Documentazione del Progetto di Software Architecture Design

GOING

Video demo

Maraniello Matteo M63/001110
Mennillo Valerio M63/001277
Orlando Gian Marco M63/001326
Perillo Marco M63/001313
Russo Diego M63/001335

Indice

Avvio del progetto **5**
1.1 Glossario 6
1.2 Parti interessate: gli utenti dell'applicazione 7
1.2.1 Utenti Non Registrati 7
1.2.2 Utenti Registrati 7
1.2.3 Utenti Autenticati 7
1.2.4 Utenti Amministratori 8
1.3 Tabella Attori-Obiettivi 8
1.4 Storie Utente 9
1.5 Requisiti generali 12
1.6 Requisiti di qualità 12
1.6.1 Usabilità (usability) 13
1.6.2 Modificabilità (modifiability) 14
1.6.3 Testabilità (testability) 15
1.6.4 Sicurezza (security) 16
1.7 Vincoli 18
1.8 Stima dei costi 18
1.9 System Context Diagram 23

Processo di sviluppo **25**
2.1 UP (Unified Process) 26
2.2 eXtreme Programming 27
2.3 Scrum 29
2.4 Tool utilizzati per la condivisione del lavori 30
2.4.1 Discord 30
2.4.2 Google Drive e Google Docs 30
2.4.3 Trello 31
2.4.4 GitHub 32
2.5 Tool e tecnologie utilizzate per lo sviluppo 33
2.5.1 Visual Paradigm 33
2.5.2 Visual Studio Code 33
2.5.3 Node.js + Express 34
2.5.4 Google Maps API 34
2.5.5 MySQL Workbench 35
2.5.6 Postman 36
2.5.7 Microsoft Azure 36
2.5.8 Photoshop 37

Fase di Analisi **38**
3.1 Analisi e Specifica dei Requisiti 40
3.1.1 Analisi Nomi-Verbi 40
3.1.2 Revisione dei Requisiti 42
3.1.3 Classificazione dei Requisiti 44

3.2 Modello dei Casi d'Uso 47
3.2.1 Attori e Casi d'Uso 47
3.2.2 Diagramma dei Casi d'Uso 49
3.2.3 Scenari dei Casi d'uso 50
3.3 Diagramma delle Classi 57
3.4 System Sequence Diagram 59
3.4.1 Ricerca di una Roadmap 59
3.4.2 Registrazione Utente 60
3.4.3 Visualizzazione di una Roadmap tramite selezione diretta 61
3.4.4 Visualizzazione di una Roadmap tramite ricerca 61
3.4.5 Creazione di una Roadmap 62
3.4.6 Gestione Segnalazione - Ban Utente Registrato 63

Architettura e Progettazione 64
4.1 Pattern Architetturali 64
4.2 Stili Architetturali 65
4.3 Stile Architetturale Client-Server 65
4.4 Pattern Server-Side MVC 66
4.5 Vista Componenti e Connettori 69
4.6 Dinamica dei componenti 71
4.6.1 Ricerca di una Roadmap 72
4.6.2 Registrazione Utente 73
4.6.3 Visualizzazione di una Roadmap tramite ricerca 75
4.6.4 Creazione di una Roadmap 76
4.6.5 Gestione Segnalazione - Ban Utente Registrato 78
4.7 Diagramma delle Classi 78
4.8 Context Diagram with Boundary 79

Implementazione 81
5.1 Organizzazione del codice 81
5.2 Descrizione dei file realizzati 86
5.2.1 View 86
5.2.2 Controller 91
5.2.3 Model 99
5.3 Design Pattern 99
5.3.1 Pattern Façade 100
5.3.2 Pattern Reactor 100
5.4 Asynchronous JavaScript and XML (AJAX) 101
5.5 Google Maps API Key 102
5.6 Diagramma Entità Relazione (ER) 103
5.7 Manuale di utilizzo 105
5.7.1 Configurazione ed installazione 105
5.7.2 Utilizzo del sito come Utente Non Registrato 106
5.7.3 Utilizzo del sito come Utente Registrato 112
5.7.4 Utilizzo del sito come Utente Autenticato 112
5.7.5 Utilizzo del sito come Utente Amministratore 117

5.8 Video demo 118

Testing **119**

6.1 Descrizione Test Suite 122

6.1.1 User 122

6.1.2 User Autenticato 123

6.1.3 Admin 124

6.1.4 Map 125

6.1.5 Content 126

6.1.6 SendView 127

Capitolo 1

Avvio del progetto

Si vuole realizzare una web app denominata *GOING* in cui gli utenti possano organizzare un proprio itinerario di viaggio (detto Roadmap) al fine di visitare una città. Ogni Roadmap è costituita da un insieme di punti di interesse.

Gli utenti potranno cercare e visualizzare le Roadmap pubbliche create da altri utenti e/o cercare e visualizzare il profilo di un altro utente registrato. Le Roadmap cercate saranno filtrate in base ad una serie di parametri tra cui - i più importanti - la città che si andrà a visitare, il tempo di permanenza in quella città ed eventuali altri interessi evidenziati dall'utente. Inoltre, l'applicazione dovrà permettere agli utenti di potersi registrare al servizio. Gli utenti registrati potranno eseguire tutte le azioni di un utente non registrato ed in più potranno:

- creare le proprie Roadmap (attribuendone ad ognuna la visibilità pubblica o privata) costituite da punti di interesse preesistenti nell'applicazione e/o da punti di interesse aggiunti ex novo dall'utente, con la possibilità di crearle anche a partire da una Roadmap pubblica già esistente;
- mantenere uno storico delle Roadmap seguite durante i propri viaggi;
- aggiungere delle Roadmap ai "preferiti";
- recensire una Roadmap tramite una valutazione numerica e/o una opinione;
- eliminare una propria Roadmap, una propria valutazione e/o la propria opinione;
- segnalare altri utenti, Roadmap, punti di interesse, recensioni e/o commenti.

Ad ogni Roadmap potrà essere assegnato un punteggio, risultato delle votazioni degli utenti.

La community viene gestita da utenti amministratori che potranno eliminare Roadmap, commenti e/o valutazioni di altri utenti. Inoltre, gli amministratori potranno anche bandire altri utenti, causando in questo modo l'eliminazione automatica delle Roadmap, commenti e valutazioni associate all'utente eliminato.

Opzionalmente, infine, il sistema potrà fornire delle caratteristiche di gamification basate su badge, titoli e achievement che gli utenti registrati potranno ottenere in base all'utilizzo della piattaforma ed ai punteggi assegnati ad ogni Roadmap creata dall'utente.

L'applicazione dovrà garantire ai clienti un alto grado di usabilità. Infatti, durante l'interazione con il prodotto, i clienti dovranno percepire un elevato livello di soddisfazione e gradimento grazie ad una grafica accattivante e ad un'interfaccia "user-friendly".
Inoltre, l'applicazione è pensata per evolversi nel tempo, agevolando l'aggiunta e la modifica di feature in tempi brevi. Infine, nonostante l'applicazione non debba necessariamente mantenere dati sensibili, si dovrà garantire un buon livello di sicurezza.

1.1 Glossario

La documentazione conterrà termini i cui significati potrebbero risultare ignoti o ambigui. Per risolvere ciò compiliamo un glossario, allegato di seguito, che descrive brevemente i suddetti termini al fine di risolvere qualsiasi ambiguità.

Termine	Alias	Descrizione
GOING		Nome dell'applicazione
Stage	POI	Nome usato per definire un punto d'interesse. Tale punto di interesse può essere creato ex-novo oppure già presente
Roadmap	Itinerario di Viaggio	Nome usato per definire un insieme degli Stage
Roadmap seguita		Termine usato per definire una Roadmap contrassegnata da un utente come svolta durante i propri viaggi
Roadmap forkata		Termine usato per definire la possibilità di creare una nuova Roadmap a partire da un'altra Roadmap già esistente
Valutazione		Nome usato per definire un indice di gradimento (da 1 a 5) che l'utente può associare ad una Roadmap pubblica
Recensione		Nome usato per definire la coppia formata da una votazione ed una eventuale opinione testuale
Opinione		Nome usato per definire una nota testuale rilasciata da un utente ad una particolare Roadmap all'atto di una recensione

1.2 Parti interessate: gli utenti dell'applicazione

Gli utenti che utilizzeranno l'applicazione si divideranno in Utenti Non Registrati, Utenti Registrati, Utenti Autenticati ed Amministratori. Ne riportiamo di seguito una breve descrizione di ognuno di essi e delle azioni a loro associate.

1.2.1 Utenti Non Registrati

L'applicazione potrà essere utilizzata anche senza necessità di registrazione. L'utilizzo dell'applicazione senza registrazione sarà costituito da una serie di limitazioni dovute alla mancata profilazione degli utenti stessi. In particolare, un Utente Non Registrato potrà effettuare ricerche delle *Roadmap* pubbliche filtrando per una serie di parametri quali città da visitare, il tempo di permanenza in una specifica città ed interessi. Un Utente Non Registrato potrà cercare e visualizzare anche il profilo pubblico degli utenti registrati. Un Utente Non Registrato, inoltre, avrà accesso alla schermata che consente di effettuare la registrazione. Una volta effettuata la registrazione l'utente diventerà un Utente Registrato e potrà effettuare il login tramite nome utente e password.

1.2.2 Utenti Registrati

Gli Utenti Registrati potranno accedere all'applicazione tramite un processo di autenticazione (login) diventando così un Utente Autenticato.

1.2.3 Utenti Autenticati

Gli Utenti Autenticati potranno, a valle di un login, accedere ai servizi offerti dall'applicazione senza limitazioni. Oltre a poter eseguire tutte le azioni di un Utente Non Registrato già descritte in precedenza nell'apposito paragrafo, gli Utenti Autenticati potranno creare *Roadmap* ex novo oppure partendo da *Roadmap* già esistenti ed eventualmente effettuando una o più modifiche e decidere se ri-pubblicare tale nuova *Roadmap*. Le *Roadmap* saranno costituite da punti di interesse già presenti nell'applicazione e/o da punti di interesse aggiunti ex novo dall'utente stesso. Gli Utenti Autenticati potranno contrassegnare una *Roadmap* come "seguita" in modo da poter andare a costituire uno storico di tutte le *Roadmap* utilizzate dall'utente durante i propri viaggi. Gli Utenti Autenticati potranno, inoltre, aggiungere ai "preferiti" le *Roadmap* di altri utenti. Infine, gli Utenti Autenticati

potranno cercare, valutare e commentare una *Roadmap*, eliminare una propria *Roadmap*, modificare o eliminare un proprio commento e/o la propria valutazione numerica assegnata ad altre *Roadmap* e segnalare altri utenti, *Roadmap*, punti di interesse, e/o commenti.
In base all'attività svolta sull'applicazione, un Utente Autenticato sarà in grado di guadagnare una serie di trofei e titoli visibili sul proprio profilo.

1.2.4 Utenti Amministratori

Gli stakeholder possono rendere Amministratori gli Utenti Registrati. L'Amministratore, a valle del login, potrà accedere ai servizi offerti dall'applicazione senza limitazioni come un qualsiasi Utente Autenticato. Inoltre, un Amministratore monitorerà la community e ne controllerà il rispetto delle regole generali di buon senso. Tramite la propria interfaccia, l'amministratore potrà eliminare *Roadmap*, bandire utenti e/o cancellare commenti di altri utenti. Inoltre, l'Amministratore avrà accesso ad una specifica sezione nell'interfaccia dove potrà gestire le varie segnalazioni arrivate da parte degli altri utenti.

1.3 Tabella Attori-Obiettivi

Di seguito definiamo la cosiddetta Tabella Attori-Obiettivi. All'interno di tale tabella evidenziamo gli Attori principali del sistema e gli Obiettivi che ognuno di questi Attori vorrà perseguire e che il sistema dovrà supportare. Ognuno di questi Obiettivi potrà costituire poi un Caso d'Uso.

Attore	Obiettivo
Utente Non Registrato	Effettuare la registrazione; Cercare una *Roadmap*; Visualizzare una *Roadmap* pubblica; Cercare un Utente Registrato; Visualizzare il profilo pubblico di un Utente Registrato;
Utente Registrato	*Obiettivi di un Utente Non Registrato (eccetto registrazione);* Effettuare il login;
Utente Autenticato	*Obiettivi di un Utente Non Registrato (eccetto registrazione);* Visualizzare le proprie *Roadmap* private;

	Creare una *Roadmap*; Creare una variante di una *Roadmap* pubblica già esistente; Eliminare una propria *Roadmap*; Segnalare una *Roadmap*; Aggiungere *Roadmap* allo storico delle *Roadmap* seguite; Eliminare *Roadmap* dallo storico delle *Roadmap* seguite; Aggiungere *Roadmap* ai preferiti; Rimuovere *Roadmap* dai preferiti; Aggiungere uno *Stage* ex novo; Segnalare uno *Stage*; Segnalare un Utente Registrato; Aggiungere una recensione ad una *Roadmap* di un altro utente; Eliminare la propria recensione ad una *Roadmap* di un altro utente; Modificare la propria recensione ad una *Roadmap* di un altro utente; Aggiungere un commento ad una *Roadmap*; Eliminare il proprio commento ad una *Roadmap*; Modificare il proprio commento ad una *Roadmap*; Effettuare il logout;
Amministratore	*Obiettivi di un Utente Autenticato;* Gestione segnalazioni; Visualizza lista di segnalazioni;

1.4 Storie Utente

Le Storie Utente descrivono in maniera semplice una funzionalità o un requisito di interesse per un utente del sistema software. Di seguito riportiamo una serie di Storie Utente realizzate per il nostro sistema che abbiamo ritenuto essere di valore per un particolare utente:

Registrazione Utente
Come Utente Non Registrato, voglio registrarmi all'applicazione in modo da poter successivamente accedere all'applicazione stessa.
Test di Accettazione 1 Considerando che l'email sia valida e che il nome utente non sia già stato utilizzato, quando l'utente conferma l'inserimento dei dati, allora il sistema deve garantire che l'utente venga correttamente registrato.
Test di Accettazione 2 Considerando che l'email sia non valida e/o che il nome utente sia già stato utilizzato, quando l'utente conferma l'inserimento dei dati, allora il sistema deve rifiutare la richiesta di registrazione dell'utente.

Autenticazione Utente
Come Utente Registrato, voglio autenticarmi nell'applicazione in modo da poter usufruire dei servizi dell'applicazione.
Test di Accettazione 1 Considerando che i dati inseriti per il login (username e password) corrispondono a uno di quelli già registrati, quando l'utente conferma l'inserimento di username e password, allora il sistema riconosce l'utente come Utente Autenticato.
Test di Accettazione 2 Considerando che i dati inseriti per il login (username e password) non corrispondono uno di quelli già registrati, quando l'utente conferma l'inserimento di username e password, allora il sistema non riconosce l'utente come Utente Autenticato e lo notifica della mancata autenticazione.

Creazione Roadmap
Come Utente Autenticato, voglio poter creare *Roadmap*.
Test di Accettazione 1 Considerando che gli *Stage* inseriti nella *Roadmap* siano almeno due, quando l'utente conferma l'inserimento dei dati, allora il sistema dovrà consentire la visualizzazione della *Roadmap* in maniera pubblica o privata in base alla visibilità impostata dall'utente.
Test di Accettazione 2 Considerando che gli *Stage* inseriti nella *Roadmap* non siano almeno due, quando l'utente conferma l'inserimento dei dati, allora il sistema non dovrà consentire la creazione di tale *Roadmap*.

Creazione variante Roadmap
Come Utente Autenticato, voglio poter creare una *Roadmap* a partire da una *Roadmap* già esistente.
Test di Accettazione 1 Considerando che venga inserito e/o modificato e/o eliminato almeno uno *Stage* dalla *Roadmap* già esistente e che la nuova *Roadmap* sia costituita da almeno due *Stage*, allora il sistema dovrà consentire la visualizzazione della *Roadmap* in maniera pubblica o privata in base alla visibilità impostata dall'utente.
Test di Accettazione 2 Considerando che non venga inserito e/o modificato e/o eliminato almeno uno *Stage* dalla *Roadmap* già esistente e/o che la nuova *Roadmap* non sia costituita da almeno due *Stage*, allora il sistema non dovrà consentire la creazione di tale *Roadmap*.

Visualizzazione Roadmap private
Come Utente Autenticato, voglio poter visualizzare le mie *Roadmap* private.
Test di Accettazione 1 Considerando che l'utente si sia correttamente autenticato, il sistema consente all'utente di visualizzare le proprie *Roadmap* private.

Aggiunta Roadmap allo storico delle Roadmap
Come Utente Autenticato, voglio aggiungere una *roadmap* allo storico delle *Roadmap* seguite.
Test di Accettazione 1 Considerando che l'utente si sia correttamente autenticato e che la *Roadmap* non sia presente nello storico delle *Roadmap* seguite, quando l'utente conferma l'inserimento di tale *Roadmap* nello storico, il sistema aggiunge la *Roadmap* allo storico delle *Roadmap* seguite.
Test di Accettazione 2 Considerando che l'utente si sia correttamente autenticato e che la *Roadmap* sia già presente nello storico delle *Roadmap* seguite, quando l'utente conferma l'inserimento di tale *Roadmap* nello storico, il sistema riconosce che la *Roadmap* è già presente nello storico e quindi non altera lo storico.

Aggiunta di uno Stage ex novo
Come Utente Autenticato, voglio aggiungere uno *Stage* ex novo.
Test di Accettazione 1 Considerando che l'utente si sia correttamente autenticato e che l'utente abbia descritto lo *Stage* con almeno un titolo e una posizione geografica allora il sistema aggiunge lo *Stage* alla lista degli *Stage* pubblicamente disponibili e quindi è adesso utilizzabile nelle successive *Roadmap* di tutti gli utenti.

1.5 Requisiti generali

1. L'applicazione deve consentire all'Utente di poter cercare e visualizzare una *Roadmap* specificando il luogo da visitare;
2. L'applicazione deve consentire all'Utente di poter visualizzare il profilo di un altro Utente Registrato in modo da visualizzare le sue *Roadmap* pubbliche, separate ;
3. L'applicazione deve consentire all'Utente Autenticato di poter creare una Roadmap ex novo oppure partendo da una *Roadmap* pre-esistente;
4. Durante il processo di creazione di una *Roadmap*, l'applicazione deve consentire all'Utente Autenticato di costruire una *Roadmap* utilizzando *Stage* già presenti nell'applicazione oppure utilizzando *Stage* creati appositamente dall'Utente Autenticato;
5. L'applicazione deve consentire all'Utente Autenticato di tener traccia delle *Roadmap* contrassegnate come *"Roadmap seguita"*;
6. L'applicazione deve consentire all'Utente Autenticato di *recensire* una *Roadmap* pubblica;
7. L'applicazione deve consentire all'Amministratore di gestire la community eliminando *Roadmap*, commenti o bandire Utenti.

1.6 Requisiti di qualità

Un attributo di qualità è una proprietà misurabile e testabile di un sistema che viene usato per indicare come il sistema è in grado di soddisfare le necessità degli stakeholders oltre alle funzionalità base del sistema. E' possibile definire un attributo di qualità attraverso gli

scenari di qualità. Uno scenario di qualità è un template adottato per stabilire i diversi attributi posseduti da un'architettura. Uno scenario di un attributo di qualità è diviso in sei parti:

1. **Stimulus**, è un evento che sollecita il sistema;
2. **Stimulus source**, è la sorgente da cui proviene entità che genera lo stimolo;
3. **Response**, è il risultato del comportamento del sistema dovuto allo stimolo;
4. **Response measure**, è la metrica utilizzata per valutare la "response";
5. **Environment**, è l'ambiente in cui il sistema è immerso nel momento in cui lo scenario avviene;
6. **Artifact**, è il destinatario dello stimolo;

Verranno di seguito descritti i vari requisiti di qualità scelti per adempiere alle esigenze degli stakeholders facendo riferimento al modello ISO/IEC 9126.

1.6.1 Usabilità (usability)

Per usabilità si intende la capacità del prodotto software di essere capito, appreso, usato e gradito all'utente, quando utilizzato sotto condizioni specificate. L'usabilità comprende diverse sotto-caratteristiche, tra cui riportiamo di seguito quelle di maggiore interesse per l'applicazione:

- **Comprensibilità** (understandability), capacità del prodotto software di mettere in grado l'utente di comprendere se il software è appropriato e come esso possa essere usato per particolari compiti e condizioni d'uso;
- **Apprendibilità** (learnability), capacità del prodotto software di permettere all'utente di apprenderne il corretto utilizzo;
- **Attrattività** (attractiveness), capacità del prodotto software di essere attraente per l'utente (ossia piacevole nell'utilizzo).

Di seguito si mostra lo scenario in cui un utente richiede l'annullamento dell'operazione che sta effettuando. Il sistema annullerà l'operazione con successo entro 1 secondo.

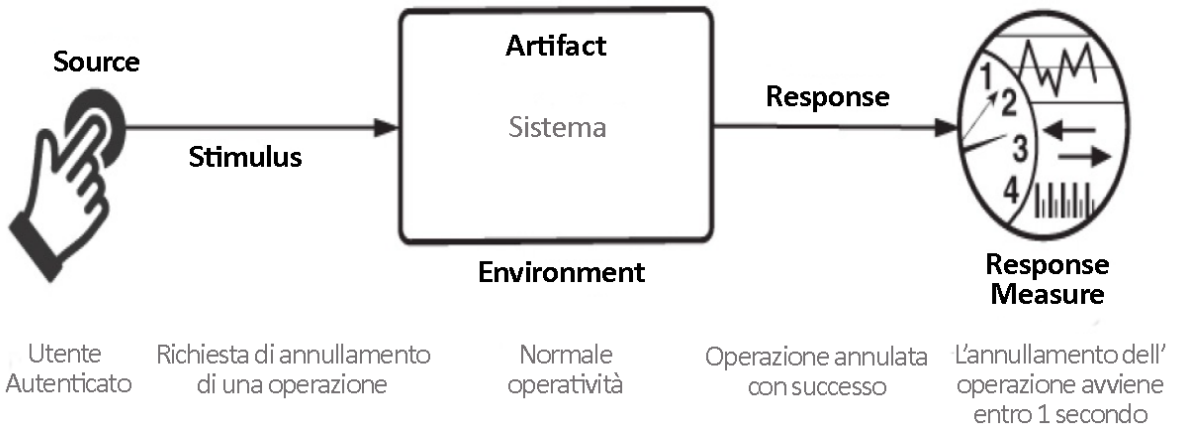


Figura 1.1 - Scenario Usabilità

L'usabilità è un requisito fondamentale per questo software in quanto l'utente deve essere in grado di eseguire facilmente tutte le operazioni messe a disposizione dall'applicazione allo scopo di agevolarne il più possibile l'utilizzo.

1.6.2 Modificabilità (modifiability)

La modificabilità è la caratteristica di un prodotto o un sistema di poter essere modificato in modo efficace ed efficiente senza introdurre difetti o degradarne la qualità.
La modificabilità comprende diverse sotto-caratteristiche, tra cui riportiamo di seguito quelle di maggiore interesse per l'applicazione:

- **Scalabilità** (scalability), capacità di un sistema di aumentare o diminuire di scala in funzione delle necessità e disponibilità. Si parla di scalabilità orizzontale se si allocano più risorse alle unità logiche. Si parla di scalabilità verticale se si allocano più risorse ad un'unità fisica.
- **Portabilità** (portability), facilità con cui il software che è stato creato per essere eseguito su una piattaforma, può essere modificato per essere eseguito su una piattaforma diversa. Si ottiene riducendo al minimo le dipendenze dalla piattaforma nel software ed isolando le dipendenze in posizioni ben identificate.

Al fine di ottenere un buon livello di modificabilità bisogna:

- Aumentare la coesione tra i moduli (cohesion), per coesione si intende la capacità del modulo di realizzare le responsabilità richieste senza aiuti esterni di altri moduli. Al diminuire della coesione, la probabilità che la modifica di un modulo abbia ripercussioni sugli altri sarà alta.

- Riduzione dell'accoppiamento tra i moduli (coupling), per accoppiamento si intende il grado con cui ciascun modulo di un software dipende dagli altri moduli. Se le responsabilità di due moduli si sovrappongono, allora una singola modifica potrebbe avere ripercussioni su entrambi.

Di seguito si mostra uno scenario in cui lo sviluppatore aggiunge una feature al sistema. Il sistema, entro 24 ore, introdurrà la nuova feature senza alcuna anomalia.

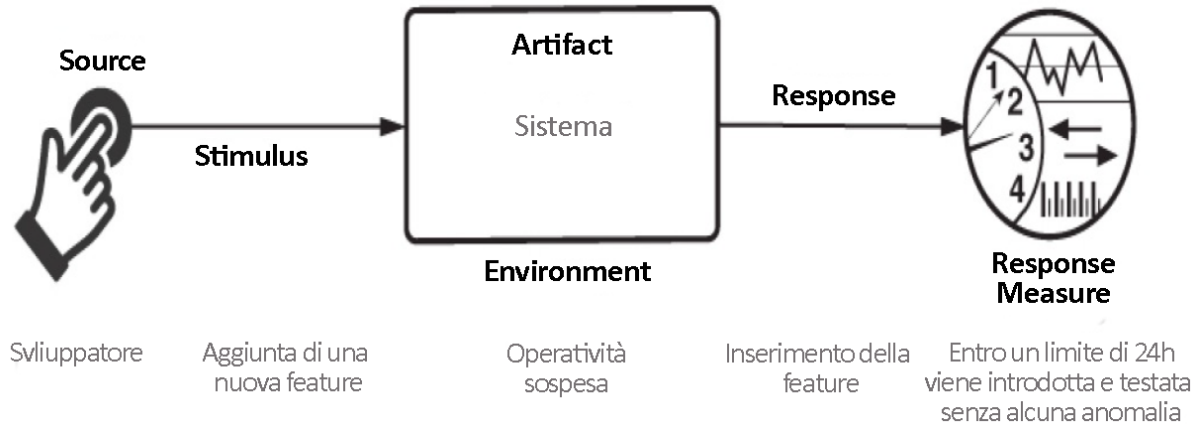


Figura 1.2 - Scenario Modificabilità

E' stato preso in considerazione il requisito di qualità di modificabilità in quanto il software è stato pensato per essere modificato ed ampliato in futuro con l'aggiunta di nuove feature.

1.6.3 Testabilità (testability)

La testabilità è la capacità del software di essere facilmente testato per validarne le modifiche. La testabilità è composta da vari fattori quali:

- **Controllabilità**, capacità del sistema di controllare lo stato del componente sottoposto a testing;
- **Osservabilità**, capacità del sistema di osservare i risultati di un test (intermedio e finale);
- **Isolabilità**, capacità del componente di essere testato in isolamento;
- **Separazione delle responsabilità**, capacità del componente sottoposto al testing di avere una responsabilità unica e ben definita;
- **Comprensibilità**, quanto il componente sottoposto a testing è documentato e/o autoesplicativo;

- **Automatizzabilità**, quanto è possibile automatizzare il testing di un componente.

Se la testabilità dell'artefatto software è elevata, è più facile trovare guasti nel sistema mediante testing. Si riporta di seguito uno scenario per il testing di unità.

Un tester di unità stimola un componente del sistema tramite test di unità. Il sistema risponderà con l'output osservabile del componente e, per misurare la risposta, si confronta l'output del componente sotto test con i risultati che ci si aspettavano dal testing. La percentuale di coverage dei test effettuati deve essere almeno del 80%.

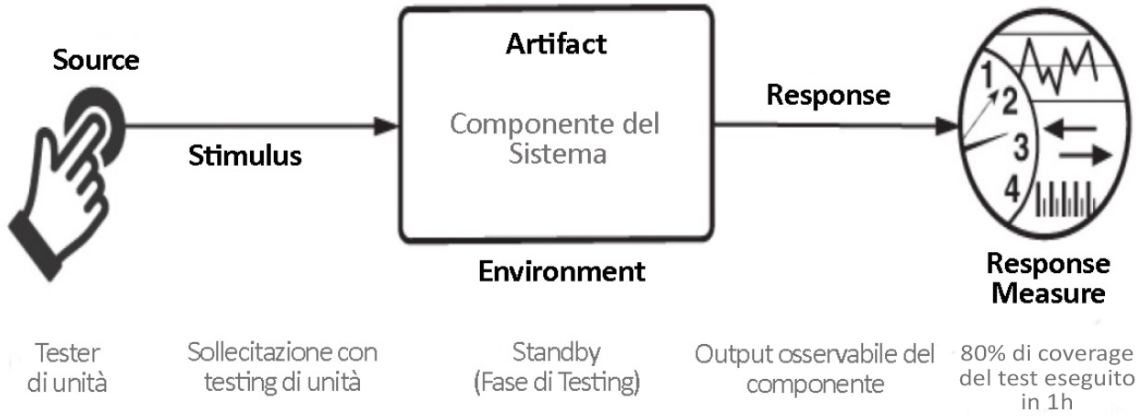


Figura 1.3 - Scenario Testabilità

E' stato preso in considerazione come requisito di qualità la testabilità perché il software è pensato per essere continuamente aggiornato. Un'elevata testabilità rende il sistema più reattivo ai cambiamenti.

1.6.4 Sicurezza (security)

La sicurezza è la misura dell'abilità di un sistema di proteggere i dati e le informazioni da accessi non autorizzati fornendo l'accesso solo a persone e sistemi con autorizzazione. La sicurezza include tre proprietà:

1. **Riservatezza** (confidentiality), dati e servizi sono protetti dagli accessi non autorizzati;
2. **Integrità** (integrity), dati e servizi non possono essere manipolati da accessi fraudolenti;
3. **Disponibilità** (availability), è la proprietà di un sistema di essere accessibile durante la sua normale operatività.

Lo scenario in figura prevede il tentativo da parte di un utente non autenticato di creare una

Roadmap. Se l'autenticazione non è avvenuta con successo allora il sistema ignora la richiesta.

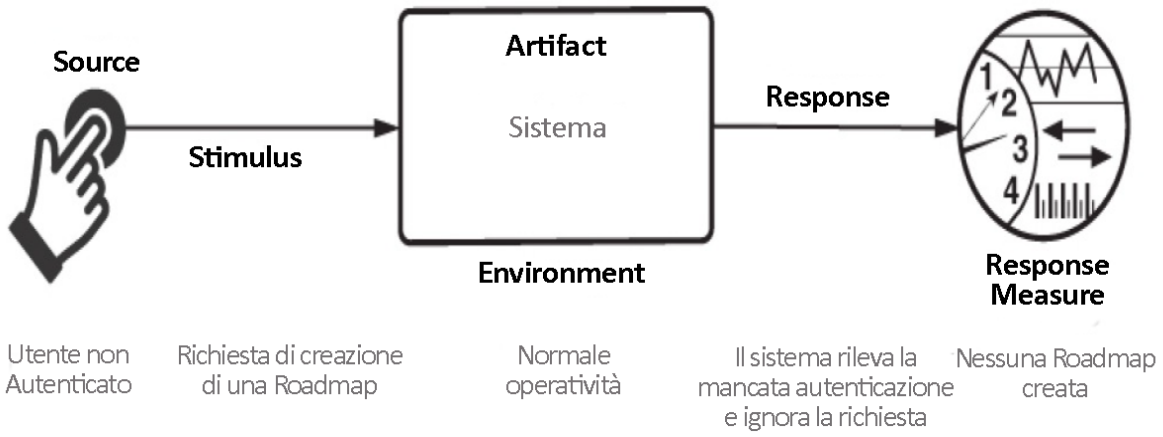


Figura 1.4 - Primo Scenario Sicurezza

Il successivo scenario prevede il tentativo da parte di un Utente Autenticato A di eliminare una *Roadmap* di un Utente B. Il sistema rileva i mancati permessi dell'Utente Autenticato A e ignora la richiesta.

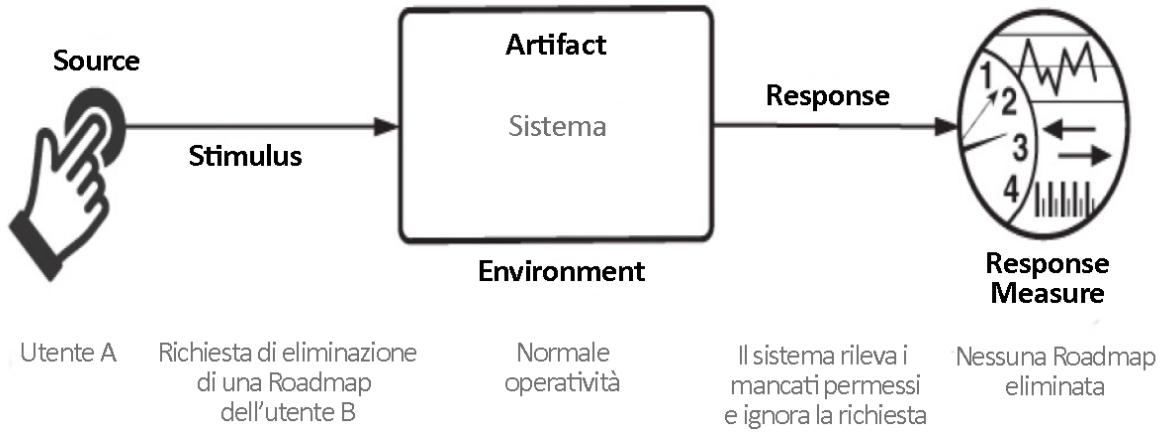


Figura 1.5 - Secondo Scenario Sicurezza

Nonostante l'applicazione non raccolga dati sensibili dell'utente, non riguardi beni economici e non mantenga file di particolare riservatezza, si è deciso comunque di scegliere la sicurezza come requisito di qualità.

Ogni Utente può effettuare particolari operazioni solo se autenticato. Qualora l'Utente non sia autenticato, il sistema deve respingere le richieste fatte in malafede, così come descritto

dal primo scenario mostrato. Inoltre, solo gli Amministratori e il creatore originario di una Roadmap possono eliminarla. C'è bisogno quindi di una gestione di permessi robusta ed efficiente. Considerando il valore delle informazioni conservate, aggiungere ulteriori specifiche di sicurezza andrebbe solo ad appesantire ulteriormente il software.

1.7 Vincoli

Per poter avere completo accesso a tutte le funzionalità dell'applicazione, l'utente deve registrarsi al servizio.
L'applicazione utilizzerà le API fornite da Google relative al servizio di mappe Google Maps e, per questo, non potrà essere utilizzato su browser Internet Explorer.
L'applicazione deve essere sviluppata rispettando le regole imposte dal GDPR (General Data Protection Regulation) per il rispetto della privacy dell'utente. I Terms of Service e la Privacy Policy verranno letti ed accettati dall'utente in fase di registrazione e saranno disponibili in una apposita pagina del sito.

1.8 Stima dei costi

Unadjusted Use Case Weight (UUCW)

Elenco dei casi d'uso con complessità stimata associata:

Caso d'uso	Complessità
Effettuare la registrazione	Average
Cercare una *Roadmap*	Simple
Visualizzare una *Roadmap* pubblica	Average
Cercare un Utente Registrato	Simple
Visualizzare il profilo pubblico di un Utente Registrato	Simple
Effettuare il login	Simple
Visualizzare le proprie *Roadmap* private	Average
Creare una *Roadmap*	Complex
Creare una variante di una *Roadmap* pubblica già esistente	Complex

Eliminare una propria *Roadmap*	Simple
Segnalare una *Roadmap*	Simple
Aggiungere *Roadmap* allo storico delle *Roadmap* seguite	Simple
Eliminare *Roadmap* dallo storico delle *Roadmap* seguite	Simple
Aggiungere *Roadmap* ai preferiti	Simple
Rimuovere *Roadmap* dai preferiti	Simple
Aggiungere uno *Stage* ex novo	Complex
Segnalare uno *Stage*	Simple
Segnalare un Utente Registrato	Simple
Aggiungere una recensione ad una *Roadmap* di un altro utente	Simple
Eliminare la propria recensione ad una *Roadmap* di un altro utente	Simple
Modificare la propria recensione ad una *Roadmap* di un altro utente	Simple
Aggiungere un commento ad una *Roadmap*	Simple
Eliminare il proprio commento ad una *Roadmap*	Simple
Modificare il proprio commento ad una *Roadmap*	Simple
Effettuare il logout	Simple
Eliminare una *Roadmap*	Simple
Eliminare un commento	Simple
Gestione Segnalazione	Average
Visualizza lista di segnalazioni	Simple

Tabella UUCW

Complessità	**Peso**	**Numero Casi d'uso**	**Prodotto**
Simple	5	22	110
Average	10	4	40
Complex	15	3	45
Totale (UUCW)			195

Unadjusted Actor Weight (UAW)

Tipi di attori

Tipo	Esempio	Peso
Simple	Sistema esterno tramite API	1
Average	Sistema esterno tramite protocollo	2
Complex	Persona tramite interfaccia grafica	3

Elenco attori con tipo stimato associato

Attore	Tipo
Utente Non Registrato	Complex
Utente Registrato	Complex
Utente Autenticato	Complex
Amministratore	Complex

Tabella UAW

Tipo	Peso	Numero Attori	Prodotto
Simple	1	0	0
Average	2	0	0
Complex	3	4	12
Totale (UAW)			12

Unadjusted Use-Case Points (UUCP)

$UUCP = UUCW + UAW = 195 + 12 = 207$

Valutazione dell'impatto dei Fattori di Complessità Tecnica (TFactor):

Fattore	Peso	Valutazione	Impatto
Sistema distribuito	2	2	4
Obiettivi di Performance	2	5	10
Efficienza end-user	1	4	4
Elaborazione Complessa	1	1	1
Codice riusabile	1	2	2
Facile da installare	0.5	5	2.5
Facile da usare	0.5	5	2.5
Portabile	2	3	6
Facile da modificare	1	3	3
Uso concorrente	1	4	4
Sicurezza	1	3	3
Accesso a terze parti	1	4	4
Necessità di addestramento	1	0	0
Totale (TFactor)			46

Technical Complexity Factor (TCF)

*TCF = 0.6 + (0.01 * TFactor)* = 1.06

Valutazione dell'impatto dei Fattori di Complessità dell'Ambiente (EFactor):

Fattore	Peso	Valutazione	Impatto
Familiare con il processo di sviluppo	1.5	1	1.5
Esperienza sull'applicazione	0.5	1	0.5
Esperienza sull'Object-Orientation	1	4	4
Capacità dell'analista	0.5	1	0.5
Motivazione	1	5	5

Requisiti stabili	2	3	6
Staff part-time	-1	0	0
Linguaggio di programmazione difficile	-1	3	-3
Totale (EFactor)			14.5

*EF = 1.4 + (-0.03 * EFactor)* = 0.965

Calcolo Finale degli Use Case Points (UCP):

*UCP = UUCP * TCF * EF* = 207 * 1.06 * 0.965 = 211.7403

Fattore	Descrizione	Peso
UUCW	Unadjusted Use Case Weight	195
UAW	Unadjusted Actor Weight	12
TCF	Technical Complexity Factor	1.06
EF	Environment Factor	0.965

Calcolo delle ore di lavoro totali:

*UCP = UCP * 15* = 3176.1045

Nota: supponendo che un singolo UC venga sviluppato in 15 ore complessive mediamente.

Numero iterazioni

Supponendo che ogni membro del team lavori per circa 60 ore a settimana (*Wh*), per 5 lavoratori otteniamo 300 ore a settimana (*TWh*). Se *Th* è il numero di ore di lavoro totali necessarie a terminare il progetto e *TWh* il numero di ore a settimana del gruppo di lavoro (team), allora il numero di settimane stimate necessarie per terminare il lavoro è dato da: $\frac{Th}{TWh} = 10,59$. Dunque, si stima che occorreranno circa 10 settimane e mezzo (ovvero 5 iterazioni da 2 settimane ciascuna) per sviluppare questo progetto nella sua interezza.

Nota: si sta supponendo che un lavoratore lavori 8 ore al giorno tutti i giorni della settimana, più qualche ora aggiuntiva la mattina o la sera durante la settimana.

1.9 System Context Diagram

Si vuole determinare il perimetro del sistema ovvero cosa il sistema comprende e cosa non comprende e con quali entità esterne comunica.

L'ambiente con cui il sistema (o una specifica parte del sistema) interagisce rappresenta il contesto. Il contesto può essere rappresentato usando un System Context Diagram.

Un System Context Diagram è un diagramma che definisce i confini tra il sistema e l'ambiente circostante mostrando le entità che interagiscono con esso. E' un diagramma che fornisce una visione ad alto livello del sistema ed è il più alto livello di astrazione che si può avere per un diagramma data flow.

Tale diagramma deve poter essere compreso dagli stakeholder in quanto si tratta di un primo strumento di confronto diretto con essi. Le frecce utilizzate per collegare entità ed interfacce rappresentano un flusso di dati, che veicolano gli input forniti dall'esterno dagli attori e gli output corrispondenti forniti dal sistema.

Purtroppo UML non ha costrutti di base per descrivere un Context Diagram perciò ci si avvale di Use Case Diagram e Class Diagram con opportuni stereotipi. Si utilizza, inoltre, il vocabolario delle viste allo scopo di rappresentare il contesto del sistema.

Di seguito si riporta il Diagramma di Contesto realizzato per l'applicazione:

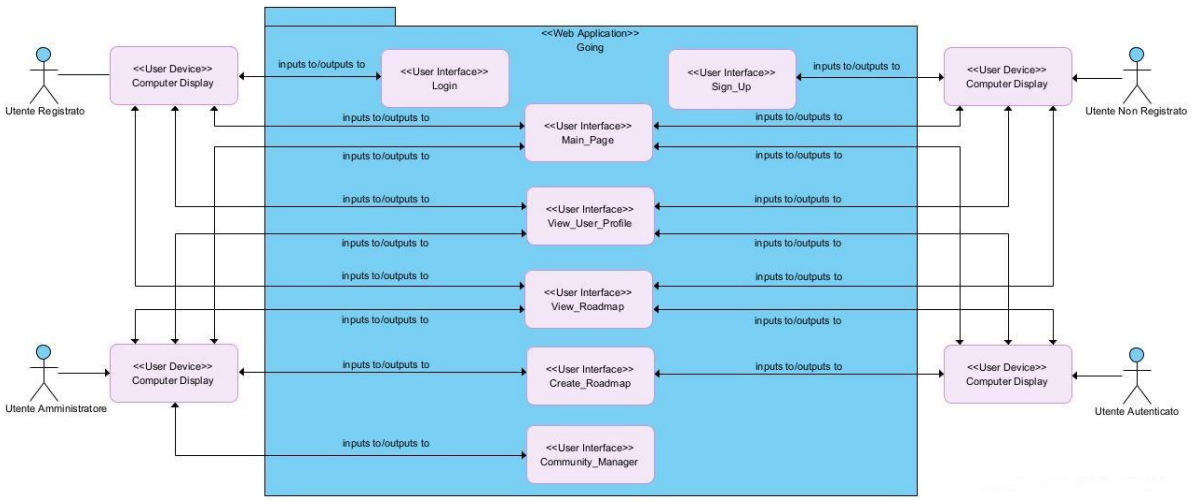


Figura 1.6 - Diagramma di Contesto

Chiunque può accedere alla *Main_Page,* dove si potranno ricercare Roadmap o Utenti, e di conseguenza essere indirizzati verso le due interfacce preposte, rispettivamente *View_Roadmap* e *View_User_Profile*.

L'*Utente Non Registrato* può fornire in input all'apposita interfaccia (*SignUp*) le informazioni con le quali intende registrarsi. Quest'ultima, in caso di corretta registrazione, mostrerà all'utente la pagina di *Login* attraverso la quale esso potrà effettuare l'accesso al servizio. A questo punto *l'Utente Registrato* può fornire alla pagina di *Login* i propri dati di accesso e, se corretti, essere reindirizzato verso l'interfaccia principale (*Main_Page*).

L'Utente Autenticato potrà usufruire delle stesse interfacce dell'*Utente Registrato* tranne quella di *Login*, potrà visualizzare il proprio profilo e modificarlo tramite l'interfaccia *View_User_Profile* ma con diverse funzionalità rispetto a quelle di un Utente Non Registrato o di un Utente Registrato ma che non ha effettuato l'autenticazione. Inoltre, l'*Utente Autenticato* potrà creare Roadmap in cui poter usare Stage già noti o crearne di propri (*Create_Roadmap*) e poi tramite la visualizzazione delle Roadmap lasciare una valutazione e/o un commento tramite l'interfaccia *View_Roadmap*.

Infine, rispetto ai normali Utenti, *l'Amministratore* può accedere ad un pannello di controllo aggiuntivo (interfaccia *Community_Manager*) attraverso il quale potrà gestire le segnalazioni da parte degli utenti.

Capitolo 2

Processo di sviluppo

L'insieme di attività necessarie legate allo sviluppo di un sistema software viene detto "Processo Software". Tra queste attività è possibile distinguere *"attività portanti"* (una serie di compiti da svolgere necessariamente) ed *"attività ausiliarie"* (le quali possono aumentare la qualità del software stesso che si vuole produrre). Tra le attività più importanti sicuramente è bene ricordare la fase di specifica del software, quella di progettazione e sviluppo, per concludere con quella di validazione del software e quelle di manutenzione ed evoluzione del prodotto dopo il rilascio.

Tra tutte le opere ingegneristiche, il software è l'unica ad essere intangibile fino al suo rilascio e, considerato ciò e l'alto grado di complessità relativo alla realizzazione di tale opera, è sicuramente necessario avere un metodo ed un'organizzazione del lavoro ben studiata al fine di evitare incomprensioni tra i membri del team durante il processo di sviluppo che potrebbero portare a conseguenze più o meno catastrofiche sulla realizzazione del prodotto finale.

Fatte tali considerazioni, abbiamo ritenuto importante riportare in questo capitolo tutte le tecnologie ed i tool che ci hanno aiutato durante il processo di sviluppo, sia relative alla semplice organizzazione e divisione del lavoro, sia relative alla realizzazione stessa del codice. Inoltre, nei seguenti paragrafi viene riportata anche una breve descrizione delle pratiche agili adottate e che hanno caratterizzato l'intero processo di sviluppo basato, così, su iterazioni di tipo incrementale e non plan-based. Tali metodologie hanno permesso all'intero team di sviluppo di adattarsi rapidamente e senza particolari difficoltà ad eventuali cambiamenti dei requisiti durante il processo di sviluppo. Ogni incremento prevede il rilascio continuo di codice funzionante. Ciò porta con sé una serie di benefici tra cui la possibilità di avere un continuo riscontro di ciò che si sta realizzando e dello "stato dei lavori", una maggiore semplicità di gestione del cambiamento ed una più semplice ricerca di eventuali errori in quanto, nel caso in cui il codice dovesse dare problemi, basterà cercare solo tra le righe di codice inserite nell'incremento più recente.

2.1 UP (Unified Process)

UP nasce come modello di riferimento per organizzare il lavoro ed è un processo di sviluppo software iterativo ed incrementale, molto flessibile ed aperto. UP incoraggia l'utilizzo di pratiche agili tra cui quelle da noi utilizzate e che verranno descritte successivamente negli appositi paragrafi (eXtreme Programming e Scrum). UP, infatti, annovera tra i suoi punti di forza proprio la grande flessibilità e compatibilità con altre pratiche agili o, in generale, di sviluppo software quali, ad esempio, l'object-oriented. UP definisce una serie di pratiche tra cui riportiamo quelle da noi utilizzate:

- affrontare i problemi a rischio maggiore nelle fasi iniziali di sviluppo;
- testare spesso, presto e realisticamente;
- utilizzare i Casi d'Uso;
- realizzare una modellazione visuale (da noi realizzata secondo gli standard UML);
- gestire i requisiti e documentarli.

In particolare, dunque, UP prevede che lo sviluppo del software proceda come un'attività guidata dalla definizione dei requisiti funzionali espressi tramite i casi d'uso. Le caratteristiche stesse del software, dunque, vengono dedotte da un'attenta analisi dei casi d'uso. La caratteristica di sviluppo incrementale adottata da questo approccio è caratterizzata da 4 fasi che costituiscono ogni iterazione:

- *inception* → durante questa fase vengono definiti i vincoli del progetto, viene realizzata la tabella Attori-Obiettivi (riportata nel capitolo 1 di questa documentazione), definiti i requisiti fondamentali che il sistema software dovrà rispettare e viene realizzata anche una stima dei costi e dei tempi necessari allo sviluppo;
- *elaborazione* → durante questa fase vengono realizzate le basi del sistema. Inoltre, viene effettuata anche un'analisi dei rischi e definito un piano di risk management al fine di ridurne o, auspicabilmente, eliminarne totalmente l'impatto sul prodotto finale;
- *costruzione* → durante questa fase, viene finalizzata ed eventualmente perfezionata la progettazione del sistema a partire dalle basi realizzate duranta la precedente fase di elaborazione. La fase di costruzione è suddivisa in più iterazioni, ognuna delle quali deve portare a un rilascio eseguibile del sistema. L'iterazione finale della fase di

costruzione permette di ottenere il sistema completo, che deve essere distribuito durante la successiva fase di transizione;

- *transizione* → durante questa fase viene realizzato il beta test ed il rilascio del prodotto agli utenti finali.

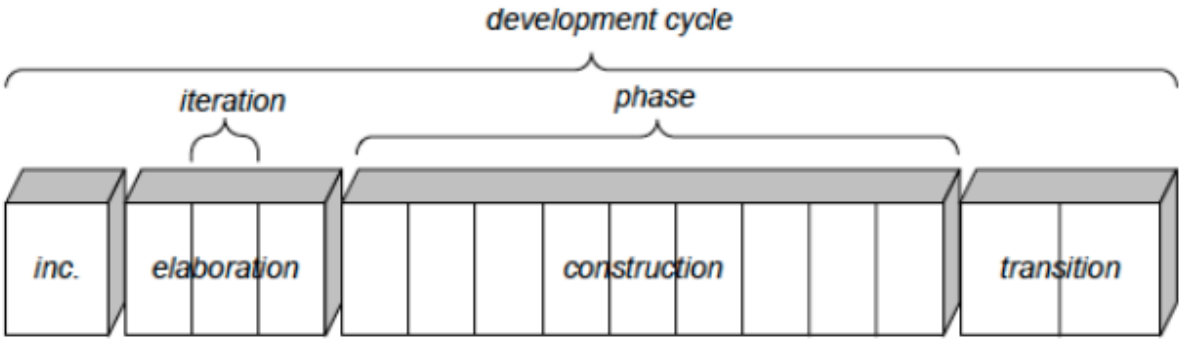


Figura 2.1 - Schema Unified Process

Come già anticipato in precedenza, fin dalle prime fasi del ciclo di sviluppo della web application da noi realizzata abbiamo sfruttato le idee di UP. Nella prima fase (quella che in UP viene definita "inception"), infatti, abbiamo posto le basi del progetto, scrivendo buona parte dei primi capitoli di questa documentazione in modo da mettere *"nero su bianco"* i requisiti che volevamo che il nostro progetto rispettasse, i vincoli con cui avremo dovuto confrontarci, il glossario - fondamentale per evitare ambiguità ed utilizzare un linguaggio chiaro, ben definito e condiviso da tutti i membri del team di sviluppo - e la prima scrittura dei casi d'uso. Terminata tale fase, l'approccio incrementale ed iterativo classico di UP non è stato accantonato ma, anzi, è stato ulteriormente ampliato dall'impiego di ulteriori metodologie agili che andremo a descrivere nel dettaglio nei prossimi paragrafi.

2.2 eXtreme Programming

eXtreme Programming (noto anche come XP) è stata una delle prime metodologie agili ad essersi affermate nella comunità di sviluppatori software. Secondo tale metodologia nuove versioni del sistema possono essere sviluppate, integrate e testate diverse volte al giorno. Uno dei vantaggi di questi continui rilasci è relativo al fatto che siccome nuove versioni del sistema possono essere rilasciate più volte al giorno, appena ci si accorge di una parte *"improvvisamente"* non funzionante sarà necessario mettere in discussione solo poche ore

di lavoro e non intere giornate o, in alcuni casi, addirittura settimane di cui si componevano i precedenti cicli di sviluppo che caratterizzavano vecchi approcci quali, ad esempio, il modello a cascata. Per poter velocizzare ulteriormente il rilascio di nuove build funzionanti, nell'approccio XP tutti i test devono essere automatizzati.

Nel pieno rispetto di XP, il lavoro da noi svolto è iniziato dalla descrizione delle cosiddette Storie Utente (riportate nell'apposito paragrafo del primo capitolo di questa documentazione). Da tali Storie sono nate una serie di discussioni tra i membri del team relative alle caratteristiche che si vogliono implementare e sui vari task di sviluppo che dovranno essere realizzati. Successivamente, siamo passati alla fase di pianificazione delle release, nella quale sono state stabilite quali sono le Storie Utente sulle quali bisognerà concentrarsi in determinati intervalli di tempo. Definito ciò, si è passati alla fase di sviluppo, integrazione e test.

L'approccio XP si basa soprattutto sull'utilizzo di 13 pratiche. Di seguito le elenchiamo tutte e sottolineiamo quelle da noi adottate più frequentemente e le approfondiamo in maggiore dettaglio:

1. i requisiti sono Storie Utente;
2. la metafora di riferimento;
3. il "planning game";
4. "Prima i test, poi il codice";
5. programmazione a coppie (pair programming);
6. integrazione continua;
7. refactoring;
8. progettazione semplice;
9. proprietà collettiva del codice;
10. cliente on-site;
11. piccoli rilasci;
12. settimana di 40 ore di lavoro;
13. uso sistematico di standard di codifica.

Oltre al già citato approccio secondo cui la maggior parte dei requisiti del sistema sono stati definiti a partire dalle Storie Utente, una delle pratiche che caratterizzano XP da noi maggiormente utilizzata è stata sicuramente quella della Programmazione a Coppie (Pair Programming). Il Pair Programming è una tecnica agile di sviluppo del software nella quale

due programmatori lavorano insieme sulla stessa postazione di lavoro (nel nostro caso, questo è stato realizzato fisicamente quando è stato possibile riunire “in presenza” il team o una parte di esso, oppure in maniera simulata tramite la condivisione del proprio schermo). Nella Programmazione a Coppie, uno dei due programmatori scrive il codice mentre l'altro svolge il ruolo di supervisore e di revisione simultanea del codice. I due programmatori hanno cambiato spesso ruolo e le coppie spesso sono state spesso “ruotate”. Questo ci ha permesso di constatare un notevole miglioramento nella diffusione della conoscenza e, di conseguenza, un accresciuto *“senso di appartenenza”* al progetto da parte di tutti i membri del team. Questa tecnica, inoltre, incoraggia enormemente il refactoring e, assieme ai caratteristici rilasci brevi e la conseguente integrazione continua, hanno permesso a tutto il team di beneficiare di un ambiente di sviluppo sano, coerente e caratterizzato da poche (ma *fisiologiche*) incomprensioni, risolte però anch’esse in brevissimo tempo.

2.3 Scrum

Scrum è un framework agile per la gestione del ciclo di sviluppo del software, iterativo ed incrementale, concepito per la gestione dei progetti e dei prodotti software. Non è una tecnica o un processo, ma un framework all’interno del quale è possibile usare più processi e tecniche. Scrum ci ha permesso, in particolare, di gestire opportunamente lo Sprint di 4 settimane necessario per realizzare il prodotto utilizzabile e potenzialmente rilasciabile. Un’altra tecnica di Scrum utilizzata nel ciclo di sviluppo è il Product Backlog. Tale backlog è un insieme di task necessari per portare a termine il progetto. Tali task sono stati organizzati per priorità ed è stato aggiornato durante l'avanzamento del progetto. Questo lavoro è stato semplificato dall’utilizzo del tool Trello che descriveremo brevemente nel successivo paragrafo.

Un’altra applicazione della metodologia Scrum è stata il Daily Meeting. Tale meeting, della durata di circa 15 minuti, è stato svolto quotidianamente ed è stato utile a tutto il team per poter sincronizzare le proprie attività e pianificare il lavoro della giornata. Inoltre, durante questo meeting ogni membro del team ha esposto cosa è andato bene durante la giornata precedente, cosa potrebbe essere migliorato e cosa ci si impegna a migliorare nella prossima iterazione.

2.4 Tool utilizzati per la condivisione del lavori

2.4.1 Discord

Discord è una piattaforma in cui gli utenti possono comunicare tra loro tramite una classica chat testuale o tramite videochiamate. Discord, inoltre, può essere utilizzato per consentire la condivisione di media e file di vario tipo. Tale piattaforma è stata installata sul computer di ogni membro del team che, una volta effettuata l'iscrizione, ha potuto iscriversi ad un cosiddetto *"Server Discord"* creato appositamente e che non è altro che un gruppo in cui ogni membro può prendere parte a diversi "canali", sia testuali che vocali. I canali testuali sono stati utilizzati per condividere link e file utili a tutti i membri del team. Tali canali testuali sono stati opportunamente suddivisi per poter essere contenitori di risorse affini tra loro in modo da poterle recuperare all'occorrenza in maniera agevole, senza eccessiva dispersione (e conseguente perdita di tempo prezioso). I canali vocali, invece, sono stati il luogo virtuale in cui si sono tenuti quotidianamente quasi tutti gli incontri del team durante l'intero ciclo di sviluppo del software (la maggior parte del lavoro, infatti, è stato svolto in modalità remota) e dove, quindi, è stato possibile realizzare chiamate e videochiamate con l'opportunità anche di condividere il proprio schermo col resto dei membri del team.

2.4.2 Google Drive e Google Docs

Google Drive è un servizio web, in ambiente cloud computing, di memorizzazione e sincronizzazione online di file e documenti condivisi con più persone contemporaneamente. Tra i vari tool messi a disposizione da questo ambiente cloud, in particolare, abbiamo utilizzato Google Docs, un editor di documenti di testo che consente di creare e collaborare su documenti online in tempo reale e da qualsiasi dispositivo. Google Docs è stato utilizzato,

dunque, per poter realizzare in maniera collaborativa la presente documentazione del software.

2.4.3 Trello

Trello è un strumento che consente ai team di gestire visivamente qualsiasi tipo di progetto, flusso di lavoro o monitoraggio dei task. La gestione di progetti sviluppati - in particolar modo - in maniera agile beneficia della struttura stessa di Trello che, tramite una serie di bacheche e schede, permette di organizzare al meglio il lavoro e la sua suddivisione tra tutti i membri del team di sviluppo. All'interno di ogni bacheca, infatti, è stato possibile creare una serie di schede all'interno delle quali inserire i task da portare a termine ed assegnare ognuno di questi ad un particolare membro del team, indicando eventualmente una data di scadenza entro cui portare a termine il lavoro e la difficoltà del lavoro stesso. Le diverse schede possono essere anche facilmente raggruppate tra loro in modo da poter avere una visione ancora più chiara dei task legati in qualche modo tra loro e, all'occorrenza, conoscere a che punto del processo di sviluppo si trova un particolare membro del team rispetto ad uno specifico compito. In questo modo, è stato possibile migliorare la produttività dell'interno gruppo di lavoro, sincronizzando al meglio anche le operazioni di comunicazione tra front-end e back-end necessarie per il corretto funzionamento dell'applicazione stessa.

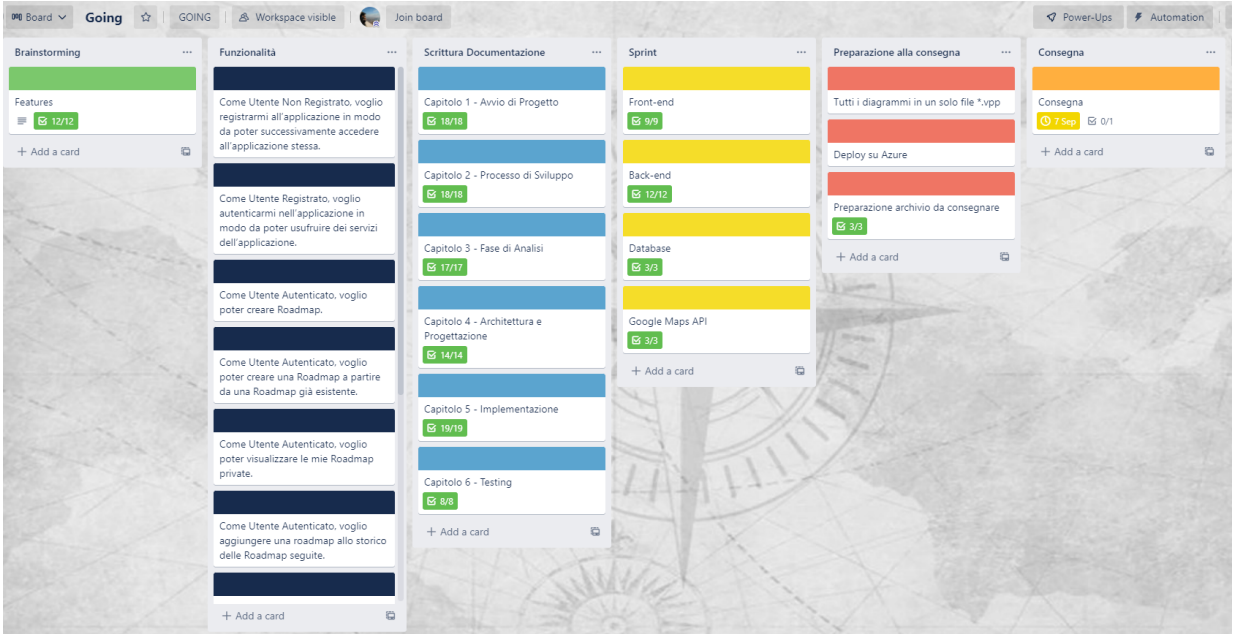


Figura 2.2 - Organizzazione Trello

2.4.4 GitHub

GitHub è un servizio di hosting per progetti software. Tale servizio è stato utilizzato per la condivisione del codice tra i diversi membri del team di sviluppo. Inoltre, un servizio come GitHub consente di sfruttare la repository del progetto anche come possibile backup del codice stesso che, in questo modo, non può venir perso. Le push sulla repository condivisa dei diversi membri del team di sviluppo sono state effettuate al termine dell'implementazione di ogni funzione, assicurandone il funzionamento tramite anche una fase di test di unità. Le pull, invece, sono state effettuate almeno una volta al giorno al fine di consentire ad ogni membro del team di sviluppo di restare sempre allineato con lo sviluppo del progetto stesso. Nel caso in cui più membri del team lavorino sugli stessi file, GitHub fornisce anche uno strumento per una agevole gestione dei conflitti. Inoltre, l'IDE utilizzato (Visual Studio Code, descritto nel paragrafo dedicato ai tool utilizzati per lo sviluppo) presenta al suo interno - tramite plugin - una completa integrazione con GitHub stesso e, dunque, della gestione della repository condivisa e di tutte le operazioni eseguibili su di essa.

Link alla repository GitHub → https://github.com/giammy677dev/going

2.5 Tool e tecnologie utilizzate per lo sviluppo

2.5.1 Visual Paradigm

Visual Paradigm è lo strumento che è stato utilizzato per la progettazione dei diagrammi UML. La modellazione di tali diagrammi è stata necessaria per una corretta descrizione e visualizzazione della nostra Architettura Software. Tutti i diagrammi prodotti tramite Visual Paradigm sono allegati alla cartella del progetto.

2.5.2 Visual Studio Code

Visual Studio Code è l'IDE utilizzato per lo sviluppo del codice del progetto. Visual Studio Code supporta una vasta gamma di linguaggi di programmazione tra cui (quelli da noi utilizzati) JavaScript, HTML e CSS. Visual Studio Code, inoltre, supporta una serie di plugin che lo rendono l'ambiente di sviluppo ideale grazie - tra le altre cose - ad un alto grado di personalizzazione dell'interfaccia ed alla sua integrazione con GitHub (come spiegato sopra) e Microsoft Azure, la tecnologia che abbiamo scelto per il deploy dell'applicazione sul web (come spiegato nell'apposito paragrafo poco più in basso).

2.5.3 Node.js + Express

Node.js è un ambiente runtime single-threaded, open-source e multipiattaforma per la creazione di applicazioni server-side e di networking veloci e scalabili. Gira sul motore di runtime JavaScript V8 e usa un'architettura I/O event-driven e non bloccante, il che lo rende efficiente e adatto ad applicazioni real-time. Nel corso del progetto sono stati aggiunti vari middleware a Node.js tra cui Express.

Express.js, o semplicemente Express, è un framework per applicazioni web per Node.js, open source sotto Licenza MIT. È stato progettato per creare web application e API ed è ormai definito il server framework standard de facto per Node.js.

2.5.4 Google Maps API

Google Maps è un servizio internet geografico sviluppato da Google che consente la ricerca e la visualizzazione di carte geografiche di buona parte della Terra. Dal 2008 Google mette a

disposizione le risorse di Google Maps attraverso API. Il servizio Google Maps API facilita l'estrapolazione di informazioni relative a determinate aree geografiche. Permette, tra le altre cose, di calcolare un possibile percorso stradale tra due punti, ottenere informazioni su determinati luoghi e così via.

2.5.5 MySQL Workbench

MySQL è un sistema di gestione di database relazionali (RDBMS). Un database relazionale organizza i dati in una o più tabelle di dati in cui i dati possono essere correlati tra loro; queste relazioni aiutano a strutturare i dati. SQL (Structured Query Language) è un linguaggio utilizzato per creare, modificare ed estrarre dati dal database relazionale, nonché per controllare l'accesso degli utenti al database. Oltre ai database relazionali e SQL, un RDBMS come MySQL gestisce gli utenti, consente l'accesso alla rete e facilita il test dell'integrità del database e la creazione di backup.

2.5.6 Postman

Postman è una piattaforma API per gli sviluppatori per progettare, creare, testare e iterare le loro API. Ad aprile 2022, Postman ha riferito di avere più di 20 milioni di utenti registrati e 75.000 API aperte, che secondo lui costituiscono il più grande hub API pubblico del mondo. Postman è stato utilizzato anche come strumento di condivisione tramite una repository Postman condivisa.

2.5.7 Microsoft Azure

Microsoft Azure è la piattaforma cloud pubblica di Microsoft, che offre servizi di cloud computing.

Tramite Azure vengono erogati servizi appartenenti a diverse categorie quali: risorse di elaborazione, archiviazione, memorizzazione, trasmissione dati e interconnessione di reti, analisi, intelligence, apprendimento automatico, sicurezza e gestione delle identità, monitoraggio e gestione, nonché servizi per lo sviluppo di applicazioni. Azure ci ha permesso di hostare un database condiviso, utilizzabile da remoto tramite MySQL Workbench. Inoltre, Microsoft Azure è stato utilizzato anche per l'hosting del server.

Link al sito di Going → https://going-app.azurewebsites.net/

2.5.8 Photoshop

Adobe Photoshop è il software di fotoritocco più utilizzato al mondo: permette di creare, elaborare, modificare e ottimizzare immagini digitali, fotografie, illustrazioni e video. Con questo software, in sintesi, si lavora su immagini raster, cioè create a partire da una “griglia” (raster, appunto) di pixel.

Nel corso degli anni, aggiornamento dopo aggiornamento, è diventato anche uno dei programmi più avanzati di grafica, oltre che di fotografia. Un programma che designer e artisti utilizzano per disegnare, fare sketching o addirittura creare dipinti digitali. L’aspetto chiave di Photoshop è la sua struttura a livelli, che permette di lavorare in modo modulare e di aggiungere modifiche all’immagine complessiva anche senza intaccare le immagini originarie.

Attraverso l’utilizzo di Photoshop, siamo stati in grado di customizzare tutti gli elementi grafici presenti all’interno del sito creandoli o adattandoli in conformità alla “color palette” del tema scelto per il sito.

Capitolo 3

Fase di Analisi

Dopo la prima interazione con gli stakeholders, ne è stata effettuata una seconda che ci ha fornito chiarimenti, disambiguazione di termini, ristrutturazioni delle frasi che esprimono più requisiti e un maggior numero di informazioni specifiche. Si riporta di seguito il documento aggiornato:

Si vuole realizzare una web app denominata *GOING* in cui gli utenti possano organizzare un proprio itinerario di viaggio (detto Roadmap) al fine di visitare una città.
Una Roadmap è costituita da un titolo, visibilità (pubblica o privata), durata, distanza di percorrenza complessiva, modalità di percorrenza (a piedi o in macchina), località di riferimento ed, opzionalmente, una descrizione. Ogni Roadmap è composta da una lista di Stage. Ogni Stage è costituito da latitudine, longitudine, indirizzo, località, nome ed, opzionalmente, un sito web ed una foto.
Gli Utenti Non Registrati potranno cercare e visualizzare le Roadmap pubbliche create da altri utenti e/o cercare e visualizzare il profilo di un altro Utente Registrato. Le Roadmap cercate saranno filtrate in base ad una serie di parametri: la località di riferimento della Roadmap, la sua durata e/o la distanza di percorrenza complessiva.
Inoltre, l'applicazione dovrà permettere agli utenti di potersi registrare al servizio. La registrazione è costituita dall'inserimento di email, nome utente, password e data di nascita. Il nome utente permette di identificare univocamente un Utente Registrato e, quindi, eventualmente di poterlo ricercare da parte di un altro utente. La password consente all'Utente Registrato, invece, di accedere all'applicazione in maniera sicura, diventando così un Utente Autenticato. Gli Utenti Autenticati potranno eseguire tutte le azioni di un Utente Non Registrato ed <u>in più</u> potranno:

- creare una Roadmap (attribuendone ad ognuna la visibilità pubblica o privata) costituite da punti di interesse (quelli che abbiamo denominato Stage e di cui è possibile trovarne la definizione nel Glossario riportato in questa documentazione) pre-esistenti nell'applicazione e/o da punti di interesse creati ex-novo dall'utente, con

la possibilità di crearle anche a partire da una Roadmap pubblica già esistente tramite la cosiddetta operazione di “fork” (intesa come definita anch’essa nel Glossario riportato in questa documentazione);

- mantenere uno storico delle Roadmap Seguite durante i propri viaggi;
- aggiungere Roadmap ai “preferiti”;
- aggiungere una recensione della Roadmap. Una recensione consiste nell’inserimento di una valutazione da 1 a 5 ad una Roadmap, di una data di pubblicazione e di una eventuale opinione;
- aggiungere un commento per richiedere informazioni agli utenti riguardo ad una specifica Roadmap. Un commento è caratterizzato da un testo e dalla data di pubblicazione;
- eliminare una Roadmap creata precedentemente;
- eliminare un commento ad una Roadmap inviato precedentemente;
- eliminare una recensione ad una Roadmap inviata precedentemente;
- modificare un commento ad una Roadmap;
- modificare una recensione ad una Roadmap;
- segnalare un utente, segnalare una Roadmap, segnalare uno Stage e/o segnalare un commento.

Ad ogni Roadmap verrà assegnato un punteggio complessivo, risultato delle votazioni degli Utenti Autenticati.

La community viene gestita da Utenti Amministratori che possono eliminare Roadmap, commenti e/o valutazioni di altri utenti e gestire le segnalazioni inviate dagli utenti. Inoltre, gli amministratori possono anche bandire un utente, causando in questo modo l’eliminazione automatica delle Roadmap, commenti e valutazioni associate all’utente eliminato.

Opzionalmente, infine, il sistema potrà fornire delle caratteristiche di gamification basate su badge ed achievement Utenti Registrati potranno ottenere in base ad una serie di azioni derivanti dall’utilizzo della piattaforma stessa.

L’applicazione dovrà garantire ai clienti un alto grado di usabilità. Infatti, durante l’interazione con il prodotto, i clienti dovranno percepire un elevato livello di soddisfazione e gradimento grazie ad una grafica accattivante e ad un’interfaccia “user-friendly”.

Inoltre, l'applicazione è pensata per evolversi nel tempo, agevolando l'aggiunta e la modifica di feature in tempi brevi. Infine, nonostante l'applicazione non debba mantenere particolari dati sensibili, si dovrà comunque garantire un buon livello di sicurezza.

3.1 Analisi e Specifica dei Requisiti

3.1.1 Analisi Nomi-Verbi

Si vuole realizzare una web app denominata *GOING* in cui gli utenti possano organizzare un proprio itinerario di viaggio (detto Roadmap) al fine di visitare una città.
Una Roadmap è costituita da un titolo, visibilità (pubblica o privata), durata, distanza di percorrenza complessiva, modalità di percorrenza (a piedi o in macchina), località di riferimento ed, opzionalmente, una descrizione. Ogni Roadmap è composta da una lista di Stage. Ogni Stage è costituito da latitudine, longitudine, indirizzo, località, nome ed, opzionalmente, un sito web ed una foto.
Gli Utenti Non Registrati potranno cercare e visualizzare le Roadmap pubbliche create da altri utenti e/o cercare e visualizzare il profilo di un altro Utente Registrato. Le Roadmap cercate saranno filtrate in base ad una serie di parametri: la località di riferimento della Roadmap, la sua durata e/o la distanza di percorrenza complessiva.
Inoltre, l'applicazione dovrà permettere agli utenti di potersi registrare al servizio. La registrazione è costituita dall'inserimento di email, nome utente, password e data di nascita. Il nome utente permette di identificare univocamente un Utente Registrato e, quindi, eventualmente di poterlo ricercare da parte di un altro utente. La password consente all'Utente Registrato, invece, di accedere all'applicazione in maniera sicura, diventando così un Utente Autenticato. Gli Utenti Autenticati potranno eseguire tutte le azioni di un Utente Non Registrato ed in più potranno:

- creare una Roadmap (attribuendone ad ognuna la visibilità pubblica o privata) costituite da punti di interesse (quelli che abbiamo denominato Stage e di cui è possibile trovarne la definizione nel Glossario riportato in questa documentazione) pre-esistenti nell'applicazione e/o da punti di interesse creati ex-novo dall'utente, con la possibilità di crearle anche a partire da una Roadmap pubblica già esistente

tramite la cosiddetta operazione di “fork” (intesa come definita anch’essa nel Glossario riportato in questa documentazione);

- mantenere uno storico delle Roadmap Seguite durante i propri viaggi;
- aggiungere Roadmap ai “preferiti”;
- aggiungere una recensione della Roadmap. Una recensione consiste nell’inserimento di una valutazione da 1 a 5 ad una Roadmap, di una data di pubblicazione e di una eventuale opinione;
- aggiungere un commento per richiedere informazioni agli utenti riguardo ad una specifica Roadmap. Un commento è caratterizzato da un testo e dalla data di pubblicazione;
- eliminare una Roadmap creata precedentemente;
- eliminare un commento ad una Roadmap inviato precedentemente;
- eliminare una recensione ad una Roadmap inviata precedentemente;
- modificare un commento ad una Roadmap;
- modificare una recensione ad una Roadmap;
- segnalare un utente, segnalare una Roadmap, segnalare uno Stage e/o segnalare un commento.

Ad ogni Roadmap verrà assegnato un punteggio complessivo, risultato delle votazioni degli Utenti Autenticati.

La community viene gestita da Utenti Amministratori che possono eliminare Roadmap, commenti e/o valutazioni di altri utenti e gestire le segnalazioni inviate dagli utenti. Inoltre, gli amministratori possono anche bandire un utente, causando in questo modo l’eliminazione automatica delle Roadmap, commenti e valutazioni associate all’utente eliminato.

Opzionalmente, infine, il sistema potrà fornire delle caratteristiche di gamification basate su badge ed achievement che gli Utenti Registrati potranno ottenere in base ad una serie di azioni derivanti dall’utilizzo della piattaforma stessa.

L’applicazione dovrà garantire ai clienti un alto grado di usabilità. Infatti, durante l’interazione con il prodotto, i clienti dovranno percepire un elevato livello di soddisfazione e gradimento grazie ad una grafica accattivante e ad un’interfaccia “user-friendly”.

Inoltre, l'applicazione è pensata per evolversi nel tempo, agevolando l'aggiunta e la modifica di feature in tempi brevi. Infine, nonostante l'applicazione non debba mantenere particolari dati sensibili, si dovrà comunque garantire un buon livello di sicurezza.

LEGENDA
Classe
Attributo
Funzionalità
Attore
Classe-Attore

3.1.2 Revisione dei Requisiti

1. Ogni Roadmap è costituita da titolo, visibilità (pubblica o privata), durata, distanza di percorrenza complessiva, località di riferimento, modalità di percorrenza (a piedi o in macchina), descrizione opzionale, una lista di Stage ed un punteggio.
2. Ogni Stage è costituito da latitudine, longitudine, indirizzo, località, nome ed, opzionalmente, un sito web ed una foto.
3. Il sistema deve offrire all'Utente una funzionalità per ricercare una Roadmap.
4. Il sistema deve offrire all'Utente una funzionalità per ricercare un Utente Registrato.
5. Il sistema deve offrire all'Utente una funzionalità per visualizzare una Roadmap.
6. Il sistema deve offrire all'Utente una funzionalità per visualizzare il profilo di un Utente Registrato.
7. Il sistema deve offrire al cliente una funzionalità di registrazione.
8. Degli Utenti Registrati, si vuole memorizzare l'email, il nome utente, la password e la data di nascita.
9. L'Utente Registrato è univocamente identificato dal suo nome utente.
10. L'Utente Registrato deve poter effettuare tutte le funzionalità dell'Utente Non Registrato eccetto l'operazione di registrazione.
11. L'Utente Registrato deve poter accedere all'applicazione con le sue credenziali divenendo così Utente Autenticato.

12. L'Utente Autenticato deve poter effettuare tutte le funzionalità di un Utente Non Registrato eccetto l'operazione di registrazione.
13. L'Utente Autenticato deve poter creare una Roadmap ex-novo o a partire da una pubblica già esistente tramite l'operazione di "fork".
14. L'Utente Autenticato deve poter mantenere uno storico delle Roadmap Seguite.
15. Il sistema deve offrire all'Utente Autenticato una funzionalità per aggiungere Roadmap ai "preferiti".
16. Il sistema deve offrire all'Utente Autenticato una funzionalità per aggiungere una recensione ad una Roadmap.
17. Una recensione consiste nell'inserimento di una valutazione e di una eventuale opinione.
18. La valutazione di una Roadmap è un numero intero compreso tra 1 e 5.
19. Il sistema deve offrire all'Utente Autenticato una funzionalità per aggiungere un commento.
20. Un commento ad una Roadmap è caratterizzato da un testo.
21. Il sistema deve offrire all'Utente Autenticato una funzionalità per eliminare una propria Roadmap.
22. Il sistema deve offrire all'Utente Autenticato una funzionalità per eliminare un proprio commento ad una Roadmap.
23. Il sistema deve offrire all'Utente Autenticato una funzionalità per eliminare una propria recensione ad una Roadmap.
24. Il sistema deve offrire all'Utente Autenticato una funzionalità per modificare un proprio commento ad una Roadmap.
25. Il sistema deve offrire all'Utente Autenticato una funzionalità per modificare una propria recensione ad una Roadmap.
26. Il sistema deve offrire all'Utente Autenticato una funzionalità per segnalare un Utente Registrato.
27. Il sistema deve offrire all'Utente Autenticato una funzionalità per segnalare una Roadmap pubblica.
28. Il sistema deve offrire all'Utente Autenticato una funzionalità per segnalare uno Stage.

29. Il sistema deve offrire all'Utente Autenticato una funzionalità per segnalare un commento.
30. Il sistema deve offrire all'Utente Autenticato una funzionalità per effettuare la disconnessione (logout) dal proprio account.
31. Il sistema deve offrire agli Utenti Amministratori una funzionalità per eliminare Roadmap.
32. Il sistema deve offrire agli Utenti Amministratori una funzionalità per eliminare commenti.
33. Il sistema deve offrire agli Utenti Amministratori una funzionalità per eliminare recensioni.
34. Il sistema deve offrire all'Utente Amministratore una funzionalità per gestire le segnalazioni.
35. L'Utente Amministratore deve poter effettuare tutte le funzionalità dell'Utente Autenticato.
36. Il sistema deve offrire un meccanismo di assegnazione di achievement.
37. L'applicazione dovrà garantire ai clienti un alto grado di usabilità.
38. L'applicazione deve agevolare l'aggiunta e la modifica di feature in tempi brevi.
39. L'applicazione deve garantire un buon livello di sicurezza.
40. Le Roadmap cercate saranno filtrate in base ad una serie di parametri.
41. I parametri per filtrare una Roadmap sono la località di riferimento della Roadmap, la durata e/o la distanza di percorrenza complessiva.

3.1.3 Classificazione dei Requisiti

Requisiti Funzionali

ID	Requisito	Origine
RF01	Il sistema deve offrire all'Utente una funzionalità per ricercare una Roadmap.	3
RF02	Il sistema deve offrire all'Utente una funzionalità per ricercare un Utente Registrato.	4
RF03	Il sistema deve offrire all'Utente una funzionalità per visualizzare una Roadmap.	5

RF04	Il sistema deve offrire all'Utente una funzionalità per visualizzare il profilo di un Utente Registrato.	6
RF05	Il sistema deve offrire al cliente una funzionalità di registrazione.	7
RF06	L'Utente Registrato deve poter effettuare tutte le funzionalità dell'Utente Non Registrato eccetto l'operazione di registrazione.	10
RF07	L'Utente Registrato deve poter accedere all'applicazione con le sue credenziali divenendo Utente Autenticato.	11
RF08	L'Utente Autenticato deve poter effettuare tutte le funzionalità di un Utente Non Registrato eccetto l'operazione di registrazione.	12
RF09	L'Utente Autenticato deve poter creare una Roadmap ex-novo o a partire da una pubblica già esistente tramite l'operazione di "fork".	13
RF10	L'Utente Autenticato deve poter mantenere uno storico delle Roadmap Seguite.	14
RF11	Il sistema deve offrire all'Utente Autenticato una funzionalità per aggiungere Roadmap ai "preferiti".	15
RF12	Il sistema deve offrire all'Utente Autenticato una funzionalità per aggiungere una recensione ad una Roadmap.	16
RF13	Una recensione consiste nell'inserimento di una valutazione e di una eventuale opinione.	17
RF14	Il sistema deve offrire all'Utente Autenticato una funzionalità per aggiungere un commento.	19
RF15	Il sistema deve offrire all'Utente Autenticato una funzionalità per eliminare una propria Roadmap.	21
RF16	Il sistema deve offrire all'Utente Autenticato una funzionalità per eliminare un proprio commento ad una Roadmap.	22
RF17	Il sistema deve offrire all'Utente Autenticato una funzionalità per eliminare una propria recensione ad una Roadmap.	23
RF18	Il sistema deve offrire all'Utente Autenticato una funzionalità per modificare un proprio commento ad una Roadmap.	24
RF19	Il sistema deve offrire all'Utente Autenticato una funzionalità per modificare una propria recensione ad una Roadmap.	25

RF20	Il sistema deve offrire all'Utente Autenticato una funzionalità per segnalare un Utente Registrato.	26
RF21	Il sistema deve offrire all'Utente Autenticato una funzionalità per segnalare una Roadmap pubblica.	27
RF22	Il sistema deve offrire all'Utente Autenticato una funzionalità per segnalare uno Stage.	28
RF23	Il sistema deve offrire all'Utente Autenticato una funzionalità per segnalare un commento.	29
RF24	Il sistema deve offrire all'Utente Autenticato una funzionalità per effettuare la disconnessione (logout) dal proprio account.	30
RF25	Il sistema deve offrire agli Utenti Amministratori una funzionalità per eliminare Roadmap.	31
RF26	Il sistema deve offrire agli Utenti Amministratori una funzionalità per eliminare commenti.	32
RF27	Il sistema deve offrire agli Utenti Amministratori una funzionalità per eliminare recensioni.	33
RF28	Il sistema deve offrire all'Utente Amministratore una funzionalità per gestire le segnalazioni.	34
RF29	L'Utente Amministratore deve poter effettuare tutte le funzionalità dell'Utente Autenticato.	35
RF30	Il sistema deve offrire un meccanismo di assegnazione di achievement.	36
RF31	Le Roadmap cercate saranno filtrate in base ad una serie di parametri.	40

Requisiti sui Dati

ID	Requisito	Origine
RD01	Ogni Roadmap è costituita da titolo, visibilità (pubblica o privata), durata, distanza di percorrenza complessiva, modalità di percorrenza (a piedi o in macchina), località di riferimento, descrizione opzionale, una lista di Stage ed un punteggio.	1
RD02	Ogni Stage è costituito da latitudine, longitudine, indirizzo, località,	2

	nome ed, opzionalmente, un sito web ed una foto.	
RD03	Degli Utenti Registrati, si vuole memorizzare l'email, il nome utente, la password e la data di nascita.	8
RD04	L'Utente Registrato è univocamente identificato dal suo nome utente.	9
RD05	La valutazione di una Roadmap è un numero intero compreso tra 1 e 5.	18
RD06	Un commento ad una Roadmap è caratterizzato da un testo.	20
RD07	I parametri per filtrare una Roadmap sono la località di riferimento della Roadmap, la durata e/o la distanza di percorrenza complessiva.	41

Vincoli/Altri Requisiti

ID	Requisito	Origine
RNF01	L'applicazione dovrà garantire ai clienti un alto grado di usabilità.	37
RNF02	L'applicazione deve agevolare l'aggiunta e la modifica di feature in tempi brevi.	38
RNF03	L'applicazione deve garantire un buon livello di sicurezza.	39

3.2 Modello dei Casi d'Uso

3.2.1 Attori e Casi d'Uso

ATTORE PRIMARIO
Utente
Utente Non Registrato
Utente Registrato
Utente Autenticato
Utente Amministratore

ATTORE SECONDARIO
Utente Registrato

ID	CASI D'USO
UC01	Ricerca Roadmap
UC02	Visualizza Roadmap
UC03	Ricerca Utente Registrato
UC04	Visualizza Utente Registrato
UC05	Effettua registrazione
UC06	Effettua login
UC07	Crea Roadmap
UC08	Contrassegna Roadmap come svolta
UC09	Aggiungi Roadmap ai preferiti
UC10	Aggiungi recensione
UC11	Aggiungi commento
UC12	Elimina Roadmap
UC13	Elimina commento
UC14	Elimina recensione
UC15	Modifica commento
UC16	Modifica recensione
UC17	Segnala Utente Registrato
UC18	Segnala Roadmap pubblica
UC19	Segnala Stage
UC20	Segnala commento
UC21	Effettua logout
UC22	Gestione segnalazioni

ID	CASI D'USO DI INCLUSIONE
UC02	Visualizza Roadmap

ID	CASI D'USO DI ESTENSIONE
UC02	Visualizza Roadmap
UC04	Visualizza Utente Registrato

3.2.2 Diagramma dei Casi d'Uso

Il diagramma dei Casi d'Uso è un tipo di diagramma UML che abbiamo utilizzato al fine di esplicitare graficamente i casi d'uso definiti in fase di analisi e le rispettive relazioni di *include* ed *extend* che li caratterizzano, oltre agli attori primari e secondari che eseguono tali casi d'uso andando così - di fatto - ad esplicitare il modo in cui tali attori interagiscono col sistema stesso. Dal grafico possiamo notare molte relazioni di *inclusione* in quanto la maggior parte delle operazioni eseguibili sul nostro sistema richiedono necessariamente la visualizzazione di una Roadmap o di un utente specifico. Tale operazione di visualizzazione è stata formalizzata a sua volta come caso d'uso stesso e, per questo, è stato necessario introdurre molte relazioni di *include*.

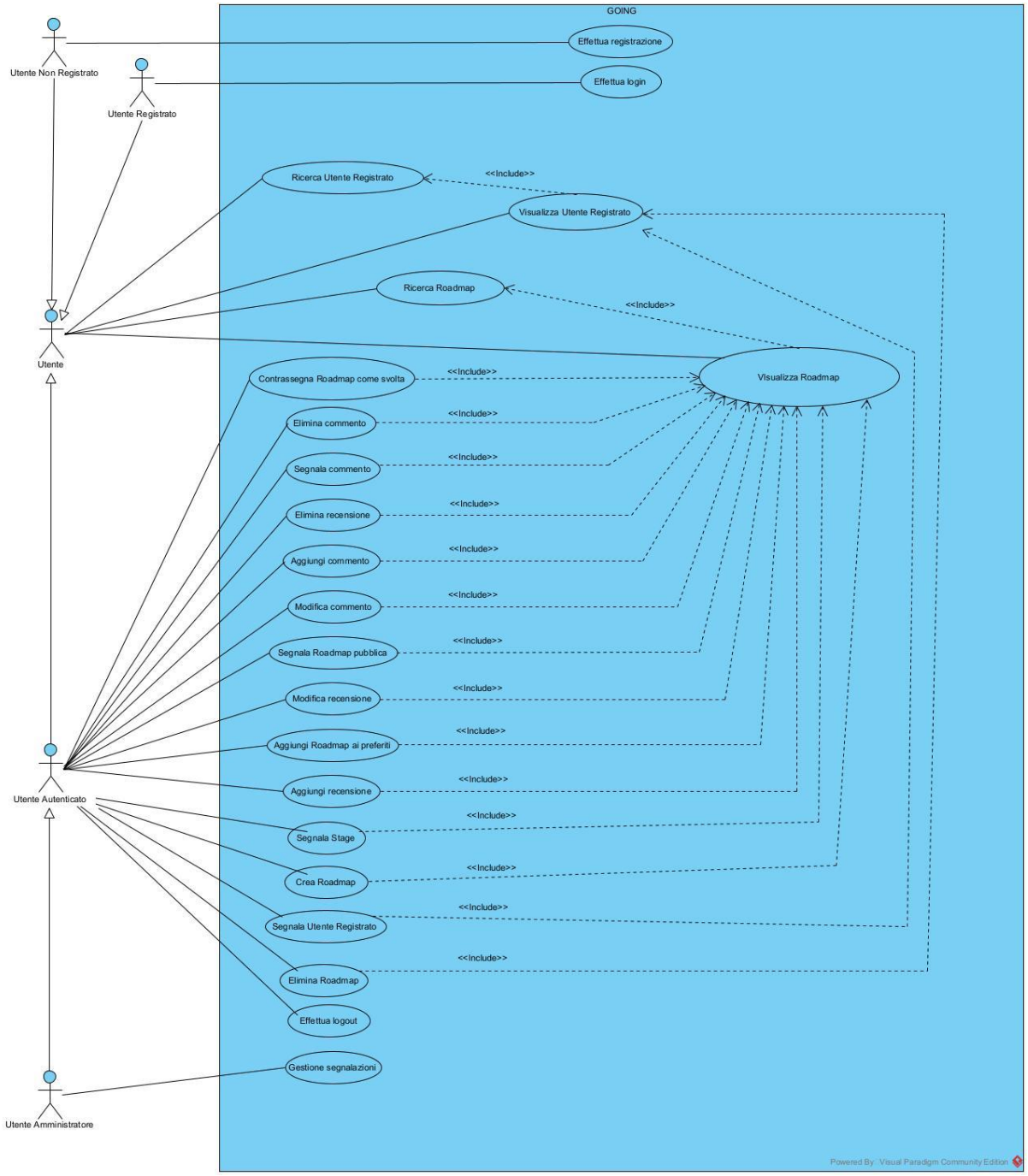


Figura 3.1 - Diagramma dei Casi d'Uso

3.2.3 Scenari dei Casi d'uso

A seguito del diagramma dei casi d'uso, si specificano alcuni casi d'uso in formato dettagliato tramite i cosiddetti Scenari.

Ricerca di una Roadmap

ID	UC01
Descrizione	Il sistema deve offrire all'Utente una funzionalità per ricercare una Roadmap.
Attore Primario	Utente.
Pre-condizioni	L'Utente seleziona la barra di ricerca.
Scenario Principale	1. L'Utente inserisce una o più parole chiave per effettuare la ricerca. 2. Il sistema mostra all'Utente una lista di Roadmap.
Post-condizioni	Il sistema ha restituito le Roadmap risultanti dalla ricerca.
Scenari Alternativi	2a. Se la connessione col server è fallita, il sistema segnala l'indisponibilità del server.

Visualizzazione di una Roadmap tramite ricerca

ID	UC02
Descrizione	Il sistema deve offrire all'Utente una funzionalità per visualizzare una Roadmap risultante da una ricerca.
Attore Primario	Utente.
Pre-condizioni	La ricerca ha restituito almeno un risultato e la Roadmap è accessibile dall'Utente.
Scenario Principale	1. **INCLUDE** (Ricerca Roadmap). 2. L'Utente richiede una delle Roadmap presentate dalla ricerca. 3. Il sistema mostra la Roadmap all'Utente.
Post-condizioni	La Roadmap è stata mostrata all'Utente.
Scenario Alternativo	1a. Se la connessione col server è fallita, il sistema segnala l'indisponibilità del server.

Visualizzazione di una Roadmap tramite selezione diretta

ID	UC02
Descrizione	Il sistema deve offrire all'Utente una funzionalità per visualizzare una Roadmap tramite selezione diretta (ad esempio, tramite link o dopo aver visitato il profilo di un Utente).
Attore Primario	Utente.
Pre-condizioni	La Roadmap è accessibile dall'Utente.
Scenario Principale	1. L'Utente seleziona una Roadmap. 2. Il sistema mostra la Roadmap all'Utente.
Post-condizioni	La Roadmap è stata mostrata all'Utente.
Scenario Alternativo	1a. Se la connessione col server è fallita, il sistema segnala l'indisponibilità del server.

Ricerca di un Utente Registrato

ID	UC03
Descrizione	Il sistema deve offrire all'Utente una funzionalità per ricercare un Utente Registrato.
Attore Primario	Utente.
Pre-condizioni	L'Utente seleziona la barra di ricerca.
Scenario Principale	1. L'Utente inserisce una o più parole chiave per effettuare la ricerca. 2. Il sistema indirizza l'Utente verso una lista di Utenti Registrati.
Post-condizioni	Il sistema ha restituito gli Utenti Registrati risultanti dalla ricerca.
Scenari Alternativi	2a. Se la connessione col server è fallita, il sistema segnala l'indisponibilità del server.

Visualizzazione di un Utente Registrato tramite ricerca

ID	UC04
Descrizione	Il sistema deve offrire all'Utente una funzionalità per visualizzare un Utente Registrato risultante da una ricerca.
Attore Primario	Utente.
Pre-condizioni	La ricerca ha restituito almeno un risultato.
Scenario Principale	1. **INCLUDE** (Ricerca Utente Registrato) 2. L'Utente richiede uno degli Utenti Registrati presentati dalla ricerca. 3. Il sistema mostra il profilo dell'Utente Registrato all'Utente.
Post-condizioni	L'Utente Registrato è stata mostrato all'Utente.
Scenario Alternativo	1a. Se la connessione col server è fallita, il sistema segnala l'indisponibilità del server.

Visualizzazione di un Utente Registrato tramite selezione diretta

ID	UC04
Descrizione	Il sistema deve offrire all'Utente una funzionalità per visualizzare il profilo di un Utente Registrato tramite selezione diretta (ad esempio tramite il nome stesso dell'Utente Registrato di cui si vuole visualizzare il profilo. Tale nome si troverà nella pagina di visualizza Roadmap, nell'autore della stessa, sui commenti o sulle recensioni create dall'Utente Registrato).
Attore Primario	Utente.
Pre-condizioni	L'Utente ha visualizzato il nome utente dell'Utente Registrato di cui vuole visitare il profilo.
Scenario Principale	1. L'Utente richiede il profilo dell'Utente Registrato. 2. Il sistema mostra il profilo dell'Utente Registrato all'Utente.
Post-condizioni	L'Utente Registrato è stata mostrato all'Utente.
Scenario Alternativo	1a. Se la connessione col server è fallita, il sistema segnala l'indisponibilità del server.

Registrazione Utente

ID	UC05
Descrizione	Il sistema deve offrire all'Utente Non Registrato la possibilità di registrarsi.
Attori Primari	Utente Non Registrato.
Pre-condizioni	L'Utente non è registrato.
Scenario Principale	1. L'Utente Non Registrato accede alla pagina di registrazione. 2. L'utente inserisce il nome utente, la password di almeno 8 caratteri, reinserisce nuovamente la password come conferma, la propria email e la propria data di nascita. 3. Il sistema conferma l'avvenuta registrazione.
Post-condizioni	L'Utente si è registrato con successo.
Scenario Alternativo 1	3.a: Se il nome utente è stato già utilizzato da un altro utente, il sistema segnala l'errore.
Scenario Alternativo 2	3.b: Se l'email è già in uso, il sistema segnala l'errore.
Scenario Alternativo 3	3.c: Se l'email è presente nella lista dei bannati, il sistema segnala l'errore.
Scenario Alternativo 4	3.d: Se l'email inserita non è di un formato valido, il sistema segnala l'errore.
Scenario Alternativo 5	3.e: Se la password inserita non corrisponde con quella inserita nel campo di "conferma password", il sistema segnala l'errore.
Scenario Alternativo 6	3.f: Se la password non è di almeno 8 caratteri, il sistema segnala l'errore.

Autenticazione Utente

ID	UC06
Descrizione	L'Utente Registrato deve poter accedere all'applicazione con le sue credenziali divenendo così Utente Autenticato.
Attori Primari	Utente Registrato.
Pre-condizioni	L'Utente è già registrato al sistema.

Scenario Principale	1. L'Utente Registrato accede alla Homepage. 2. L'utente inserisce il nome utente e la password. 3. Il sistema conferma l'avvenuta autenticazione.
Post-condizioni	L'Utente Registrato diviene Utente Autenticato.
Scenario Alternativo 1	3a. Se il nome utente inserito non è corretto, il sistema segnala l'errore 4. L'Utente Registrato reinserisce il nome utente fino a quando non è corretto
Scenario Alternativo 2	3b. Se la password inserita non corrisponde a quel particolare nome utente, il sistema segnala l'errore di "password non corrispondente al nome utente". 4. L'utente inserisce nuovamente la password fino a quando essa non sarà corrispondente al nome utente.
Scenario Alternativo 3	3c. Se la connessione col server è fallita il sistema segnala l'indisponibilità del server.

Creazione di una Roadmap

ID	UC07
Descrizione	L'Utente Autenticato deve poter creare Roadmap.
Attori Primari	Utente Autenticato.
Pre-condizioni	L'Utente è già autenticato al sistema.
Scenario Principale	1. L'Utente Autenticato accede alla pagina di creazione di una Roadmap. 2. L'Utente Autenticato aggiunge un titolo alla Roadmap, ne imposta la visibilità (pubblica o privata), la modalità di percorrenza (a piedi o in macchina) ed, opzionalmente, una descrizione. L'Utente Autenticato aggiunge poi almeno due Stage alla Roadmap. 3. Il sistema conferma la pubblicazione della Roadmap.
Post-condizioni	L'Utente Autenticato ha pubblicato una Roadmap con successo.
Scenario Alternativo 1 (Fork)	1a. **INCLUDE** (Visualizza Roadmap). 2. Il sistema reindirizza l'Utente Autenticato alla pagina di creazione, inizializzata con le informazioni relative agli Stage ed alla modalità di percorrenza della Roadmap "forkata". 3. L'Utente Autenticato aggiunge un titolo alla Roadmap, ne

	imposta la visibilità (pubblica o privata) ed, opzionalmente, una descrizione. L'Utente Autenticato può aggiungere nuovi Stage o rimuoverne altri tra quelli già presenti. Il minimo numero di Stage per poter confermare la Roadmap è sempre due. 4. Il sistema conferma la pubblicazione della Roadmap.
Scenario Alternativo 2	3a. Se la connessione col server è fallita il sistema segnala l'indisponibilità del server.

Recensione di una Roadmap

ID	UC10
Descrizione	Il sistema deve offrire all'Utente Autenticato una funzionalità per aggiungere una recensione ad una Roadmap.
Attori Primari	Utente Autenticato.
Pre-condizioni	L'Utente è autenticato.
Scenario Principale	1. **INCLUDE** (Visualizza Roadmap). 2. L'Utente Autenticato inserisce la recensione. 3. Il sistema conferma l'avvenuta pubblicazione della recensione.
Post-condizioni	La recensione è stata pubblicata con successo.
Caso d'uso correlati	UC02
Scenario Alternativo	3a. Se la connessione col server è fallita il sistema segnala l'indisponibilità del server.

Gestione Segnalazione - Ban Utente Registrato

ID	UC23
Descrizione	Il sistema deve offrire all'Utente Amministratore una funzionalità per bandire un Utente Registrato.
Attore Primario	Utente Amministratore.
Attore Secondario	Utente Registrato.
Pre-condizioni	L'Utente è già autenticato come Utente Amministratore.

Scenario Principale	1. L'Utente Amministratore accede al pannello di amministrazione. 2. L'Utente Amministratore seleziona la segnalazione relativa al ban dell'Utente Registrato. 3. Il sistema aggiunge l'email dell'Utente Registrato alla lista dei bannati. 4. Il sistema conferma l'avvenuto ban dell'Utente Registrato.
Post-condizioni	L'Utente Registrato viene bannato (il suo profilo e tutti i contenuti da lui creati vengono rimossi dalla web app).
Caso d'uso correlati	UC04
Scenario Alternativo	3a. Se la connessione col server è fallita il sistema segnala l'indisponibilità del server.

3.3 Diagramma delle Classi

Il Diagramma delle Classi (Class Diagram) consente di descrivere tipi di entità, con le loro caratteristiche e le eventuali relazioni fra questi tipi.

Durante la fase di analisi, un Diagramma delle Classi viene realizzato per poter avere una vista statica delle entità del dominio del problema. L'attenzione viene rivolta, in particolare, su *"cosa"* il sistema deve fare e non su *"come"* lo fa. Abbiamo, dunque, realizzato tale Diagramma delle Classi (che, al momento, viene detto System Domain Model in quanto ci troviamo ancora in fase di analisi) al fine di esplicitare graficamente gli attributi principali e le associazioni che caratterizzano le diverse classi del sistema, senza scendere troppo in dettagli implementativi.

Si descrivono, di seguito, le classi e le loro associazioni presenti nel diagramma:

- *Utente Registrato:* si tratta dell'utilizzatore del servizio di web app GOING. Accede al servizio utilizzando la coppia username-password determinata in fase di registrazione. Ogni utente registrato è associato alla Roadmap, ai relativi Commenti e Recensioni;
- *Roadmap:* è la classe che descrive l'insieme di Stage da percorrere;
- *Stage:* si tratta dei punti di sosta nella Roadmap, descritti da latitudine, longitudine, titolo, descrizione, ed è associato alla Roadmap;
- *Commento:* si tratta di un commento ad una Roadmap, descritto da testo e data di

pubblicazione, è in associazione all'Utente Registrato ed in relazione di composizione alla Roadmap;

- *Recensione:* si tratta della coppia di valutazione ed eventualmente un'opinione di un Utente ad una Roadmap , è in associazione all'Utente Registrato e in relazione di composizione alla Roadmap.
- *Segnalazione*: tiene traccia delle segnalazioni effettuate dall'Utente Registrato, descritta da motivazione e tipo oggetto segnalato.

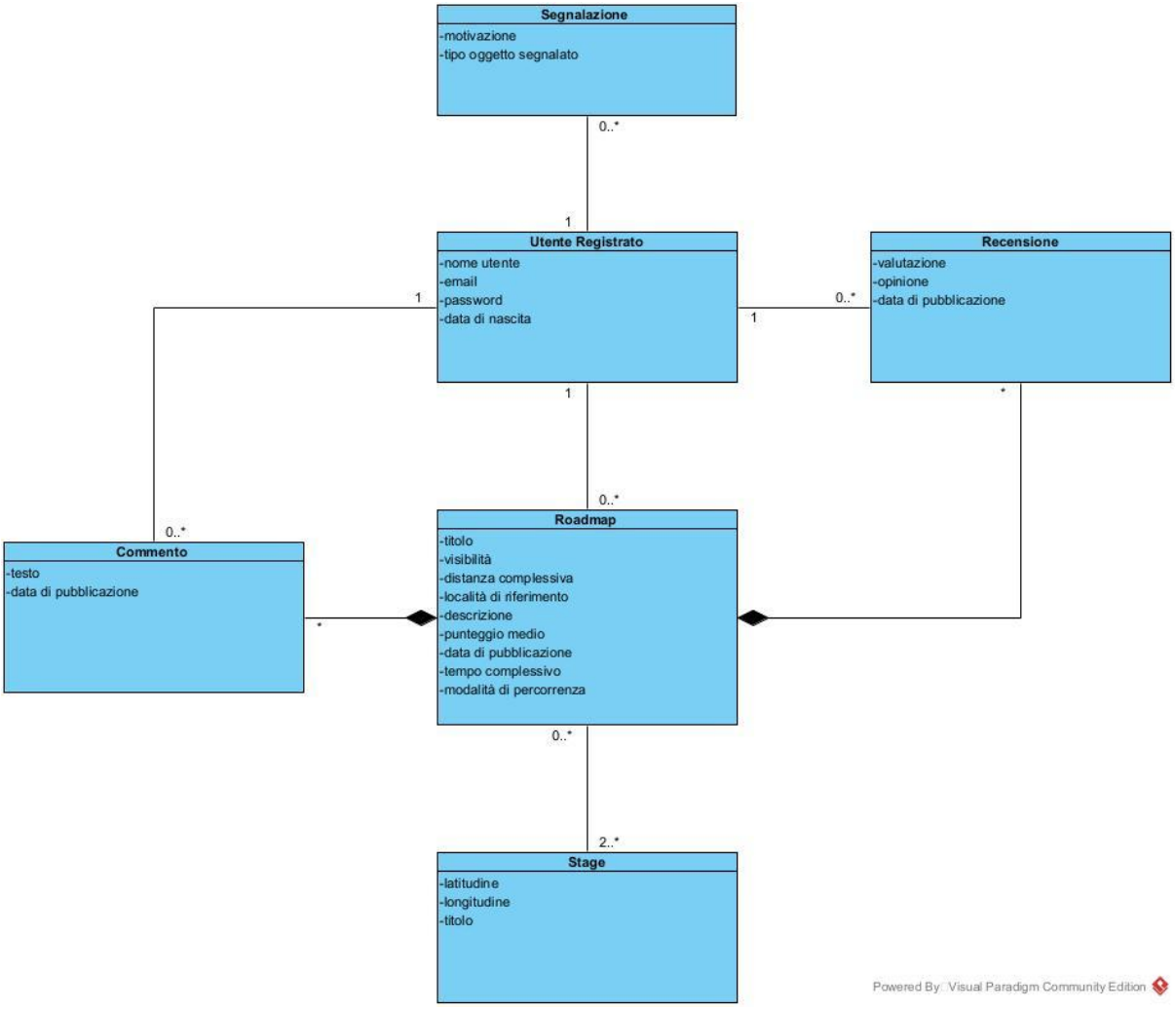


Figura 3.2 - Diagramma delle Classi

3.4 System Sequence Diagram

Il Diagramma di Sequenza (Sequence Diagram) modella le interazioni tra attori e parti di un sistema software, nell'ambito dell'esecuzione di uno scenario di esecuzione di uno Use Case. Di seguito riportiamo i Sequence Diagram realizzati per i Casi d'Uso ritenuti di maggiore interesse.

3.4.1 Ricerca di una Roadmap

Un Utente generico può fare una ricerca ed il sistema restituisce una lista di Roadmap.

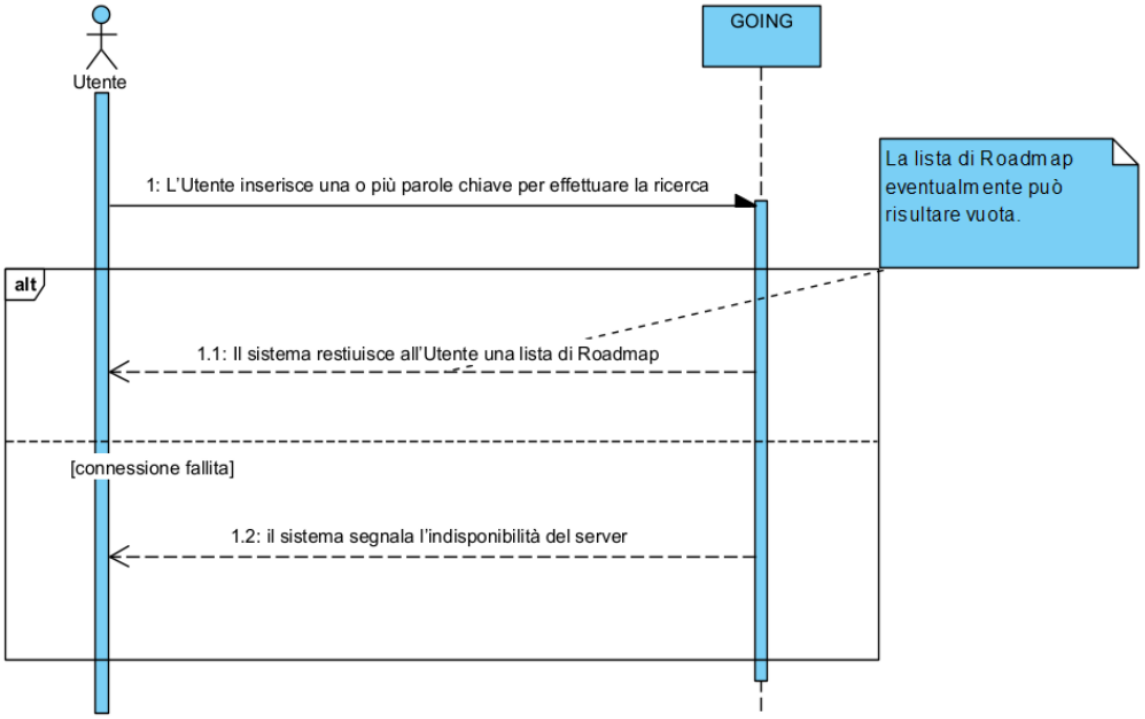


Figura 3.3 - Sequence Diagram Ricerca di una Roadmap

3.4.2 Registrazione Utente

L'Utente Non Registrato si registra, inserendo i suoi dati ed il sistema registra l'utente dopo un controllo sui dati da egli inseriti.

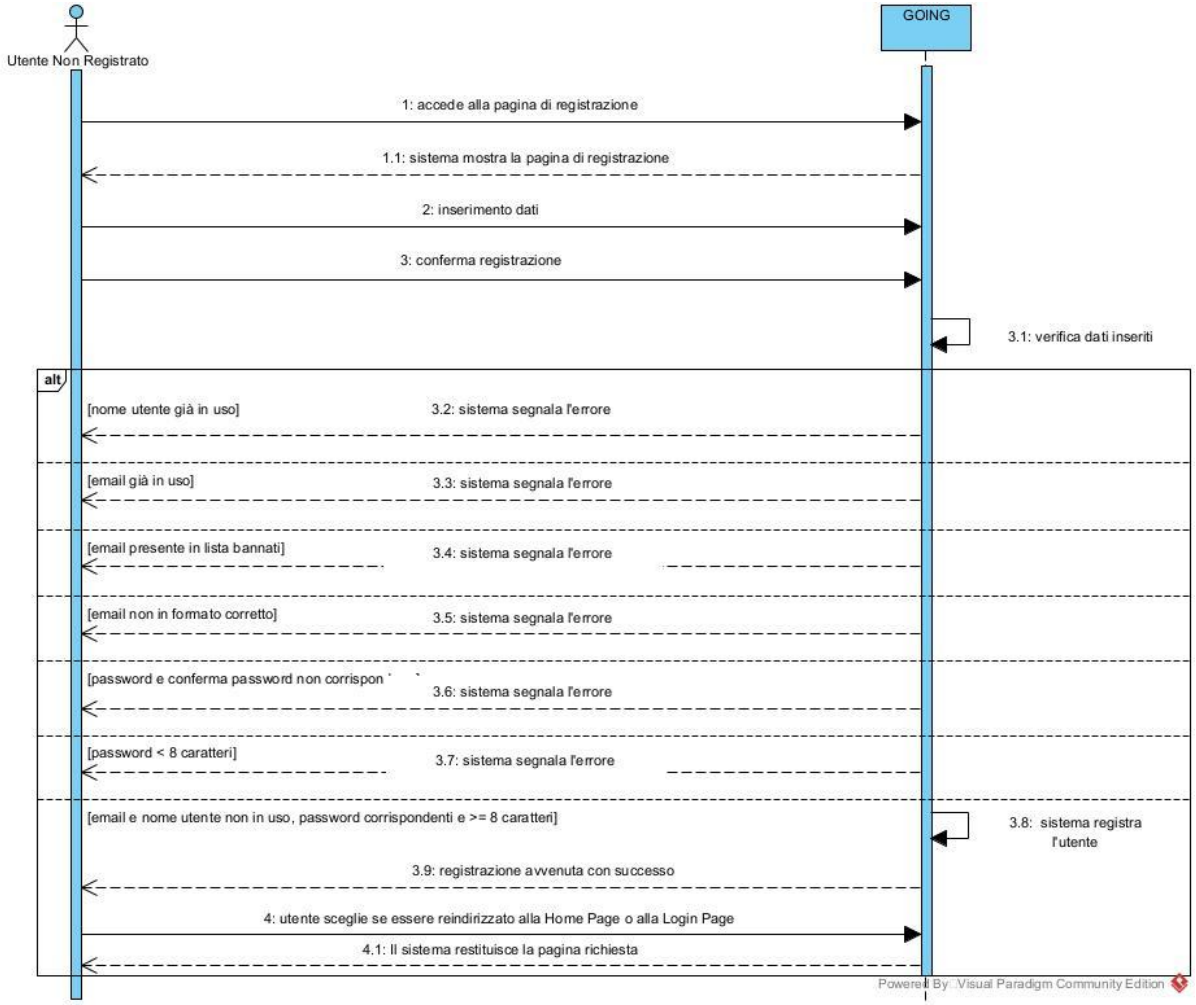


Figura 3.4 - Sequence Diagram Registrazione Utente

3.4.3 Visualizzazione di una Roadmap tramite selezione diretta

Un Utente richiede una Roadmap pubblica al sistema tramite selezione diretta.

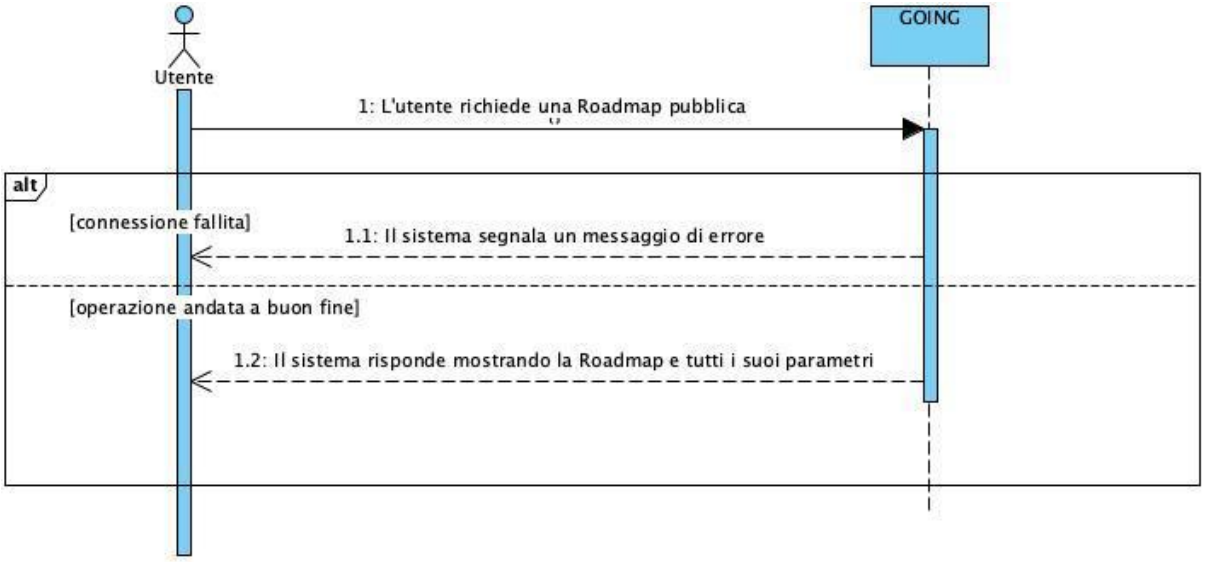


Figura 3.5 - Sequence Diagram Visualizzazione di una Roadmap tramite selezione diretta

3.4.4 Visualizzazione di una Roadmap tramite ricerca

Un Utente richiede una Roadmap pubblica al sistema dopo una ricerca.

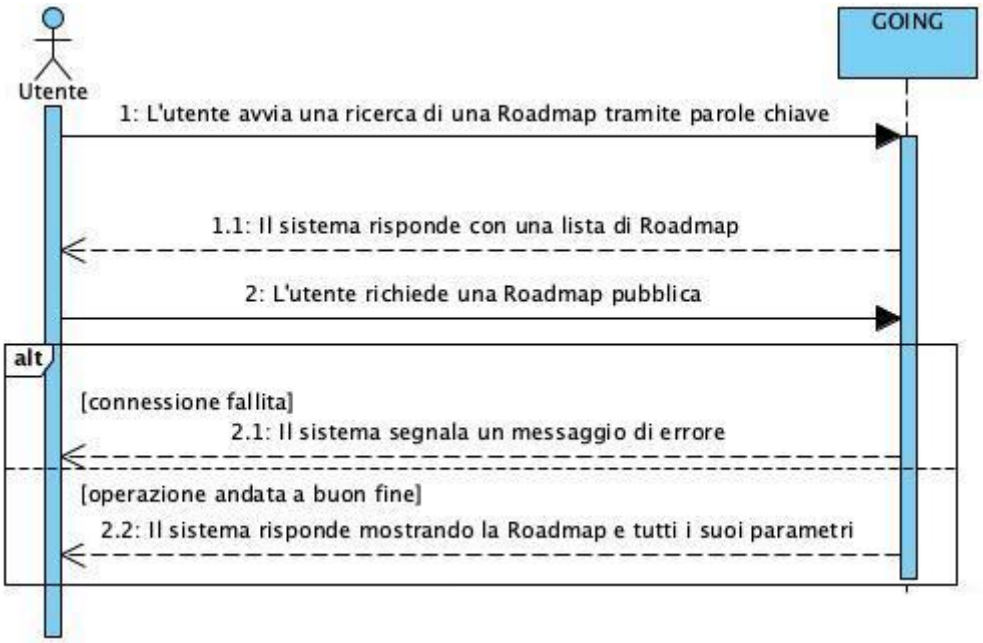


Figura 3.6 - Sequence Diagram Visualizzazione di una Roadmap tramite ricerca

3.4.5 Creazione di una Roadmap

Un Utente crea una Roadmap, inserendo titolo, modalità di percorrenza, descrizione (opzionale) e gli Stage che la compongono, con annessa durata.

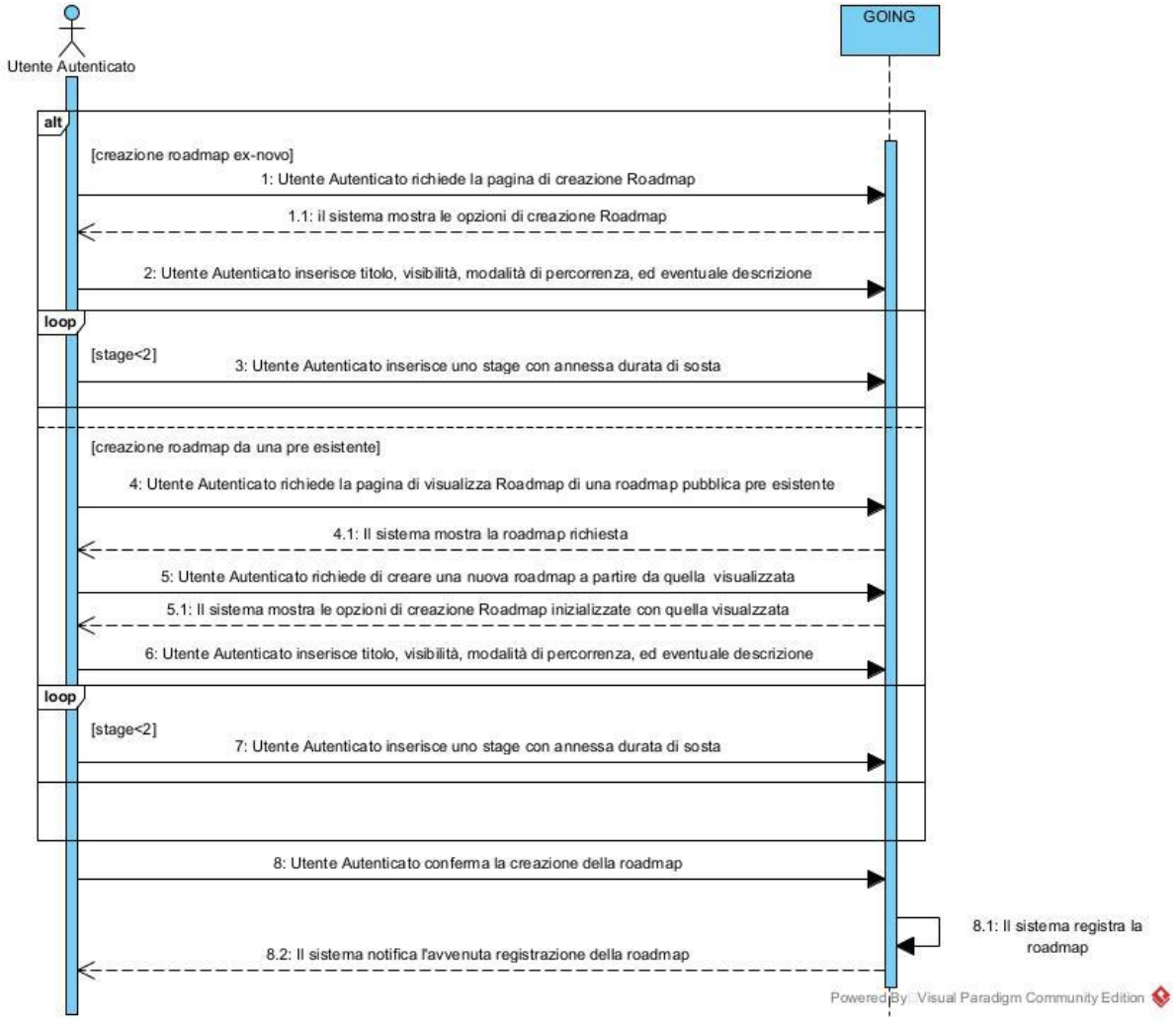


Figura 3.7 - Sequence Diagram Creazione di una Roadmap

3.4.6 Gestione Segnalazione - Ban Utente Registrato

Un Utente Amministratore accede al pannello amministratori per richiedere un ban di un Utente Registrato.

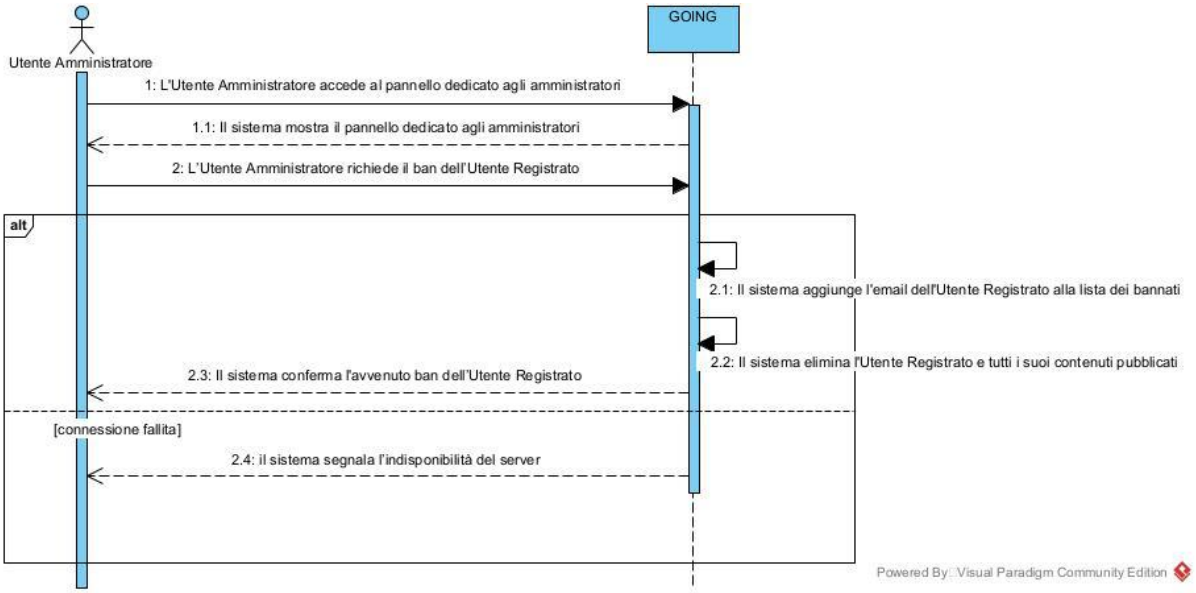


Figura 3.8 - Sequence Diagram Ban Utente Registrato

Capitolo 4

Architettura e Progettazione

Al termine della fase di analisi, si passa alla fase di definizione dell'architettura e di progettazione del software vero e proprio. A questo punto, infatti, risulta fondamentale definire un'architettura che costituisce la struttura stessa del sistema, il suo insieme di componenti ed il modo in cui essi interagiscono tra loro per realizzare le funzionalità da implementare. Al fine di godere di una serie di vantaggi quali un maggior riuso e, di conseguenza, un minor numero di errori sia durante la fase di sviluppo che nel prodotto finale stesso, si è pensato di applicare una serie di Stili e Pattern Architetturali già noti. Nei successivi paragrafi di questo capitolo, tali Stili e Pattern saranno spiegati nel dettaglio e verranno utilizzati anche per ottenere versioni raffinate dei diagrammi già realizzati e presentati in fase di analisi. Notiamo che, nel pieno rispetto dei principi agili di adattamento a requisiti mutevoli durante l'intero processo di sviluppo, ogni diagramma riportato in questo capitolo è il risultato finale di un lavoro di continuo raffinamento degli stessi.

4.1 Pattern Architetturali

I Pattern Architetturali sono soluzioni a problemi ricorrenti che si incontrano in applicazioni reali. Nel campo dell'ingegneria del software, i Pattern Architetturali sono vincolanti sull'architettura e quindi hanno un'applicabilità a livello del progetto dell'architettura e non di tutto il sistema. Notiamo che i Pattern Architetturali dipendono molto dal dominio applicativo in quanto non sono adatti alla realizzazione di qualsiasi tipo di applicazione si voglia sviluppare.

L'utilizzo di Pattern Architetturali ci ha permesso di applicare una soluzione generica a problemi diffusi ed adattarla al nostro problema. Questo comporta un maggior riuso e, di conseguenza, meno errori.

4.2 Stili Architetturali

Conclusa la trattazione dei Pattern Architetturali, passiamo agli Stili Architetturali. Anche gli Stili Architetturali sono modelli di esperienza che è possibile riutilizzare nella realizzazione di sistemi software. Bisogna notare, però, che mentre il Pattern ha un'applicabilità piuttosto specifica e, dunque, può essere utilizzato per specifici problemi ben definiti, lo Stile può essere utilizzato in molti più ambiti e fornisce una serie di indicazioni e di vincoli applicabili sia al livello dell'architettura che al livello della realizzazione finale del sistema ed anche al livello della programmazione. Dunque, lo Stile Architetturale risulta essere più *"trasversale"* rispetto al Pattern Architetturale.

Come accaduto per i Pattern Architetturali, adottare determinati Stili Architetturali comporta una serie di vantaggi quali - tra i più importanti ricordiamo - un maggiore riuso del codice ed una maggiore comprensibilità della struttura del sistema (ad esempio, nel successivo paragrafo descriveremo lo stile "client-server", la quale da sola porta con sé tante informazioni ed evita ambiguità, aumentando esponenzialmente la facilità di comunicazione tra i membri del team di sviluppo).

4.3 Stile Architetturale Client-Server

Lo Stile Architetturale scelto per la realizzazione del sistema è lo Stile Client-Server.

Lo Stile Architetturale Client-Server rientra nella categoria di Stili Architetturali detti *"Stili a Livelli"*. Uno Stile a Livelli è uno Stile che si utilizza quando si vuole decomporre un sistema più grande immaginando di stratificare l'applicazione attraverso un insieme di moduli. I moduli di un certo livello non conoscono come sono state implementate le operazioni realizzate dai moduli di un altro livello. Si effettua così un'astrazione dei vari livelli dei quali si conoscerà solo la loro interfaccia (API). Questo tipo di approccio ci permette di separare i moduli che hanno una grande coesione.

In particolare, lo Stile Client-Server è uno Stile Architetturale in cui un client (cliente) si connette ad un server allo scopo di usufruire di un certo servizio. I computer client forniscono un'interfaccia per consentire ad un client di richiedere i servizi del server e di visualizzare i risultati restituiti dal server. I server aspettano che le richieste arrivino dai client e quindi rispondono ad esse. I client non devono essere necessariamente a conoscenza delle

specifiche del sistema che fornisce loro il servizio. I client sono spesso situati su workstation o su personal computer, mentre i server si trovano altrove sulla rete, di solito su macchine più potenti. Questo modello di elaborazione è particolarmente efficace quando i client ed il server hanno ciascuno compiti distinti che eseguono regolarmente. Più client possono richiedere il servizio allo stesso server ed il server fornirà a tutti gli utenti il servizio richiesto. A sua volta, il server può eventualmente avvalersi di servizi esterni.

Tale scelta architetturale è giustificata da numerosi vantaggi, di seguito elencati:

- connessione tra Client e Server stabilita dinamicamente;
- basso accoppiamento tra Server e i Client;
- il numero di Client può scalare facilmente, anche se ciò dipende dalle capacità del Server, il quale può scalare a sua volta;
- Client e Server possono evolvere indipendentemente.

Notiamo che la serie di vantaggi da noi appena elencati hanno determinato la scelta consapevole di adottare lo Stile Client-Server nonostante fossimo a conoscenza del fatto che tale scelta può comportare potenzialmente possibili ritardi nella comunicazione dovuti alla congestione/degradazione della rete.

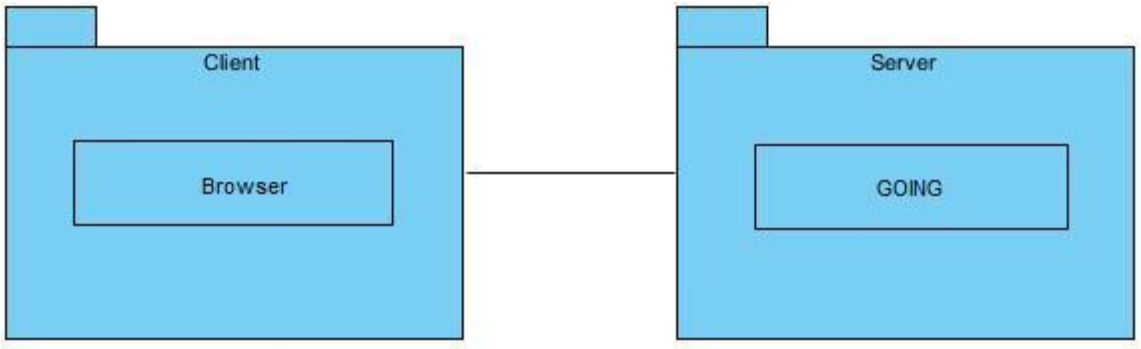


Figura 4.1 - Stile Architetturale Client-Server

4.4 Pattern Server-Side MVC

Il Pattern Architetturale scelto è il Pattern MVC (Model-View-Controller), in particolare nella sua implementazione *"Server-Side"* (spiegata nel dettaglio più avanti in questo stesso paragrafo). Tale Pattern - nella sua accezione più generale - permette di creare delle applicazioni dove la gestione dei dati è disaccoppiata dalle modalità con cui essi vengono presentati agli utenti. Tale Pattern consente, quindi, di realizzare un'architettura dove i

componenti di presentazione dei dati sono separati dai componenti che gestiscono i dati. Il pattern MVC è caratterizzato dai componenti Model, View e Controller:

- **Model**, ha il compito di mantenere e modificare i dati in modo da poter effettuare delle operazioni su di essi. Ha la responsabilità di notificare ai componenti della View eventuali aggiornamenti verificatisi in seguito a richieste del Controller, al fine di permettere alle View di presentare dati sempre aggiornati;
- **View**, ha il compito di gestire la logica di presentazione dei dati. Ogni View può implementare viste diverse sui dati e modalità diverse di interazione coi dati;
- **Controller**, ha il compito di gestire l'interazione tra Model e View. Implementa quindi la logica di controllo dell'applicazione che permette di effettuare le richieste sul Model e di indicare alla View quali dati devono essere aggiornati e visualizzati.

Da quanto detto, dunque, è possibile evincere che il Model non conosce quali Controller e quali View siano presenti nel pattern (e, per questo, è disaccoppiato). Le View, dunque, richiedono al Model di essere aggiornate sullo stato dei dati da mostrare. Nel nostro caso, tale aggiornamento avviene utilizzando la cosiddetta modalità "pull" in quanto è la View stessa a richiedere il dato al Model tramite la mediazione del Controller.

Possiamo concludere, dunque, che tale pattern è stato scelto in quanto vogliamo che l'applicazione sia facilmente modificabile, dando la possibilità di aggiungere nuove viste oppure modificando viste già esistenti in maniera agevole, e deve essere di semplice utilizzo da parte degli utenti.

Il pattern MVC è, ad oggi, uno dei pattern più utilizzati in ambito web e, dunque, si sposa perfettamente con le nostre esigenze. Ora, però, facciamo alcune considerazioni: in una applicazione web, il software deve essere eseguito sul server mentre l'utente (client) interagisce tramite il proprio browser. In tale ambito, dunque, la programmazione e la strutturazione stessa del codice risulta essere più difficoltosa del classico Pattern MVC in quanto il programma deve essere distribuito in più parti: alcuni frammenti di codice devono essere eseguiti lato client per permettere l'interazione con l'utente mentre altri devono essere eseguiti lato server per permettere l'interazione con la base di dati. Questo implica che da un lato l'applicazione deve effettuare il render dei dati ricevuti dal server in una maniera che possa permettere al client di visualizzarli, mentre dall'altro lato le informazioni relative all'utente e le sue interazioni con l'applicazione devono essere mandate, al contrario, da client a server. Fatte tali considerazioni, è proprio a questo punto, dunque, che

si pone un problema di decisione sul come organizzare le varie parti che compongono una struttura di tipo Model-View-Controller su un'architettura Client-Server e sul come permettere la comunicazione tra le parti stesse. In particolare, ci siamo domandati se la View ed il Controller dovessero essere eseguiti sul server o sul client, come abbiamo cercato di rappresentare nell'immagine sottostante.

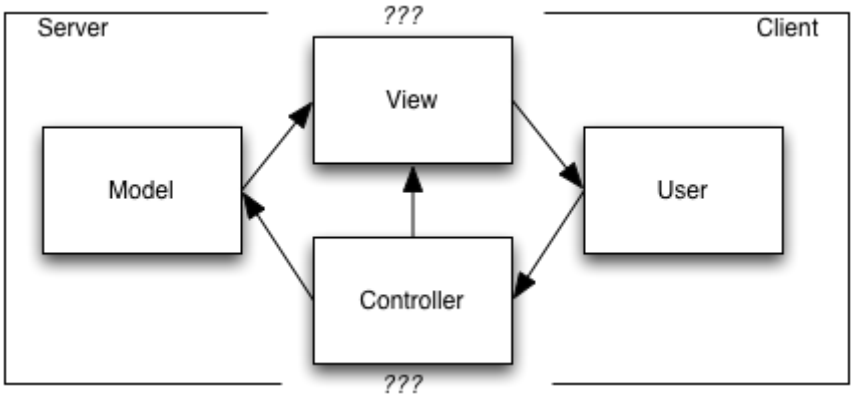


Figura 4.2 Problema di allocazione di Controller e View

A partire dal problema sopra esposto, dunque, abbiamo fissato dei punti cardine da rispettare:

- il Model sicuramente dovrà essere eseguito lato server in quanto esso è responsabile della logica di mantenimento dell'integrità degli oggetti e, dunque, dovrà essere necessariamente immune a possibili problematiche di spoofing lato client. Inoltre, il Model dovrà trovarsi necessariamente lato server anche per sincronizzare cambiamenti simultanei ai dati eseguiti da più client;
- indipendentemente da dove si troverà la View, questa dovrà sicuramente gestire la visualizzazione delle informazioni richieste dal client;
- il Controller dovrà sicuramente essere in grado di ricevere eventi. Tali eventi rappresentano l'attività dell'utente.

Fatte, dunque, le considerazioni sopra ampiamente esposte, abbiamo deciso di adottare il **Pattern Server-Side MVC** in quanto si sposava perfettamente con le nostre esigenze. Tale Pattern prevede che Model, View e Controller vengano eseguiti tutti lato server. Uno dei vantaggi più importanti del Pattern Server-Side MVC per lo sviluppatore è che se il Model è sempre in esecuzione sul server, questa architettura consente allo sviluppatore di utilizzare

lo stesso linguaggio, librerie e strutture di dati per scrivere il resto dell'applicazione e di utilizzare chiamate di procedure locali per realizzare la comunicazione tra i componenti.
In questo particolare Pattern, dunque, la View genera una pagina HTML, la quale viene inviata al client come una risposta HTTP. Al contrario, il browser del client codifica le interazioni dell'utente (URL, query, parametri POST e così via) in richieste HTTP. Il Controller, sul server, interpreta queste informazioni come cambiamenti da passare al Model e, alla fine, la View risponde o con un'altra pagina HTML oppure con risposte secondo il formato JSON che viene restituito al client.

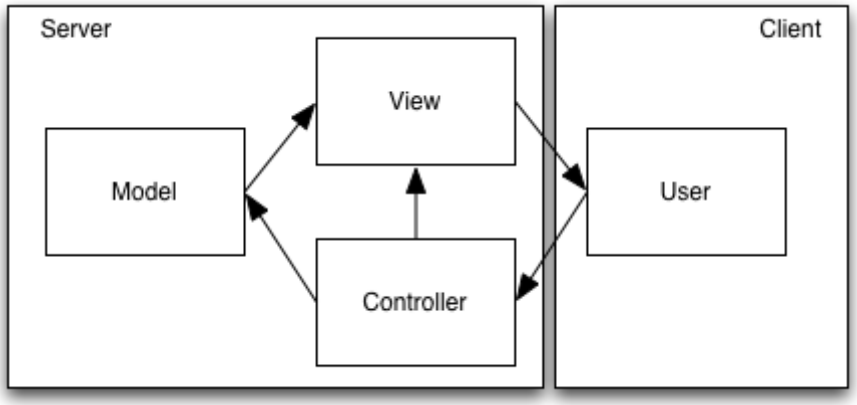


Figura 4.3 - Schema del Pattern Server-Side MVC

Volendo essere ancora più specifici, notiamo che la View non sta effettivamente mostrando le informazioni allo schermo dell'utente bensì sta gestendo dati che servono ad istruire il browser dell'utente a visualizzare tali informazioni. Dal punto di vista delle interazioni dell'utente, invece, il client non sta leggendo lo stato di tali input bensì è il browser a leggere tale stato e racchiuderlo in una richiesta HTTP. Ovviamente, l'utente non interagisce direttamente con i protocolli HTTP: è proprio il browser a mediare l'interazione dell'utente con i componenti server dell'applicazione.

4.5 Vista Componenti e Connettori

Una volta definiti i Pattern Architetturali e gli Stili Architetturali da adottare, abbiamo ritenuto opportuno realizzare come primo passo di definizione dell'architettura stessa una vista Componenti e Connettori. In questo modo abbiamo potuto definire come sarà strutturato l'insieme dei componenti che interagiscono per realizzare le funzionalità

analizzate e specificate nelle fasi precedenti, in relazione anche al rispetto dei requisiti non funzionali richiesti. Riportiamo, di seguito, la vista C&C realizzata:

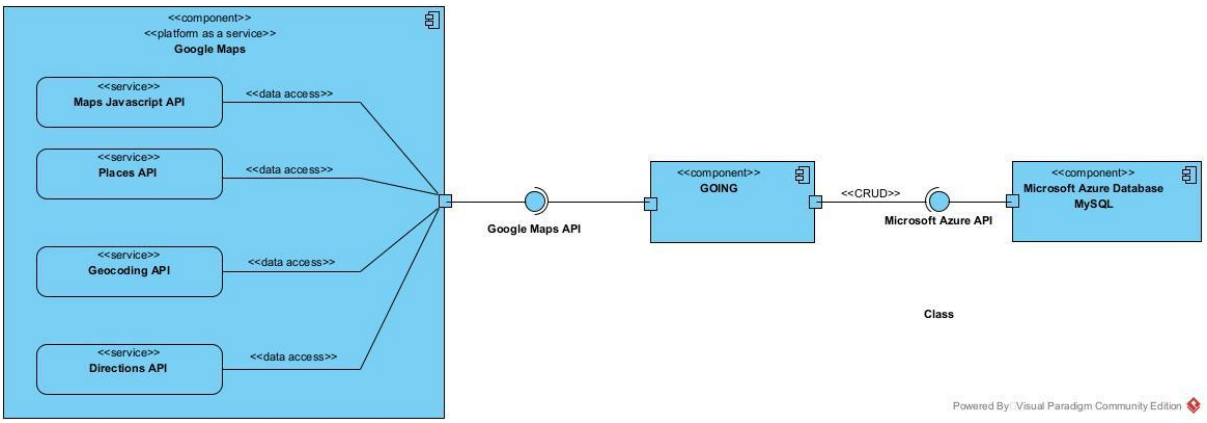


Figura 4.4 - Vista Componenti e Connettori

Riportiamo, inoltre, una breve analisi dei componenti previsti:

- *Going* → è il componente che rappresenta il sistema, il quale è stato rappresentato da un unico prodotto software nonostante la sua struttura Client-Server interna precedentemente descritta;
- *Microsoft Azure Database MySQL* → è il componente che rappresenta il servizio di archiviazione di tutte le informazioni utilizzate dal sistema;
- *Google Maps API* → stereotipato come *<<platform as a service>>* è il componente che rappresenta la piattaforma esterna che offre una serie di servizi utili alla gestione e l'utilizzo delle mappe, fondamentali per il corretto funzionamento del sistema. Siccome è sempre il nostro sistema a richiedere informazioni a tali servizi, le interazioni tra queste due entità sono state collegate al porto di uscita del componente in questione mediante una relazione di tipo *<<data access>>*. Per completezza, riportiamo di seguito una breve descrizione dei servizi utilizzati:
    - *Maps JavaScript API* → è il servizio che ci ha permesso di personalizzare la mappa con i nostri contenuti;
    - *Places API* → è il servizio che ci ha permesso di ottenere le informazioni (indirizzo, città e così via) relative ai punti di interesse (già esistenti o creati ex-novo);
    - *Geocoding API* → è il servizio che ci ha permesso di convertire indirizzi in coordinate geografiche (latitudine e longitudine) utilizzabili per posizionare i

marker sulla mappa o, viceversa, di ottenere l'indirizzo di un particolare posto a partire dalle sue coordinate geografiche tramite il servizio duale di Reverse Geocoding;

- *Directions API* → è il servizio che ci ha permesso di ottenere il percorso e le indicazioni stradali tra due Stage.

Riportiamo, infine, una breve analisi dei connettori previsti:

- *Going - Google Maps* → l'interazione tra questi componenti è stata rappresentata mediante la notazione *"lollipop"*. Tale notazione ci permette di implementare il protocollo di interfaccia richiesta/interfaccia fornita. La nostra applicazione, infatti, richiede l'interfaccia di accesso e di gestione delle mappe e Google Maps fornisce esattamente questa interfaccia;
- *Going – Microsoft Azure Database MySQL* → anche in questo caso l'interazione tra questi componenti è stata rappresentata mediante la notazione *"lollipop"* in quanto la nostra applicazione richiede l'interfaccia di accesso e di gestione delle informazioni archiviate nel database remoto ed il servizio di Microsoft fornisce le funzioni di questa interfaccia. Notiamo anche l'utilizzo dello stereotipo *<<CRUD>>* in quanto Going utilizza queste API per eseguire proprio questo tipo di operazioni.

4.6 Dinamica dei componenti

In questo paragrafo riporteremo nuovi diagrammi - detti Sequence Diagram *"di progettazione"* - i quali, caratterizzati stavolta da un livello medio di astrazione, ci hanno aiutato a definire meglio la dinamica delle interazioni che avvengono tra i componenti del sistema fin qui descritti in specifici casi d'uso ritenuti di maggiore interesse. In alcuni casi, al fine di specificare ancora meglio tali interazioni, abbiamo deciso di allegare una seconda vista dinamica: gli Activity Diagram.

4.6.1 Ricerca di una Roadmap

Sequence Diagram di progettazione

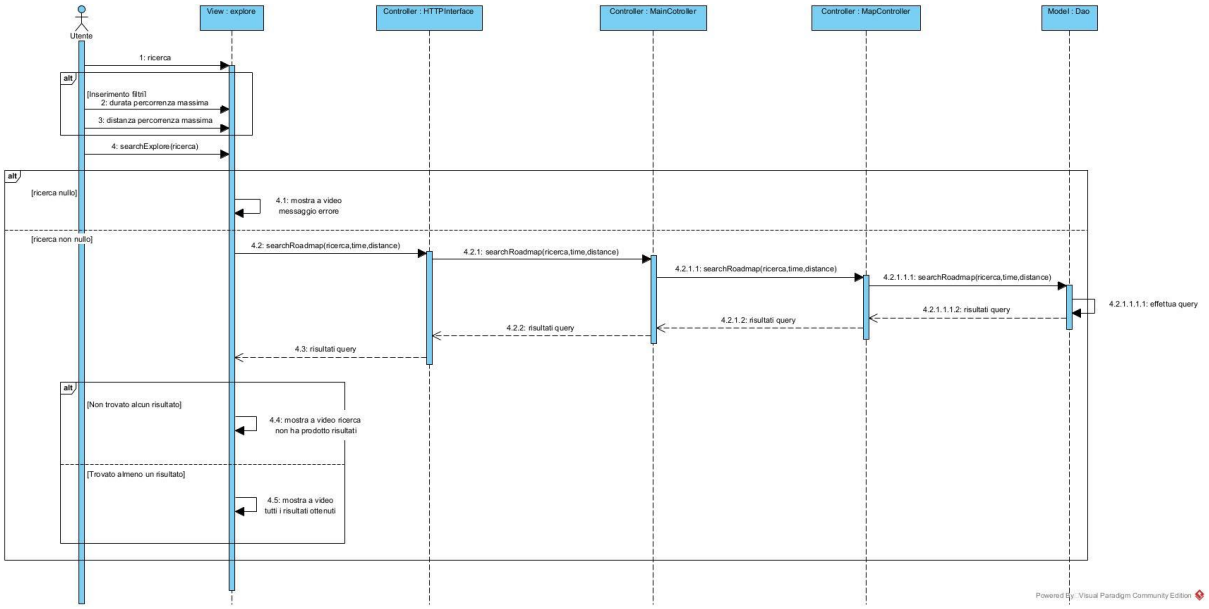


Figura 4.5 - Sequence Diagram di Progettazione Ricerca di una Roadmap

Nel diagramma di sequenza raffinato di ricerca di una Roadmap notiamo come un qualsiasi Utente può visitare la pagina *"Esplora"* e, tramite l'inserimento di una parola chiave ed eventuali ulteriori parametri di filtraggio, viene innescata la ricerca tramite la funzione *searchExplore(ricerca)*. Tale richiesta viene catturata dal Controller *HTTPInterface* se il dato inserito è non nullo, che inoltra al *mainController* ed al *mapController*, il quale interagirà con il Model *DAO* per avere i risultati della query richiesta, da far poi stampare a video nella View. Se la query non ha prodotto risultati, avremo un messaggio a video.

4.6.2 Registrazione Utente

Sequence Diagram di progettazione

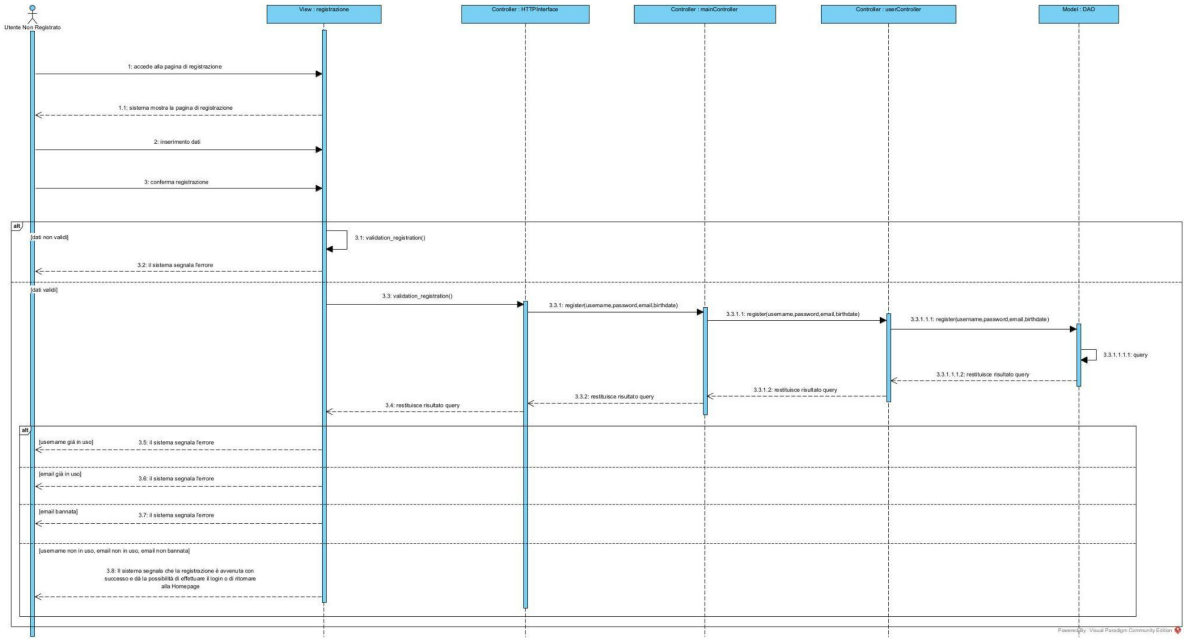


Figura 4.6 - Sequence Diagram di Progettazione Registrazione Utente

Nel diagramma di sequenza raffinato dell'operazione di Registrazione Utente vediamo come un Utente Non Registrato richiede la pagina di registrazione ed inserisce tutti i dati (username, password, password di conferma, email e data di nascita). Vengono, dunque, avviati i controlli sui dati inseriti. Avremo vari casi alternativi che portano ad un messaggio di errore a video: email non in formato valido, password e password di conferma non coincidono, password non di almeno 8 caratteri. Se la password è di almeno 8 caratteri, viene richiesta al sistema la registrazione dell'Utente. Si noti come al ritorno dal Model *DAO*, viene riportato anche un eventuale tentativo di registrazione tramite una email bannata, email già in uso o username già in uso. Se lo scenario è questo, avremo un messaggio di errore, altrimenti l'utente viene correttamente inserito nella lista degli Utenti Registrati. L'Utente, successivamente, sceglie dove essere reindirizzato.

Activity Diagram

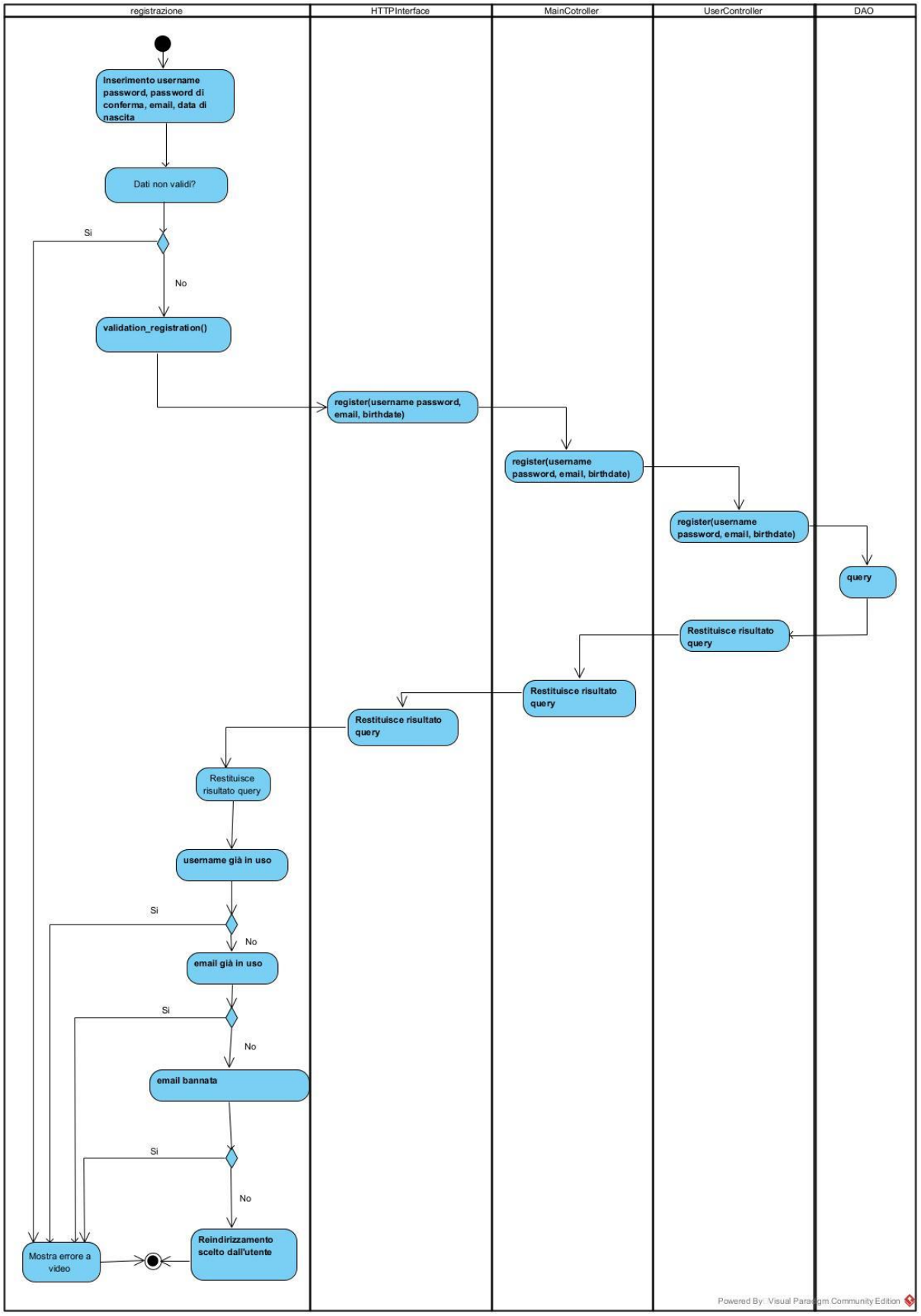


Figura 4.7 - Activity Diagram Registrazione Utente

Per specificare meglio questo caso d'uso, oltre al Sequence Diagram di Progettazione, abbiamo deciso di aggiungere una seconda vista, un Activity Diagram. In tale diagramma abbiamo una descrizione delle azioni richieste per la registrazione di un utente, distinguendo i vari scenari tra i quali quelli di input non validi e/o email nella lista dei bannati, che portano ad un messaggio di errore o, ancora, quello di registrazione effettuata con successo dove, invece, l'utente viene correttamente inserito nella lista degli Utenti Registrati.

4.6.3 Visualizzazione di una Roadmap tramite ricerca

Sequence Diagram di progettazione

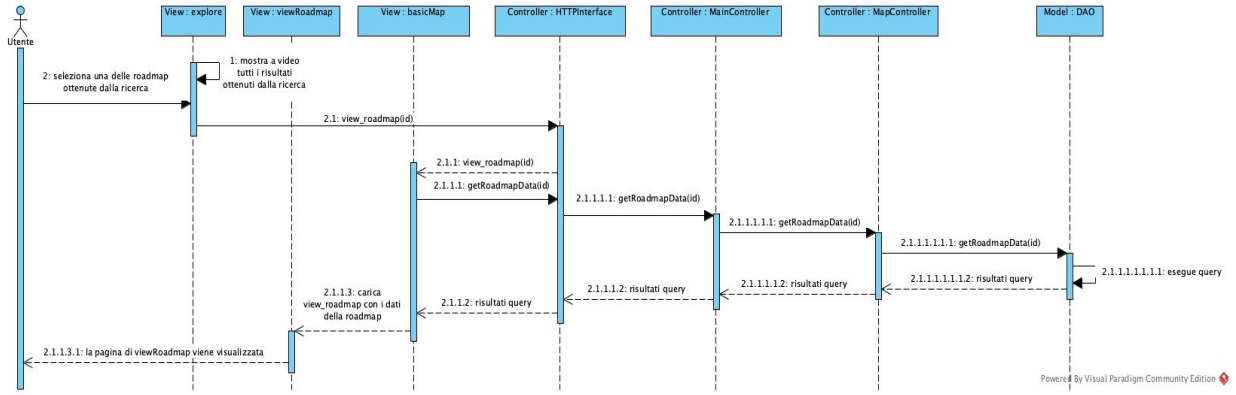


Figura 4.8 - Sequence Diagram di Progettazione Visualizzazione di una Roadmap tramite ricerca

Nel diagramma di sequenza raffinato dell'operazione di Visualizzazione di una Roadmap tramite ricerca notiamo come un qualsiasi Utente che abbia effettuato una ricerca nella pagina *"Esplora"* ed abbia ottenuto almeno un risultato può selezionare tale risultato al fine di visualizzare la Roadmap scelta. Facendo click sulla Roadmap, infatti, viene richiamato il Controller *HTTPInterface*, il quale manda l'id della Roadmap da visualizzare alla View *basicMap*. Tramite poi la funzione *getDataRoadmap(id)*, verrà poi richiamato il Model *DAO*, il quale effettuerà la query. Non abbiamo scenari alternativi, poiché la Roadmap che vogliamo visualizzare esiste sicuramente, essendo risultante della ricerca effettuata.

4.6.4 Creazione di una Roadmap

Sequence Diagram di progettazione

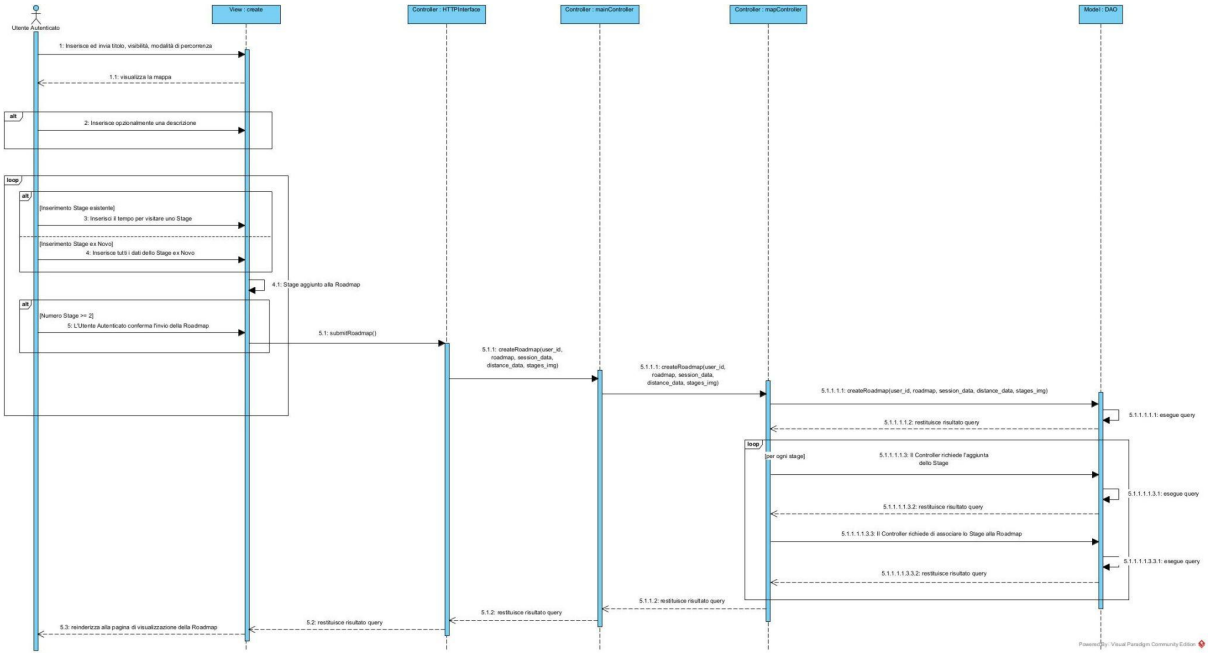


Figura 4.9 - Sequence Diagram di Progettazione Creazione di una Roadmap

Nel diagramma di sequenza raffinato dell'operazione di Creazione di una Roadmap notiamo come solo un Utente Autenticato possa accedere a questo caso d'uso, tramite la pagina *"Create"*. L'Utente Autenticato prima di tutto inserisce titolo, visibilità e modalità di percorrenza; dopodiché verrà mostrata la mappa e la possibilità di inserire una eventuale descrizione. Con la visualizzazione della mappa, l'utente potrà inserire gli *Stage* che vorrà visitare, con annessa *durata di sosta,* sia se uno stage è ex-novo (in questo caso con altri dati possibili da inserire) sia se già esistente. Dopo aver inserito almeno due Stage, potrà essere innescata dall'Utente Autenticato la *submitRoadmap()*.

La View *Create* andrà a richiamare il Controller *HTTPInterface* tramite la funzione *createRoadmap(user_id,roadmap,session_data,distance_data,stages_img)*, fino ai Controller *mainController* e *mapController.* Quest'ultimo chiederà al Model *DAO* di effettuare la query con tutti i dati della Roadmap; dopodiché, richiederà l'aggiunta di ogni Stage nel nostro database. Dunque, una query di aggiunta dello Stage al database verrà sempre effettuata e sarà essa stessa ad inserire effettivamente lo Stage nel caso in cui non fosse presente nel database. In caso contrario, lo Stage già presente nel database non verrà

inserito nuovamente. Inoltre, viene eseguita una ulteriore query per associare ogni Stage che compone la Roadmap appena creata al fine di poterla correttamente visualizzare in seguito. Dopo quest'ultima query, viene restituita all'utente la pagina di visualizzazione della Roadmap appena creata.

Activity Diagram

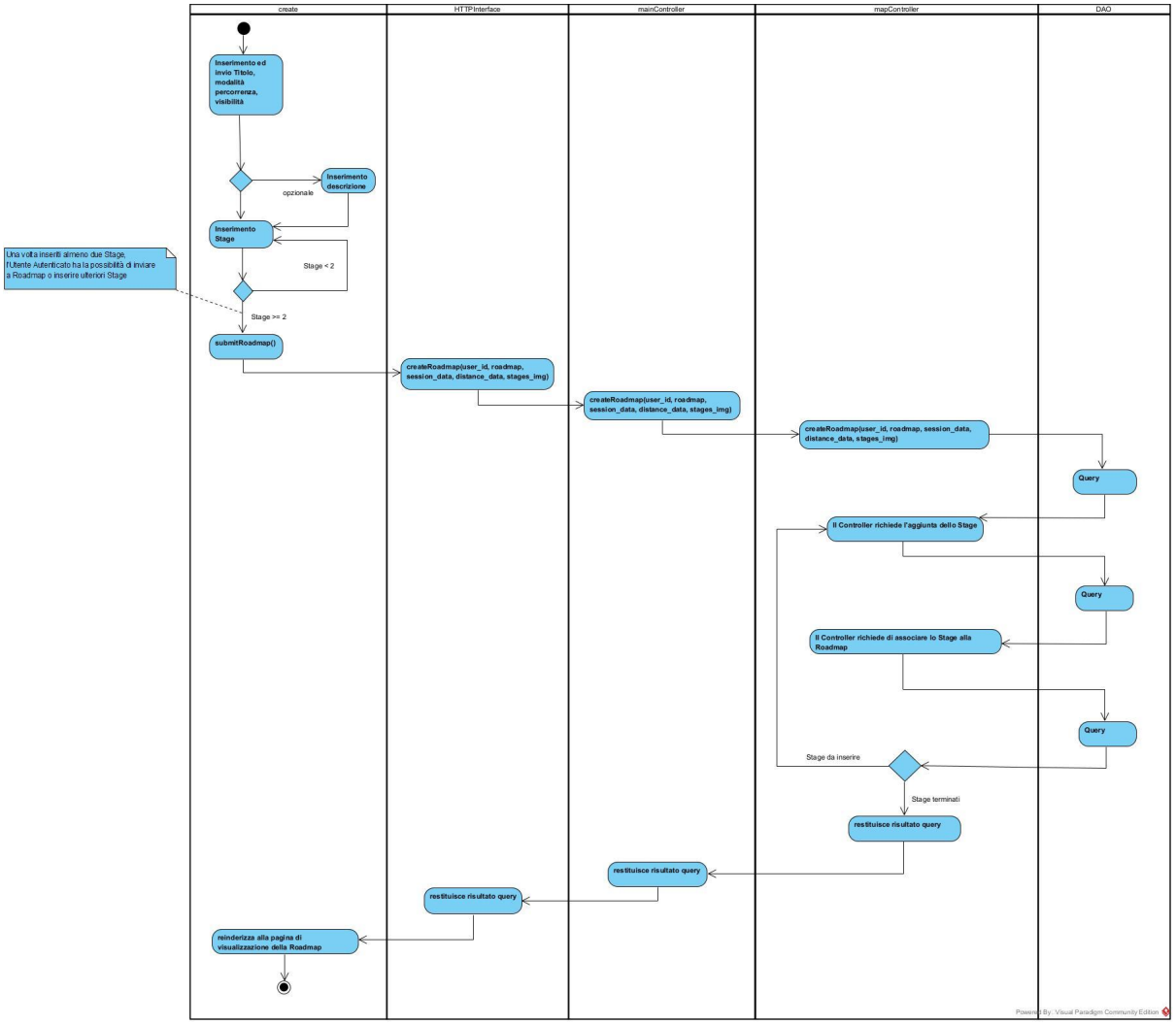


Figura 4.10 - Activity Diagram Creazione di una Roadmap

Per specificare meglio questo caso d'uso, oltre al Sequence Diagram di Progettazione, abbiamo deciso di aggiungere, anche in questo caso, una seconda vista, un Activity Diagram, dove possiamo vedere le attività effettuate e l'ordine di queste.

4.6.5 Gestione Segnalazione - Ban Utente Registrato

Sequence Diagram di progettazione

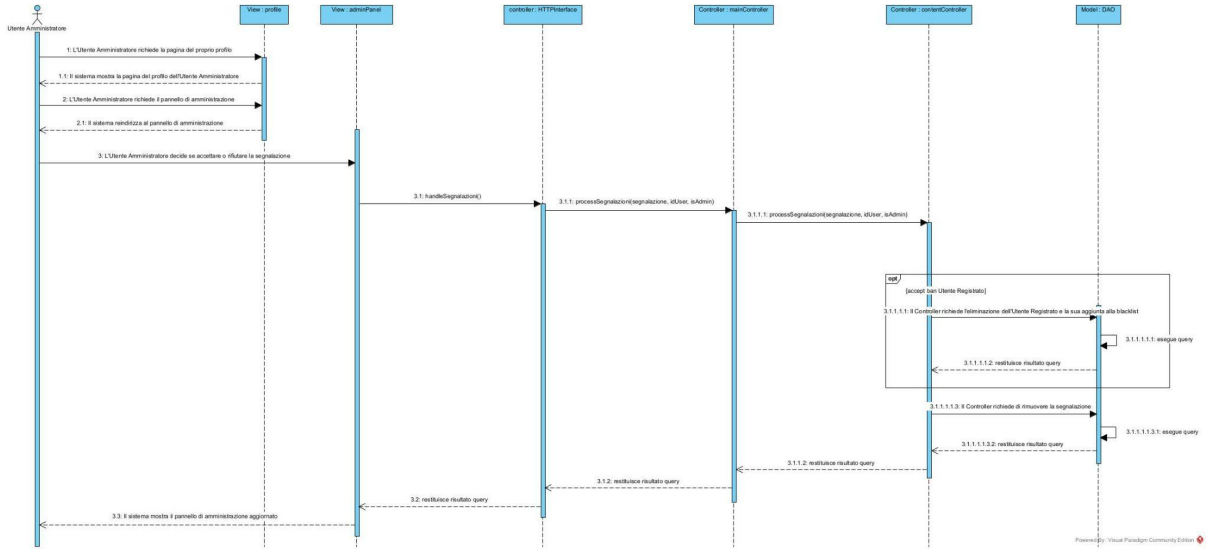


Figura 4.11 - Sequence Diagram di Progettazione Ban Utente Registrato

Nel diagramma di sequenza raffinato dell'operazione di Ban di un Utente Registrato evidenziamo come un Utente Amministratore - accedendo al proprio profilo e poi all'apposito pannello di amministrazione - possa prendere le proprie decisioni in merito alle segnalazioni giunte da altri Utenti Registrati. Dopodiché, viene propagata la funzione *processSegnalazioni(segnalazioni, idUser, isAdmin)* verso *l'HTTPInterface,* il *mainController* e *userController*. Se l'Utente Amministratore ha declinato la segnalazione, verrà solo cancellata questa. Altrimenti, oltre a questa azione, avremo la cancellazione di tutti i contenuti creati dall' Utente bannato e l'aggiunta della sua email alla blacklist.

4.7 Diagramma delle Classi

Come già spiegato nel paragrafo 4.4 di questa documentazione, il sistema è stato realizzato secondo il Pattern Architetturale Server-Side MVC. In particolare, tale Pattern ci ha permesso di implementare lato server la classica struttura Model-View-Controller. Tale struttura viene descritta nel dettaglio nel Diagramma delle Classi riportato di seguito e realizzato al fine di evidenziare il tipo degli oggetti che compongono il sistema e le relazioni statiche esistenti tra di essi.

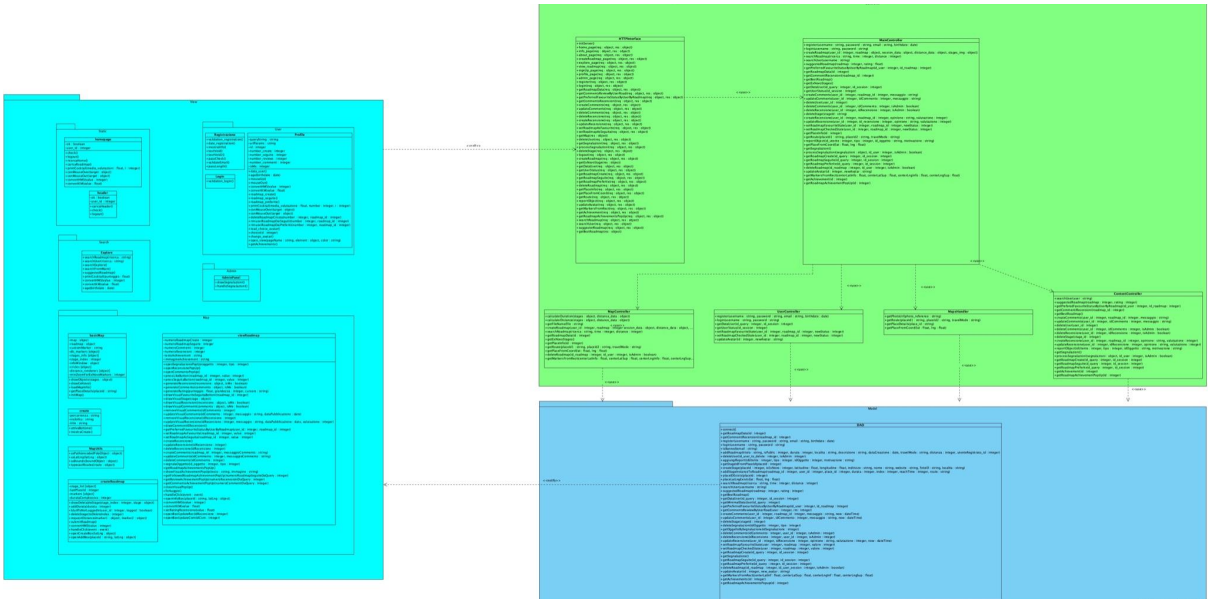


Figura 4.12 - Class Diagram

4.8 Context Diagram with Boundary

In questo paragrafo riportiamo un ulteriore diagramma che risulta utile a stabilire le relazioni che il sistema può avere con l'esterno. Trovandoci nella fase di progettazione, in tale diagramma - detto Context Diagram with Boundary - a differenza del Diagramma di Contesto mostrato nel paragrafo 1.9 di questa documentazione, abbiamo ritenuto importante evidenziare le interazioni tra il nostro sistema e gli attori, i sottosistemi ed i servizi con cui è possibile interagire e le classi utilizzate per la realizzazione di tale interazione. Notiamo che in questo particolare diagramma il sistema viene rappresentato in maniera *"parzialmente black-box"* in quanto vengono evidenziati soltanto i dettagli relativi alle classi necessarie a tale interfacciamento con l'esterno e, appunto, le entità esterne.

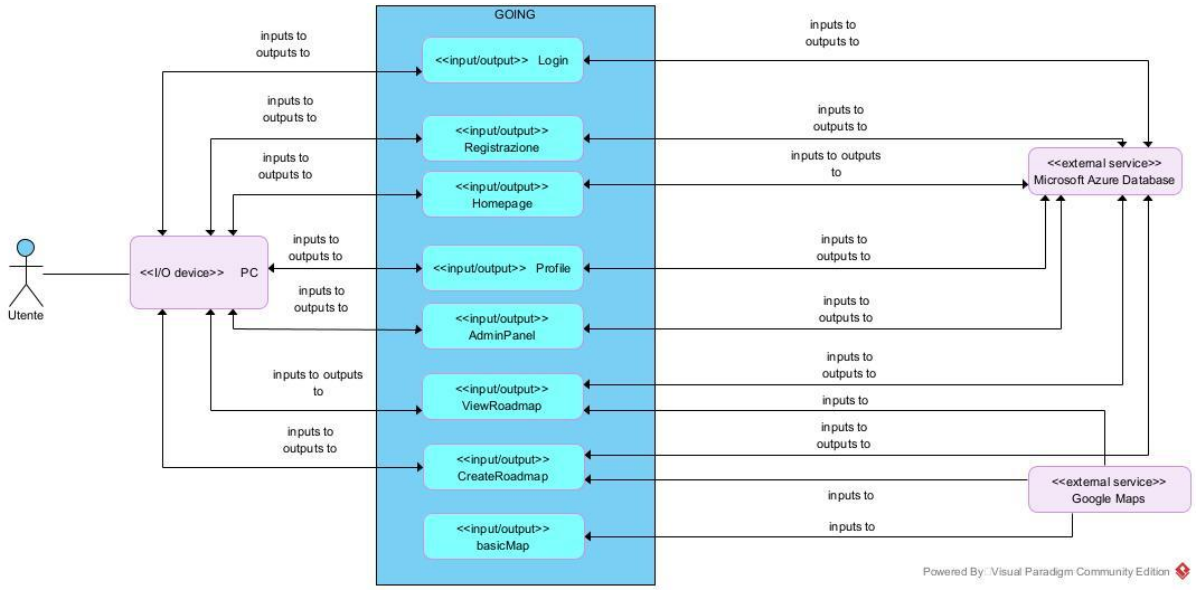


Figura 4.13 - Context Diagram with Boundary

L'unico attore esterno ad interagire col sistema è l'utente in quanto (a differenza del Diagramma di Contesto riportato nel paragrafo 1.9) abbiamo voluto racchiudere tutte le tipologie di Utente (Non Registrato, Registrato, Autenticato, Amministratore) in un unico attore al fine di favorire la chiarezza del diagramma stesso e focalizzare - come detto - l'attenzione sulle interazioni del sistema con l'esterno. Tale attore si interfaccia col sistema tramite l'utilizzo del proprio PC connesso alla rete. Tale dispositivo è stato stereotipato nel grafico come *<<I/O Device>>*. L'interfacciamento con l'utente viene gestito da classi etichettate come *<<inputs to/outputs to>>* in quanto ognuna di esse può sia ricevere gli input dall'utente che mostrare gli output corrispondenti alle richieste ricevute. I due servizi esterni, invece, indicati come *<<external service>>* sono quelli relativi a Google Maps ed al database hostato su Microsoft Azure. Nel primo caso, le interazioni vengono indicate solo con lo stereotipo di *<<inputs to>>* in quanto - tramite le apposite API descritte precedentemente - andiamo a prelevare i dati necessari alla creazione, gestione ed utilizzo della mappa, fondamentali per il corretto funzionamento del sistema. Nel secondo caso, infine, abbiamo indicato le interazioni tra il nostro sistema ed i servizi di database di Microsoft Azure tramite lo stereotipo di *<<inputs to/outputs to>>* in quanto, ovviamente, una serie di informazioni vengono sia prelevate che inviate al database stesso.

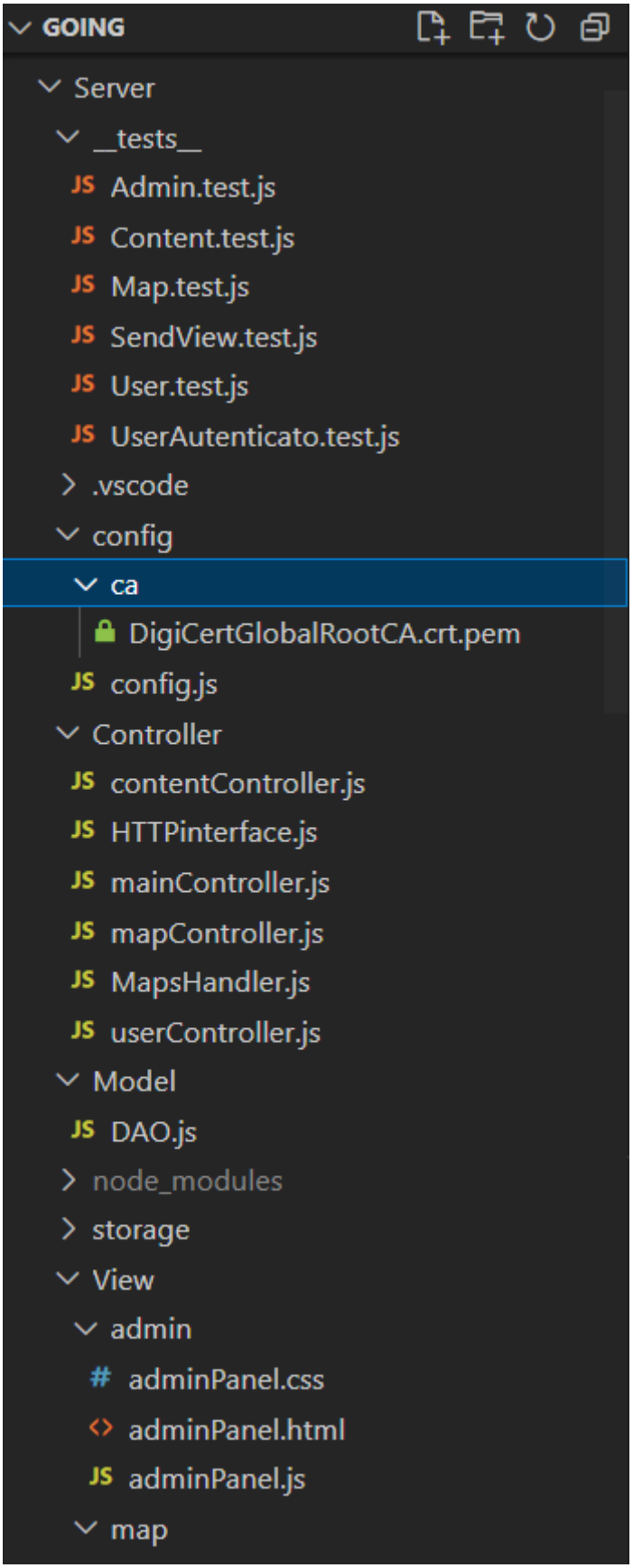
Capitolo 5

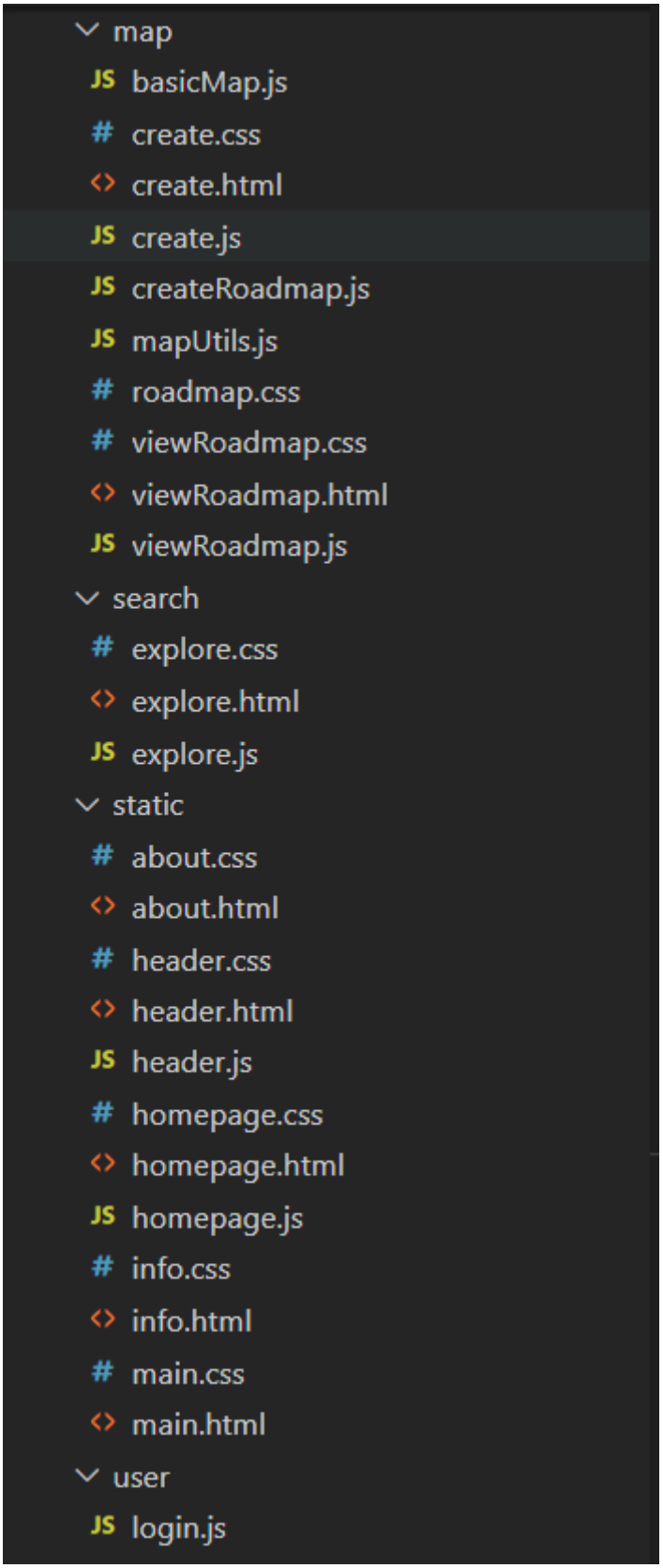
Implementazione

Terminata la fase di progettazione, si passa alla fase di implementazione vera e propria del sistema. In questo capitolo riportiamo una descrizione dettagliata di tutto il codice realizzato, approfondendo in particolare gli aspetti implementativi dei file ritenuti di maggiore interesse. Inoltre, al termine del capitolo, si riporta anche un manuale d'uso dell'applicazione realizzata al fine di chiarire il flusso di navigazione e di utilizzo del sistema stesso.

5.1 Organizzazione del codice

Nel pieno rispetto del Pattern Architetturale da noi scelto, abbiamo deciso di suddividere il nostro codice in tre cartelle: Model, View e Controller. Il linguaggio di programmazione da noi utilizzato è Javascript. Inoltre, per la realizzazione della struttura logica delle pagine web abbiamo utilizzato HTML. Come buona norma, infine, tale strutturazione logica delle pagine web è stata totalmente disaccoppiata dalla sua rappresentazione stilistica ed infatti abbiamo utilizzato anche CSS per definire la formattazione dei file HTML realizzati.
Riportiamo, di seguito, uno screenshot di presentazione dei package utilizzati al fine di evidenziare la loro organizzazione.


GOING
Server
__tests__
Admin.test.js
Content.test.js
Map.test.js
SendView.test.js
User.test.js
UserAutenticato.test.js
.vscode
config
ca
DigiCertGlobalRootCA.crt.pem
config.js
Controller
contentController.js
HTTPinterface.js
mainController.js
mapController.js
MapsHandler.js
userController.js
Model
DAO.js
node_modules
storage
View
admin
adminPanel.css
adminPanel.html
adminPanel.js
map


map
basicMap.js
create.css
create.html
create.js
createRoadmap.js
mapUtils.js
roadmap.css
viewRoadmap.css
viewRoadmap.html
viewRoadmap.js
search
explore.css
explore.html
explore.js
static
about.css
about.html
header.css
header.html
header.js
homepage.css
homepage.html
homepage.js
info.css
info.html
main.css
main.html
user
login.js

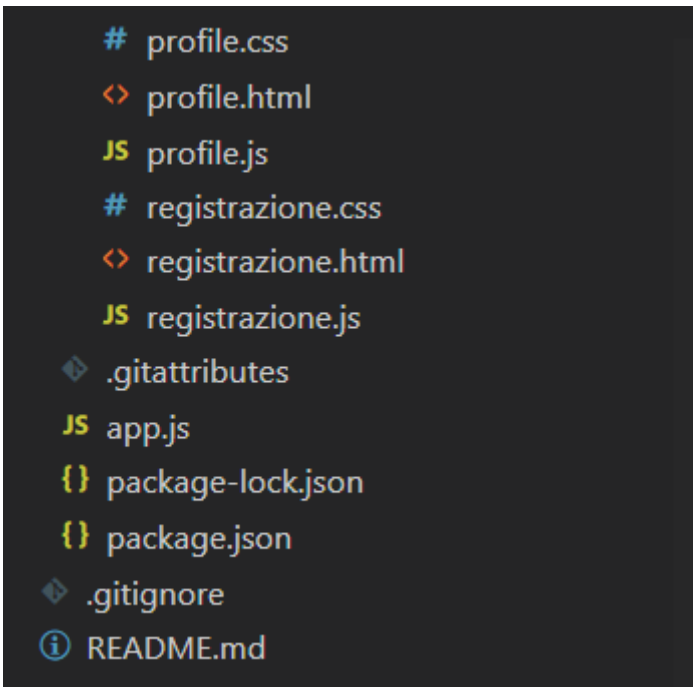


Figura 5.1 - Organizzazione Directory

Di seguito si descrivono, inoltre, le caratteristiche più importanti di ogni package:

- *model* → contiene le classi necessarie alla comunicazione con il database per il prelievo e l'inserimento dei dati da/nel database stesso;
- *view.admin* → contiene la classe necessaria all'implementazione delle funzioni relative agli Utenti Amministratori;
- *view.map* → contiene le classi necessarie all'implementazione di tutte le funzioni necessarie alla gestione, visualizzazione ed interazione con la mappa, necessaria per le assolvere tutte le funzionalità più importanti offerte dall'applicazione stessa;
- *view.search* → contiene la classe necessaria all'implementazione della pagina *"Esplora"* da cui qualsiasi Utente (Registrato o Non Registrato) può effettuare una ricerca di Roadmap o di altri Utenti Registrati;
- *view.static* → contiene le classi relative alle pagine "statiche" del sito, tra cui l'homepage, l'header (che viene richiamato in ogni altra pagina del sito stesso), la pagine di *"Info"* in cui è possibile trovare alcune informazioni utili riguardo l'applicazione e quella di *"Chi Siamo"* in cui, invece, è possibile contattare gli sviluppatori e visualizzarne una breve bio;
- *view.user* → contiene le classi necessarie all'implementazione delle funzioni più strettamente correlate all'utente stesso. In particolare, troviamo le classi relative alla

registrazione al sito, al login e, infine, di visualizzazione del proprio profilo o di quello di un altro Utente Registrato;

- *controller* → contiene le classi necessarie all'implementazione delle funzioni relative alla intermediazione tra View e Model nel pieno rispetto del Pattern MVC in cui da un lato il Model fornisce dati utili alla realizzazione della user interface (UI) mentre dall'altro lato gli input provenienti dalla View (e determinati dalle interazioni dell'utente) vengono convertiti in azioni da eseguire sul Model.

Notiamo, inoltre, che come è possibile evincere dalla precedente immagine, oltre alle cartelle strettamente correlate all'organizzazione relativa al Pattern MVC, sono state realizzate anche altre risorse:

- *config* → cartella utilizzata per inserire i file di configurazione del sistema. In particolare, al suo interno troviamo il certificato Digicert per la comunicazione SSL con il database hostato sulla piattaforma Microsoft Azure ed un file relativo alla configurazione dell'accesso a tale database, oltre che le due chiavi per poter accedere alle API di Google Maps come spiegheremo successivamente in un apposito paragrafo di questa documentazione;
- *storage* → cartella utilizzata per archiviare tutti gli assets ed altri oggetti grafici relativi all'applicazione;
- *gitattributes, gitignore* → file relativi alla corretta configurazione della repository Git utilizzata per permettere la condivisione del codice tra tutti i membri del team di sviluppo;
- *vscode, node_modules, package, package-lock* → file e cartelle relative alla configurazione dell'IDE utilizzato, all'installazione di moduli utili a Node.js.

5.2 Descrizione dei file realizzati

Di seguito si riporta una spiegazione di tutti i file realizzati, suddivisi per i propri package di appartenenza sopra descritti:

5.2.1 View

Nella View sono presenti le diverse pagine visualizzabili dall'utente mentre naviga nell'applicazione. Nella seguente descrizione i file sono stati raccolti in sottoparagrafi al fine di migliorarne la leggibilità e la chiarezza. Tali sottoparagrafi sono totalmente conformi alle sottocartelle in cui il codice stesso è stato suddiviso e la cui strutturazione è stata mostrata e descritta nel dettaglio nel paragrafo 5.1.

View → admin

- *adminPanel* → costituito da un file JavaScript, un file HTML ed un file di stile CSS, presenta le funzioni necessarie per la visualizzazione del pannello di controllo a cui solo un Utente Amministratore può accedere. All'interno troviamo tutte le funzioni necessarie alla corretta visualizzazione delle segnalazioni provenienti dagli altri Utenti Registrati ed alla loro gestione. Un Utente Amministratore, infatti, potrà decidere di accettare tale segnalazione (rimuovendo, così, dalla piattaforma il contenuto segnalato) oppure rifiutare la segnalazione (conservando, quindi, il contenuto segnalato all'interno della piattaforma senza effettuare ulteriori modifiche).

View → map

- *basicMap* → costituito da un file JavaScript, presenta le funzioni necessarie alla corretta inizializzazione della mappa per quanto riguarda le funzionalità di creazione di una nuova Roadmap da parte degli Utenti Registrati. Oltre alla funzione *initMap* che permette di inizializzare la configurazione basilare per la creazione della mappa e di impostare alcune personalizzazioni della stessa (tra cui - ma non solo - la tipologia di interfaccia della mappa, il livello di zoom iniziale e così via), tale classe contiene anche le funzioni necessarie per disegnare diversi oggetti sulla mappa tra cui gli Stage che l'Utente che sta creando la Roadmap aggiunge man mano durante il processo di creazione della Roadmap stessa (funzione *drawObjects*) o, ancora, gli

Stage ex-novo già creati precedentemente da altri Utenti Registrati e che, una volta inseriti nel nostro database, sono visualizzabili ed utilizzabili da altri Utenti Registrati come se fossero normali punti di interesse presenti nella classica mappa di Google Maps (funzione *drawExNovoStages*). Infine, le funzioni *getRoadmapData* e *getPlaceDetails* permettono rispettivamente di ottenere tutti i dettagli visualizzabili dall'Utente durante la creazione di una Roadmap (numero di Stage aggiunti, dettagli relativi ai singoli Stage quali durata di sosta, distanza di percorrenza e durata complessiva della Roadmap) ed anche informazioni relative proprio allo Stage preso in considerazione (indirizzo, località, eventuali informazioni aggiuntive quali foto o sito web e così via);

- *create* → costituito da un unico file JavaScript, si occupa di gestire tutto il flusso di creazione di una nuova Roadmap. In particolare, un Utente Registrato dovrà necessariamente inserire prima un titolo alla Roadmap ed indicarne la visibilità e la modalità di percorrenza (a piedi o in macchina) per poi poter inserire una descrizione (opzionale) e poter passare all'interazione stessa con la mappa che consente la creazione vera e propria della Roadmap tramite l'aggiunta dei diversi Stage che la compongono e che, invece, è gestita tramite le funzioni della classe *createRoadmap*, analizzata subito dopo;
- *createRoadmap* → costituito da un file JavaScript, un file HTML ed un file di stile CSS, contiene tutte le funzioni relative all'interazione da parte dell'Utente Registrato con la mappa al fine di creare una nuova Roadmap da pubblicare sulla piattaforma. In particolare, notiamo la presenza della funzione *blurIfNotLoggedIn* alla quale vengono passati due parametri relativi all'id dell'Utente Registrato ed al suo stato (loggato oppure no). Se, infatti, l'utente non dovesse essere loggato, il sistema presenterà un popup che inviterà l'utente a fare l'accesso al proprio account (se già registrato) o ad iscriversi al sistema. Come specificato in fase di analisi, infatti, vogliamo che tutti gli Utenti (Registrati o Non Registrati) possano visualizzare una qualsiasi Roadmap (con visibilità pubblica) ma la funzionalità di creazione di una nuova Roadmap, invece, dovrà essere esclusiva degli Utente Registrati. Una volta effettuato l'accesso, dunque, l'Utente Autenticato potrà creare una nuova Roadmap. L'utente potrà creare una nuova Roadmap interagendo con la mappa. Si potranno creare Stage partendo da quelli già messi a disposizione da Google interagendo con i marker pre-esistenti

oppure creando marker ex novo. Considerato che ogni utente può aggiungere degli Stage ex novo alla propria Roadmap, gli utenti successivi potranno riutilizzare gli stessi Stage interagendo con i rispettivi marker. Questo potrebbe portare a problemi di scalabilità in quanto, il client dovrebbe richiedere al server la lista di tutti gli Stage ex novo al caricamento della mappa e renderizzarli successivamente. Per evitare ciò, una funzione denominata *getMarkersFromRect,* richiamata sul caricamento della pagina e ogni volta che ci si sposta nella mappa, ha lo scopo di individuare un'area geografica leggermente più estesa dell'area visualizzata al momento dall'utente. Tale area geografica sarà quindi delimitata da una finestra scorrevole. Da essa, ne estrae tutti gli stage ex novo presenti nel database corrispondenti a tale area e solo quelli vengono spediti all'utente. Si risolve così un importante problema di scalabilità. La fase di creazione di una nuova Roadmap è gestita tramite una serie di funzioni quali *drawDeletableStage* che permette di inserire nella punto di interazione dell'Utente con la mappa un nuovo Stage, aggiungendone anche dettagli quali la durata di permanenza e - nel caso in cui lo Stage sia creato ex-novo - un nome, un'immagine (opzionale) ed un sito web (opzionale). La funzione *deleteStage*, invece, permette di rimuovere uno Stage già aggiunto alla Roadmap. Tale funzione ricalcola automaticamente la posizione degli altri Stage all'interno della Roadmap tramite un apposito indice (*toDeleteIndex*) ed, inoltre, vengono ricalcolate anche le informazioni relative al percorso da seguire ed alla durata complessiva della Roadmap. La distanza tra due nodi, invece, viene calcolata tramite la funzione *requestDistance*. Infine, la funzione *submitRoadmap* permette di confermare la pubblicazione della Roadmap sulla piattaforma e di renderla accessibile a chiunque se la visibilità sarà stata impostata su "pubblica". Nel caso in cui la visibilità della Roadmap sarà impostata su "privata", invece, soltanto il creatore della Roadmap potrà visualizzarla;

- *mapUtils* → costituito da un unico file JavaScript, contiene una serie di funzioni di utilità relative alla mappa.
- *viewRoadmap* → costituito da un file JavaScript, un file HTML ed un file di stile CSS, contiene tutte le funzioni relative alla visualizzazione di una Roadmap. Oltre alle funzioni di visualizzazione delle informazioni e degli Stage che costituiscono la Roadmap stessa (funzione *drawVisualStage*), troviamo altre funzionalità aggiuntive che costituiscono il cuore delle funzionalità di interazione della community. Tra

queste troviamo le funzionalità di aggiunta della Roadmap ai Preferiti o ai Seguiti. In particolare la funzione *drawVisualFavouriteSeguitaBottoni* gestisce la visualizzazione e l'interfaccia grafica dei relativi bottoni, le funzioni *pressLikeButton* e *pressSeguitaButton* vengono richiamate nel momento in cui un Utente Autenticato preme su uno dei due bottoni corrispondenti e, a loro volta, richiamano rispettivamente le funzioni *setRoadmapAsFavourite* e *setRoadmapAsSeguita* le quali permettono di effettuare una chiamata POST per impostare la Roadmap considerata rispettivamente come Preferita o come Seguita. Infine, la funzione *getPreferredFavouriteStatusByUserByRoadmap* permette, tramite una chiamata GET, di ottenere le informazioni relative all'Utente Autenticato rispetto alla particolare Roadmap visualizzata al fine di conoscere se quest'ultima sia stata aggiunta tra le Preferite/Seguite da parte dell'utente stesso. Altre funzionalità messe a disposizione della community sono la possibilità di lasciare una propria recensione (costituita da valutazione ed opinione) e/o un proprio commento. Queste features sono assolte dalle relative funzioni di creazione, modifica e cancellazione di recensioni e commenti e dalle corrispondenti funzioni per la visualizzazione dell'interfaccia grafica che le caratterizza. Agli Utenti Autenticati viene anche offerta la possibilità di segnalazione agli Utenti Amministratori di qualsiasi contenuto ritenuto poco consono (funzione *segnalaOggetto*). Un'altra caratteristica che contraddistingue la piattaforma è la possibilità di "forkare" una Roadmap pubblica esistente e di modificarla a proprio piacimento per, eventualmente, ri-pubblicarla a proprio nome. Questa feature è assolta dalla apposita funzione *forkaggio*. Infine, l'applicazione offre anche funzionalità relative alla gamification e, per questo, al caricamento della pagina di visualizzazione di una Roadmap (funzione *onLoad*) vengono opportunamente calcolate le informazioni relative all'Utente Autenticato che sta visualizzando tale Roadmap al fine di assegnargli - al soddisfacimento di particolari requisiti - uno o più achievement. Tale funzionalità si risolve con l'assegnazione di tali achievement tramite la comparsa a schermo di un popup a seconda dell'achievement ottenuto (funzioni *getRoadmapAchievementsPopup*, *showVisualAchievementPopup*, *getFollowedRoadmapAchievementPopup*, *getReviewAchievementPopup*, *getCommentAchievementPopup*) e della sua scomparsa automatica dopo pochi secondi (funzione *closeVisualPopup*).

View → search

- *explore* → costituito da un file JavaScript, un file HTML ed un file di stile CSS, presenta tutte le funzioni necessarie per l'operazione di ricerca da parte di un qualsiasi Utente (Registrato o Non Registrato) di Roadmap pubbliche o di altri Utenti Registrati al fine di visualizzarne il profilo. Al caricamento della pagina (funzione *onLoad*) vengono visualizzate alcune Roadmap suggerite. Appena l'utente effettua una ricerca per parola chiave vengono visualizzate le Roadmap e/o gli Utenti Registrati corrispondenti ai requisiti richiesti. Per quanto riguarda le Roadmap, viene data all'utente che effettua la ricerca anche la possibilità di filtrare in base ad una serie di parametri quali la massima durata di percorrenza e la massima distanza di percorrenza.

View → static

- *header* → costituito da un file JavaScript, un file HTML ed un file di stile CSS, viene richiamato in ogni pagina del sito in quanto presenta le funzioni e l'interfaccia grafica dell'header del sito stesso. La funzione *check*, in particolare, verifica se l'utente ha effettuato il login. In base a ciò, in alto a destra viene eventualmente mostrata l'immagine profilo dell'utente in modo da fornire, al click di tale immagine, un collegamento diretto alla pagina del profilo dell'utente stesso. Inoltre, l'header presenta anche un bottone di logout che, tramite l'apposita funzione, permette all'Utente Autenticato di effettuare la disconnessione;
- *homepage* → costituito da un file JavaScript, un file HTML ed un file di stile CSS, è la pagina principale dell'app e, per questo, presenta una serie di collegamenti a tutte le sezioni del sito. Inoltre, tramite la funzione *caricaRoadmap* vengono visualizzate a schermo alcuni suggerimenti tra le migliori Roadmap presenti nella piattaforma (in particolare, quelle con valutazione maggiore). Dalla homepage, inoltre, tramite la funzione *ricercaHome* è possibile effettuare una ricerca che rimanda direttamente alla pagina Esplora già descritta in precedenza.

View → user

- *login* → costituito da un unico file JavaScript, contiene le funzioni necessarie per le operazioni di accesso al sito da parte di un Utente Registrato. In particolare, la funzione *validation_login* gestisce tali operazioni di autenticazione;
- *profile* → costituito da un file JavaScript, un file HTML ed un file di stile CSS, è la pagina che permette di mostrare il profilo di un Utente Registrato. Al suo interno presenta varie funzioni che gestiscono la visualizzazione a schermo di diverse informazioni relative a tale utente quali il suo username, l'età e la mail con la quale si è registrati al sistema. Inoltre, le funzioni *roadmap_create*, *roadmap_seguite* e *roadmap_preferite* permettono di visualizzare i rispettivi elenchi di Roadmap create, seguite e preferite dell'utente. In particolare, il parametro *isMe* consente di verificare se l'utente che sta visualizzando il profilo è proprio il proprietario di quel profilo in modo tale da permettere la visualizzazione non solo delle Roadmap pubbliche ma anche di quelle private. Tale parametro, inoltre, viene utilizzato anche per mettere a disposizione - tramite la funzione *change_avatar* - la possibilità di modificare l'avatar del proprio profilo. Infine, la funzione *getAchievements* consente di visualizzare i badge relativi agli achievement guadagnati dall'utente.
- *registrazione* → costituito da un file JavaScript, un file HTML ed un file di stile CSS, presenta tutte le funzioni necessarie per guidare l'utente durante il processo di registrazione alla piattaforma. In particolare, troviamo funzioni relative alla gestione di campi obbligatori nulli (*textVoid*), al controllo dell'uniformità del campo "password" e "conferma password" (*passCheck*) e della sua lunghezza (*passLength*) ed alla validazione della email inserita (*validateEmail*).

5.2.2 Controller

Nel Controller troviamo i controllori che gestiscono la logica dell'applicazione.

- *HTTPInterface* → rappresenta l'interfaccia che il Controller espone per la comunicazione con il package View. In tale file, infatti, troviamo tutte le pagine front-end che vengono mostrate all'utilizzatore del sito e tutte le chiamate back-end che, invece, vengono poi passate al *requestController* (spiegato poco più in basso);

- *MapsHandler* → è il controllore che gestisce l'interazione con le API di Google Maps. In questo file, infatti, troviamo svariate funzioni che ci consentono di ottenere le coordinate (latitudine e longitudine) di un certo punto di interesse, le informazioni relative a quel punto e, ancora, il percorso minimo per andare da un punto all'altro (tali punti di interesse, come ampiamente spiegato nel corso di questa documentazione, sono quelli da noi ri-denominati "Stage"). Notiamo la presenza di due differenti API Key di Google Maps: alcune funzioni fanno uso di una chiave ed altre, invece, dell'altra. La scelta di separare la chiave back-end da quella front-end verrà ampiamente giustificata nell'apposito paragrafo di questo capitolo;
- *mainController* → è il controllore principale che instrada le richieste verso il controller responsabile di tale richiesta;
- *mapController* → è il controller a cui vengono affidate le richieste riguardanti informazioni geografiche ed in generale sulle roadmap;
- *userController* → è il controller a cui vengono affidate le richieste riguardanti l'utente come utilizzatore della piattaforma;
- *contentController* → è il controller a cui vengono affidate le richieste riguardanti tutti i tipi di contenuti afferenti la community.

Notiamo, inoltre, che per tutte le funzioni presenti all'interno del Controller abbiamo cercato di uniformare le interfacce logiche mantenendo sempre il seguente paradigma di ritorno: *return { ok: <status>, error: <error>, data: <data> }.* All'interno del campo data è possibile inserire diversi tipi di oggetti in base all'oggetto richiesto dalla funzione che effettua la chiamata. Per completezza, invece, riportiamo di seguito le interfacce fisiche realizzate per ogni chiamata:

Metodo	**URL**	**Request Body**	**Response Body**

POST	/register	{
		"username":<string>,
		"password":<string>,
		"email":<string>,
		"birthdate":<string>
		}
			{
			"ok":<boolean>,
			"error":<integer>,
			"data":
			{
			"id":<integer>
			}
			}

POST	/auth	{
		“username”:<string>,
		“password”:<string>
		}
			{
			“ok”:<boolean>,
			“error”:<integer>,
			“data”:
			{
			“id”:<integer>,
			“username”:<string>,
			“isAdmin”:<boolean>
			}
			}

POST	/logout		{
			“ok”:<boolean>,
			“error”:<integer>,
			“data”:
			{
			}
			}

POST	/searchRoadmap	{
		“ricerca”:<string>,
		“time”:<integer>,
		“distance”:<integer>
		}
			{
			“ok”:<boolean>,
			“error”:<integer>,
			“data”:
			{
			“roadmaps”:<[object]>
			}
			}

GET	/searchUser?usernam
	e=...
			{
			“ok”:<boolean>,
			“error”:<integer>,
			{
			“users”:<[object]>
			}
			}

POST	/suggestedRoadmap	{
		“roadmap”:<integer>,
		“rating”:<integer>
		}
			{
			“ok”:<boolean>,
			“error”:<integer>,
			“data”:
			{
			“roadmaps”:<[object]>
			}
			}

GET	/getDataUser?id=...		{ “ok”:<boolean>,

			“error”:<integer>,
			“data”:
			{
			“info”:<[object]>,
			“isMe”:<boolean>
			}
			}

GET	/getUserStatus?id=...		{
			“ok”:<boolean>,
			“error”:<integer>,
			“data”:
			{
			“info”:<[object]>
			“isAdmin”: integer,
			“logged” : boolean
			}
			}

GET	/getRoadmapCreate?id=...		{
			“ok”:<boolean>,
			“error”:<integer>,
			“data”:
			{
			“roadmaps”:<[object]>
			}
			}

GET	/getRoadmapSeguite?id=...		{
			“ok”:<boolean>,
			“error”:<integer>,
			“data”:
			{
			“roadmaps”:<[object]>
			}
			}

GET	/getRoadmapPreferite?id=...		{
			“ok”:<boolean>,
			“error”:<integer>,
			“data”:
			{
			“roadmaps”:<[object]>
			}
			}

POST	/deleteRoadmap	{	{

		“id_roadmap”:<integer>
		}
			“ok”:<boolean>,
			“error”:<integer>,
			“data”:
			{
			“results”:<integer>
			}
			}

GET	/getPlaceInfo?placeid=...		{
			“ok”:<boolean>,
			“error”:<integer>,
			“data”:
			{
			“results”:<[object]>
			}
			}

GET	/getPlaceFromCoords?lat=... & lng=...		{
			“ok”:<boolean>,
			“error”:<integer>,
			“data”:
			{
			“res”:<[object]>
			}
			}

POST	/getRoute	{
		“origin”:<string>,
		“destination”:<string>,
		“travelMode”:<string>
		}
			{
			“ok”:<boolean>,
			“error”:<integer>,
			“data”:
			{
			“data_from_google”:<object>
			}
			}

GET	/getRoadmapData?id=...		{
			“ok”:<boolean>,
			“error”:<integer>,
			“data”:
			{
			“roadmap”:<[object]>,
			“user”:<[object]>
			}
			}

GET	/getCommentiRecensi		{

	oni?id=...		“ok”:<boolean>,
			“error”:<integer>,
			“data”:
			{
			“recensioni”:<[object]>,
			“commenti”:<[object]>
			}
			}

POST	/updateCommento	{
		“user_id”:<integer>,
		“id_commento”:<integer>
		, “messaggio”:<string>
		}
			{
			“ok”:<boolean>,
			“error”:<integer>,
			“data”:
			{
			“id_commento”:<integer
			>
			“now”:<date>
			}
			}

POST	/deleteCommento	{
		“id_commento”:<integer>
		}
			{
			“ok”:<boolean>,
			“error”:<integer>,
			“data”:
			{
			“res”:<[object]>
			}
			}

POST	/deleteStage	{
		“idStage”:<integer>
		}
			{
			“ok”:<boolean>,
			“error”:<integer>,
			“data”:
			{
			}
			}

POST	/createRecensione	{
		“roadmapId”:<integer>,
		“opinione”:<string>,
		“valutazione”:<integer>
		}
			{
			“ok”:<boolean>,
			“error”:<integer>,
			“data”:
			{
			“idRecensione”:<integer>,
			“now”:<date>,
			“numeroRecensioniUtent
			e”:<integer>
			}

			}

POST	/updateRecensione	{
		“idRecensione”:<integer>,
		“opinione”:<string>,
		“valutazione”:<integer>
		}
			{
			“ok”:<boolean>,
			“error”:<integer>,
			“data”:
			{
			“idRecensione”:<integer>,
			“now”:<date>
			}
			}

POST	/deleteRecensione	{
		“idRecensione”:<integer>
		}
			{
			“ok”:<boolean>,
			“error”:<integer>,
			“data”:
			{
			“res”:<object>
			}
			}

POST	/setRoadmapAsFavourite	{
		“roadmap_id”:<integer>,
		“newStatus”:<integer>
		}
			{
			“ok”:<boolean>,
			“error”:<integer>,
			“data”:
			{
			“results”:<integer>
			}
			}

POST	/setRoadmapAsSeguita	{
		“roadmap_id”:<integer>,
		“newStatus”:<integer>
		}
			{
			“ok”:<boolean>,
			“error”:<integer>,
			“data”:
			{
			“results”:<integer>
			}
			}

POST	/report	{
		“tipo”:<integer>,
		“motivazione”:<string>
		}
			{
			“ok”:<boolean>,
			“error”:<integer>,
			}

GET	/getAchievements?id=...		{ “ok”:<boolean>, “error”:<integer>, “data”:

			{
			“results”:<[integer]>
			}
			}

POST	/getRoadmapAchievementsPopup		{
			“ok”:<boolean>,
			“error”:<integer>,
			“data”:
			{
			“results”:<integer>
			}
			}

GET	/getSegnalazioni		{
			“ok”:<boolean>,
			“error”:<integer>,
			“data”:
			{
			“results”:<[object]r>
			}
			}

POST	/updateSegnalazioni		{
			“ok”:<boolean>,
			“error”:<integer>,
			“data”:
			{
			“result”:<object>
			}
			}

GET	/getPreferredFavouriteStatusByUserByRoadmap?user_id= …
	&roadmap_id= …
			{
			“ok”:<boolean>,
			“error”:<integer>,
			“data”:
			{
			“result”:<[integer]>
			}
			}

POST	/createRoadmap	{
		“roadmap”:<object>
		}
		
		Nota: in questa chiamata passiamo anche file (Stage
			{
			“ok”:<boolean>,
			“error”:<integer>,
			“data”:
			{
			}

		ed immagini)	}

POST	/updateAvatar	{
		“new_avatar_index”:<string>
		}
			{
			“ok”:<boolean>,
			“error”:<integer>,
			“data”:
			{
			}
			}

POST	/getMarkersFromRect	{
		“centerLatInf”:<float>,
		“centerLatSup”:<float>,
		“centerLngInf”:<float>,
		“centerLngSup”:<float>
		}
			{
			“ok”:<boolean>,
			“error”:<integer>,
			“data”:
			{
			“results”:<[object]>
			}
			}

POST	/deleteUser	{
		“user_id”:<integer>
		}
			{
			“ok”:<boolean>,
			“error”:<integer>,
			“data”:
			{
			}
			}

5.2.3 Model

- *DAO* → nel Model troviamo tutte le funzioni che presentano le richieste che vengono inviate al database sotto forma di query.

5.3 Design Pattern

I Design Pattern, a differenza dei Pattern Architetturali e degli Stili Architetturali, vengono solitamente utilizzati come soluzione a livello implementativo del codice e quindi il loro scope di applicabilità è, in particolar modo, il livello di programmazione. Inoltre, i Design Pattern hanno l’applicabilità più *trasversale* possibile perché non dipendono dal dominio applicativo (o la loro dipendenza è, comunque, minima).

5.3.1 Pattern Façade

Il Pattern Façade permette, attraverso un'interfaccia più semplice, l'accesso a sottosistemi che espongono interfacce complesse e diverse tra loro. In particolare, nel nostro caso, abbiamo utilizzato questo Pattern nel package Controller. Il *mainController*, infatti, fa da Façade alle classi della View che, in questo modo, non dovranno conoscere necessariamente i dettagli implementativi dei singoli controller più interni a cui afferiscono. Questo consente di ottenere un basso accoppiamento tra i due package.

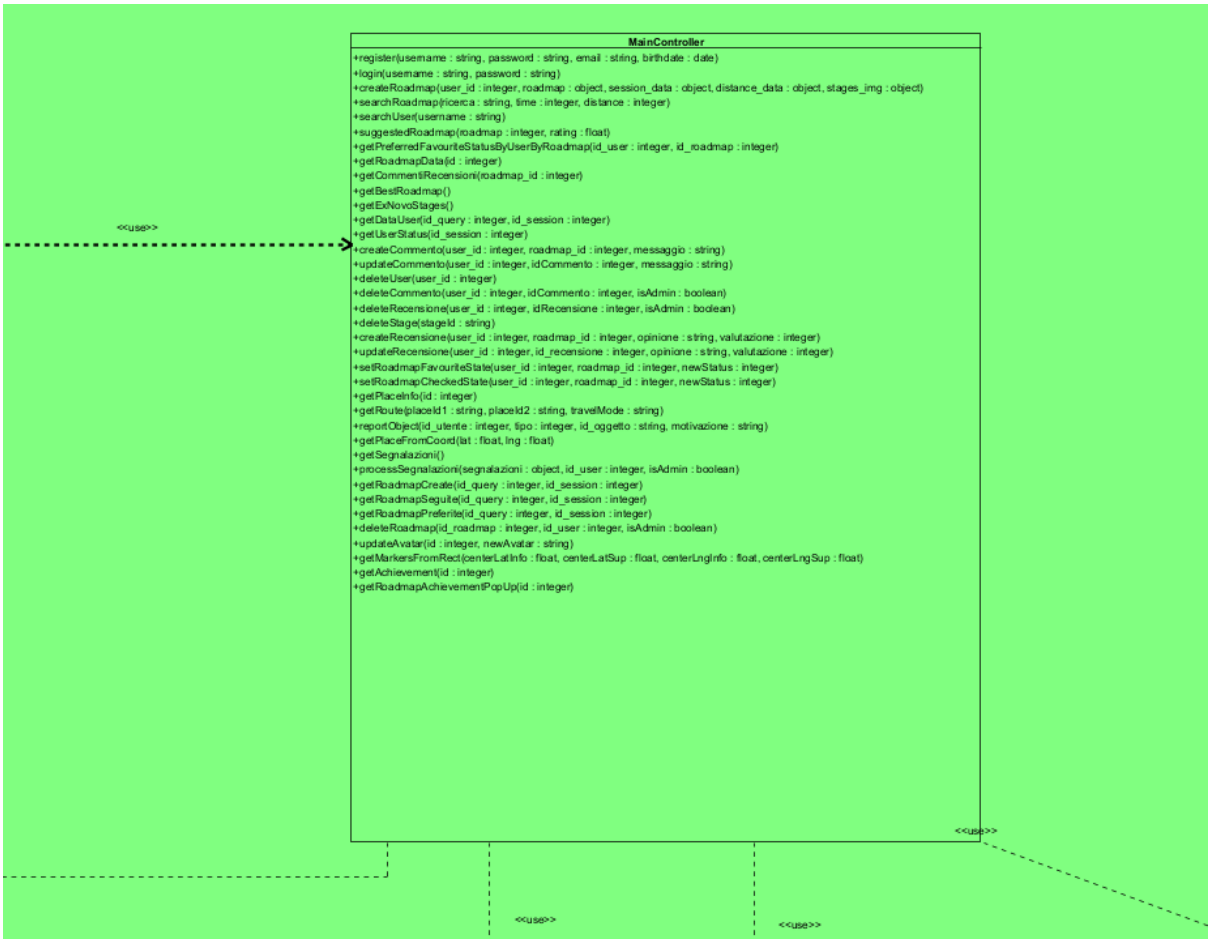


Figura 5.2 - MainController come Façade

5.3.2 Pattern Reactor

Il Pattern Reactor è un Design Pattern per la programmazione orientata agli eventi e rientra nella categoria dei pattern concorrenti. Si applica ad un modello Client-Server in cui le richieste da parte dei client possono arrivare in qualsiasi momento ed in maniera

concorrente. Viene anche chiamato dispatcher di eventi o notificatore. Sapendo che il cuore di Node.js è la sua natura asincrona, durante la fase di implementazione ci siamo resi conto che sarebbe stato possibile utilizzare questo particolare Design Pattern. L'idea è quella di avere un gestore per ogni operazione di I/O. Tale gestore viene invocato appena viene prodotto un evento. Nel nostro caso, il gestore è proprio Node.js. Vediamo, quindi, cosa succede quando un'applicazione utilizza il Pattern Reactor:

- tramite un'operazione di I/O viene generato un nuovo evento. L'applicazione identifica un gestore di tale evento;
- quando tutte le operazioni di I/O sono terminate, per ogni evento viene invocato il corrispondente gestore che è stato identificato in precedenza;
- il gestore completa le sue esecuzioni ed il ciclo ricomincia.

La natura asincrona dell'applicazione, dunque, a questo punto è chiara: l'applicazione esprime l'interesse ad accedere ad una particolare risorsa e fornisce un gestore il quale verrà invocato per completare tale richiesta.

5.4 Asynchronous JavaScript and XML (AJAX)

AJAX è un insieme di tecniche di sviluppo Web che utilizza varie tecnologie Web lato client per creare applicazioni Web asincrone. Con AJAX, le applicazioni web possono inviare e recuperare dati da un server in modo asincrono (in background) senza interferire con la visualizzazione e il comportamento della pagina esistente. Disaccoppiando il livello di scambio dati dal livello di presentazione, AJAX consente alle pagine Web e, per estensione, alle applicazioni Web, di modificare il contenuto in modo dinamico senza la necessità di ricaricare l'intera pagina. Le moderne implementazioni utilizzano comunemente JSON invece di XML, come nel nostro caso.
La pagina Web può essere modificata da JavaScript per consentire all'utente di interagire con le nuove informazioni. HTML e CSS possono essere utilizzati in combinazione per contrassegnare e definire lo stile delle nuove informazioni.

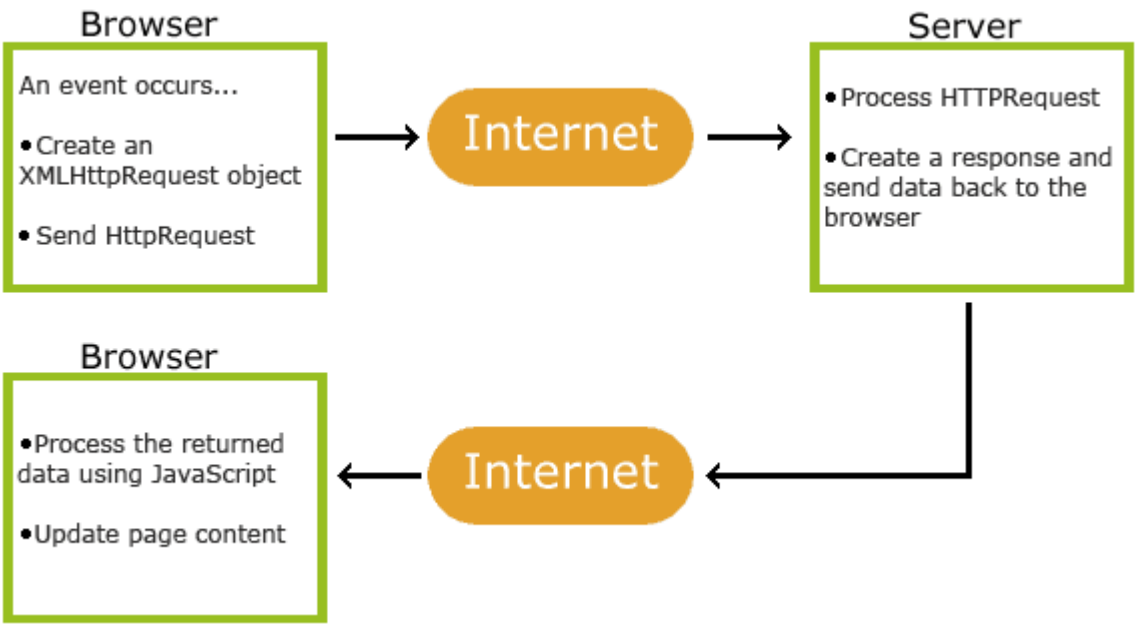


Figura 5.3 - Schema di funzionamento di AJAX

5.5 Google Maps API Key

Come già anticipato nel corso di questa documentazione, fondamentale per il pieno funzionamento di tutte le features offerte da Going è sicuramente l'utilizzo delle API offerte da Google per il proprio servizio di mappe Google Maps. Una volta ottenuta la API Key per accedere al servizio, però, ci siamo resi conto che utilizzare tale chiave pubblicamente avrebbe esposto il sito stesso a potenziali vulnerabilità e problemi di sicurezza. Ora, sicuramente il client avrà in ogni caso bisogno di tale chiave per poter renderizzare correttamente la mappa e tutte le informazioni ad essa associate. Per questo, abbiamo deciso di richiedere due differenti chiavi per accedere alle API: una utilizzata "client-side" e l'altra, invece, utilizzata "server-side". Una volta ottenute le chiavi, alla prima sono state associate maggiori restrizioni in termini di utilizzo e di funzionalità in modo da evitarne eventuali abusi e, per questo, è impostata in modo tale da garantire semplicemente i permessi di visualizzazione della mappa. L'altra chiave, invece, è dotata di meno restrizioni e, per questo, deve essere mantenuta segreta. Inoltre, una volta effettuato il deploy della web app su un server remoto si potrebbe decidere anche di impostare un HTTP Referrer in modo tale da accettare richieste provenienti solo da quel server remoto o, ancora, impostare degli Usage Limits di tale chiave e, quindi, evitare utilizzi malevoli o impropri, aumentandone ulteriormente la sicurezza.

5.6 Diagramma Entità Relazione (ER)

In questo paragrafo riportiamo il diagramma entità relazione (ER) realizzato. Tale diagramma permette di esplicitare le entità (che rappresentano concetti complessi e di rilievo che descrivono classi di oggetti con esistenza autonoma) e le relazioni che intercorrono tra loro (le quali rappresentano legami logici, significativi per la realtà modellata, tra due o più entità). Ciascuna entità può essere caratterizzata da diversi attributi che ne definiscono le proprietà. Il diagramma, dunque, è stato realizzato per la rappresentazione delle entità in gioco che modellano i database sottostanti e che vengono sfruttati per la realizzazione delle varie funzionalità.

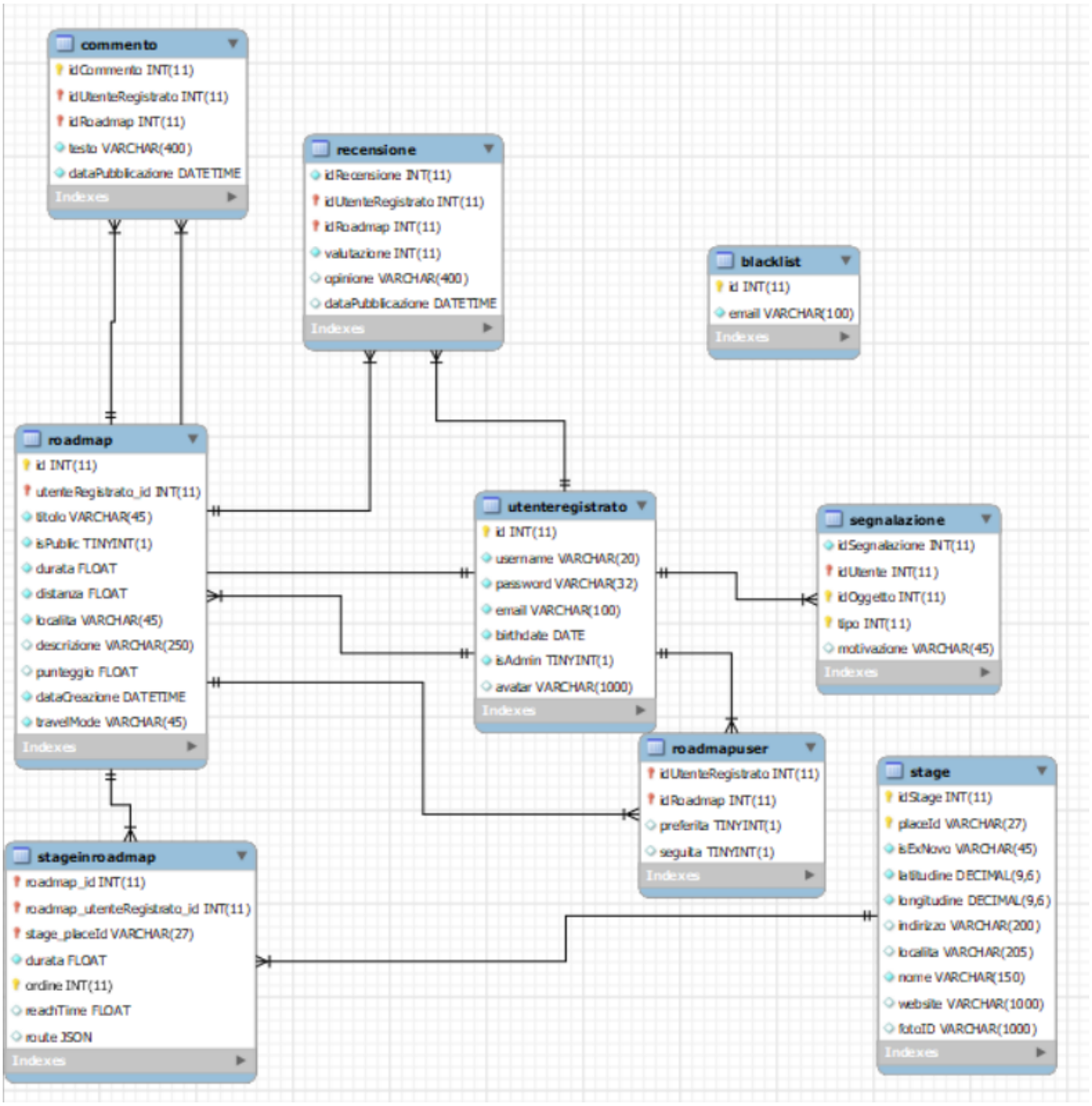


Figura 5.4 - Diagramma ER

Riportiamo, di seguito, una breve descrizione di ogni entità e, se necessario, delle relazioni che la legano con le altre per maggiore chiarezza:

- *utenteregistrato* → mantiene tutte le informazioni relative agli utenti che decidono di iscriversi al sito. Tra le diverse relazioni che legano tale tabella alle altre troviamo, ovviamente, quella relativa alle *roadmap* realizzate dall'Utente Registrato (uno a molti in quanto una Roadmap appartiene ad uno ed un solo Utente Registrato ma un Utente Registrato può creare più Roadmap) e quella con la tabella *roadmapuser* (spiegata più avanti), anche questa di tipo uno a molti. Inoltre, un Utente Registrato è legato anche alla tabella *segnalazione* e *recensione*;

- *roadmap* → mantiene tutte le informazioni relative alle roadmap;
- *stageinroadmap* → mantiene tutte le informazioni relative agli Stage che compongono una singola roadmap. In particolare, notiamo la relazione con la apposita tabella *stage* che ci consente di ottenere le informazioni relative al singolo Stage e che vanno mostrate all'interno della Roadmap in cui esso viene utilizzato;
- *stage* → mantiene le informazioni relative agli Stage;
- *roadmapuser* → mantiene tutte le informazioni relative alle Roadmap seguite e/o preferite degli Utenti Registrati. In questa tabella, infatti, vengono riportate le azioni di un Utente Registrato effettuate non solo sulle proprie Roadmap ma anche su quelle relative alle Roadmap create da altri Utenti Registrati. Le chiavi esterne utilizzate per mantenere tali relazioni sono proprio l'id dell'Utente Registrato e l'id della Roadmap;
- *commento* → mantiene le informazioni relative ai commenti rilasciati dagli Utenti Registrati alle Roadmap;
- *recensione* → mantiene le informazioni relative alle recensioni (composte da valutazione e/o opinione) rilasciate dagli Utenti Registrati alle Roadmap;
- *segnalazione* → mantiene le informazioni relative alle segnalazioni ricevute dagli Utenti Registrati e che un Utente Amministratore dovrà gestire;
- *blacklist* → mantiene le informazioni relative agli utenti bannati e che, per questo, non dovranno più potersi iscrivere alla piattaforma.

5.7 Manuale di utilizzo

Giustificate tutte le scelte implementative, riportiamo a questo punto un sintetico manuale di utilizzo del software al fine di mostrare tutte le funzionalità offerte dallo stesso. Contestualmente ad una descrizione testuale verranno riportati anche una serie di screenshot al fine di migliorare ulteriormente la chiarezza di quanto spiegato.

5.7.1 Configurazione ed installazione

Il software realizzato è una web app per cui presenta il vantaggio di non dover essere installato. Basterà, infatti, tramite un qualsiasi browser che rispetti i vincoli evidenziati

nell'apposito paragrafo del capitolo 1 di questa documentazione visitare il seguente link per avere accesso alla homepage del sito → https://going-app.azurewebsites.net/

Visitando tale link, si verrà indirizzati alla homepage del sito.

Figura 5.5 - Homepage

5.7.2 Utilizzo del sito come Utente Non Registrato

Come già ampiamente spiegato nel corso di questa documentazione, il sito offre alcune funzionalità anche agli Utenti Non Registrati. Oltre a visitare la homepage e le pagine *"Info"* e *"Chi Siamo"* (dove è possibile trovare alcune informazioni utili), anche gli Utenti Non Registrati potranno cercare Roadmap create dagli Utenti Registrati per poterle visualizzare. Gli Utenti Non Registrati potranno effettuare tale ricerca dall'apposita barra presente direttamente in homepage oppure visitando la pagina *"Esplora"*.

Figura 5.6 - Pagina Esplora

È possibile visitare la pagina *"Esplora"* anche non effettuando alcuna ricerca. In questo caso verranno mostrate alcune Roadmap suggerite selezionate in base ad alcuni criteri di qualità tra quelle presenti nella piattaforma.

Effettuando, invece, una ricerca per parole chiave, verranno mostrate tutte le Roadmap ed i profili degli Utenti Registrati che corrispondono a tali parametri. Riportiamo uno screenshot di ciò che si ottiene effettuando una ricerca per parole chiave.

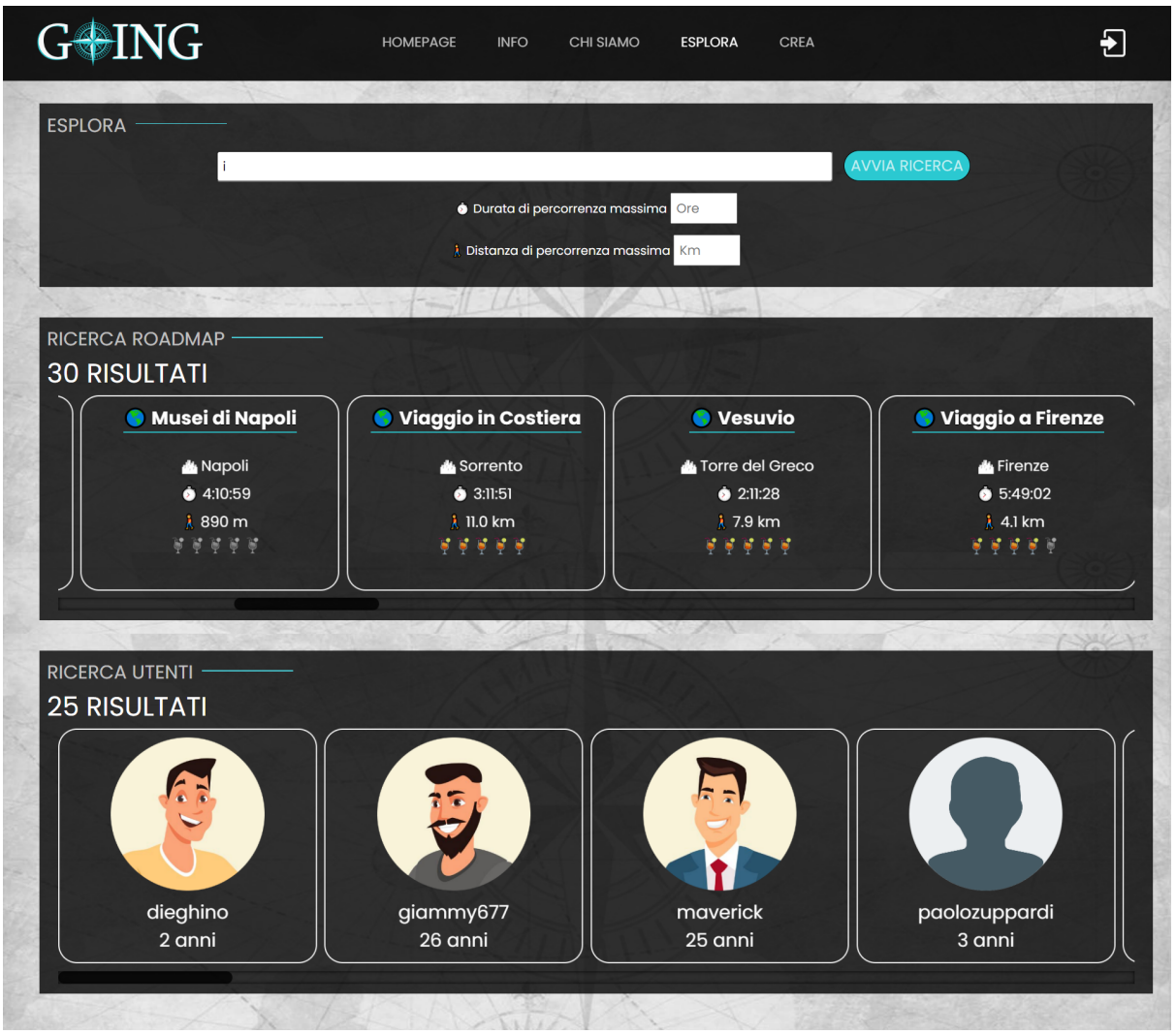


Figura 5.7 - Pagina Esplora effettuando una ricerca

Tramite gli appositi campi presenti subito sotto la barra di ricerca, è possibile filtrare i risultati inserendo ulteriori parametri. Di seguito riportiamo un ulteriore screenshot di ciò che si ottiene se alla stessa ricerca mostrata nello screenshot precedente si inserisce una durata di percorrenza massima delle Roadmap di 5 ore ed una distanza di percorrenza massima di 8 km.

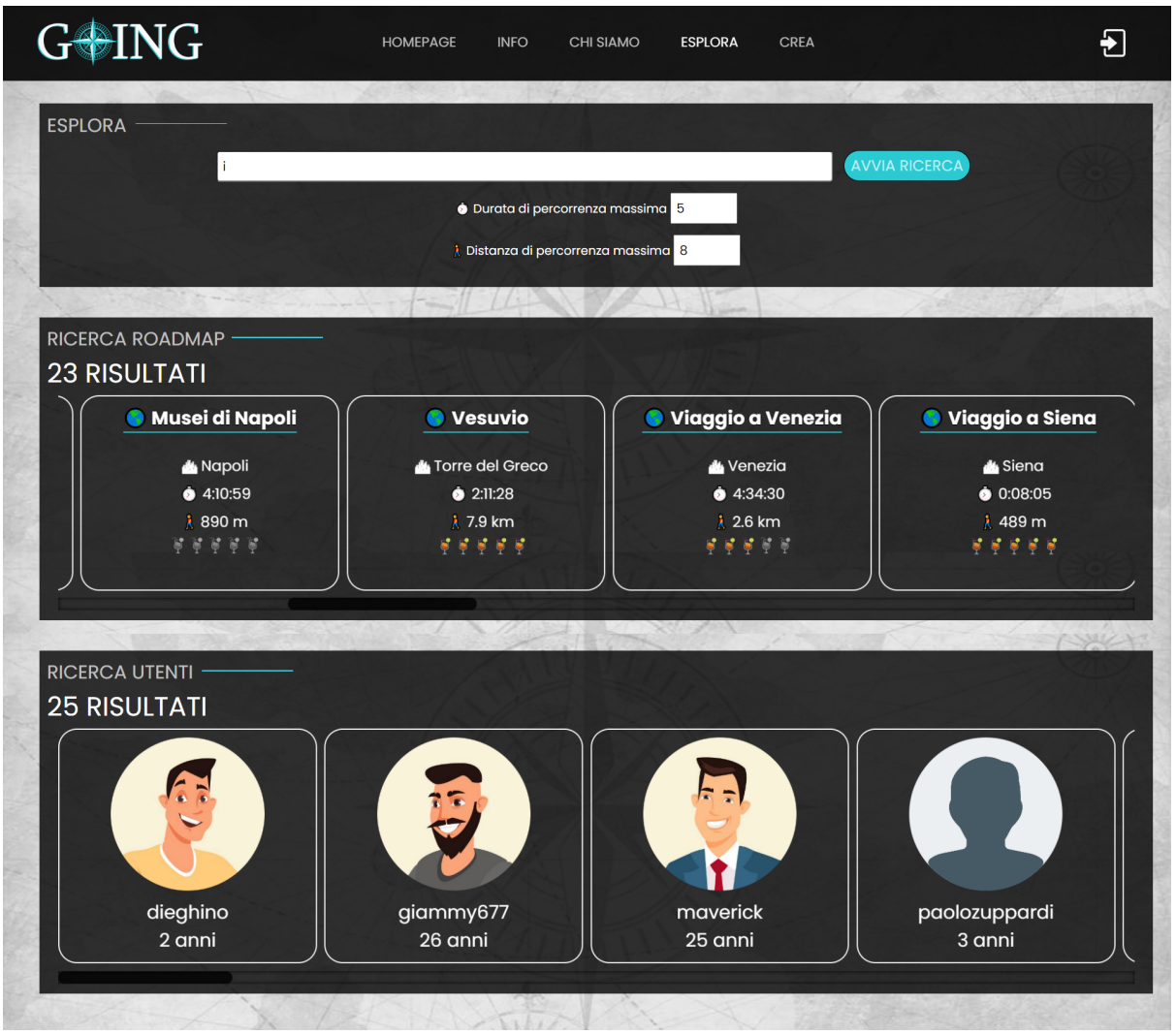


Figura 5.8 - Pagina Esplora effettuando una ricerca con filtri

Notiamo come il numero di Roadmap ottenute da tale ricerca sia sceso da 30 a 23 inserendo, appunto, i filtri relativi alla durata ed alla distanza.

Conclusa la ricerca, sarà possibile visualizzare una Roadmap o il profilo di un utente semplicemente con un click del mouse sulla relativa scheda. Mostriamo, di seguito, la schermata che si presenta quando si decide di voler visualizzare una Roadmap.

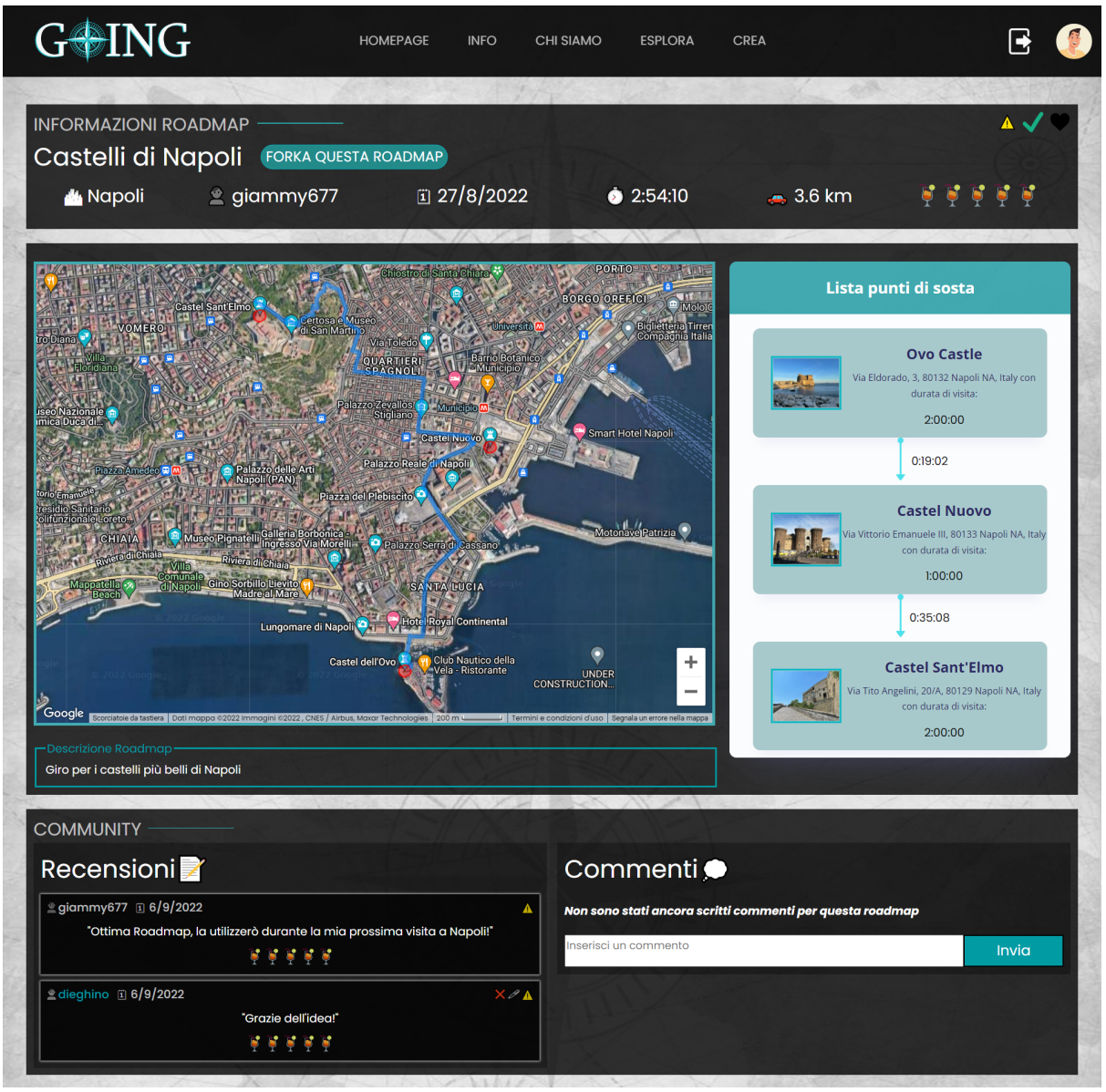


Figura 5.9 - Pagina di visualizzazione di una Roadmap

Di seguito, invece, mostriamo la schermata di visualizzazione del profilo di un Utente Registrato.

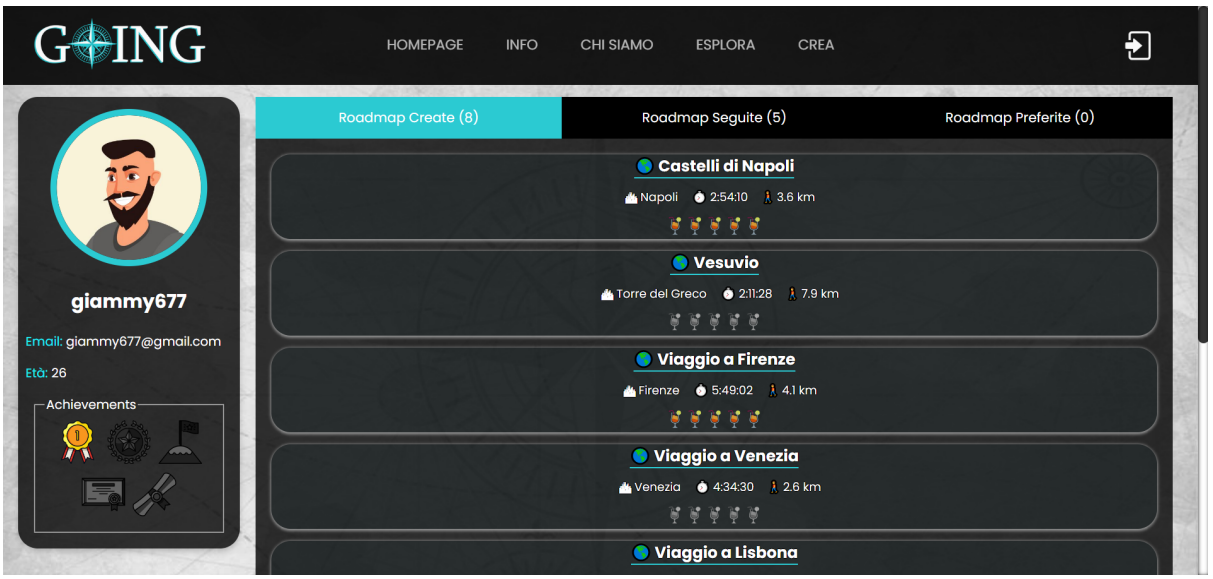


Figura 5.10 - Pagina Profilo di un Utente Registrato

Oltre alle informazioni come avatar, username, email ed età, dal profilo utente è possibile vedere gli achievement sbloccati ed avere un collegamento diretto alle Roadmap pubbliche dell'utente stesso.

Vediamo adesso, invece, cosa accade se un Utente Non Registrato prova ad accedere alla pagina "Crea" nella quale è possibile creare una nuova Roadmap.

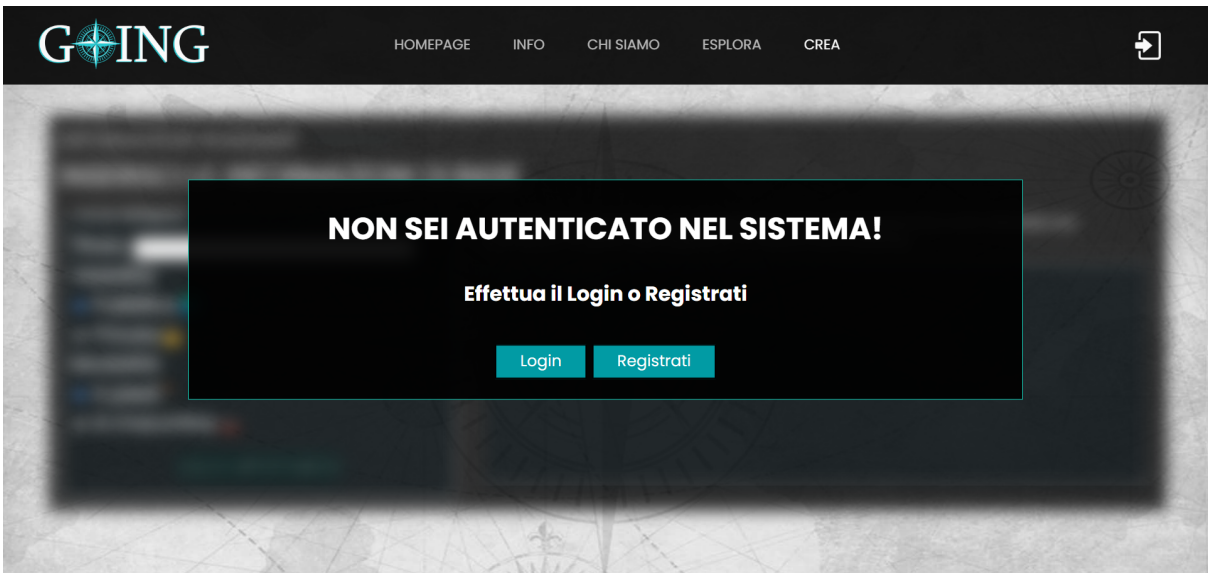


Figura 5.11 - Pagina Crea "blurrata" per Utenti Non Registrati

Notiamo che il sistema rileva che l'Utente non è attualmente Autenticato all'interno del sistema. Tutte le funzionalità di creazione sono bloccate ed un popup che invita ad effettuare

il login o la registrazione viene mostrato a schermo. La stessa cosa avviene anche nel caso in cui un Utente Non Registrato tenti di creare una Roadmap a partire dalla visualizzazione di una già esistente (la nostra funzionalità di *"fork"* già descritta in precedenza).

L'Utente Non Registrato può, dunque, decidere di iscriversi premendo sull'apposito bottone. Si verrà, dunque, re-indirizzati alla pagina di registrazione nella quale sarà possibile inserire i propri dati.

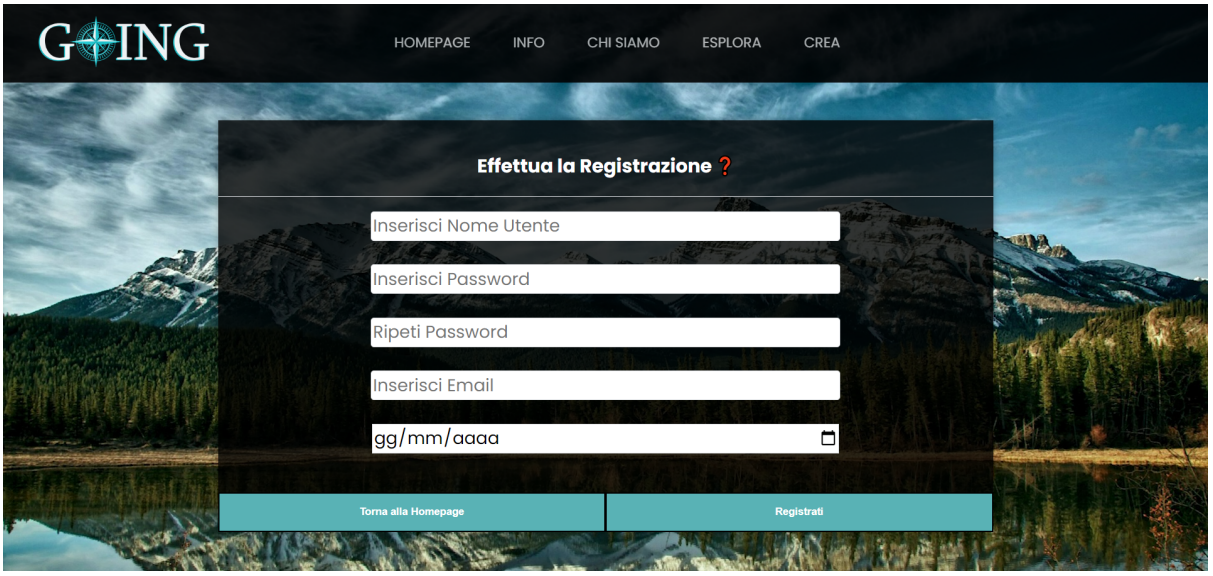


Figura 5.12 - Pagina di registrazione

5.7.3 Utilizzo del sito come Utente Registrato

Una volta completata la fase di registrazione, il sito effettuerà il redirect sulla homepage. Da qui, tramite l'apposito bottone, sarà possibile effettuare il login.

5.7.4 Utilizzo del sito come Utente Autenticato

Una volta effettuato il login, un Utente Autenticato ha pieno accesso a tutte le funzionalità del sito (escluse, ovviamente, quelle di amministrazione). Oltre a tutte le funzionalità messe a disposizione anche degli Utenti Non Registrati già descritte in precedenza nell'apposito paragrafo, un Utente Autenticato avrà diritto anche alla funzionalità di creazione di una nuova Roadmap. Visitando, infatti, l'apposita pagina (*"Crea"*) oppure premendo il bottone *"Forka questa Roadmap"* presente nella pagina di visualizzazione di un'altra Roadmap, la schermata che verrà presentata all'Utente Autenticato sarà la seguente:

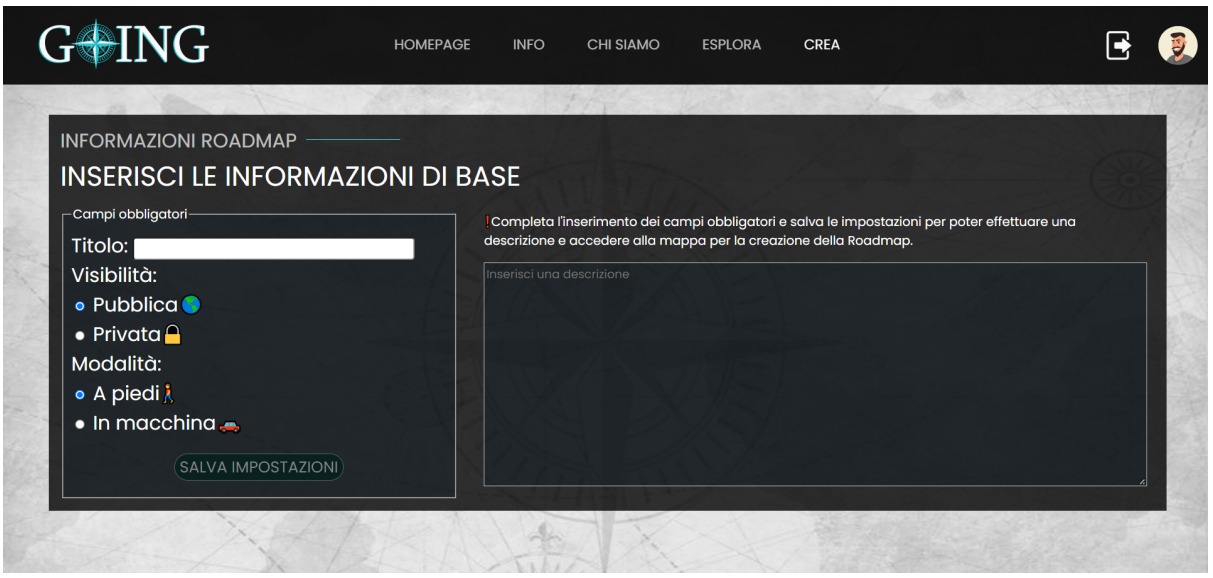


Figura 5.13 - Pagina Creazione di una Roadmap (prima parte)

L'Utente Autenticato potrà, quindi, inserire le informazioni basilari relative alla Roadmap che si vuole realizzare quali titolo, visibilità (pubblica o privata), la modalità di percorrenza (a piedi o in macchina). Premendo il bottone "Salva impostazioni" la schermata sbloccherà automaticamente la sezione in cui, opzionalmente, sarà possibile inserire una descrizione. Scendendo verso il basso, invece, verrà sbloccata la sezione in cui viene presentata la mappa.

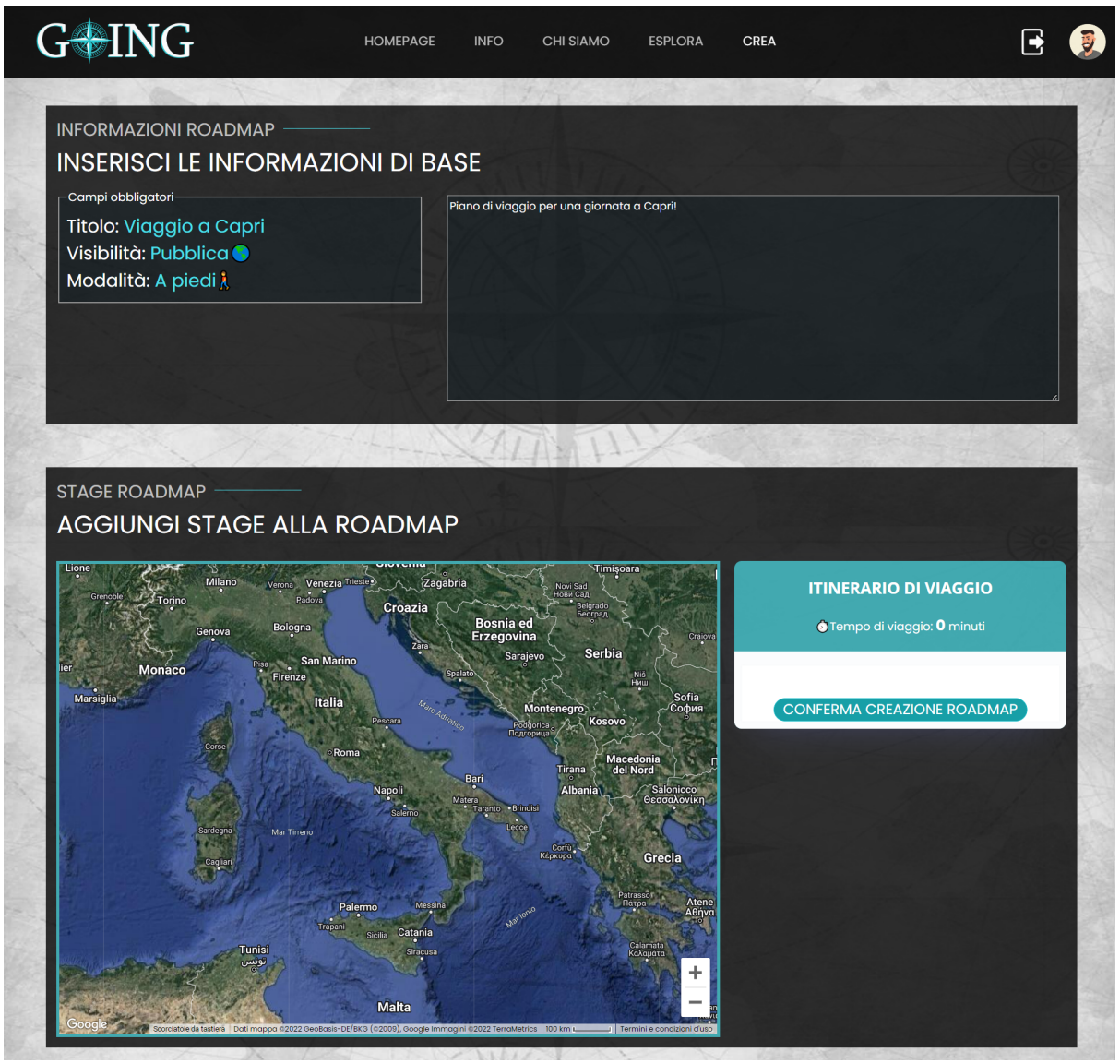


Figura 5.14 - Pagina Creazione di una Roadmap (completa)

A questo punto, interagendo direttamente con la mappa sarà possibile aggiungere Stage alla propria Roadmap. Facendo click su un punto di interesse già evidenziato sulla mappa sarà possibile selezionare proprio quel punto come nuovo Stage senza dover inserire alcuna informazione ulteriore se non la durata di permanenza in cui si prevede di sostare in tale Stage.

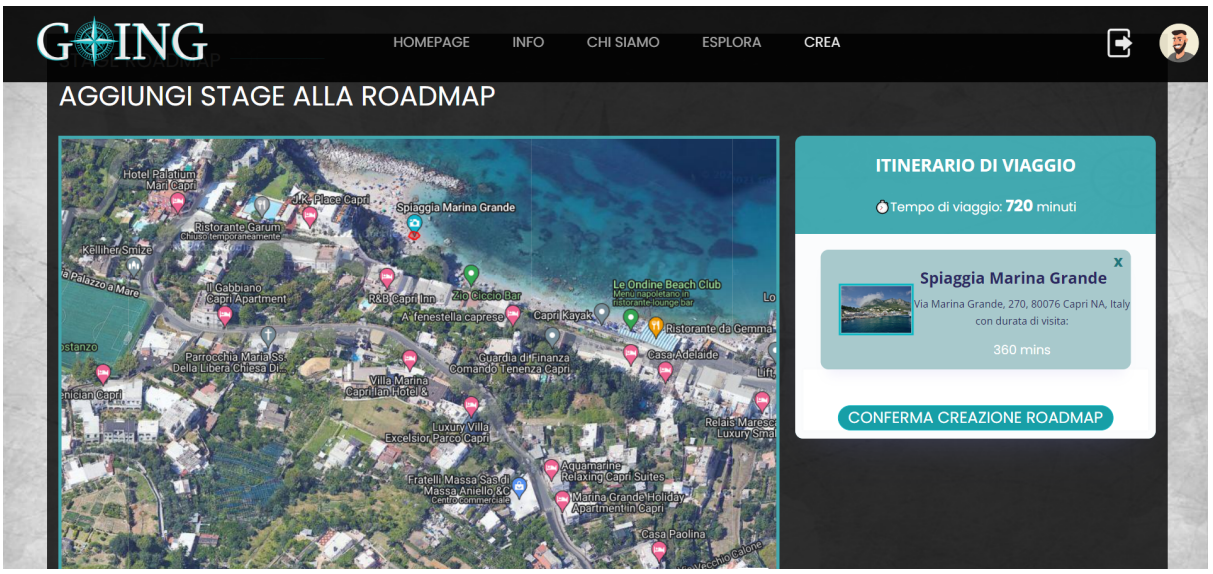


Figura 5.15 - Pagina Creazione di una Roadmap (interazione con la mappa)

Facendo click, invece, su un qualsiasi punto della mappa che non è già evidenziato sarà possibile inserire uno Stage ex-novo. Tali Stage vengono creati appositamente dall'Utente Autenticato e salvati nel database locale. Per questo, successivamente qualsiasi altro Utente Autenticato potrà utilizzare tale Stage come se fosse un qualsiasi altro Stage. All'atto di creazione di un nuovo Stage sarà richiesto di inserire un nome e, opzionalmente, sarà possibile arricchirlo con ulteriori informazioni opzionali quali una foto, un sito web e la durata di sosta.

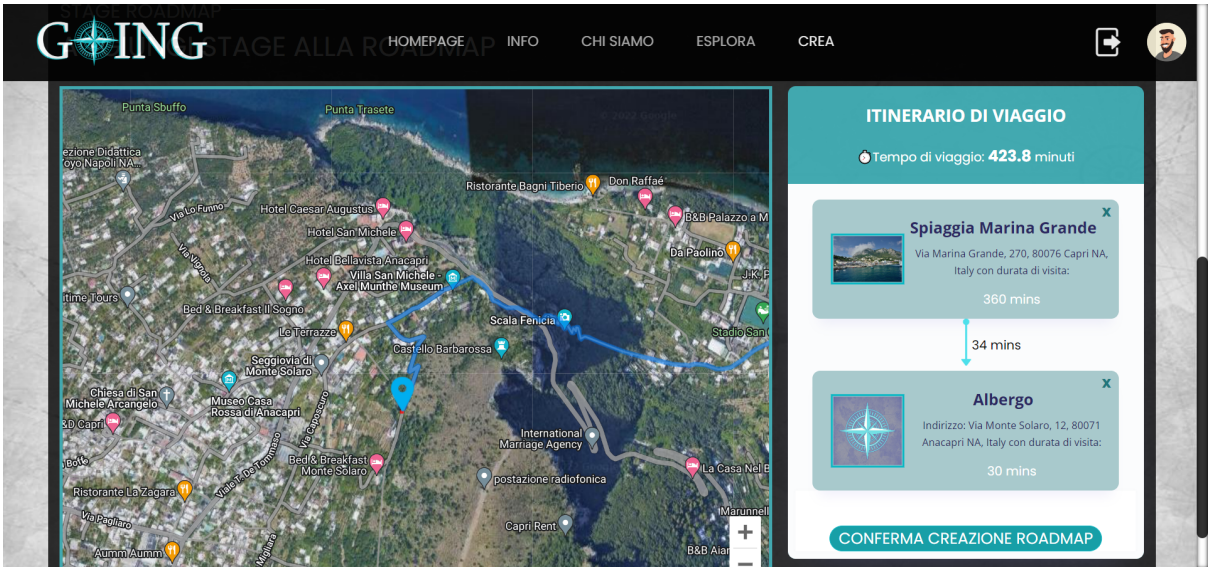


Figura 5.16 - Pagina Creazione di una Roadmap (aggiunta Stage)

Tali Stage verranno aggiunti automaticamente all'elenco presente sulla destra ed in maniera del tutto automatica verranno calcolati il percorso e la durata di percorrenza tra uno Stage e l'altro.

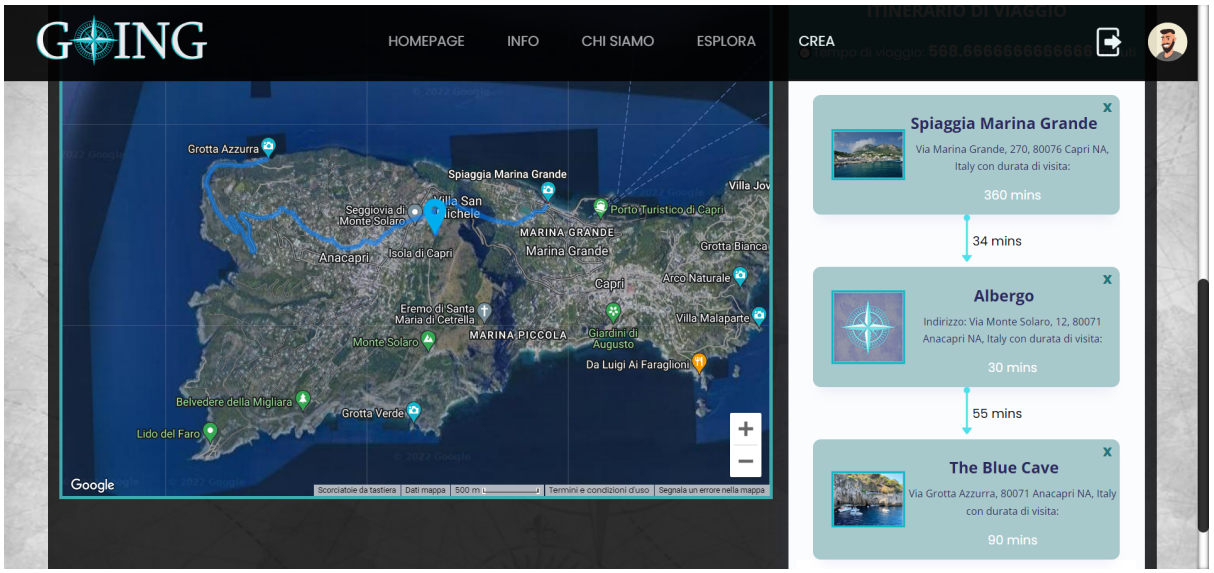


Figura 5.17 - Pagina Creazione di una Roadmap (aggiunta Stage ex-novo)

Una volta terminata la propria Roadmap, l'Utente Autenticato può pubblicarla tramite l'apposito bottone di conferma. Alla pressione di tale bottone, l'Utente Autenticato sarà automaticamente indirizzato alla pagina di visualizzazione della Roadmap appena creata.

Ulteriori funzionalità esclusive degli Utenti Autenticati sono quelle relative all'aggiunta di una Roadmap al proprio elenco di Roadmap Preferite e/o Seguite o, ancora, quelle relative alla pubblicazione di un commento o una recensione ad una Roadmap (possibile tramite la pagina di visualizzazione della Roadmap stessa). Inoltre, interagendo in svariati modi col sito gli Utenti Autenticati potranno guadagnare diversi achievement relativi, ad esempio, alla pubblicazione di un certo numero di nuove Roadmap, all'aggiunta di un certo numero di Roadmap ai Preferiti o ai Seguiti o, ancora, alla pubblicazione di un certo numero di commenti o recensioni. Inoltre - accedendo al proprio profilo premendo sul proprio avatar presente in alto a destra su qualsiasi pagina del sito una volta che si è effettuato l'accesso - ogni Utente Autenticato potrà visualizzare le proprie Roadmap private o, ancora, modificare il proprio avatar facendo click su quello corrente.

Figura 5.18 - Modifica Avatar Utente Registrato

5.7.5 Utilizzo del sito come Utente Amministratore

Accedendo alla pagina del proprio profilo, un Utente Amministratore avrà la possibilità - tramite un apposito bottone - di collegarsi al pannello di amministrazione in cui potrà trovare tutte le segnalazioni ancora da gestire giunte dagli altri Utenti Autenticati. Riportiamo, di seguito, uno screenshot del pannello.

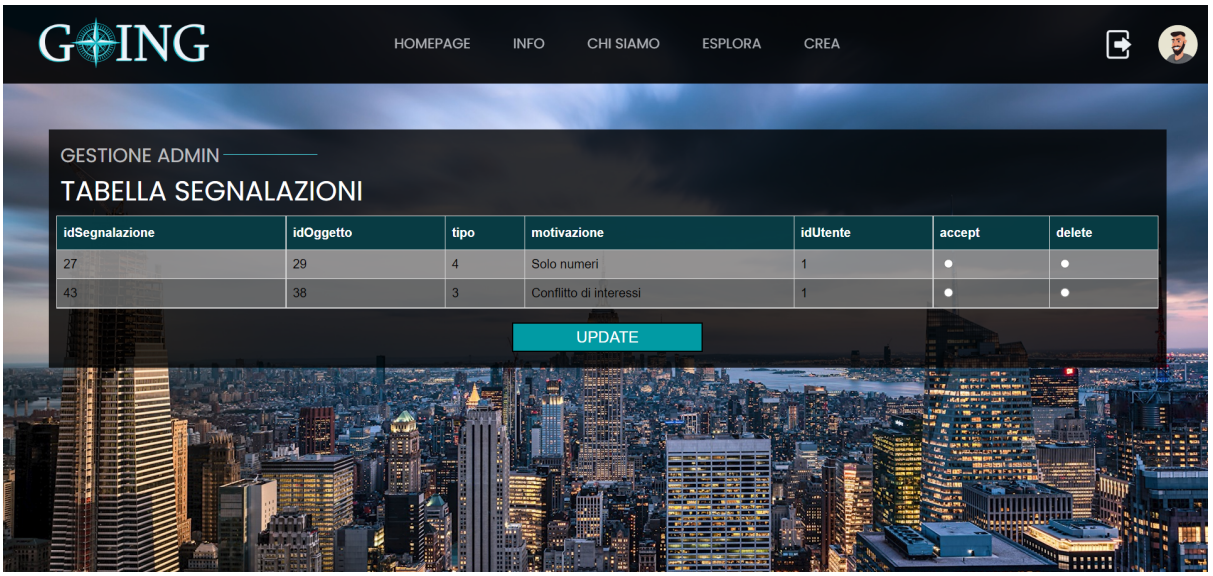


Figura 5.19 - Pannello di Amministrazione

5.8 Video demo

Un video dimostrativo dell'applicazione realizzata e delle sue funzionalità può essere trovato al seguente link YouTube → https://www.youtube.com/watch?v=jMba2l-n3x8

Capitolo 6

Testing

Il testing è un'attività fondamentale da svolgere all'interno del ciclo di vita di un qualsiasi software. Tale attività ha l'obiettivo di scovare difetti e dimostrare che un sistema soddisfi i suoi requisiti funzionali e non funzionali.

Prima di scrivere un test effettivo, comprendiamo la struttura generale di un blocco di test:

si definisce un blocco di test un insieme di test riguardanti la stessa funzionalità. Non è sempre semplice associare un'unica funzionalità ai singoli casi di test, a maggior ragione considerando che il nostro sistema è ricco di funzionalità.

Il processo di testing inizia con l'instanziare i componenti utili per l'esecuzione del test.

Tali componenti devono poi essere inizializzati ad uno stato specifico coincidente con lo stato iniziale utile per eseguire il test. Tale inizializzazione può avvenire *"fittiziamente"* iniettando valori all'interno degli oggetti oppure *"realmente"* utilizzando proprio l'interazione col sistema per raggiungere tale stato. Durante il testing ci si è focalizzati maggiormente sull'esecuzione di test "reali", nonostante siano stati più lenti (ma si parla sempre in termini di millisecondi), visto che in questo modo ripercorriamo proprio le chiamate che il front-end (e quindi il client attraverso chiamate AJAX) fa al server.

Visto che i test inevitabilmente agiscono sul database, ci si è assicurati che ogni blocco di test, qualora crei un contenuto, lo rimuova a termine del blocco di test allo scopo di ripristinare lo stato originale del database.

I componenti da testare, una volta istanziati ed inizializzati, eseguono un'operazione i cui risultati vengono confrontati con quelli prefissati che ci aspettiamo dal sistema. Nel caso in cui i risultati coincidano, si conclude che il test è andato a buon fine. Altrimenti, il test viene considerato fallito.

La suddivisione dei blocchi di testing è effettuata cercando di separare le principali responsabilità. Si è suddivisa l'attività di testing quindi in 6 gruppi:

1) *User* (riguardante le funzionalità specifiche di un Utente Non Registrato);
2) *UserAutenticato* (riguardante le funzionalità specifiche di un Utente Autenticato);
3) *Admin* (riguardante le funzionalità specifiche di un Utente Amministratore);

4) *Map* (riguardante le funzionalità specifiche delle mappe);
5) *Content* (riguardante le funzionalità specifiche di commenti, recensioni etc.);
6) *SendView* (riguardante le funzionalità specifiche atte a restituire la View al client).

La strategia di testing utilizzata per la validazione della nostra applicazione è stata quella di procedere testando tutte le possibili interfacce del server visto che, nella nostra applicazione specifica, il server vive solo per rispondere al cliente ed esso non prende mai iniziativa. Le interfacce chiameranno poi una ad una tutte le funzioni del controller e così via in cascata per poi testare praticamente tutto il sistema backend.

Il modulo utilizzato per effettuare i test è il modulo *Jest*. Dopo un semplice *import* all'interno del modulo Jest nei moduli di Node.js è stato davvero intuitivo iniziare il lavoro. Si allega uno screenshot dell'output di Jest.

File	% Stmts	% Branch	% Funcs	% Lines
All files	83.33	67.4	92.69	83.18
Controller	88.57	72.64	92.64	88.42
HTTPinterface.js	91.1	57.69	86.79	91.06
MapsHandler.js	80.95	70	100	80.95
contentController.js	81.18	82.05	100	81
mainController.js	97.91	100	97.29	97.91
mapController.js	87.34	83.33	83.33	86.48
userController.js	92.85	81.25	100	92.85
Model	73.93	46.42	92.85	73.86
DAO.js	73.93	46.42	92.85	73.86
config	100	100	100	100
config.js	100	100	100	100

Figura 6.1 - Coverage fornito da Jest

Si noti come si sia correttamente superato l'**80% di coverage** di linee di codice. Nonostante ciò, si nota che la percentuale di "Branch" coperti è più bassa degli altri parametri, soprattutto per il DAO. La ragione di questo è che nei casi di test non si è contemplato il fatto che il database possa non rispondere. Infatti, il database durante i casi di test è stato sempre disponibile. Tali branch di codice non coperti coincidono con le routine di gestione dei casi in cui il server dove è hostato il database restituisce errore. E' buona norma, quindi, avere questo tipo di gestione delle eccezioni. Questi sono i casi di *Too Many Connections* o quando

la connessione verso il database viene a mancare. Abbiamo deciso di non approfondire tali casi di test.

```
Test Suites: 6 passed, 6 total
Tests:       65 passed, 65 total
Snapshots:   0 total
Time:        3.911 s, estimated 4 s
Ran all test suites.
```

Figura 6.2 - Esito Test

Avendo suddiviso in blocchi differenti i casi di test ci siamo resi conto di un problema relativo all'instanziazione del server. In *HTTPInterface* si istanzia un server in ascolto sulla porta 3000. In realtà, essendo la modalità di esecuzione dei test di Jest - per ovvi motivi - in parallelo, ci si è ritrovati ad avere problemi di conflitto sulla porta 3000 di tipo `EADDRINUSE - port already in use` visto che ogni Test Suite o blocco di test re-instanziava un server, ma sempre sulla stessa porta. Il problema poteva essere risolto in due modi:

1) eseguire i blocchi di test sequenzialmente e non più in parallelo, perdendo molto in termini di velocità;
2) specificare all'interfaccia *HTTPInterface* che se si è in un contesto di testing si deve hostare il server sulla porta 0. Per porta 0 si definisce la prima porta libera.

Abbiamo deciso di adottare il secondo metodo di risoluzione allo scopo di non perdere in termini di velocità. Rimane il fatto che all'atto di un deploy reale il server verrà comunque hostato sulla porta 3000, se non specificata nelle variabili di ambiente del servizio di hosting.
I test sono andati tutti a buon fine e, di seguito, si descrive più approfonditamente quelli più complessi e meno intuitivi.

6.1 Descrizione Test Suite

6.1.1 User

```
PASS  __tests__/User.test.js
  Test User (block #1)
    √ test di register con email già utilizzata (71 ms)
    √ test di register con username già utilizzato (8 ms)
    √ test di register con email bannata (7 ms)
    √ Test user general info quando NON loggato (44 ms)
    √ Test di login con credenziali errate (15 ms)
    √ test di logout senza aver effettuato il login (5 ms)
    √ Test login (account esistente e dati corretti) (9 ms)
    √ Test user general info quando loggato (10 ms)
    √ Test preleva roadmap create da utente registrato (14 ms)
    √ Test preleva roadmap seguite da utente registrato (11 ms)
    √ Test preleva roadmap preferite da utente registrato (10 ms)
```

Figura 6.3 - Esito test del blocco User

TC	Descrizione	Input	Output

3	Registrazione con una email bannata	Dati di registrazione (username, password, email e data di nascita)	{
			ok:false,
			error:-777,
			data:{}
			}

Una delle feature già discusse è la possibilità per un Amministratore di bandire un utente registrato. Quando ciò avviene, la sua email finirà in una tabella del database denominata "*blacklist*". Nessuno potrà più registrarsi con tale email in GOING.

6.1.2 User Autenticato

```
PASS  __tests__/UserAutenticato.test.js
  Test User Autenticato (SEZIONE COMMENTI/RECENSIONI) (block #2)
    √ Test user general info quando loggato (15 ms)
    √ Test creazione commento invalido (messaggio vuoto) (6 ms)
    √ Test creazione commento valido (12 ms)
    √ Test modifica commento (11 ms)
    √ Test creazione recensione invalida (6 ms)
    √ Test creazione recensione valida (12 ms)
    √ Test creazione recensione valida (ma già pubblicata) (8 ms)
    √ Test modifica recensione (12 ms)
    √ Test delete commento valido (11 ms)
    √ Test delete recensione valida (11 ms)
    √ Test delete commento valida di un altro utente (8 ms)
    √ Test delete recensione valida di un altro utente (8 ms)
    √ Test aggiungi Roadmap ai preferiti (9 ms)
    √ Test rimuovi Roadmap dai preferiti (10 ms)
    √ Test aggiungi Roadmap alle roadmap seguite (10 ms)
    √ Test rimuovi Roadmap dalle roadmap seguite (9 ms)
    √ Test set Roadmap as checked (7 ms)
    √ Test prendi info da user getDataUser (7 ms)
    √ Test login (ma già loggato) (4 ms)
    √ Test update avatar (7 ms)
    √ Test di logout avendo effettuato il login (4 ms)
```

Figura 6.4 - Esito test del blocco UserAutenticato

TC	Descrizione	Input	Output
12	Informazioni su utente loggato (me)		Informazioni su utente loggato (id dell'utente, username, avatar, isAdmin, isLogged)

Questo test chiede al server le informazioni che esso ha nella sessione relativa all'utente loggato.

TC	Descrizione	Input	Output

18	Creazione recensione valida ma già pubblicata	[*recensione obj*]	{
			ok:false,
			error:1062,
			data:{}
			}

A valle del login da parte di un Utente Registrato, tale test richiede al server di inserire una recensione all'interno del database. Tale recensione non viene però registrata in quanto un Utente Registrato può fare solo una recensione su una roadmap e quindi, avendone già pubblicata una nel test precedente il database rifiuta l'inserimento della recensione. Il test, dunque, può dirsi superato.

6.1.3 Admin

```
PASS  __tests__/Admin.test.js
  Test Admin (block #3)
    √ Test admin banna utente esistente (29 ms)
    √ Test admin banna inesistente/già bannato (10 ms)
    √ Test user segnala + admin process bulk segnalazioni (81 ms)
```

Figura 6.5 - Esito test del blocco Admin

TC	Descrizione	Input	Output

34	Ban utente esistente	idUtente	{
			ok:true,
			error:0,
			data:{}
			}

Questo test chiede al server di rimuovere un Utente Registrato dalla piattaforma. Tale azione, se effettuata dall'Amministratore (validato con il parametro *isAdmin* nella sessione) consente di eliminare tale Utente dalla piattaforma. L'eliminazione comporterà la rimozione di ogni suo contenuto dalla piattaforma e l'aggiunta della email dell'Utente nella tabella *"blacklist"*.

TC	Descrizione	Input	Output

36	Admin processa le segnalazioni	[*action obj*]	{
			ok:true,
			error:0,
			data:{}
			}

Questo caso di test è quello più importante per il corretto funzionamento del blocco di segnalazioni. Come spiegato precedentemente, si preferisce nel testing andare ad agire con stimoli reali. Quello che si fa, quindi, è creare dei contenuti (commenti, recensioni, roadmap, utenti) per poi effettuare le segnalazioni da parte di un Utente Registrato. L'Amministratore tramite il suo pannello può rimuovere o accettare le segnalazioni. Se accetta le segnalazioni, si rimuove il contenuto da GOING e si rimuovono tutte le segnalazioni che fanno riferimento a quello specifico oggetto. Se rifiuta le segnalazioni, si procede solamente a rimuovere tutte le segnalazioni relative all'oggetto senza rimuoverlo dalla piattaforma. All'atto del click del submit dell'amministratore, esso manderà al server un array contenente *<idSegnalazione, action>* dove action è l'azione da intraprendere. Action può essere "accept" o "ignore". Il server agirà sulle singole segnalazioni come già spiegato per poi mostrare un messaggio di conferma all'amministratore.

6.1.4 Map

```
PASS  __tests__/Map.test.js
  Test ROADMAP (block #4)
    √ Test migliori roadmap (15 ms)
    √ Test prendi i marker da finestra scorrevole rettangolare (11 ms)
    √ Test calcola percorso tra 2 posti raggiungibili tra loro (177 ms)
    √ Test calcola percorso tra 2 posti irraggiungibili tra loro (161 ms)
    √ Test prendi info di un posto esistente (e già nel database) (8 ms)
    √ Test prendi info di un posto esistente (e non nel database) (160 ms)
    √ Test prendi info di un posto inesistente (75 ms)
    √ Test prendi info di un posto dalle sue coordinate (106 ms)
    √ Test creazione roadmap con nodi ex-novo + non nel db (cache miss) (driving) (450 ms)
    √ Test prendi info della roadmap appena creata (11 ms)
    √ Test creazione roadmap con nodi pre-esistenti (cache hit) (walking) (131 ms)
    √ Test rimozione roadmap (8 ms)
    √ Test delete stage (43 ms)
```

Figura 6.6 - Esito test del blocco Map

TC	Descrizione	Input	Output

45	Creazione roadmap con nodi ex-novo e nodi POI non nel DB	[*roadmap obj*]	{
			ok:true,
			error:0,
			data:
			{roadmapId:X}
			}

Lo scopo di questo test è quello di verificare la corretta creazione della Roadmap. Viene costruita una Roadmap partendo da un titolo, descrizione e modalità (a piedi o in macchina) per poi creare degli Stage che saranno aggiunti all'Array Stages. Una volta assemblata la Roadmap, il test effettua la richiesta POST *createRoadmap* al server con annessa la Roadmap e automaticamente la Roadmap aggiunge i nodi ex novo e quelli di Google (ma non presenti nella tabella Stage locale) alla tabella Stage. Dopodiché, ogni utente potrà utilizzare anche gli Stage ex-novo nelle successive Roadmap.

6.1.5 Content

```
PASS  __tests__/Content.test.js
  Test Achievement (block #5)
    √ Test preleva achievement completati da user (101 ms)
    √ Test preleva achievement numero roadmap create da utente (14 ms)
    √ Test ricerca utente vuota (5 ms)
    √ Test ricerca utente valida (8 ms)
    √ Test suggestedRoadmap (25 ms)
    √ Test suggestedRoadmap invalid params (5 ms)
    √ Test stato like/follow su roadmap fissato utente (9 ms)
    √ Test preleva commenti e recensioni da roadmap (11 ms)
```

Figura 6.7 - Esito test del blocco Content

Tale gruppo di test serve a verificare il corretto funzionamento delle caratteristiche relative agli achievement, alla ricerca ed all'aggiunta ai Preferiti/Seguiti delle Roadmap.

6.1.6 SendView

```
PASS  __tests__/SendView.test.js
  Test retrieving html (block #6)
    √ Test HomePage retrieving (61 ms)
    √ Test info_page retrieving (8 ms)
    √ Test about_page retrieving (5 ms)
    √ Test create_page retrieving (5 ms)
    √ Test explore_page retrieving (6 ms)
    √ Test view_roadmap retrieving (5 ms)
    √ Test signup_page retrieving (5 ms)
    √ Test profile_page retrieving (5 ms)
    √ Test admin_page retrieving (7 ms)
```

Figura 6.8 - Esito test del blocco SendView

Tale gruppo di test serve a verificare il corretto trasferimento delle pagine della View al browser web del cliente.