualizzerà nella sua homepage le sue vecchie strutture assieme alla nuova appena inserita, altrimenti potrà ritentare l'inserimento.



**Figura 4.4.** Diagramma attività *create a structure*.

Per poter inviare un messaggio ai suoi clienti prenotati il manager dovrà prima recarsi sulla sua homepage, poi caricare la corretta pagina di invio del messaggio, e infine comporre il testo dello stesso compilando l'apposito campo ed infine inviarlo. Nel momento in cui il messaggio è stato inviato correttamente esso sarà visibile cliccando su *view messages*. Il flusso è schematizzato in **figura 4.5**.



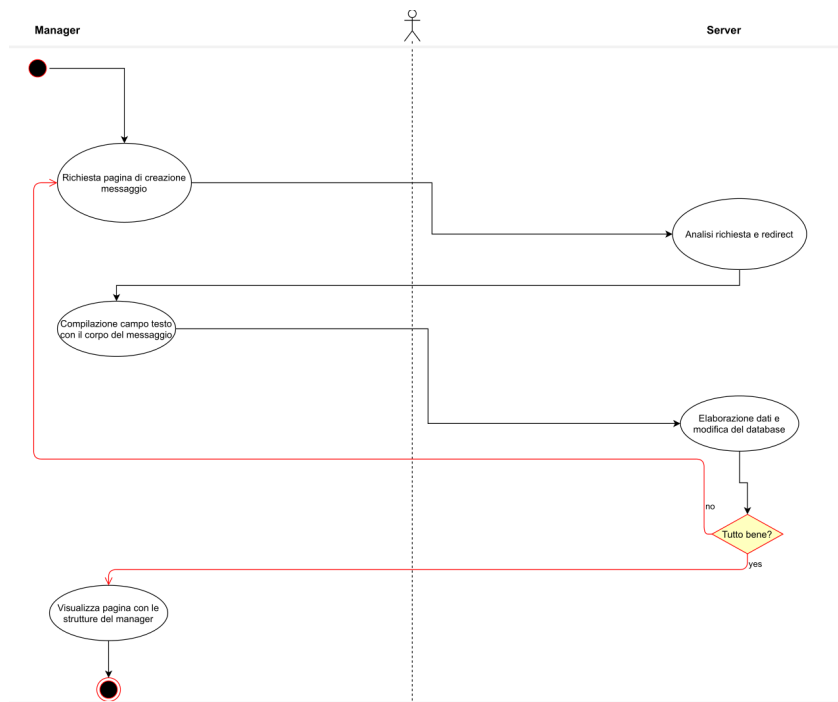
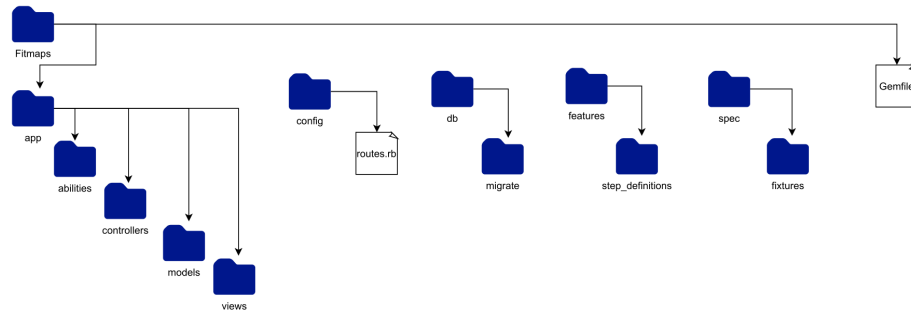


Figura 4.5. Diagramma attività *send a message*.

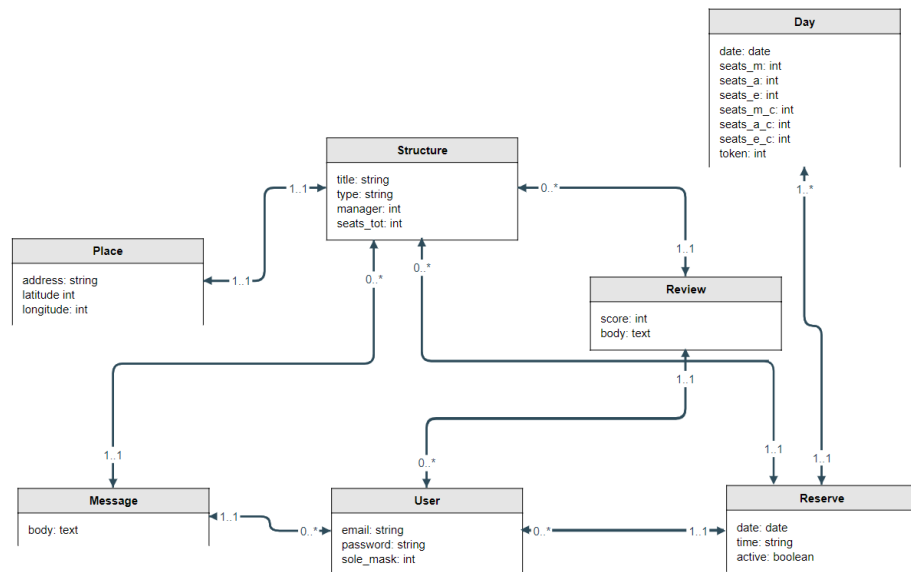
## 4.2 Architettura interna del sistema

In figura 4.6 c'è la rappresentazione del *package diagram* dove sono evidenziati file e directory che sono stati modificati almeno una volta nel corso dello sviluppo dell'applicazione. Partendo dalla root del progetto abbiamo il *Gemfile* che contiene tutte le informazioni sulle gemme da installare per l'applicazione. Nella cartella */app* sono contenuti tutti i: *controllers* (che si occupano dell'elaborazione delle richieste producendo l'output appropriato), *models* (ovvero le "classi di Ruby") e le *views* (quello che l'utente vede, quindi pagina con codice *HTML*, *JavaScript* e *Ruby*); sempre sullo stello livello troviamo la cartella */app/abilities* che contiene i file dove sono indicate le autorizzazioni delle varie tipologie di utenze. In */config* abbiamo il file di *routes.rb*, grazie ad esso il *Rails router* può analizzare gli URL e inoltrarli al controller dell'azione [6]. In */db/migrate* sono contenute le *Rails Migrations* che consentono di definire dei cambiamenti al database [9]. Per quanto riguarda la parte dei test, focalizziamo la nostra attenzione su */features*, in questa cartella sono contenuti i files che definiscono le varie funzionalità testate (mediante test di accettazione e integrazione), e */features/step\_definitions* che contiene le implementazioni dei vari passaggi dei test. In */spec* contiene i test unitari dell'applicazione e nella sotto cartella *fixtures* sono contenute le istanze delle classi di prova del db.

Figura 4.6. *Package diagram.*

### 4.3 Progettazione logica del Data Layer

Iniziamo con il descrivere la sua struttura, in **figura 4.7** possiamo osservare uno schema *UML* che sintetizza il database e le dipendenze tra le classi con le relative cardinalità.

Figura 4.7. Diagramma *UML*.

Possiamo osservare come gli *Users* siano caratterizzati da email e password, entrambi di tipo stringa. Inoltre è presente anche la role-mask (rappresentata con un intero) che identifica la tipologia di utenza. I *Messages* sono identificati unicamente dal corpo del messaggio (*body*) di tipo *text*; le *Reviews* hanno anche esse un body ma in più è presente anche un punteggio (*score*) della recensione. Per quanto riguarda le strutture (*Structures*) esse hanno come attributi: *title* e *type* espressi entrambi con delle stringhe, questi indicano rispettivamente il nome della struttura e la tipologia della stessa (palestra o piscina). Passiamo ora alla tabella di *Place*: questa è una classe di default introdotta per consentire l'integrazione delle API di *Google Maps* ed

è caratterizzata da un indirizzo e le relative coordinate (latitudine e longitudine). Le *Reserves* sono state rappresentate con una data (di tipo *date*), una corrispondente fascia temporale (*time*) di tipo stringa ed infine un booleano (*active*) che rappresenta lo stato della prenotazione. Per modellare al meglio il sistema di prenotazioni è stato fondamentale introdurre la classe *Day*, caratterizzata da una data, un *token* (rappresentato con un intero) per effettuare la conferma delle prenotazioni per quello specifico giorno, ed infine il numero di posti (confermati e non) per le corrispettive tre fasce temporali (*morning*, *afternoon*, *evening*).



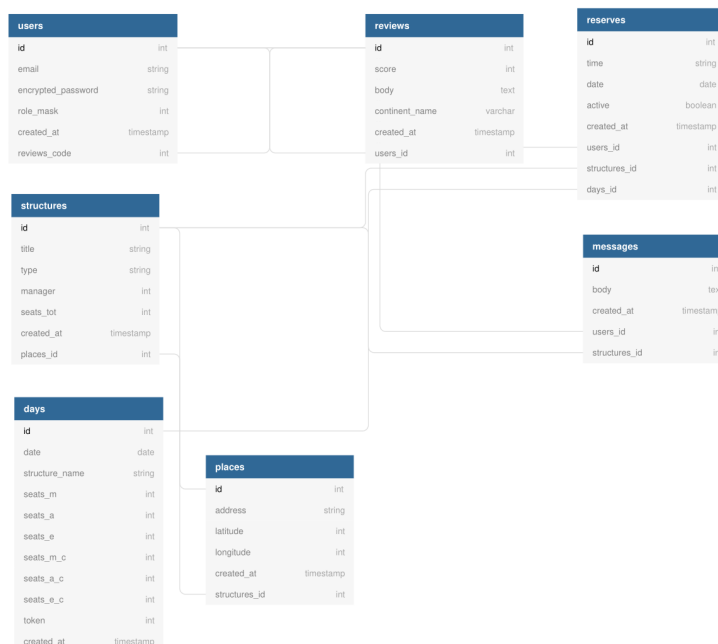
## Capitolo 5

# Realizzazione del sistema

In questo capitolo ci soffermeremo sulla realizzazione del sistema, in particolare differenzieremo i tre layer (*data layer*, *application layer* e *presentation layer*), analizzandone l'implementazione.

### 5.1 Data Layer

In questa sezione descriveremo come i vincoli imposti nello schema ER si mappano concretamente all'interno dell'applicazione, inoltre indicheremo quali sono le situazioni critiche e come esse vengano risolte. In **figura 5.1** possiamo visualizzare la struttura del database implementato sull'applicazione.



**Figura 5.1.** Data Layer.

Il *database*, come il resto dei layer dell'applicazione, è gestito dalla piattaforma mediante le apposite gemme. *Ruby on Rails* offre un'interfaccia per separare la

gestione del database dalla sua implementazione: in questo modo, indipendentemente da quale implementazione si scelga, la modalità di accesso e aggiornamento rimarrà la medesima. In particolare nel nostro caso abbiamo scelto *Sqlite3* per la gestione dei dati. Per quanto riguarda l'aspetto autorizzativo, esso è stato implementato con le gemme di *Canard* e tramite delle apposite sezioni di codice prodotte da noi per l'amministrazione un po' più specifica; ad esempio la gemma non offriva la possibilità di controllare l'accesso alle risorse sulla base del proprietario, quindi questo aspetto lo abbiamo dovuto curare noi. Come già detto, abbiamo tre tipologie di utenze: il *client*, che ha la possibilità di visualizzare le strutture ma non può crearle, modificarle o eliminarle. È in grado di creare recensioni e prenotazioni ma può modificare ed eliminare naturalmente solo quelle di sua proprietà. I messaggi che vengono inviati dai manager delle strutture sono ovviamente solo visualizzabili dagli utenti che hanno una prenotazione attiva per il centro in questione.

La seconda tipologia è il *manager*, che può creare strutture, ma può modificare e/o eliminare solo quelle di sua proprietà. È in grado di visualizzare tutte le recensioni delle sue strutture, ma logicamente non può modificarle o eliminarle; discorso analogo vale per le prenotazioni. Mentre ovviamente può inviare messaggi solo ai suoi iscritti e modificare ed eliminare unicamente i suoi messaggi.

Continuiamo con l'*admin*, questo detiene molte più autorizzazioni rispetto alle utenze precedenti; infatti è in grado di leggere ed eliminare strutture (tutte quelle presenti nel database), utenti, recensioni, prenotazioni e messaggi ma non può creare nessuna istanza di tali entità. Ovviamente questo ruolo verrà assegnato solo ad una persona fidata che si occupi della consistenza e sicurezza del database.

I vincoli dello schema logico vengono realizzati sia sfruttando le gemme che controllano le autorizzazioni, ma anche con degli opportuni moduli software inseriti all'interno dei vari *controller*. Iniziamo con i messages: ogni funzione di *messages\_controller.rb* è protetta dal metodo *authorize*, come possiamo vedere ad esempio in **figura 5.2** ma questo non gestisce le singole autorizzazioni all'interno della stessa tipologia di utente; abbiamo quindi introdotto dei controlli basati su *query sql* che proteggano l'invocazione di tali funzioni da chi non ne detiene i permessi.

```
25  def edit
26    authorize! :edit, Message, message: "You are not authorized!"
27    @structure=Structure.find(params[:structure_id])
28    @message = @structure.messages.find(params[:id])
29    if current_user.roles_mask==2 && @structure.manager != current_user.id
30      redirect_to structures_path
31    end
32  end
```

**Figura 5.2.** funzione edit in *messages\_controller.rb*.

Si noti che questo controllo non si limita a eliminare i link di accesso alle pagine per le utenze non autorizzate; anche se il metodo dovesse venire invocato direttamente dalle *routes* effettuando una richiesta verso l'URL corretto, la funzione non sarebbe comunque eseguita nel caso in cui l'utenza non sia autorizzata. Il controllo in questo caso è fatto nelle righe 29-30: osserviamo che qualora si cerchi di invocare tale metodo senza passare per le vie legittime, si verrà reindirizzati all'homepage della specifica utenza. Questa tipologia di controllo si ritrova in tutti quei metodi dove è

necessario controllare la proprietà della risorsa prima di garantirne l'accesso. Per quanto riguarda il vincolo che impone il limite superiore sulle prenotazioni che possono essere effettuate presso una specifica struttura in una determinata data e fascia temporale, è gestito dal modulo *reserves\_controller.rb*, in particolare nel metodo *create* (figura 5.3) se questo viene invocato nel momento in cui i posti sono esauriti allora si verrà reindirizzati nuovamente sulla pagina delle prenotazioni.

```
7     def create
8       authorize! :create, Reserve, message: "You are not authorized!"
9       @structure = Structure.find(params[:structure_id])
10      @reserve = @structure.reserves.create(reserve_params)
11      reserve_day = Day.where(:date => @reserve.date)
12
13
14      if (reserve_day.empty?)
15        reserve_day = Day.create(:date => @reserve.date,
16                                :structure_name => @structure.title,
17                                :seats_m => @structure.seats_tot,
18                                :seats_a => @structure.seats_tot,
19                                :seats_e => @structure.seats_tot,
20                                :seats_m_c => 0,
21                                :seats_a_c => 0,
22                                :seats_e_c => 0,
23                                :token => rand(100..999))
24      else
25        reserve_day = reserve_day.first
26      end
27      @reserve.day_id = reserve_day.id
28      @reserve.save
29      ok = true
30      if (@reserve.time == "morning")
31        reserve_day.seats_m = reserve_day.seats_m - 1
32        if (reserve_day.seats_m == -1)
33          ok = false
34        end
35      elsif (@reserve.time == "afternoon")
36        reserve_day.seats_a = reserve_day.seats_a - 1
37        if (reserve_day.seats_a == -1)
38          ok = false
39        end
40      else
41        reserve_day.seats_e = reserve_day.seats_e - 1
42        if (reserve_day.seats_e == -1)
43          ok = false
44        end
45      end
46      if (ok)
47        reserve_day.save
48        redirect_to reserves_2_path
49      else
50        @reserve.destroy
51        flash[:fail] = 'Seats Over'
52        redirect_to structure_2_path(@structure.id)+"/#Seats"
53      end
54    end
```

Figura 5.3. funzione create in *reserves\_controller.rb*.

La somma degli attributi *morning*, *afternoon* e *evening seats* di ogni oggetto *Day*

deve essere pari al numero di prenotazioni che sono state effettuate quel determinato giorno; il rispetto di questo vincolo è garantita dal fatto che i valori degli attributi vengono incrementati sempre al momento dell'avvenuta prenotazione, e decrementati quando questa viene cancellata. La limitazione sul tetto massimo delle conferme di prenotazione è garantita direttamente dalla consistenza del vincolo precedente; se ci sono tante prenotazioni quante indicate sull'oggetto *Day* allora necessariamente le conferme non possono eccedere tale valore.

Come già detto precedentemente le condizioni di avere una determinata role-mask sono già garantite da *Canard* e dai controlli effettuati sui controllers. Gli attributi *date* di *Day* e *Reserve* sono sempre identici, questo è assicurato dall'implementazione del metodo *create* che si trova in *reserves\_controller.rb* (figura 5.3) in particolare alla riga 15 vediamo come il *Day* è stato creato direttamente a partire dalla data della prenotazione.

Per il rispetto delle cardinalità sono stati inseriti vincoli di eliminazione a cascata a seguito della cancellazione di un dato elemento dal database come nell'esempio in figura 5.4, ma in alcuni casi è stato necessario introdurre dei moduli che si occupassero della consistenza del database anche dopo un'eliminazione non "pulita" come in figura 5.5 dove se si prova ad eliminare una struttura prima di aver prima annullato tutte le prenotazioni associate ad essa, questo viene fatto direttamente nella chiamata al metodo *destroy* di *structures\_controller.rb*.

```
1  class Structure < ApplicationRecord
2    has_many :reviews, dependent: :destroy
3    has_many :reserves, class_name: 'Reserve', dependent: :destroy
4    has_many :messages, dependent: :destroy
5  end
```

Figura 5.4. Classe Structure in *structure.rb*.

```
58  def destroy
59    authorize! :destroy, Structure, message: "You are not authorized!"
60    @structure = Structure.find(params[:id])
61    if (current_user.id == @structure.manager || current_user.is_admin?)
62      @structure.destroy
63      @place = Place.find(@structure.place_id)
64      @place.destroy
65      reserves_day = Day.where(:structure_name => @structure.title)
66      if (!reserves_day.empty?)
67        for reserve_day in reserves_day
68          reserve_day.destroy
69        end
70      end
71    end
72
73    if (current_user.is_manager?)
74      redirect_to structures_path
75    else
76      redirect_to structures_3_path
77    end
78  end
```

Figura 5.5. funzione destroy in *structures\_controller.rb*.



## 5.2 Application Layer

Proseguiamo con l'*application layer*: tutti i servizi offerti al presentation layer sono implementati come metodi CRUD dei vari controllers, vale a dire create, read, update e delete. Questi metodi sono accessibili mediante *REST route* (figura 5.6) dal presentation layer attraverso la richiesta diretta tramite URL oppure navigando nelle *views* dell'applicazione.

```
1  Rails.application.routes.draw do
2    resources :places
3    #get 'reviews/new'
4    #get 'reviews/create'
5    #get 'reviews/destroy'
6    devise_for :users, controllers: { omniauth_callbacks: 'users/omniauth_callbacks' }
7    # For details on the DSL available within this file, see https://guides.rubyonrails.org/routing.html
8    resources :structures do
9      resources :reviews
10     resources :reserves
11     resources :messages
12     resources :places
13   end
14
15   get '/structures_2' => 'structures#index2', as: :structures_2
16   get '/structures_2/:id' => 'structures#show2', as: :structure_2
17   get '/structures_3/:id' => 'structures#show3', as: :structure_3
18   get '/reserves_2' => 'reserves#index2', as: :reserves_2
19   get '/structures_3' => 'structures#index3', as: :structures_3
20   post '/structures_upgrade' => 'structures#upgrade', as: :structures_upgrade
21   post '/structures/:structure_id/reserves_confirm/:id' => 'reserves#confirm', as: :reserve_confirm
22
23   resources :roles
24   resources :clients
25   resources :admins
26
27
28   root 'structures#index' #fare pagine di root per sicurezza
29
30 end
```

Figura 5.6. *routes.rb*.

Di seguito descriveremo alcune scelte realizzative e mostreremo come avviene l'interazione tra la client e server, quali dati vengono scambiati e quali controller invocati.

Prendiamo ad esempio la creazione di una struttura da parte del *manager*;

- l'interazione inizia con il manager che effettua una richiesta GET della pagina di creazione della struttura, con URL *localhost:3000/structures/new*.
- questo tipo di richiesta invoca il metodo *new* dello *structures controller* che controlla le autorizzazioni del manager e crea un nuovo oggetto *Structure* non inizializzato. A questo punto il server risponde al manager con la pagina richiesta.
- attraverso l'apposito *form* il manager esegue una richiesta POST all'URL *localhost:3000/structures* contenente gli opportuni dati per la creazione della struttura e a questo punto il controller elabora la richiesta attraverso l'esecu-

zione del metodo *create* che popola l'oggetto con i dati inviati dal manager e invia a quest'ultimo la pagina della struttura appena creata.

Possiamo analizzare anche interazioni tra client e server per la modifica di una specifica recensione.

- il client inizia con una richiesta in GET all'indirizzo *localhost:3000/structures/:structure\_id/reviews/:id*, in questo caso abbiamo due parametri (la structure e la review) che sono necessari per identificare univocamente la review stessa.
- il server analizza la richiesta invocando il metodo di *edit* del *reviews controller* e cerca nel database la recensione opportuna salvandola, a questo punto invia all'utente la pagina di modifica della recensione.
- il client, attraverso l'apposito form, esegue una PUT all'URL *localhost:3000/structures/:structure\_id/reviews/:id* contenente i dati per la modifica della stessa.
- a questo punto il server aggiorna la review sul database attraverso il metodo *update* del controller e invia come risposta all'utente la pagina relativa alla struttura corrispondente alla recensione appena modificata.

Infine analizziamo la richiesta di delete per il messaggi.

- il manager manda una richiesta di DELETE all'URL *localhost:3000/structures/:structure\_id/messages/:id* con i parametri corretti.
- il server analizza la richiesta e invoca il metodo di *destroy* del *messages controller* che cancella l'istanza dal database, a questo punto invia all'utente la pagina relativa ai messaggi.

Già nel **capitolo 2.3** abbiamo parlato di alcuni servizi REST esterni, tra cui l'*Oauth* di *Google*; nella trattazione di quest'ultimo non ci siamo però soffermati nel dettaglio nell'interazione tra client e provider nel corso della procedura di autenticazione, ma in questa sezione la analizzeremo [4].

- il client manda una richiesta GET ad uno specifico URL fornito da *Google* che include i *query parametres*, ovvero i parametri opzionali inclusi direttamente nell'indirizzo, con indicato il tipo di accesso richiesto.
- il provider gestisce la richiesta di autenticazione e fornisce un *authorization code* (diverso dal token).
- il client invia il precedente codice al server.
- il server risponde con un *token* e un *refresh token*: il primo viene usato immediatamente per chiamare le API mentre il refresh token verrà scambiato per un nuovo token nel momento in cui quello precedente sarà scaduto.

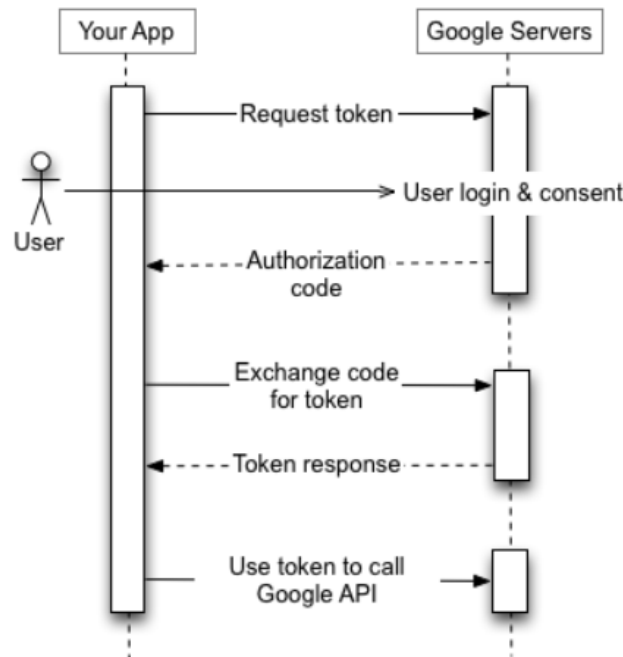


Figura 5.7. Esempio interazione con *Google* per autenticazione esterna.

### 5.3 Presentation Layer

Fino ad ora ci siamo occupati di illustrare il *back-end* e tutto quello che riguarda la gestione delle richieste dal punto di vista interno, ora invece descriveremo come viene presentata l'interfaccia e quali sono le motivazioni che hanno portato a determinate scelte.

Iniziamo dal *front-end* del client: la schermata che si trova di fronte, oltre a mostrare l'elenco delle strutture vicine alla sua posizione, offre anche diverse opzioni; Tutte le funzionalità offerte al client sono accessibili attraverso gli appositi *link* inseriti all'interno pagina e sono evidenziate in **figura 5.8** attraverso una sottolineatura del testo di colore blu oppure viola. Le strutture ricercate sono visibili in una pratica tabella con le opzioni che possono essere selezionate sempre mediante degli appositi link.

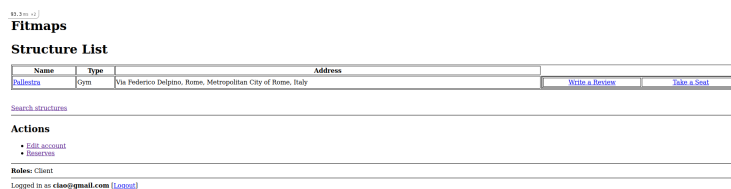
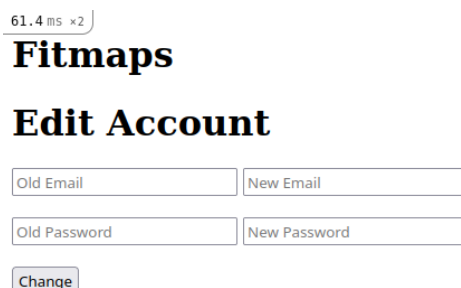


Figura 5.8. Client Homepage.

Per quanto riguarda l'interazione con il database, è necessario che siano forniti dei dati e questo viene eseguito con degli appositi *field* che vanno riempiti oppor-

tunamente: un esempio lo possiamo vedere in **figura 5.9** dove viene illustrata la pagina per la modifica dell'account



61.4 ms x2

## Fitmaps

### Edit Account

Old Email New Email

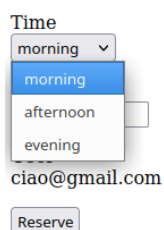
Old Password New Password

Change

**Figura 5.9.** Edit account Page.

Per le selezioni multiple si è scelto di utilizzare dei menù a tendina e per la selezione della data per la prenotazione abbiamo pensato ad un *calendar input*; questo permette all'utente di avere un'interazione più semplice, in quanto viene guidato passo passo per aiutarlo a fornire i dati correttamente; possiamo vedere un esempio in **figura 5.10 e 5.11**. Le informazioni vengono poi inoltrate al server mediante gli appositi pulsanti di *save*.

### Take a Seat



Time

morning

morning

afternoon

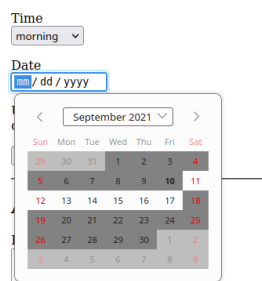
evening

ciao@gmail.com

Reserve

**Figura 5.10.** Esempio menù a tendina.

### Take a Seat



Time

morning

Date

dd / yyyy

September 2021

Sun	Mon	Tue	Wed	Thu	Fri	Sat
29	30	31	1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	1	2
3	4	5	6	7	8	9

**Figura 5.11.** Calendar Input.

Gli utenti dell'applicazione interagiscono con la posizione di una struttura (durante una ricerca oppure durante la creazione del centro) mediante le mappe di *Google* (**figura 5.12**): basta che sul browser si selezioni la barra di ricerca e si inserisca l'indirizzo, attraverso l'*autocomplete* si sceglie la posizione desiderata e si manda la richiesta al server.

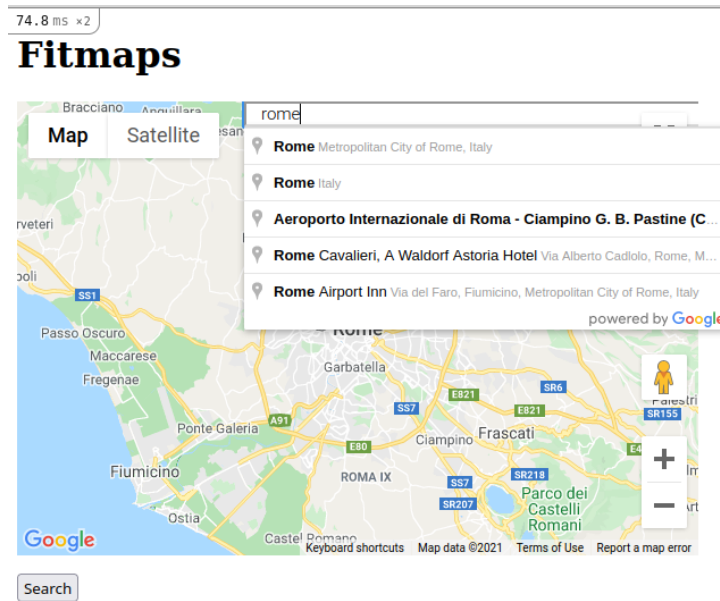


Figura 5.12. *Google Maps.*



## Capitolo 6

# Validazione e dispiegamento

In questo capitolo descriveremo le modalità di *testing* utilizzate e faremo una stima sulle porzioni di software coperte dai test; inoltre spiegheremo anche come poter utilizzare l'applicazione a partire dal codice che risiede su *GitHub*.

### 6.1 Testing

L'ambiente di *Ruby on Rails* permette di automatizzare il processo di testing in modo da poter controllare il codice prodotto frequentemente e soprattutto velocemente, infatti se scrivo dei test che non mutano con lo sviluppo dell'applicazione, almeno quelli sulle funzionalità già implementate, riesco a verificare efficientemente che con l'introduzione di nuove *features* non abbia invalidato quelle inserite in precedenza. I test sono stati divisi in due diverse categorie:

**accettazione e integrazione** serve per collaudare l'integrazione di uno o più moduli

**unitari** collaudo delle singole procedure software

Iniziamo con la descrizione dei test di *accettazione e integrazione*, che sono stati implementati attraverso le gemme di *Cucumber* e *Capybara*. Ogni test è composto un file che contiene una sequenza di *step* (**figura 6.1**) che ne descrivono una funzionalità e separatamente sono implementati i singoli passaggi consultabili in **figura 6.2**. I test sono divisi per comodità in due file distinti, uno per client e uno manager.

```

22 Scenario: Write a Review
23   Given manager creates a structure
24   Given client set position
25   Given I am authenticated as a client
26   Then I should see link "Write a Review"
27   When I click "Write a Review"
28   Then I should be on review page
29   When I fill review body with "test" and score with "1"
30   When I "Save Review"
31   Then I should see a review with body "test"

```

Figura 6.1. Feature write a review.

```

25 Given('client set position') do
26   if(User.where(:email=>"prova@user.com").empty?)
27     @u = User.create(:email=>"prova@user.com", :password => "password", :roles_mask => 1)
28   end
29   @u.update(:latitude => 0.41902784e2, :longitude => 0.12496366e2)
30 end
31
32 Given('/"manager creates a structure"/') do
33   if(User.where(:email=>"prova@manager.com").empty?)
34     @um = User.create(:email=>"prova@manager.com", :password => "password", :roles_mask => 2)
35   end
36   @s = Structure.create(:title => "Palestra", :gym_pool => "Gym", :manager => @um.id, :seats_tot => 10)
37   p = Place.create(:address => "Roma, Metropolitan City of Rome, Italy", :latitude => 0.41902784e2, :longitude => 0.12496366e2, :structure_id => @s.id)
38   @s.update(:place_id => p.id)
39 end
40

```

Figura 6.2. Step definition example.

Descriveremo per primi i test che riguardano le funzionalità offerte al *client*. Nell'esempio delle immagini precedenti abbiamo testato la funzionalità di scrittura di una recensione da parte del client (*write a review*), per prima cosa popola il database con una struttura di test, si prosegue impostando la posizione di default del client ed entrando con le opportune credenziali, si cerca la struttura di test e si compone la recensione; dopo aver salvato, affinché il test sia andato a buon fine, si dovrebbe visualizzare la propria recensione sulla pagina della struttura.

Un ulteriore test di interazione che abbiamo implementato è quello di *edit account* e la sua definizione è la stessa del test precedente: si inizia con la creazione di una struttura di test e con l'autenticazione, poi si accede al link di modifica e si modifica la posizione di default con l'indirizzo della struttura creata precedentemente; a questo punto si dovrebbe visualizzare nella home del client la struttura di test e questo indica che il test si è concluso correttamente.

L'ultimo test inerente al client è quello della prenotazione del posto (*take a seat*): si inizia sempre creando la struttura per il test, settando la posizione e autenticandosi con le credenziali del client, poi si accede alla pagina di prenotazione della struttura di test e si riempiono i campi con i dati corrispondenti; se il test è stato completato correttamente dovremmo vedere sulla pagina delle prenotazioni quella appena effettuata.

Proseguiamo con i test per il *manager*. La prima *feature* testata è quella della creazione di una struttura (*add a structure*): dopo essersi autenticati come manager e aver caricato la pagina per la creazione di una nuova struttura, si compilano i campi con le informazioni opportune e si inoltra la richiesta; a questo punto si



verrà reindirizzati nella pagina di configurazione della posizione e dopo averla scelta dovremmo visualizzare la nuova struttura nella homepage del manager: questo indicherà la corretta conclusione del test.

Concludiamo illustrando il test per l'invio del messaggio (*send a message*): dopo la consueta autenticazione con le credenziali opportune di accede alla pagina per l'invio del messaggio, si compone il corpo di prova e dopo il suo invio, se tutto si è svolto correttamente, si dovrebbe vedere la sezione *view messages* e al suo interno proprio il messaggio appena inviato.

I test *unitari* mettono alla prova i singoli metodi offerti al presentation layer, oltre alla loro implementazione (nel nostro caso abbiamo creato un file specifico per ogni controller testato) hanno anche dei file con estensione *.yml* che contengono delle istanze di prova dei vari modelli, hanno una specifica formattazione in modo che possano essere usati proprio questi per popolare il database per i test. In **figura 6.3** possiamo vedere un esempio di test unitario del metodo `update` del `Reviews Controller`, mentre in **figura 6.4** osserviamo il file *yml* corrispondente.

```
1 require 'rails_helper.rb'
2
3 describe ReviewsController, type: :controller do
4   fixtures [:users, :structures, :reviews]
5
6   #Client
7   context "with Client privileges" do
8     before :each do
9       client = users(:client)
10      sign_in client
11    end
12
13    #Update
14    it "should update reviews" do
15      new_rev = reviews(:one)
16      new_struct = structures(:one)
17      client = users(:client)
18      Review.find(new_rev.id).update(:user_id => client.id, :structure_id => new_struct.id)
19      params = {:review=>{:body => "Change"}, :structure_id => new_struct.id, :id => new_rev.id}
20      get :update, :params => params
21      m_tst = Review.find(new_rev.id)
22      expect(m_tst.body).not_to eql(new_rev.body)
23    end
24  end
25 end
```

Figura 6.3. *reviewcontroller\_spec.rb*.

```
7  one:
8    body: "Test1"
9    score: 5
10   user_id: 1
11   structure_id: 1
12
13  two:
14    body: "Test2"
15    score: 5
16    user_id: 1
17    structure_id: 1
```

Figura 6.4. *review.yml*.

I test sono stati implementati per i controller di: *messages*, *reviews* e *structures* e sostanzialmente sono tutti strutturati come l'esempio in **figura 6.3**, quindi abbiamo testato le funzionalità CRUD di tutti i precedenti moduli.

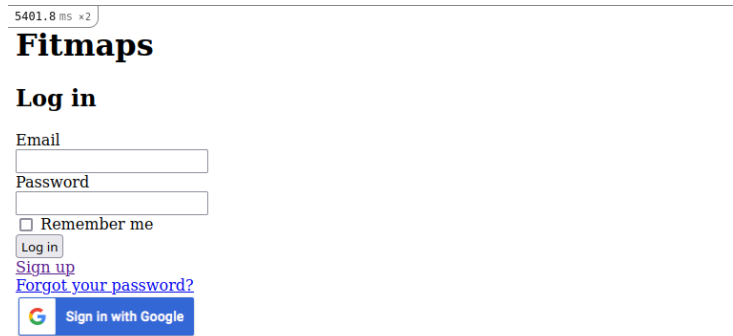
Parliamo ora di copertura dei test: i test di *accettazione e integrazione* coprono una buona percentuale di codice (circa 70%) infatti solo una funzionalità per il client non è stata testata, così come per il manager. I test unitari hanno una copertura leggermente maggiore (circa 75 %) e l'unico modulo non testato è il controller delle prenotazioni anche perché è stato fatto ampiamente con la tipologia di test citata precedentemente.

## 6.2 Installazione dell'applicazione

Partendo dal *codebase* caricato sulla *repository* di *GitHub* si procede eseguendo il comando da terminale: `git clone https://github.com/whoami-Lory271/fitmaps` in una cartella qualsiasi. Una volta fatto ciò si accede a `./fitmaps` e si lancia il comando di `bundle install` per l'installazione delle gemme specificate dal *Gemfile*; si esegue `rails db:migrate` per popolare il database con le entità descritte nei file di migrazione. Purtroppo ci sono alcuni moduli che non vengono inclusi quando si condivide una directory di *Git*, per questo motivo tali moduli vanno reintegrati eseguendo `rails webpacker:install`, a questo punto la nostra applicazione è pronta per essere utilizzata.

## 6.3 Esecuzione

Per poter usufruire di *Fitmaps* si procede in primo luogo aprendo la *root* dell'applicazione con il terminal ed avviando il server (lanciando `rails s`); poi si avvia un qualsiasi browser che fungerà da client e si naviga all'indirizzo `localhost:3000`, a questo punto ci troveremo sull'homepage del sito pronti per creare il nostro primo utente.



**Figura 6.5.** *Fitmaps* Homepage.

Per poter effettivamente provare tutte le specifiche sono necessari almeno due utenti: un *client* e un *manager*, creare una struttura con il manager e poi provare le varie funzionalità offerte al client. Il server riesce a gestire solamente una sessione alla volta, quindi è necessario che la dimostrazione non sia fatta su due finestre aperte in contemporanea ma piuttosto accedendo con le due utenze in differenti momenti.



## Conclusioni

Vorrei concludere la trattazione con alcune riflessioni sul progetto: purtroppo il *framework* di *Ruby on Rails* non mi ha entusiasmato particolarmente, l'ho trovato molto complesso e difficile da personalizzare, molte funzionalità erano nascoste e la documentazione delle gemme non era facile da reperire né molto esauriente. Inoltre non è una piattaforma molto popolare per lo sviluppo di applicazioni e questo complica la ricerca e risoluzione di errori poco comuni. In ogni modo sono rimasto molto soddisfatto dello svolgimento del progetto stesso, in quanto mi ha sicuramente aiutato a migliorare le mie abilità nel lavoro di gruppo, ma soprattutto mi ha fatto capire quanto è importante farlo: un'unica persona che lavora su un progetto in maniera monolitica senza un compagno o un collega che ricontrolli il codice, ha molte più possibilità di fare errori o comunque di produrre un risultato non ottimale. Il lavoro di squadra è fondamentale anche per confrontarsi e accrescere la propria conoscenza, quindi sono molto contento di averlo fatto in un ambiente protetto, prima di avvicinarmi al mondo del lavoro. Vorrei sottolineare come la nostra applicazione possa essere utilizzata anche per modellare diverse altre realtà che hanno una gestione molto simile a quella di palestre e centri sportivi: un esempio può essere quello di grandi luoghi pubblici o privati quali biblioteche, teatri o cinema; queste sono solo alcune delle numerose funzioni offerte dal nostro progetto. Trovo che tale idea abbia un grande potenziale, mi piacerebbe proseguire e ampliare *Fitmpas*, preferibilmente migrandola su un'altra piattaforma. Ci sono una serie di funzionalità che mi sarebbe piaciuto implementare: ad esempio la possibilità di poter integrare l'applicazione con il segnale del GPS del proprio telefono, il che renderebbe l'esperienza d'uso ancora più immediata; oppure integrare un *tool* per la creazione di schede per l'allenamento così che i clienti possano avere tutte le informazioni relative alla palestra comodamente in un'unica applicazione. Si potrebbe sviluppare una versione per *smartphones* che consenta l'integrazione con dei moduli hardware esterni quali ad esempio *smartband* e *smartwatch* per poter condividere i propri progressi in tempo reale direttamente con il trainer, in questo modo potrebbe apportare le modifiche opportune alla scheda del cliente che sarebbe così sempre aggiornata. Per poter utilizzare l'applicazione al meglio ci sarebbe bisogno di un server *multi thread* che consenta a più utenze di interagire con l'applicazione contemporaneamente oltre che l'integrazione con un sistema per il pagamento online degli abbonamenti e lezioni. Naturalmente il nostro progetto non vuole essere un software professionale, però è sicuramente un punto di partenza e, grazie alla possibile introduzione di nuove funzionalità, si potrebbe progettare, un giorno, un prodotto professionale affidabile e totalmente sicuro.



# Bibliografia

- [1] ARMANDO FOX, D. P. *Engineering software as a service an agile approach using cloud computing*. Strawberry Canyon LLC (2013). ISBN 1735233803.
- [2] GOOGLE. Place autocomplete (2021). Visited on: 02/09/2021. Available from: <https://developers.google.com/maps/documentation/places/web-service/autocomplete>.
- [3] GOOGLE. Place autocomplete (2021). Visited on: 03/09/2021. Available from: <https://developers.google.com/maps/documentation/javascript/examples/places-autocomplete>.
- [4] GOOGLE. Using oauth 2.0 to access google apis (2021). Visited on: 10/09/2021. Available from: <https://developers.google.com/identity/protocols/oauth2>.
- [5] JÖRG W MITTAG. Stack overflow user comment about openid (2010). Visited on: 02/09/2021. Available from: <https://stackoverflow.com/questions/4230821/if-openid-is-dead-what-is-out-there-to-take-its-place/4230970#4230970>.
- [6] RAILS. Rails routing from the outside in (2021). Visited on: 05/09/2021. Available from: <https://guides.rubyonrails.org/routing.html>.
- [7] ROBERTO BERALDI, M. C. *Slides del corso*.
- [8] ROGER A. GRIMES, JOSH FRUHLINGER. What is oauth? how the open authorization framework works (2019). Visited on: 02/09/2021. Available from: <https://www.csoonline.com/article/3216404/what-is-oauth-how-the-open-authorization-framework-works.html>.
- [9] TUTORIALSPPOINT. Ruby on rails - migrations (2021). Visited on: 05/09/2021. Available from: <https://www.tutorialspoint.com/ruby-on-rails/rails-migrations.htm>.
- [10] WIKIPEDIA. OAuth (2021). Visited on: 01/09/2021. Available from: <https://en.wikipedia.org/wiki/OAuth>.