

# UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

---

Dipartimento di Scienze Fisiche Informatiche e Matematiche  
CORSO DI LAUREA TRIENNALE IN INFORMATICA

## **Dashboard Estate - Progettazione e Sviluppo di una Web Application per l'Analisi sul Mercato Immobiliare**

*Relatore:*

*Laureando:*

Prof. RICCARDO MARTOGLIA

LORENZO BARCELLONA

---

ANNO ACCADEMICO 2019-2020

# Indice

<b>I</b>	<b>Stato dell'arte</b>	<b>4</b>
<b>1</b>	<b>Contesto di riferimento</b>	<b>5</b>
1.1	70 Division s.r.l. . . . .	5
1.2	Obiettivi e scelte progettuali . . . . .	5
<b>2</b>	<b>Studio delle tecnologie</b>	<b>7</b>
2.1	Database . . . . .	7
2.1.1	MongoDB . . . . .	9
2.2	JSON . . . . .	11
2.3	JavaScript . . . . .	12
2.4	Framework . . . . .	12
2.4.1	Angular . . . . .	13
2.4.2	React . . . . .	13
2.4.3	Vue . . . . .	14
2.5	React-Redux . . . . .	15
<b>II</b>	<b>Progetto e Sviluppo</b>	<b>19</b>
<b>3</b>	<b>Progettazione</b>	<b>20</b>
3.1	Analisi dei Requisiti dell'applicazione . . . . .	20
3.1.1	Terminologia . . . . .	20
3.1.2	Requisiti Funzionali . . . . .	21
3.2	Casi d'uso . . . . .	23
3.3	Diagramma delle attività . . . . .	25
3.4	Progettazione Database . . . . .	26
3.4.1	Costruzione delle entità . . . . .	27
<b>4</b>	<b>Implementazione</b>	<b>30</b>
4.1	Struttura del progetto . . . . .	30
4.2	Dashboard . . . . .	33
4.2.1	Login . . . . .	33
4.2.2	Pagina iniziale . . . . .	34
4.2.3	Ricerca . . . . .	35
4.2.4	Dettaglio . . . . .	38
<b>5</b>	<b>Conclusioni e sviluppi futuri</b>	<b>40</b>

# Introduzione

Lo studio del mercato immobiliare è sempre stato un punto d'interesse per la società moderna, molte sono infatti le aziende che operano su di esso.

Insieme a 70 Division abbiamo deciso di implementare "Dashboard Estate", con lo scopo di fornire uno strumento utile ai professionisti per analizzare il mercato immobiliare e poter compiere scelte oculate. L'applicazione presenta varie funzionalità, specialmente di ricerca e confronto di dati, che forniscono un'idea di come si muove il mercato. Si è cercato inoltre di darle un'interfaccia utente semplice e intuitiva.

Come database da utilizzare per lo sviluppo della dashboard è stato utilizzato MongoDB, database di tipo NoSQL. Per quanto riguarda l'implementazione è stato utilizzato il framework React integrandolo con Redux. Le prime settimane di tirocinio sono state dedicate allo studio di tale framework.

La struttura della tesi si divide in due parti principali:

- Stato dell'arte;
- Progetto e sviluppo;

## Stato dell'arte

Questa parte della tesi è suddivisa in due capitoli:

- **Capitolo 1:** In questo capitolo viene descritto il contesto in cui si è lavorato e quali fossero gli obiettivi da raggiungere.
- **Capitolo 2:** In questo capitolo si descrivono le tecnologie utilizzate, partendo dall'analisi dei database disponibili sul mercato e illustrando perchè si è deciso di utilizzare MongoDB. A seguire è stata fatta un'analisi sui tre principali framework JavaScript in circolazione: Angular, React e Vue. Infine è stato approfondito lo studio su React e sull'architettura di Redux.

## Progetto e sviluppo

Questa parte della tesi si suddivide in tre capitoli:

- **Capitolo 3:** Questo capitolo tratta della progettazione della dashboard. Vengono analizzati i requisiti funzionali che l'applicazione deve fornire, analizzato il comportamento del sistema grazie al diagramma sul caso d'uso e al diagramma delle attività. Viene inoltre fornita una panoramica sulla progettazione del database e delle entità che lo popolano.
- **Capitolo 4:** In questo capitolo viene mostrata la struttura del progetto e l'implementazione dell'applicazione.
- **Capitolo 5:** Nell'ultimo capitolo si traggono le conclusioni sul lavoro svolto e si suggerisce come migliorare il prodotto software.

# Parte I

## Stato dell'arte

# Capitolo 1

## Contesto di riferimento

### 1.1 70 Division s.r.l.

*70 Division s.r.l.* è un'azienda informatica con sede a di Fabbrico (RE) che dal 2011 si occupa principalmente di consulenza digital.

Nel corso degli anni l'azienda ha sviluppato siti-web, e-commerce, pubblicità in digital etc.

L'azienda è costituita da un ufficio corredato degli strumenti necessari per lo sviluppo dei software ed è formata da personale qualificato in tecnologie informatiche.

### 1.2 Obiettivi e scelte progettuali

All'epoca in cui si è svolto il tirocinio, *70 Division s.r.l.*, forniva consulenza ad una società specializzata nell'analisi massiva dei crediti deteriorati<sup>1</sup>.

La suddetta società ha come oggetto la valutazione e la compravendita di crediti deteriorati nel mercato immobiliare, come ad esempio le aste giudiziarie. Necessitava dunque di uno stru-

---

<sup>1</sup>i crediti deteriorati sono crediti delle banche che i debitori non riescono più a ripagare regolarmente o del tutto

mento con cui effettuare un controllo efficace nella fase di valutazione del credito di un immobile, per confrontare il credito deteriorato in base al mercato.

E' stato dunque deciso di collaborare alla creazione di una dashboard per studiare il mercato delle proprietà in vendita per confrontare il valore della base d'asta con la stima della rivendita di un immobile, considerando l'andamento del mercato.

Si intende quindi sviluppare dunque la "Dashboard Estate" con le seguenti caratteristiche:

- Un database contenente un vasto numero di proprietà, in modo da effettuare stime di mercato su grandi numeri.
- Una pagina che mostri le informazioni delle proprietà in vendita sulla base di determinati parametri di ricerca.
- Finestre che mostrino nel dettaglio le singole proprietà.
- Una mappa che rappresenti graficamente le proprietà in vendita.
- Un'interfaccia utente facile ed intuitiva.

# Capitolo 2

## Studio delle tecnologie

In questo capitolo si analizzano le tecnologie utilizzate per l'implementazione del progetto. Verrà trattato prima lo studio del database ed infine il framework e il linguaggio di programmazione adottati.

### 2.1 Database

Vengono ora analizzati alcuni modelli di database disponibili sul mercato per identificare quello più adatto alle esigenze del progetto.

**Modello Relazionale:** gli elementi di un database relazionale sono organizzati sotto forma di set di tabelle composte da colonne e righe. La tecnologia di questo tipo di database offre la soluzione più efficiente e flessibile per accedere alle informazioni strutturate.

- utilizzo del linguaggio SQL per archiviazioni e interrogazioni.
- rappresentazione tramite tabelle.



- una volta definita la struttura dei dati non è possibile effettuare alcuna modifica.

**Database orientato agli oggetti:** è un modello di base di dati in cui l'informazione è rappresentata in forma di oggetti come nei linguaggi di programmazione ad oggetti.

- rappresentazione dei dati tramite veri e propri oggetti, come accade nei linguaggi di programmazione ad oggetti.
- Set di dati complessi possono essere memorizzati e richiamati in modo semplice e veloce.
- In alcune situazioni l'elevata complessità può portare a problemi di prestazione.

**Database orientato ai documenti:** le basi di dati orientate ai documenti sono utilizzate per la gestione dei dati semi-strutturati.

- i documenti non hanno una struttura rigida, ma dati semi-strutturati.
- Ogni record è memorizzato come un documento con determinate caratteristiche e può differire dagli altri.
- le informazioni sono archiviate come un insieme di coppie di chiave-valore dove una chiave rappresenta un identificatore univoco.
- non sono utilizzate tabelle per memorizzare le informazioni.
- sono utilizzate delle operazioni CRUD: creazione(Creation), recupero (Retrieval), aggiornamento (Update) ed eliminazione (Delete).
- principale categoria dei database NoSQL.

Nella sezione seguente si analizza nello specifico un database orientato ai documenti scelto per il progetto, ovvero MongoDB.

### 2.1.1 MongoDB

MongoDb è un sistema di gestione di basi di dati open source per applicazioni e infrastrutture internet, scritto in C++ e sviluppato da *MongoDB Inc.* (in principio 10gen). Progettato per essere potente, flessibile e scalabile e per la sua capacità di integrare le migliori caratteristiche dei database orientati ai documenti e dei database relazionali, che presentano un ricco modello di dati e un potente linguaggio di interrogazione.[1]

Di seguito le principali caratteristiche di MongoDB che lo rendono un'ottima alternativa ai modelli relazionali:

- **Modello orientato ai documenti**

- i documenti corrispondono a tipi di dati nativi di molti linguaggi di programmazione; il modello dei dati basato su documenti memorizzati in formato JSON/BSON che sono facili da codificare e da gestire, data la mancanza di uno schema fisso, consentendo di ottenere eccellenti performance nelle interrogazioni di raggruppamento di dati.
- l'incorporazione dei documenti e degli array riduce il bisogno dei costosi join.
- i documenti non seguono uno schema rigido.

- **Prestazioni elevate**

- l'assenza di join consente operazioni di lettura e scrittura più veloci.
- utilizza molte proprietà dei database relazionali quali: indici secondari, interrogazioni dinamiche, sorting e aggregazione, senza rinunciare alla flessibilità e alla scalabilità dei modelli non relazionali.

- **Disponibilità elevata**

- grazie al processo di sincronizzazione dei dati tra più server (detto replica), che fa in modo di avere più copie dei dati su diversi server di database. In MongoDB si ha un set di repliche formate da due o più nodi (almeno 3 nodi di solito). Uno di questi è il nodo primario e i nodi rimanenti sono detti secondari.

- **Scalabilità dei dati**

- ridimensionamento orizzontale delle informazioni, detto *sharding*, che permette di distribuire i dati su più macchine e facilitare operazioni ad alta produttività per una grande quantità di dati, invece di affidarsi a un singolo server ad alta velocità e alta capacità.

- **Facilità d'uso**

MongoDB è stato creato per essere facile da installare, configurare, mantenere e usare.

MongoDB utilizza una struttura differente da quella classica basata su tabelle, utilizzando documenti di tipo JSON e le *collection*, dette anche raccolte, che contengono insiemi di documenti e funzionano in maniera analoga alle tabelle classiche. Le *collection* non impongono alcuna sorta di schema, i documenti al suo interno possono avere qualsiasi tipo di struttura e differire tra loro.

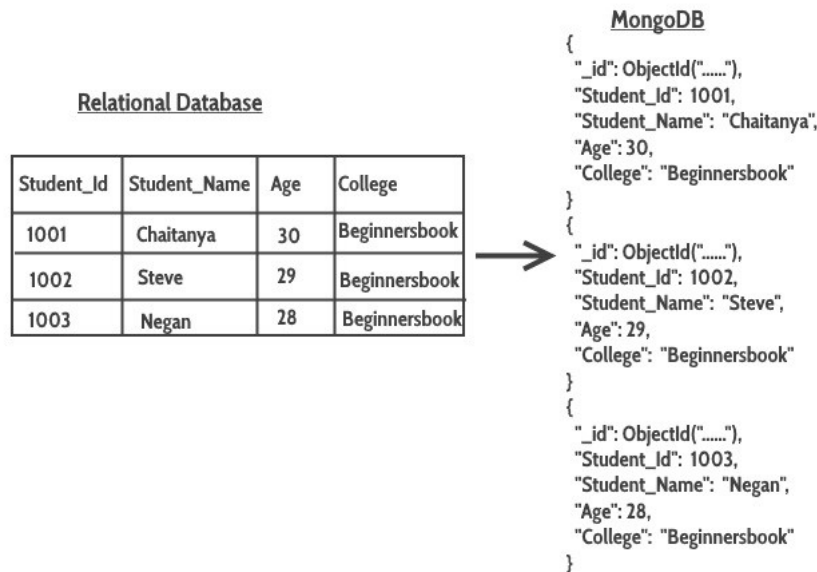


Figura 2.1: Confronto modello relazionale e modello orientato ai documenti

## 2.2 JSON

JSON (JavaScript Object Notation) è un semplice formato per lo scambio di dati. Facile da leggere e scrivere per i programmatori e facile da generare ed analizzarne la sintassi per le macchine. Si basa su un sottoinsieme del Linguaggio di Programmazione JavaScript, Standard ECMA-262. JSON è un formato di testo completamente indipendente dal linguaggio di programmazione, ma utilizza convenzioni conosciute dai programmatori di linguaggi della famiglia del C, Questa caratteristica fa di JSON un linguaggio ideale per lo scambio di dati. JSON è basato su due strutture: un insieme di coppie nome/valore, rappresentabili come un oggetto o un record e da un elenco ordinato di valori, che si realizza di solito tramite un array, un vettore o un elenco. Queste sono strutture di dati universali e i linguaggi di programmazione moderni le supportano entrambe.[2]

## 2.3 JavaScript

Per lo sviluppo dell'applicativo si è scelto come linguaggio di programmazione JavaScript.

JavaScript è un linguaggio di programmazione orientato agli oggetti e agli eventi.

Viene utilizzato nella programmazione Web lato client per la creazione di siti e applicazioni web aventi effetti dinamici interattivi.[3]

Le sue caratteristiche principali sono:

- è un linguaggio interpretato: il codice non viene compilato, ma eseguito direttamente.
- la sintassi è simile a quella dei linguaggi C, C++, Java.
- è debolmente tipizzato, le variabili vengono tutte definite dal tipo comune "var", non viene quindi specificato il vero tipo della variabile nel codice.
- è debolmente orientato agli oggetti, il meccanismo di ereditarietà differisce da quello del linguaggio Java.
- il codice viene eseguito direttamente sul client e non sul server.
- può valutare le espressioni regolari.

## 2.4 Framework

In questa sessione si illustrano tre framework basati sui componenti, così da scegliere quello che più si addice alle esigenze del progetto: Angular, React e Vue.

Le performance di 3 framework sono più o meno equivalenti, quindi questi dettagli non saranno oggetto di analisi approfondita.[4]

### 2.4.1 Angular

Angular è un framework JavaScript basato su TypeScript supportato da Google. È un framework estremamente popolare per lo sviluppo front-end. Gli aggiornamenti ed il rilascio di nuove versioni sono molto frequenti, ma le migrazioni sono ben supportate. Un aggiornamento sostanziale è rilasciato ogni 6 mesi. Angular è meno flessibile di React e Vue, in quanto è un full-framework, considerato tale perchè fornisce un ottima base d'inizio per creare un'implementazione con la configurazione completa e senza l'uso di librerie.

Angular implica l'inserimento di dipendenze, un concetto in cui un oggetto fornisce le dipendenze a un altro oggetto, rendendo il codice più pulito e facile da capire. D'altra parte utilizza l'architettura MVC, che divide il progetto in tre componenti: Model, View e Controller.

Angular utilizza un approccio *two-way Data Binding*, meccanismo in cui i campi dell'interfaccia utente sono associati per modellare dinamicamente. Quando gli elementi dell'interfaccia utente cambiano, i dati del modello vengono modificati di conseguenza. Questo approccio fornisce una struttura di codifica efficiente e gli sviluppatori trovano facile lavorarci.

### 2.4.2 React

React è un set di librerie JavaScript indipendenti veloci ed in evoluzione. Utilizzato per implementare componenti UI per web-application. E' gestito e supportato da Facebook, ma utilizzato anche da Netflix, Uber, Twitter, Paypal, Reddit e molti altri. Molte compagnie hanno preferito migrare su React per le

sue eccezionali caratteristiche e funzioni.

E' molto più flessibile rispetto ad angular visto che è un insieme di librerie e non un full-framework, ciò permette di controllare ogni singolo modulo che non è più supportato.

Anche in React si hanno aggiornamenti e rilasci di versione molto frequenti e le migrazioni sono ben supportate.

React è molto concentrato su JavaScript, per questo motivo alcuni sviluppatori preferiscono lavorare con Angular o Vue.

Con React, si possono eseguire più integrazioni poiché si può accoppiare, scambiare e integrare librerie con altri strumenti disponibili.

React funziona fuori dagli schemi grazie alla sua flessibilità per offrire un'integrazione perfetta ma ci sono più possibilità di sbagliare e richiede più dipendenze.

React è orientato al *one-way Data Flow*, ovvero significa che esiste un'unica fonte di verità: il modello. I dati di un'applicazione fluiscono in un'unica direzione e solo il modello può modificare lo stato dell'app. Questo approccio offre una migliore panoramica e comprensione perché i dati vengono gestiti in una sola direzione.

React fa uso di un DOM-virtuale, modello molto utile in termine di prestazioni.

### 2.4.3 Vue

Vue è uno dei framework JavaScript più discussi e in rapida crescita. E' stato creato da uno sviluppatore di Google per semplificare Angular. Ottiene un grande successo grazie all'abilità di costruire UI attraenti utilizzando HTML, CSS e JavaScript. Quando avvengono delle migrazioni si può utilizzare uno strumento di supporto che le semplifica, ma nelle app di grandi dimensioni, potrebbe causare un problema in quanto non esiste una roadmap adeguata che si concentri sul controllo delle ver-

sioni e sui loro piani.

Vue è il più pulito tra questi tre framework. Aiuta a mantenere efficiente il codice con il perfetto equilibrio tra dipendenze interne e flessibilità.

Vue può utilizzare sia il meccanismo *two-way Data Binding* sia il *one-way Data Flow*.

Anche Vue, come React, utilizza un modello di DOM-virtuale.

## 2.5 React-Redux

E' stato scelto React per:

- la sua capacità di integrarsi con qualsiasi tecnologia. E' quindi possibile scrivere codice in React all'interno di altri framework di lavoro ed anche di all'interno di server che usano Node;
- per l'Atomic Design, filosofia che prevede la creazioni di semplici componenti che uniti tra loro formano il prodotto finale;
- infine gode di una community molto numerosa

I punti fondamentali di React sono illustrati come segue:

- **Componente**: Il codice di React è costituito da entità denominate componenti, che possono essere sottoposti a rendering su un particolare elemento nel DOM . Quando si esegue il rendering di un componente, si possono passare valori noti come "props".
- **JSX**: utilizza il linguaggio JavaScript XML, un'estensione di JavaScript. Applicata a React descrive come l'interfaccia utente dovrebbe apparire. JSX crea elementi React che



andranno a popolare il DOM. Non saranno dunque necessari più file per scrivere il codice JavaScript, CSS e HTML, ma ne servirà uno solo.

- **DOM virtuale** : è un concetto di programmazione dove una rappresentazione virtuale del UI è mantenuta in memoria e sincronizzata con il DOM reale, tramite alcune librerie, tipo ReactDOM.
- **Componente funzionale** : è una semplice funzione JavaScript che ritorna un elemento JSX. Noti come componenti *stateless*.
- **Componenti basati su classi** : classe JavaScript che estende React.Component e ritorna un elemento JSX tramite il metodo render.  
Noti anche come componenti *stateful*, perchè possono avere uno stato che contiene valori in tutto il componente.
- **Metodo del ciclo di vita** : Ogni componente è caratterizzato da un ciclo di vita. All'interno dei componenti basati su classi abbiamo accesso a dei metodi particolari (*lifecycle hooks*) che potremo usare per intercettare alcuni istanti del ciclo di vita di un componente e stabilire un determinato comportamento per quel componente.

## Redux

Redux è un'architettura per scrivere applicazioni web basate su React. Tale architettura permette di gestire uno stato globale all'interno dell'applicazione. Il suo scopo è rendere prevedibile il cambio di stato tramite alcune restrizioni su come e quando questo può avvenire. La gestione dello stato e i suoi cambiamenti sono astratti rispetto all'interfaccia utente[5].

Il concetto è avere un unico stato rappresentato da un oggetto

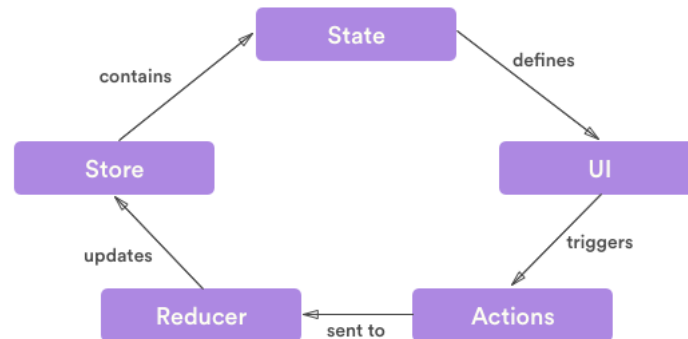


Figura 2.2: Logica di Redux

JSON e conservato in uno *store*. Lo stato può mutare solo in seguito a delle azioni e la modifica avviene tramite l'invocazione di una funzione pura denominata *reducer*.

Tre principi fondamentali su cui si basa Redux:

- **Singola fonte di verità**: lo stato è l'unica fonte di verità a cui fa riferimento la UI.
- **Lo stato è read-only**: lo stato è in sola lettura e muta con intenti ben definiti, ovvero partendo da un'azione che passa tramite il *reducer*.
- **Cambiamenti di stato tramite funzioni pure**: i cambiamenti avvengono tramite funzioni pure, ovvero funzioni che preso un input restituiscono lo stesso output. Tali funzioni vengono utilizzate per gestire i *reducer*. [1]

Si illustrano di seguito altri concetti di Redux:

- **Stato**: lo stato è quello che conosciamo già in React, cioè lo stato che viene renderizzato nella nostra UI. È buona pratica, e Redux serve proprio a questo, mantenere lo stato generale dell'applicazione il più centralizzato possibile.

Redux e la centralizzazione dello stato vengo utilizzate perché lo stato locale non scala bene all'aumentare degli sviluppatori.

- **Store:** è la parte dell'architettura che mantiene lo stato dell'applicazione.
- **Azioni:** sono funzioni che fanno il *dispatch* di azioni, come ad esempio una invocazione ad un servizio esterno. Queste azioni possono contenere un body, che potrebbe essere utilizzato, nei reducer, per eseguire la mutazione dello stato.
- **Reducer:** Qui viene gestito lo stato. Si tratta di un grande *switch* che esegue una variazione dello stato in base all'azione, ed eventualmente, al suo contenuto.
- **Connect:** questa è una funzione che, invocata quando si esporta il componente, serve a collegarlo alle azioni e alla parte dello stato e delle azioni di cui ha bisogno.
- **Selector:** Si può notare come la selezione dello stato soffra del problema dell'usare una stringa che può causare problemi nel refactoring. Per ovviare a questo si possono creare delle funzioni in un unico luogo che vanno a compiere quell'operazione.
- **Container:** il concetto può essere interpretato come il componente contenitore di altri componenti che vanno a formare quello che nelle vecchie applicazione web chiameremmo pagina.
- **Middleware:** Il middleware è una *higher-order function* che riceve una funzione di *dispatch* e ne restituisce un'altra potenziata.

# Parte II

## Progetto e Sviluppo

# Capitolo 3

## Progettazione

Questo capitolo tratterà della progettazione del prodotto software, che prevede quindi un'analisi dei requisiti funzionali che il software deve offrire all'utente e le relative caratteristiche. Verranno dunque illustrate le scelte fatte riguardo alle funzioni che il software deve svolgere senza entrare nei dettagli implementativi.

### 3.1 Analisi dei Requisiti dell'applicazione

Mostriamo in seguito la terminologia e i requisiti funzionali dell'applicazione.

#### 3.1.1 Terminologia

- **Dashboard** : una vista di informazioni che consente di monitorare eventi, osservare tendenze al fine di adottare decisioni più mirate.
- **Proprietà** : edifici residenziali in vendita attualmente.
- **Utente** : personale dell'azienda cui è consentito l'accesso alla dashboard.

- **Dettaglio** : anteprima di una singola proprietà con le relative informazioni.
- **Mappa** : rappresentazione grafica delle proprietà in vendita.
- **Zone OMI** : Organizzazione del Mercato Immobiliare.

### 3.1.2 Requisiti Funzionali

In questa sezione si descrivono tutte le funzioni che svolgerà il software spiegandone l'uso e il procedimento.

- **RF 1 Login**: il software deve fornire un'azione per accedere alla dashboard tramite una procedura di autenticazione. Sono quindi necessari i seguenti campi:
  - **Lista Utenti** : lista di tutti gli utenti che possono accedere alla dashboard.
  - **Username**: una casella di testo dove inserire l'username, sequenza alfanumerica di caratteri che identifica univocamente l'utente.
  - **Password** : una casella di testo dove inserire la password, sequenza alfanumerica di caratteri utilizzata per accedere in modo esclusivo al profilo dell'utente.
- **RF 2 Ricerca semplice** : funzione che permetta di effettuare una ricerca semplice sulle proprietà inserendo solo il luogo.
  - **Lista proprietà** : una lista con le proprietà in vendita sulla quale effettuare la ricerca.
  - **Luogo** : casella di testo dove inserire la città o l'indirizzo in cui cercare le proprietà in vendita.
  - **Bottone** : bottone per svolgere la procedura di ricerca.

- **RF 3 Ricerca con filtri** : funzione che permetta di effettuare una ricerca più dettagliata sulle proprietà inserendo dei filtri, quale prezzo e dimensione.
  - **Lista proprietà** : una lista con le proprietà in vendita sulla quale effettuare la ricerca.
  - **Luogo** : casella di testo dove inserire la città o l'indirizzo in cui cercare le proprietà in vendita.
  - **Filtri** : quattro caselle di testo per filtrare le proprietà in base al prezzo massimo o minimo e alla dimensione massima o minima.
  - **Bottone** : bottone per svolgere la procedura di ricerca con filtri.
  - **Mappa** : rappresentazione grafica della lista proprietà.
- **RF 4 Stampa dei risultati** : il software deve ovviamente mostrare i risultati della ricerca semplice o con filtri, fornendo sia una lista che una mappa di questi.
  - **Lista** : una lista che stampi tutte le proprietà inerenti alla ricerca.
  - **Mappa** : rappresentazione grafica della lista proprietà.
- **RF 4 Dettaglio** : il software deve fornire la visualizzazione di una proprietà nello specifico.
  - **Proprietà** : id dell'immobile che si vuole valutare.
  - **Dati** : informazioni relative alla proprietà nel dettaglio; verranno mostrati indirizzo, prezzo, dimensione della casa più l'URL del sito che mostra la vendita dell'immobile.
  - **Zona OMI**: viene inoltre fornita una statistica relativa alla zona OMI dell'immobile.

- **RF5 Logout**: il software deve ovviamente permettere la procedura di logout.
  - **Bottone** : basterà un semplice bottone che permetta di effettuare la procedura.

## 3.2 Casi d'uso

Dopo aver analizzato i requisiti della dashboard possiamo ora andare a mostrare il diagramma dei casi d'uso, ovvero la specifica di una sequenza di azioni, incluse eventuali sequenze alternative o di errore, che un sistema può eseguire interagendo con un attore esterno.[6]

In questo caso l'attore è l'utente che interagisce con la dashboard, mentre le azioni sono i requisiti funzionali elencati nella sezione 3.1.2.

In figura 3.1 viene mostrato il diagramma dei casi d'uso del software



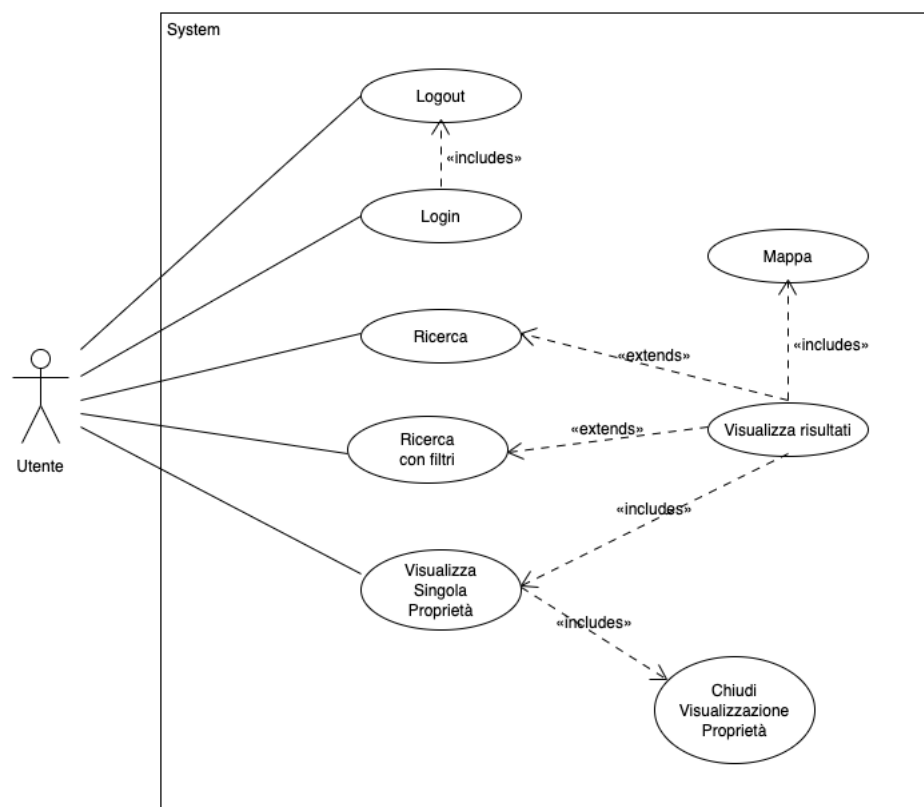


Figura 3.1: Diagramma dei casi d'uso

### 3.3 Diagramma delle attività

Il diagramma delle attività è un tipo di diagramma che permette di descrivere un processo attraverso dei grafi in cui i nodi rappresentano le attività e gli archi l'ordine con cui vengono eseguite. Il diagramma parte con il nodo iniziale rappresentato da un cerchio pieno e termina nel nodo finale raffigurato da un cerchio bordato. Le attività vengono descritte dai nodi azione, rettangoli smussati con una breve descrizione dell'azione. Per i nodi di decisione si utilizzano dei rombi.

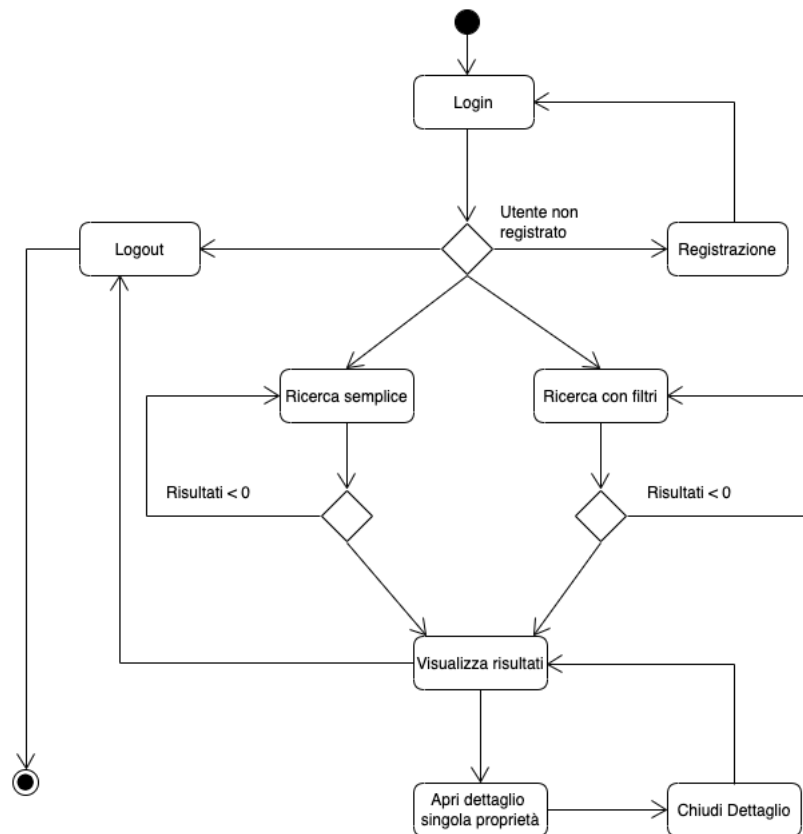


Figura 3.2: Diagramma delle attività

### 3.4 Progettazione Database

Come accennato nel paragrafo 2.1.1 è stato scelto MongoDB per la creazione del database. Avendo dunque a che fare con un database di tipo NoSQL la progettazione logica del progetto non è stata realizzata con il classico schema ER (Entità e Relazioni), strumento di modellazione comune per la progettazione concettuale e logica dei database relazionali. La progettazione del database del progetto è stata perciò effettuata seguendo la logica dei database NoSQL, dove ogni dato è stato immagazzinato nei file JSON.

Vengono di seguito mostrate, in figura 3.3, tutte le entità che compongono il software e le loro associazioni.

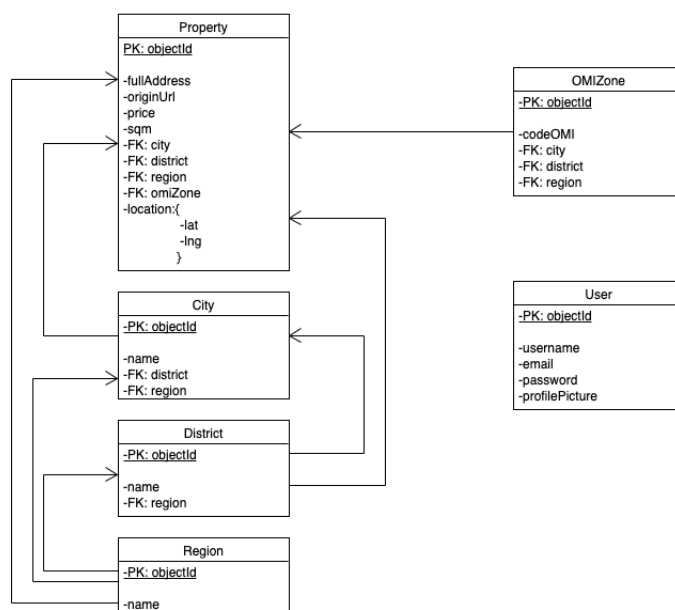


Figura 3.3: Diagramma completo dell'entità del software

### 3.4.1 Costruzione delle entità

**Proprietà':** L'entità Proprietà rappresenta un immobile in ven-



Figura 3.4: Entità delle proprietà

dita all'interno dell'applicazione. L'attributo chiave che contraddistingue una proprietà dall'altra è *objectId*. Dopo di che ci sono una serie di attributi che definiscono le caratteristiche dell'immobile come il prezzo, la dimensione, l'indirizzo e la locazione. L'attributo *location* contiene le coordinate della proprietà che risulteranno utili per rappresentazione grafica. l'*omiZone*, invece, definisce in quale zona del mercato immobiliare si trova la proprietà.

**Città:** L'entità Città viene anch'essa contraddistinta dall'at-

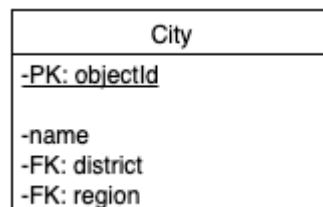


Figura 3.5: Entità delle città

tributo chiave *objectId*, così come descritto precedentemente per la proprietà. Il campo *name* contiene il nome della città; i campi *district* e *region* identificano la provincia e la regione di appartenenza della città.

Le entità delle province e delle regioni (in figura 3.3 *District* e *Region*), hanno una struttura simile all'entità Città precedentemente descritta.

**OMI Zone:** Anche in questo caso la zona OMI viene identi-

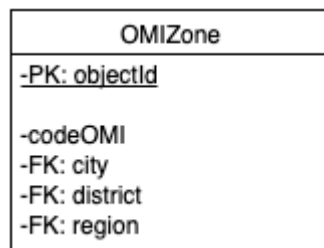


Figura 3.6: Entità delle zone OMI

ficata dal valore chiave *objectId*. L'attributo *codeOMI* definisce il codice della zona, rappresentato da una sequenza alfanumerica. Infine anche in questa entità ci sono attributi che specificano dove si trova la zona OMI.

**Utente:** L'entità Utente rappresenta un utente registrato alla

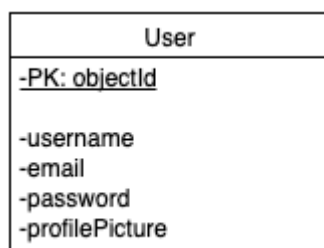


Figura 3.7: Entità delle città

dashboard ed è identificata dal valore chiave *objectId*. L'attributo *username* rappresenta il nome dell'utente all'interno del

software, che insieme all'attributo *password* permette l'accesso alla dashboard.

La costruzione del diagramma è molto intuitiva, dato che si è cercato di ridurre al minimo le entità memorizzate, evitando di riempire il database con informazioni non necessarie.

Come si può vedere in figura 3.3 ogni proprietà può appartenere ad 1 sola zona OMI. Ovviamente ogni proprietà sarà collegata ad 1 sola città, mentre le città possono avere da 0 a N proprietà, 0 nel caso non ci sia alcun immobile in vendita nella città.

Si può banalmente intuire che le regioni avranno collegamenti 1 a N con le province e le città, mentre province e città saranno collegate ad 1 sola regione. Le province saranno collegate in modo analogo alle città.

# Capitolo 4

## Implementazione

In questo capitolo verrà mostrato come è stata sviluppata la dashboard cercando di rispettare i requisiti funzionali richiesti.

### 4.1 Struttura del progetto

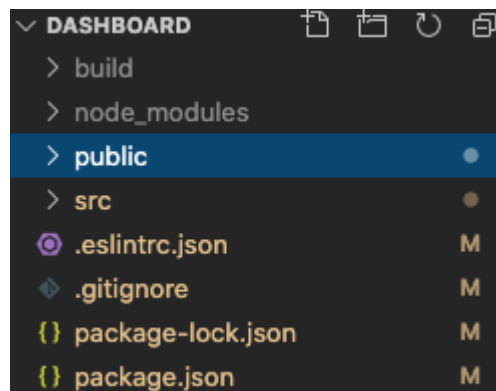


Figura 4.1: Struttura del progetto

- **Package.json**

La cartella di progetto include un set di package già installati, facenti parti dell'ecosistema di Node.js. Tali file sono contenuti nella directory speciale `node_modules`.

Una parte dei package sono necessari per l'esecuzione dell'applicazione, dato che contengono i tipi e le funzioni

a cui si fa riferimento nei file sorgenti per costruire l'infrastruttura dell'app. Questi file vengono distribuiti con l'applicazione in modo che possa materialmente funzionare.

- **File sorgenti** La cartella `src` del progetto contiene infine i file sorgenti veri e propri, che costituiscono l'applicazione: essi comprendono file JavaScript con il codice dei componenti, file CSS con i fogli di stile, eventuali "asset" (immagini, icone, font, ecc.) e i reducer.

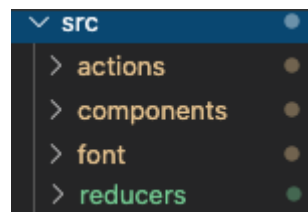


Figura 4.2: file sorgenti

– **actions:**

nella cartella `actions` troviamo due file: `index.js`, contenente tutte le definizioni delle azioni che svolge l'applicazione, e `types.js`, che contiene i nomi delle azioni.

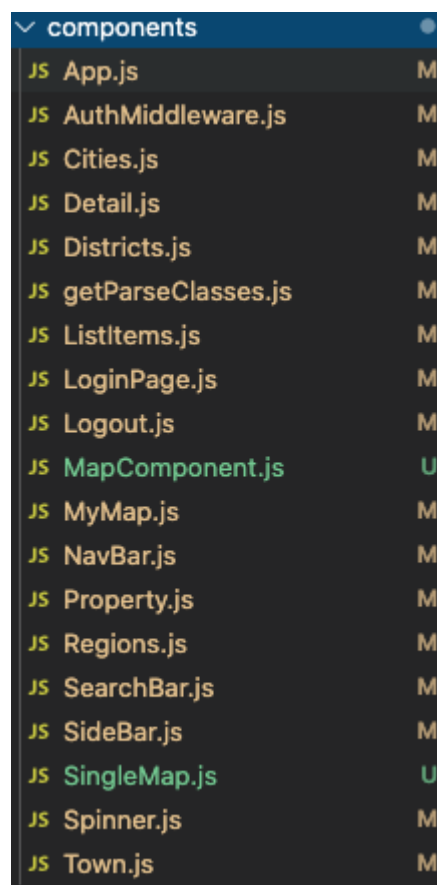


Figura 4.3: azioni del progetto

– **components:**

Nella cartella `components` troviamo tutti i componenti che compongono l'applicazione, ad esempio il file `App.js`, già predisposto dal tool `create-react-app`, che rappresenta il componente principale dell'applicazione





A screenshot of a file explorer interface with a dark theme. The 'components' folder is expanded, showing a list of JavaScript files. Each file is preceded by a 'JS' icon and followed by a status letter (M or U). The files 'MapComponent.js' and 'SingleMap.js' are highlighted in green, indicating they are the active or selected files.

components	
JS App.js	M
JS AuthMiddleware.js	M
JS Cities.js	M
JS Detail.js	M
JS Districts.js	M
JS getParseClasses.js	M
JS ListItems.js	M
JS LoginPage.js	M
JS Logout.js	M
JS MapComponent.js	U
JS MyMap.js	M
JS NavBar.js	M
JS Property.js	M
JS Regions.js	M
JS SearchBar.js	M
JS SideBar.js	M
JS SingleMap.js	U
JS Spinner.js	M
JS Town.js	M

Figura 4.4: componenti del progetto

e che ospiterà al suo interno tutti i componenti figli che andranno a comporre l'interfaccia utente.

– **reducers:**

Nella cartella *reducers* troviamo il file `index.js`, che contiene tutti gli stati globali dell'applicazione, e i file dei reducer che andranno a modificare ognuno il proprio stato.

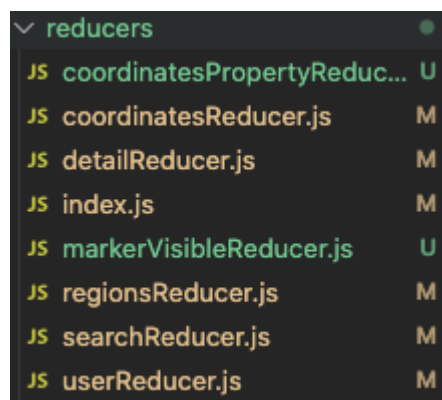


Figura 4.5: reducer del progetto

## 4.2 Dashboard

### 4.2.1 Login

Un utente registrato nel database può effettuare il login inserendo il nome utente e la password, negli appositi campi, come si può vedere in figura 4.6. Se le credenziali corrispondono ad un utente del database il login avviene con successo e l'utente potrà accedere alla dashboard e alle sue funzionalità.

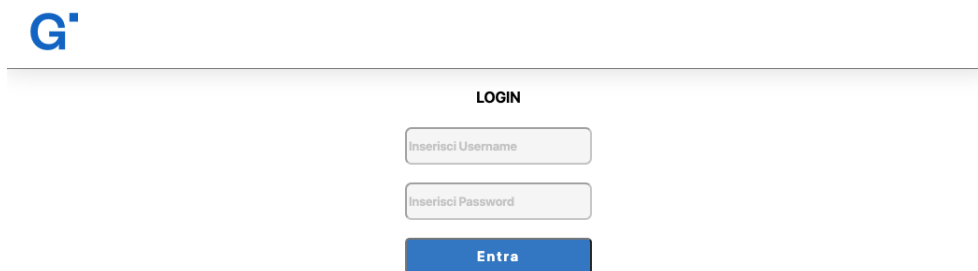


Figura 4.6: Schermata di login

### 4.2.2 Pagina iniziale

La pagina iniziale della dashboard viene mostrata in figura 4.7. In alto abbiamo la barra di ricerca, ovvero un componente di React, composta dal logo dell'azienda, una casella di testo dove inserire il luogo da ricercare, un bottone per avviare la ricerca, e le informazioni dell'utente.

Sulla sinistra invece si trova un altro componente che definisce la ricerca con filtri, composto da più caselle di testo per i filtri e dal bottone che avvia una ricerca.

Al centro troviamo un altro componente contenente una mappa, ottenuta tramite le API di google che andrà a mostrare graficamente gli immobili, e una lista che mostrerà i risultati all'esito della ricerca.

Nel file `App.js` mostrato in figura 4.8 si può apprezzare l'Atomic Design di React. Infatti si può vedere come con sole poche righe di codice si possa ottenere la schermata principale dell'applicazione. All'interno del file troviamo tre componenti: *NavBar*, *SideBar* e *MainContainer* che rispettivamente rappresentano nella schermata principale, la barra di ricerca in alto, la barra di ricerca laterale e la zona centrale con la mappa e i risultati.

Ogni componente può essere composto da altri componenti che

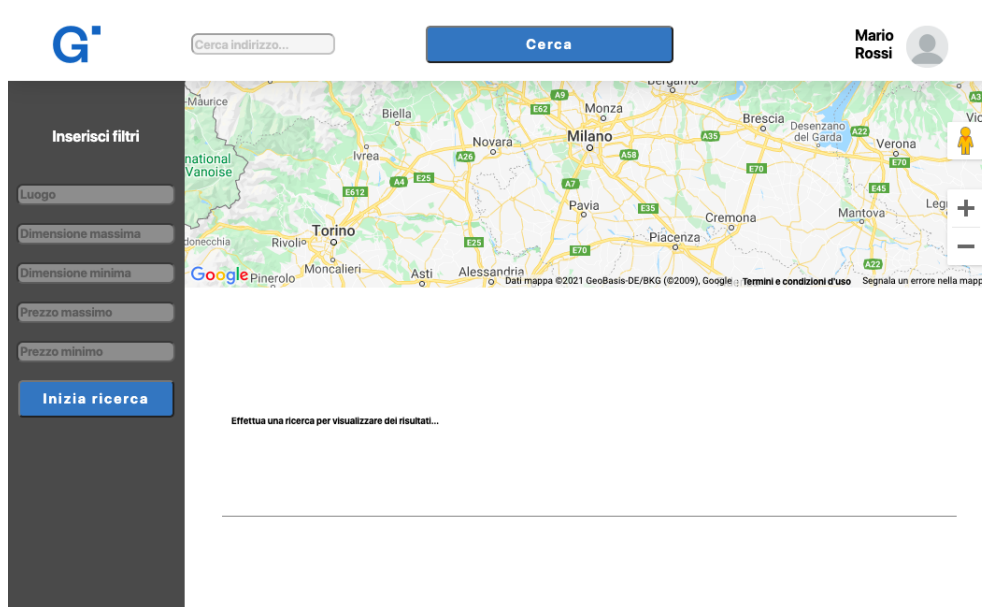


Figura 4.7: Schermata iniziale

a loro volta possono contenere ulteriori componenti eventuali.

### 4.2.3 Ricerca

In figura 4.9 si vede la dashboard a seguito di una ricerca semplice.

Per effettuarla l'utente deve inserire il luogo nel quale intende cercare gli immobili e premere il bottone *Cerca* o il tasto "invio" della tastiera. Una volta avviata la ricerca il sistema cercherà nel database tra gli immobili e presenterà una lista con qualche informazione di base sui risultati.

La mappa invece andrà a mostrare graficamente dove si trovano tutti gli immobili trovati nel database, rappresentandoli con il classico *balloon* di google.

In figura 4.10 viene mostrato come si comporta la dashboard quando l'utente posiziona il mouse su un elemento della lista immobili che si ottiene dopo una ricerca. L'elemento della li-

```


const App = () => {
  let online = window.navigator.onLine;
  return (
    <Fragment>
      <GlobalStyle />
      {online ? (
        <BrowserRouter>
          <AuthMiddleware>
            <NavBar />
            <SideBar />

            <Switch>
              <Route path="/login" exact component={LoginPage} />
              <MainContainer>
                <Route path="/" component={MapComponent} />
                <Route path="/" component={ListItems} />
              </MainContainer>
            </Switch>
          </AuthMiddleware>
        </BrowserRouter>
      ) : (
        <OfflineContainer>
          <OfflineExpression>connessione assente</OfflineExpression>
        </OfflineContainer>
      )}
    </Fragment>
  );
};

export default App;


```

Figura 4.8: File App.js

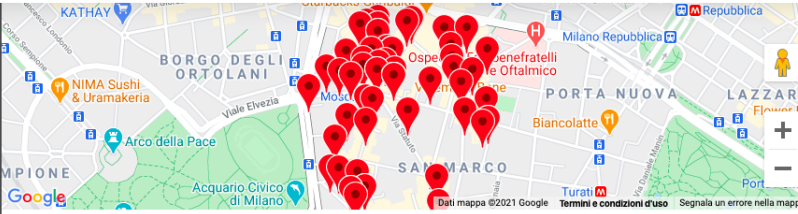




Mario Rossi



Inserisci filtri








	Indirizzo: Via della Moscova, 39, 20121 Milano MI, Italy	Prezzo: €2,300,000	Dimensione: 260 m <sup>2</sup>
	Indirizzo: Via della Moscova, 39, 20121 Milano MI, Italy	Prezzo: €540,000	Dimensione: 60 m <sup>2</sup>
	Indirizzo: Via della Moscova, 39, 20121 Milano MI, Italy	Prezzo: €895,000	Dimensione: 150 m <sup>2</sup>
	Indirizzo: Via della Moscova, 39, 20121 Milano MI, Italy	Prezzo: €2,300,000	Dimensione: 250 m <sup>2</sup>
	Indirizzo: Via della Moscova, 39, 20121 Milano MI, Italy	Prezzo: €895,000	Dimensione: 150 m <sup>2</sup>

Figura 4.9: Dashboard dopo una ricerca

sta viene evidenziato e nella mappa viene mostrata la singola posizione dell'immobile. Spostando il puntatore su un altro elemento esso verrà evidenziato, mentre quello precedente tornerà al colore base, e sulla mappa verrà mostrata la posizione del nuovo immobile. Per tornare a mostrare tutte le posizioni, come in figura 4.9, bisognerà spostare il puntatore del mouse fuori dalla lista immobili.

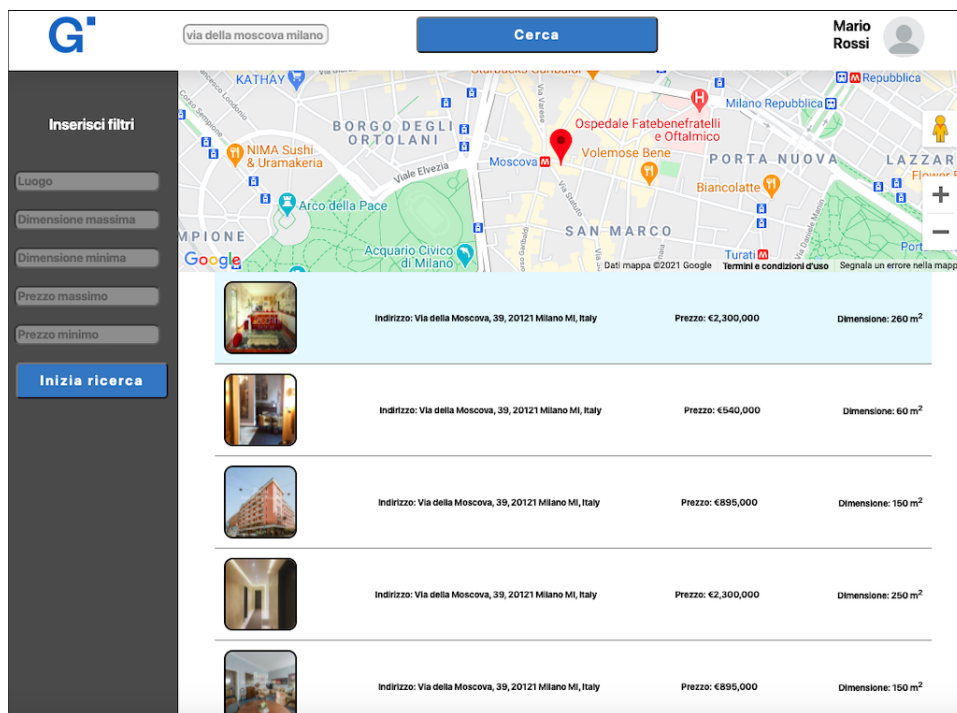


Figura 4.10: Mouse su un elemento della lista

#### 4.2.4 Dettaglio

Per poter visualizzare un singolo immobile e avere delle informazioni su di esso bisognerà selezionare un elemento dalla lista immobili e fare click con il tasto sinistro del mouse. A questo punto si aprirà una finestra sulla destra della dashboard che mostrerà le informazioni sull'immobile. Le informazioni fornite sono: l'indirizzo dell'immobile, il suo prezzo relativo e la sua

dimensione.

Viene inoltre fornita una statistica sulla zona OMI relativa all'immobile, che definisce il prezzo medio al metro quadro degli immobili di tale zona.

Infine, si troverà l'url della pagina web che mostra la casa in vendita.

The screenshot displays a real estate website interface. On the left, a sidebar titled "Inserisci filtri" (Add filters) includes input fields for "Luogo" (Location), "Dimensione massima" (Maximum size), "Dimensione minima" (Minimum size), "Prezzo massimo" (Maximum price), and "Prezzo minimo" (Minimum price), along with an "Inizia ricerca" (Start search) button. The main content area features a map of Milan with a red pin indicating the property location. Below the map, a large image shows the interior of a house. To the right of the image, the following details are listed:

- Indirizzo:** Via della Moscova, 39, 20121 Milano MI, Italy
- Prezzo:** €2,300,000
- Dimensione:** 260 m<sup>2</sup>
- OMI zone:** €7,400/m<sup>2</sup>

Below these details, a URL is provided: "URL: Via della Moscova, 39, 20121 Milano MI, Italy". At the bottom, there is a map view toggle with "Mappa" (Map) and "Satellite" options, and a larger map showing the surrounding area of Milan, including landmarks like the Ospedale Fatebenefratelli e Oftalmico and the Volemosse Bene building.

Figura 4.11: Finestra che mostra le informazioni di un immobile nel dettaglio



## Capitolo 5

### Conclusioni e sviluppi futuri

Lo sviluppo e la realizzazione della "Dashboard Estate" rappresenta un percorso di autoapprendimento che ha portato alla creazione di una Web-application che possa fornire una panoramica sul mercato immobiliare attraverso un'interfaccia utente intuitiva e semplice, come desiderato nella fase introduttiva della tesi.

Per quanto riguarda gli sviluppi futuri del software varie sono le migliorie che si possono apportare. Ad esempio si è proposto all'azienda di implementare un'ulteriore schermata dedicata ad altre statistiche riportate sotto forma di grafici o liste, così da poter analizzare al meglio quando un immobile sia conveniente o meno.

Inoltre si potrebbe estendere l'utilizzo della dashboard anche ad utenti non registrati fornendo loro un piano "base" da cui sono escluse funzioni più specifiche.

# Bibliografia

- [1] Luca Merelli. “Analisi delle performance dei database non-relazionali: il caso di studio di MongoDB”. In: (2012).
- [2] *www.JSON.org*.
- [3] *<https://it.wikipedia.org/wiki/JavaScript>*.
- [4] Antik Kumar. “React vs Angular vs Vue. js: A Complete Comparison Guide”. In: (2018).
- [5] Davide Cerbo. “Redux in parole semplici”. In: (2018).
- [6] *www.uml.org*.