



Documentazione di progetto

Lavoro di gruppo - Corso di Informatica III -
Modulo Progettazione e Algoritmi

Studenti: Nome-A Cognome-A, Matr. 0000000

Nome-B Cognome-B, Matr. 0000000

Nome-C Cognome-C, Matr. 0000000

Docente: Prof.ssa Patrizia Scandurra

CdL Magistrale in Ing. Informatica

Facoltà di Ingegneria

Università degli Studi di Bergamo

Bergamo, Italia - Gennaio 2020

Indice

Indice generale	vii
Indice delle figure	viii
Indice delle tabelle	ix
Iterazione 0	11
1 Introduzione	13
1.1 Panoramica	13
2 Analisi dei requisiti	15
2.1 Analisi del contesto	15
2.2 Studio di fattibilità	15
3 Usecases	17
3.1 Descrizione delle usecases	17
3.2 Diagramma comportamentale	25
4 Specifiche	27
4.1 Introduzione specifiche	27
4.2 Specifiche funzionali	27
5 Architettura	29
5.1 Deployment Diagram	29
5.1.1 Architettura Hardware	29
5.2 Architettura Software	32
5.2.1 Component Diagram	32

5.2.2	Class Diagram	32
I	Iterazione 1	35
6	Parser	37
6.1	Scelta funzioni da implementare	37
II	Iterazione 2	39
7	Template HTML	41
7.1	Variazione linguaggio di programmazione	41
7.2	Template	41
7.3	Progettazione dei Test	43
7.4	Template Sequence Diagram	43
7.5	Risultato dei Test	44
III	Iterazione 3	45
8	Algoritmo	47
8.1	Algoritmo e Flowchart	47
8.1.1	Analisi di complessità	47
8.1.2	Pseudocodice	49
8.1.3	Diagrammi di attività	55
8.2	Progettazione dei Test	56
8.3	Risultato dei Test	58
IV	Iterazione 4	59
9	Visualizzazione Percorso	61
9.1	Progettazione dei Test	61
9.2	Pseudocodice	61
9.3	Risultato dei Test	64
10	Manuale Utente	65
10.1	Installazione	65

10.2 Utilizzo	68
11 Toolchain	73
11.1 Tools	73
11.2 Librerie	74
Bibliografia	75

Elenco delle figure

3.1 Diagramma dei casi d'uso	18
3.2 Diagramma comportamentale	26
5.1 Deployment Diagram	30
5.2 Architectural Diagram	31
5.3 Client-Server Architecture	31
5.4 Component Diagram	32
5.5 Class Diagram Architecture	33
7.1 Template Deployment Diagram	42
7.2 Template Sequence Diagram	44
7.3 Test iterazione 2	44
8.1 Flowchart Algorithm	48
8.2 Algorithm Sequence Diagram	56
8.3 Methods	57
8.4 Test iterazione 3	58
9.1 Template Deployment Diagram	63
9.2 Test iterazione 3	64
9.3 Test completo	64
10.1 Virtual server in esecuzione	66
10.2 "yarn serve" sito versione development in esecuzione	67
10.3 "yarn build" compilazione del sito versione di produzione	68
10.4 Schermata Homepage	69
10.5 Schermata Esempio Planner	69
10.6 Schermata Esempio Planner 2	70
10.7 Schermata Esempio Planner 3	71

Elenco delle tabelle

4.1 Specifiche funzionali, priorità e implementazione	28
8.1 Funzioni implementate nella suite di test	58

Parte

Iterazione 0

Capitolo 1

Introduzione

1.1 Panoramica

Il problema del riscaldamento climatico globale e dei disastri ambientali degli ultimi anni, ha portato alla crescente rigidezza delle normative sulle emissioni e quindi ad un aumento di volume del mercato delle auto elettriche e ibride a scapito di quelle termiche.

Insieme all'arrivo dei veicoli elettrificati sono sorti anche i problemi della ricarica di tali veicoli dovuti alla scarsa estensione della rete di colonnine atte a tale scopo.

L'obiettivo del nostro progetto è quello di creare una Web App che semplifichi gli spostamenti agli utilizzatori di veicoli elettrici, consentendo loro di pianificare itinerari tenendo in considerazione anche l'esigenza di ricarica.

Capitolo 2

Analisi dei requisiti

2.1 Analisi del contesto

Allo stato attuale l'unico Software in grado di creare un planner per veicoli elettrici è quello installato nei veicoli Tesla, la nota casa automobilistica americana, leader in questo settore. Questo Software inoltre si integra con la vettura e consente di monitorarne lo stato e controllarla da remoto.

Purtroppo però il suo utilizzo è riservato ai soli possessori di un'auto dello stesso marchio; rimangono quindi scoperti tutti coloro che hanno acquistato un veicolo elettrico di un diverso produttore. Per loro l'unica applicazione disponibile è quella reperibile al link [http://evroute.
controtex.com](http://evroute.controtex.com) che, però, fa riferimento solo al territorio inglese. Non è quindi disponibile alcun servizio relativo al suolo italiano.

2.2 Studio di fattibilità

Per realizzare le funzionalità che abbiamo ideato avremo bisogno di 4 elementi: una mappa, un pianificatore del percorso iniziale, un database delle stazioni di ricarica e infine una componente che si occupi di unire il percorso fornito con la posizione delle colonnine per generare il percorso con le eventuali fermate.

Capitolo 3

Usecases

3.1 Descrizione delle usecases

In questa sezione andremo a schematizzare i requisiti funzionali del software da sviluppare, essi sono indipendenti dalla tipologia di tecnologia utilizzata, dall'architettura, dalla piattaforma e dalla tipo di linguaggio di programmazione scelto.

Nel nostro caso avremo un attore Utente che interagisce con il sistema, questi potrà invocare una Richiesta Percorso(UC1) che a sua volta include un altro caso d'uso che si occuperà della generazione del percorso(UC2).

Generazione percorso fa utilizzo di due API esterne: Open Street Map si occuperà di fornire una mappa del percorso (planner) mentre Open Charge Map è un database contenente tutte le informazioni sulla posizione delle colonnine di ricarica.

L'Utente potrà Registrarsi(UC8) ed effettuare un Login(UC10) all'applicazione web in modo da ottenere dei Servizi Aggiuntivi(UC3).

I servizi aggiuntivi offerti sono il Salvataggio dei Luoghi Preferiti(UC4), Modifica Dati Utente(UC5), Esportare il Percorso(UC6), Generazione dei Report(UC7), Cancellazione Utente(UC9) e il Logout(UC11).

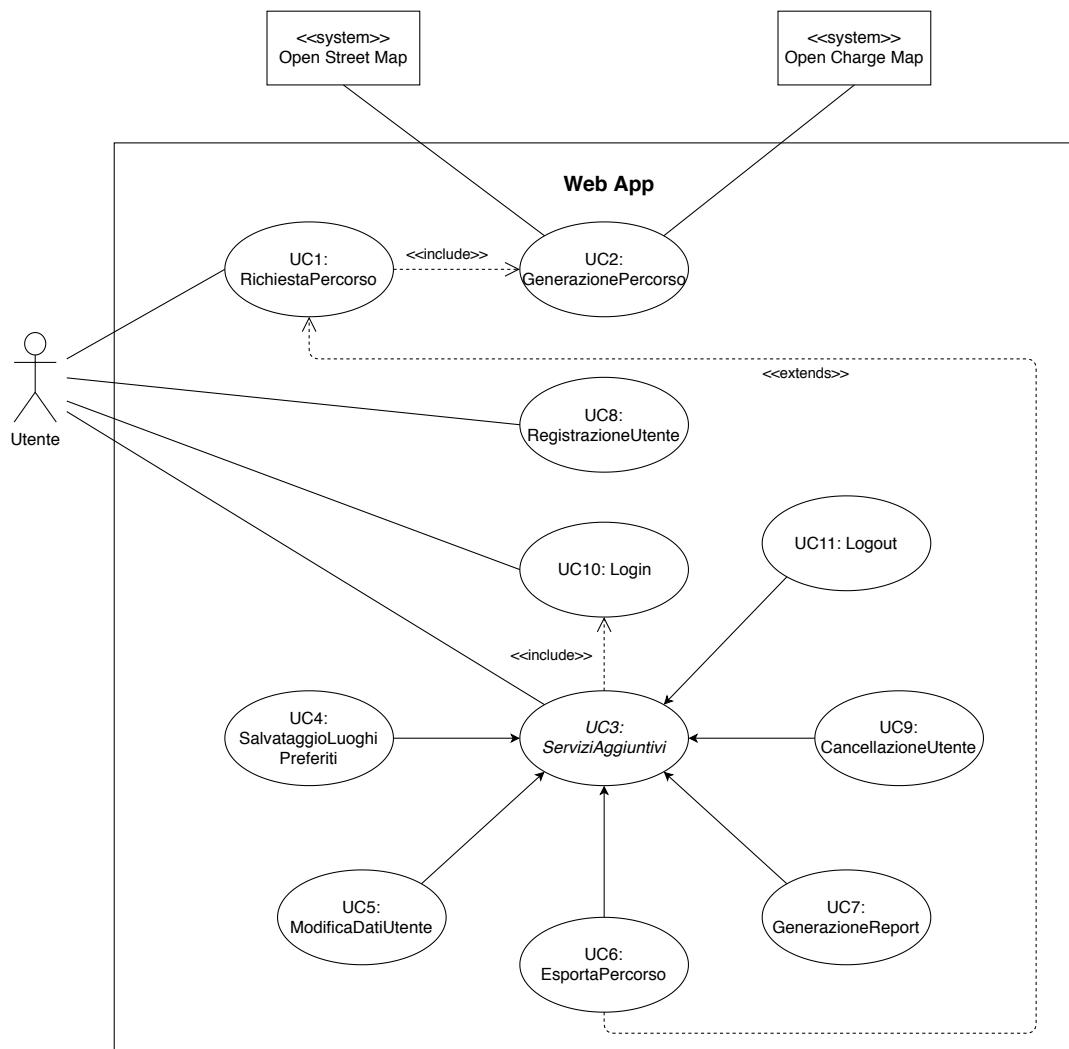


Figura 3.1: Diagramma dei casi d'uso

UC1: RichiestaPercorso

1. Descrizione

L'utente desidera ottenere il tragitto fra due località comprensivo delle soste da effettuare presso le colonnine di ricarica.

2. Requisiti coperti

/

3. Attori

Utente

4. Precondizioni

L'utente ha accesso alla pagina web ed è pronto ad inserire i dati richiesti

5. Passi principali

- (a) L'utente inserisce la località di partenza, quella di destinazione desiderata e l'autonomia dell'auto
- (b) Controllo che le località di inizio, fine percorso e autonomia sono inserite con la giusta formattazione
- (c) Controllo che le località inserite sono valide e che l'autonomia sia stata inserita correttamente(valori negativi non ammessi)
- (d) Le informazioni circa le località inserite dall'utente vengono processate attraverso Open Street Map al fine di convertire gli indirizzi in coordinate
- (e) Le coordinate vengono inviate al modulo di analisi con il relativo campo autonomia

6. Situazioni eccezionali

/

7. Postcondizioni

Coordinate formattate

UC2: GenerazionePercorso

1. Descrizione

Generazione del percorso finale comprensivo di soste per le ricariche dell'auto

2. Requisiti coperti

R1

3. Attori

/

4. Precondizioni

Coordinate formattate

5. Passi principali

- (a) Le coordinate ricevute vengono fornite a Open Street Map per ottenere il percorso fra i due punti
- (b) Il percorso ottenuto viene combinato con i dati delle colonnine per generare il percorso definitivo
- (c) Il risultato viene presentato all'utente sia in forma grafica che testuale

6. Situazioni eccezionali

Non esiste un percorso stradale fra le due località

7. Postcondizioni

Percorso generato

UC3: ServiziAggiuntivi

1. Descrizione

Use Case astratto che rappresenta i servizi ulteriori a cui un utente registrato può accedere

2. Requisiti coperti

/

3. Attori

Utente

4. Precondizioni

Utente loggato

5. Passi principali

/

6. Situazioni eccezionali

/

7. Postcondizioni

/

UC4: SalvataggioLuoghiPreferiti

1. Descrizione

L'utente può salvare i suoi luoghi preferiti (i.e. i luoghi maggiormente visitati), in modo da potervi accedere più facilmente (non sarà necessario digitare ogni volta l'indirizzo completo)

2. Requisiti coperti

R7

3. Attori

Utente

4. Precondizioni

Utente loggato

5. Passi principali

- (a) L'utente inserisce il luogo da salvare
- (b) L'utente dà un nome al luogo in questione (es. Casa, Lavoro, etc.)
- (c) L'utente preme sul pulsante salva dati
- (d) Il sistema risponde con un avvenuto salvataggio

6. Situazioni eccezionali

Dati inseriti non validi

7. Postcondizioni

/

UC5: ModificaDatiUtente

1. Descrizione

L'utente può modificare i dati inseriti in fase di registrazione

2. Requisiti coperti

R3

3. Attori

Utente

4. Precondizioni

Utente loggato

5. Passi principali

- (a) L'utente inserisce i dati da modificare nel sistema
- (b) L'utente preme sul pulsante salva dati
- (c) Il sistema risponde con un avvenuto salvataggio

6. Situazioni eccezionali

Dati inseriti non validi

7. Postcondizioni

/

UC6: EsportaPercorso

1. Descrizione

L'utente può esportare il percorso generato dall'applicazione in diversi formati o può condividerlo su diverse piattaforme social

2. Requisiti coperti

R4

3. Attori

Utente

4. Precondizioni

Utente loggato, Percorso generato

5. Passi principali

- (a) L'utente può esportare la pianificazione tramite un apposito pulsante "esporta" che fornisce diverse opzioni tramite un menù a comparsa

6. Situazioni eccezionali

/

7. Postcondizioni

/

UC7: GenerazioneReport

1. Descrizione

L'utente può visualizzare alcune statistiche di suo interesse circa lo storico dei suoi viaggi (es. totale kilometri percorsi, strade maggiormente frequentate, tempo di sosta medio alle colonnine, frequenza media di sosta, etc.)

2. Requisiti coperti

R5

3. Attori

Utente

4. Precondizioni

Utente loggato

5. Passi principali

/

6. Situazioni eccezionali

/

7. Postcondizioni

/

UC8: RegistrazioneUtente

1. Descrizione

L'utente può registrarsi all'applicazione in modo da avere accesso ad ulteriori funzioni

2. Requisiti coperti

R2

3. Attori

Utente

4. Precondizioni

/

5. Passi principali

- (a) L'utente preme il pulsante di registrazione presente nell'interfaccia
- (b) L'utente inserisce i propri dati nella schermata di registrazione
- (c) L'utente preme sul pulsante invio dati al sistema
- (d) Il sistema risponde con un avvenuta registrazione e salva i dati dell'utente

6. Situazioni eccezionali

Utente già registrato

7. Postcondizioni

Utente registrato

UC9: CancellazioneUtente

1. Descrizione

L'utente può cancellare il suo account ed eliminare i suoi dati

2. Requisiti coperti

/

3. Attori

Utente

4. Precondizioni

Utente loggato

5. Passi principali

- (a) L'utente preme il pulsante di cancellazione account

- (b) Il server informa l'utente circa l'irreversibilità dell'operazione e la perdita dei dati
- (c) L'utente preme sul pulsante conferma
- (d) Il sistema richiede la password dell'utente
- (e) L'utente inserisce la password
- (f) Il server verifica la correttezza della password, elimina l'account e invia una conferma all'utente sul buon esito dell'operazione

6. Situazioni eccezionali

 Password fornita errata

7. Postcondizioni

 Utente cancellato

UC10: Login

1. Descrizione

 L'utente accede al sistema con le sue credenziali

2. Requisiti coperti

 /

3. Attori

 Utente

4. Precondizioni

 Utente registrato

5. Passi principali

- (a) L'utente preme sul pulsante di login
- (b) L'utente inserisce i propri dati nella schermata
- (c) L'utente preme sul pulsante invio dati al sistema
- (d) Il sistema risponde con una conferma

6. Situazioni eccezionali

 Dati forniti errati

7. Postcondizioni

 Utente loggato

UC11: Logout

1. Descrizione

 L'utente esce dal suo account

2. Requisiti coperti

/

3. Attori

Utente

4. Precondizioni

Utente loggato

5. Passi principali

- (a) L'utente preme sul pulsante di logout
- (b) Il sistema risponde con una conferma

6. Situazioni eccezionali

/

7. Postcondizioni

Utente non loggato

3.2 Diagramma comportamentale

Il diagramma comportamentale descrive come evolve il software (Fig.3.2).

In particolare un utente interroga il server e questi risponde con il caricamento della pagina web richiesta (Web page loaded). L'utente inserisce i dati richiesti nei campi e sottomette la richiesta al sistema(Request submitted). Una volta calcolato il percorso migliore il sistema risponde a schermo con la visualizzazione della mappa.

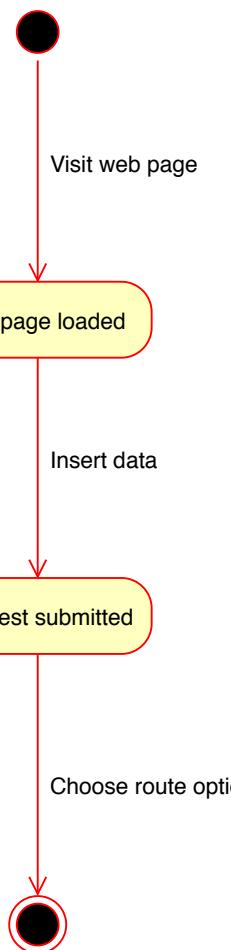


Figura 3.2: Diagramma comportamentale

Capitolo 4

Specifiche

4.1 Introduzione specifiche

In questo capitolo analizzeremo le specifiche funzionali e non funzionali della applicazione da sviluppare.

La priorità di una funzione è scelta in base a quante funzioni dipendono da essa:

1. **Alta**: è fondamentale implementarla prima di procedere con il macroblocco delle funzioni con media priorità. Sono funzioni da implementare principalmente l'una in serie con l'altra.
2. **Media**: è fondamentale implementarla prima di procedere con il macroblocco delle funzioni con bassa priorità. È possibile implementare parallelamente queste funzioni.
3. **Bassa priorità**: da loro non dipende nessuna funzione.

4.2 Specifiche funzionali

Le specifiche funzionali dell'applicazione sono le seguenti:

1. **Calcolo del percorso**: una volta fornito la mappa delle colonnine e il percorso da un punto A ad un punto B l'app deve rielaborare i dati in modo da calcolare il nuovo percorso comprensivo di eventuali soste.
2. **Iscrizione utente**: l'applicazione deve poter permettere all'utente l'iscrizione in modo da avere ulteriori servizi.
3. **Modifica dati utente**: l'applicazione deve poter permettere all'utente la modifica dei propri dati all'interno del proprio profilo.
4. **Esportazione tragitto**: quando l'applicazione restituisce un percorso all'utente, questi deve essere in grado di esportarlo in un formato opportuno.

5. **Visualizzazioni statistiche:** l'utente dal proprio profilo è in grado di visualizzare le statistiche relative ai propri viaggi.
6. **Mostra alternative percorso:** l'applicazione può mostrare all'utente le alternative al percorso precedentemente calcolato.
7. **Memorizzazione luoghi preferiti:** l'utente è in grado di salvare i percorsi preferiti nel proprio profilo
8. **Visualizzazione percorso:** l'utente è in grado di vedere il percorso calcolato dall'algoritmo su una mappa.
9. **Visualizza dati utente:** l'utente è in grado di vedere i propri dati nell'apposito profilo utente.

Sono riportati in Tabella 4.1 tutte le specifiche funzionali con le relative priorità.

Codice	Nome	Priorità	Implementato
R1	Calcolo percorso	A	SI
R2	Iscrizione utente	M	NO
R3	Modifica dati utente	B	NO
R4	Esportazione tragitto	M	NO
R5	Visualizzazione statistiche	B	NO
R6	Mostra alternative percorso	A	NO
R7	Memorizzazione luoghi preferiti	M	NO
R8	Visualizza percorso	A	SI
R9	Visualizza dati utente	M	NO

Tabella 4.1: Specifiche funzionali, priorità e implementazione

Capitolo 5

Architettura

5.1 Deployment Diagram

Il deployment diagram mostrato nella Figura 5.1 del sistema evidenzia quali dispositivi vengono considerati, le interconnessioni tra loro e dove le componenti software verranno istanziate.

Nel nostro caso abbiamo 4 "device":

Un device è rappresentato dal Client che tramite protocollo http invia una richiesta al device Server, esso al suo interno presenta una componente Web Browser che si interfacerà con il Server.

Sono presenti due device di supporto alla nostra applicazione: Open Charge Map Server e Open Street Map Server queste due componenti restituiscono tramite una richiesta di tipo REST la posizione delle colonnine e il percorso da un punto di partenza ad un punto di destinazione richiesto dall'utente. Il cuore del nostro software è presente dentro il device Server in cui sono presenti 5 componenti. Il componente charger_location_data_feeder si occupa di scaricare le colonnine tramite l'API di Open Charge Map e le passa al modulo general_data_bank che contiene i metodi atti a generare la struttura dati che svolgerà il ruolo di database locale. itinerary_feeder è il modulo che calcola il percorso attraverso l'API di Open Street Map. Il componente algorithm_calculation incorpora l'algoritmo che si occuperà di trovare le colonnine a cui fermarsi. Infine client_communication si occupa di ricevere gli input dall'utente e di restituire il risultato a schermo.

5.1.1 Architettura Hardware

L'architettura che è emersa è di tipo client-server a 2 livelli. Nel primo livello il client è impersonato dall'utilizzatore finale che richiede un servizio al nostro server attraverso il protocollo

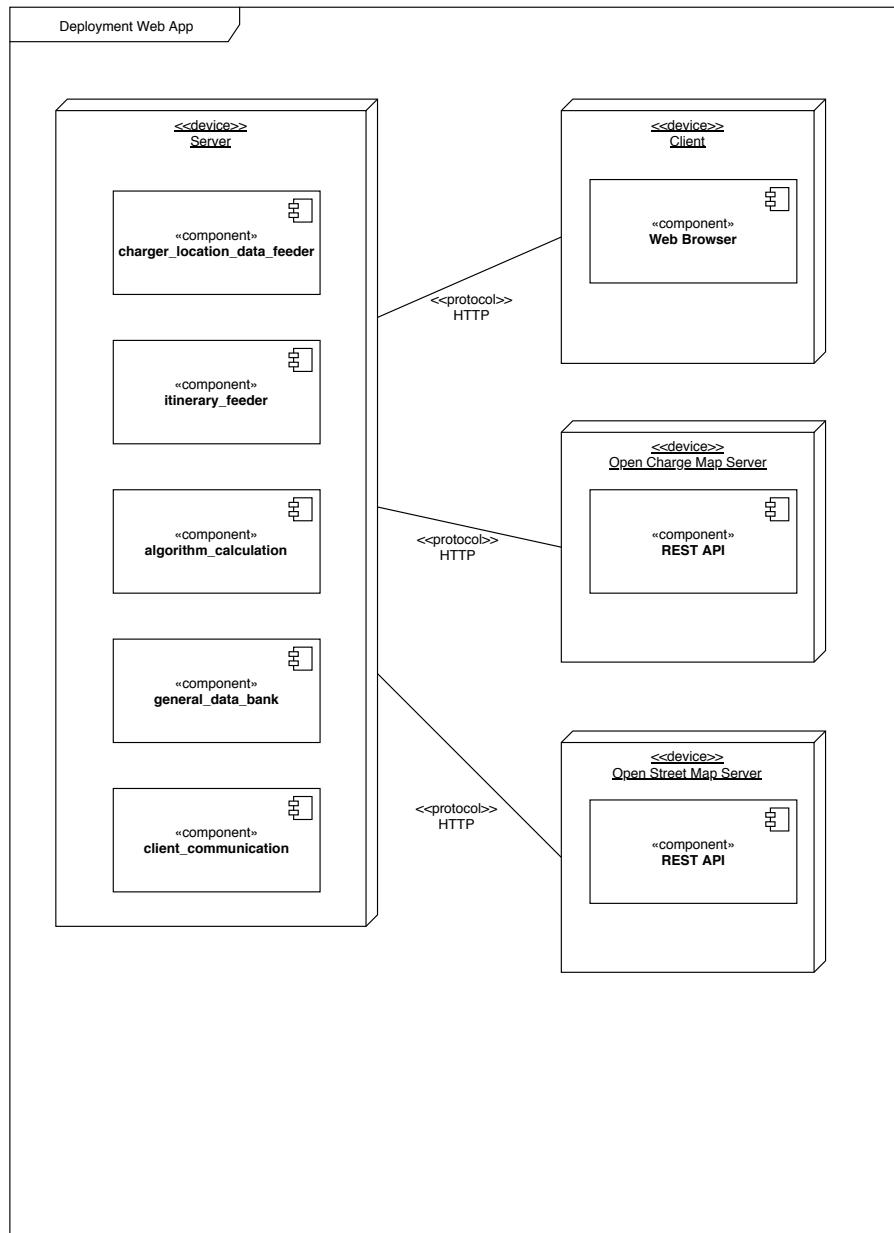


Figura 5.1: Deployment Diagram

http. Nel secondo livello il nostro server recita la parte del client, richiedendo dei servizi alle due API a cui ci appoggiamo. (Figura 5.2 e Figura 5.3)^[1]

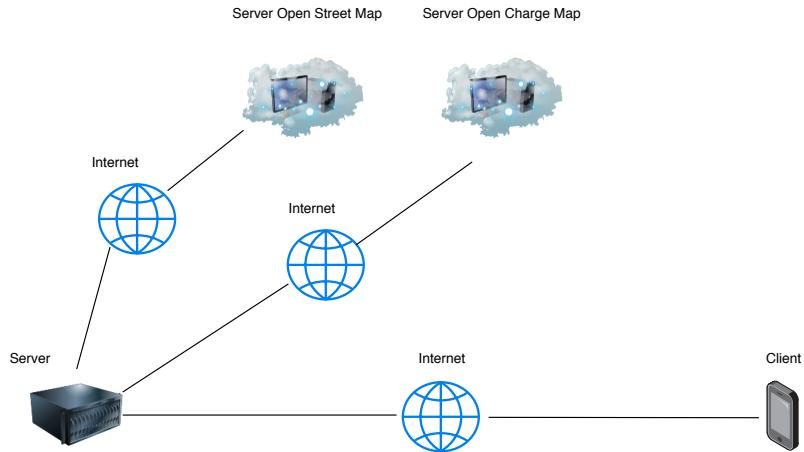


Figura 5.2: Architectural Diagram

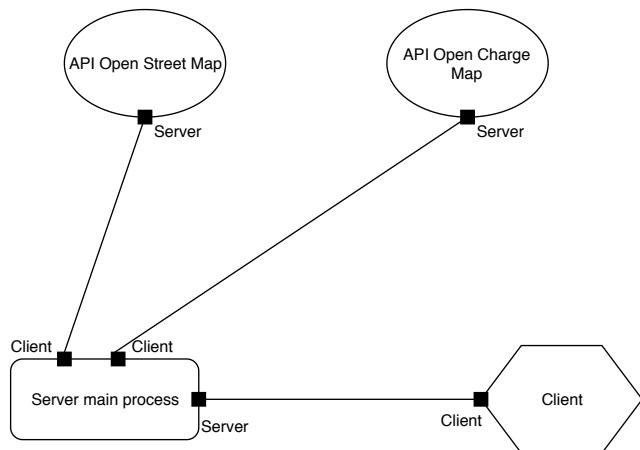


Figura 5.3: Client-Server Architecture

^[1] Len Bass, Paul Clements e Rick Kazman. *Software Architecture in Practice, 3rd Edition.* Inglese. ISBN: 9332502307.

5.2 Architettura Software

L'architettura software si divide in due diagrammi Component Diagram e Class Diagram.

5.2.1 Component Diagram

Il diagramma delle componenti ha lo scopo di rappresentare la struttura interna del sistema software modellato in termini dei suoi componenti principali e delle relazioni fra di essi. (Figura 5.4)

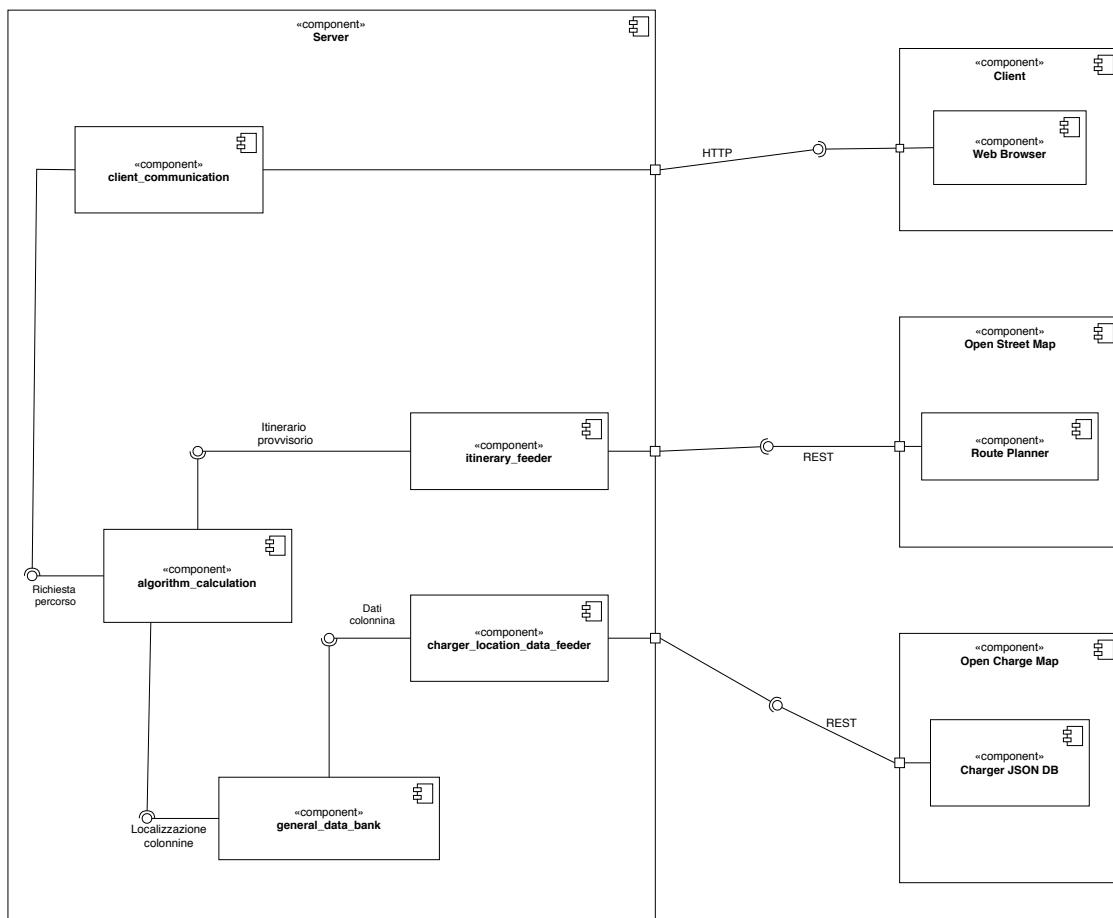


Figura 5.4: Component Diagram

5.2.2 Class Diagram

Il diagramma delle classi descrive il tipo degli oggetti che compongono il sistema e le relazioni statiche esistenti tra loro. (Figura 5.5)

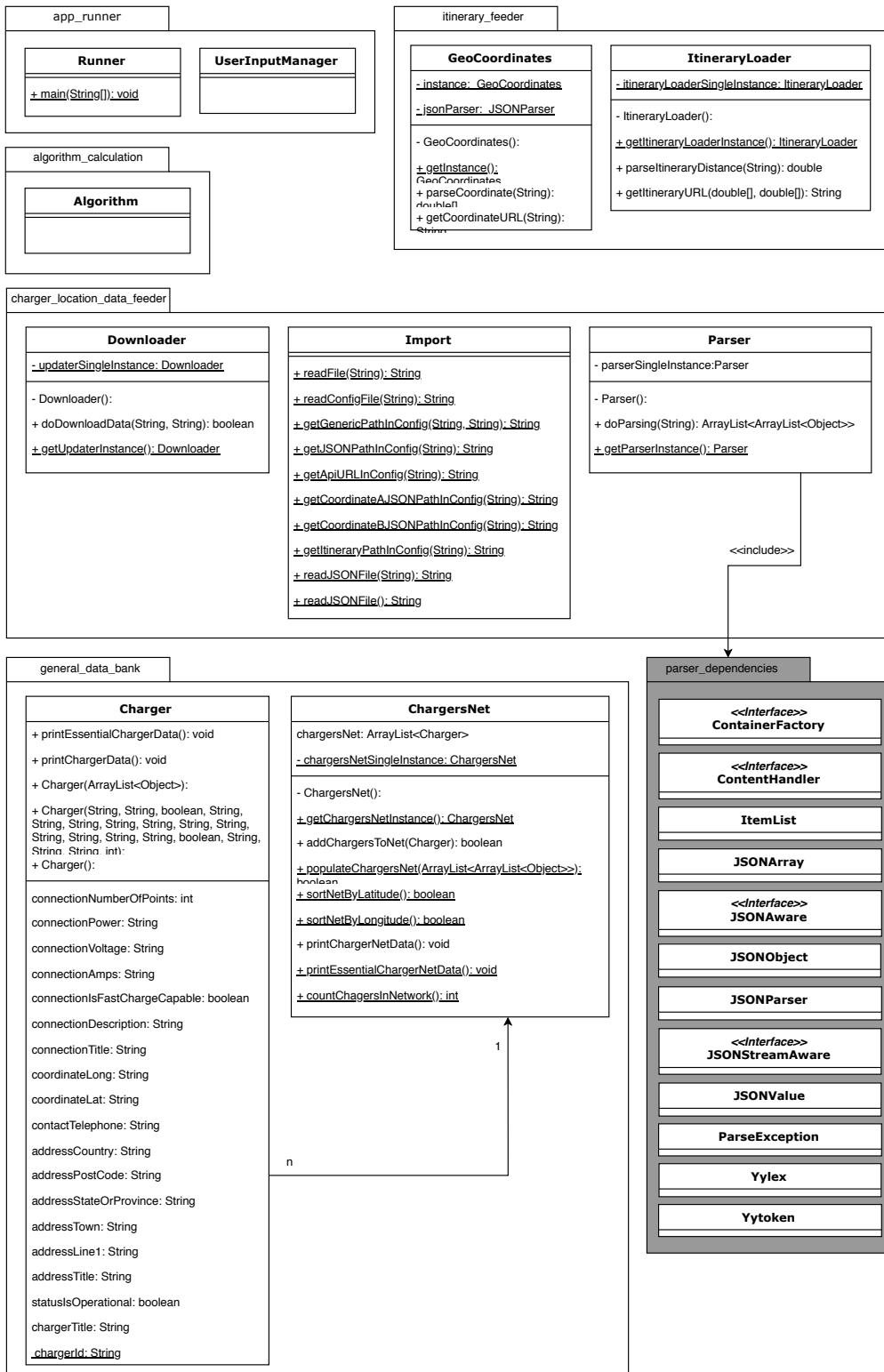


Figura 5.5: Class Diagram Architecture

[2]

[2] Mauro Ferrari e Giovanni Pighizzini. *Dai fondamenti agli oggetti: Corso di programmazione java.* 2008. 646 pp.
ISBN: 9788871924489.

Parte I

Iterazione 1

Capitolo 6

Parser

6.1 Scelta funzioni da implementare

Si è scelto di iniziare ad implementare le funzioni indispensabili del sistema da sviluppare tra cui la specifiche funzionali: R1, R6, R8 e i casi d'uso UC1 e UC2.

Per la parte di comunicazione tra l'applicazione e le API esterne, si è scelto di utilizzare il formato JSON, mentre per lo scaricamento e l'aggiornamento delle colonnine di ricarica viene usata una normale connessione http.

Lo scopo dell'applicazione è: date le colonnine di ricarica e il percorso da un punto A ad un punto B, essa calcolerà il percorso tenendo conto delle eventuali fermate per ricaricare l'auto.

I dati forniti all'algoritmo per l'elaborazione sono:

1. la posizione delle colonnine di ricarica
2. il percorso da un punto A ad un punto B

Per la parte di comunicazione tra il nostro Server e le API è stato sviluppato un Parser che permette una volta che il file viene scaricato dalle API di filtrarlo in base alle informazioni interessanti per il nostro progetto

INTERRUZIONE

Durante l'esecuzione di questa iterazione ci siamo resi conto che l'utilizzo del linguaggio Javascript per l'intero progetto avrebbe reso più agevole la realizzazione dello stesso. Abbiamo quindi deciso di interrompere questa iterazione e iniziare immediatamente la successiva dove illustreremo le modifiche effettuate all'iterazione 0 del processo.

Parte II

Iterazione 2

Capitolo 7

Template HTML

7.1 Variazione linguaggio di programmazione

Siccome per la parte di sviluppo della WebApp si è reso necessario utilizzare il linguaggio Javascript, la struttura e l'architettura del software sono state riviste al fine di utilizzare gli strumenti adatti allo sviluppo con tale linguaggio. Rimangono invariati rispetto a prima gli Use Cases, i requisisti funzionali del progetto e l'architettura hardware. Subisce invece variazioni l'architettura Software (deployment, component e class diagram).^[3]

Il nuovo deployment diagram si presenta come mostrato in Figura 7.1. Sono presenti 4 device: device Server che contiene il file Planner.vue che dovrà contenere il software sviluppato, la componente Leaflet API che avrà il compito di comunicare con le API di Open Street Map Server tramite protocollo REST. Inoltre servirà utilizzare il device Open Charge Map Server e il device Client per lo scaricamento della posizione delle colonnine sulla mappa e la visualizzazione della WebApp sul pc come nel Deploy precedente.

7.2 Template

Per lo sviluppo del software è stato sviluppato una parte di codice in HTML dove vengono definite le tre Textbox visualizzate a schermo, i due bottoni di Search e Reset e la visualizzazione della mappa a schermo centrata all'apertura sulla cartina geografica dell'Italia.^[4]

^[3] P Bill Scott e Theresa Neil. *Designing Web Interfaces: Principles and Patterns for Rich Interactions*. Inglese. Gen. 2009. 334 pp. ISBN: 9780596516253.

^[4] Jeremy Sydik. *Design Accessible Web Sites*. Inglese. Nov. 2007. 328 pp. ISBN: 9781934356029.

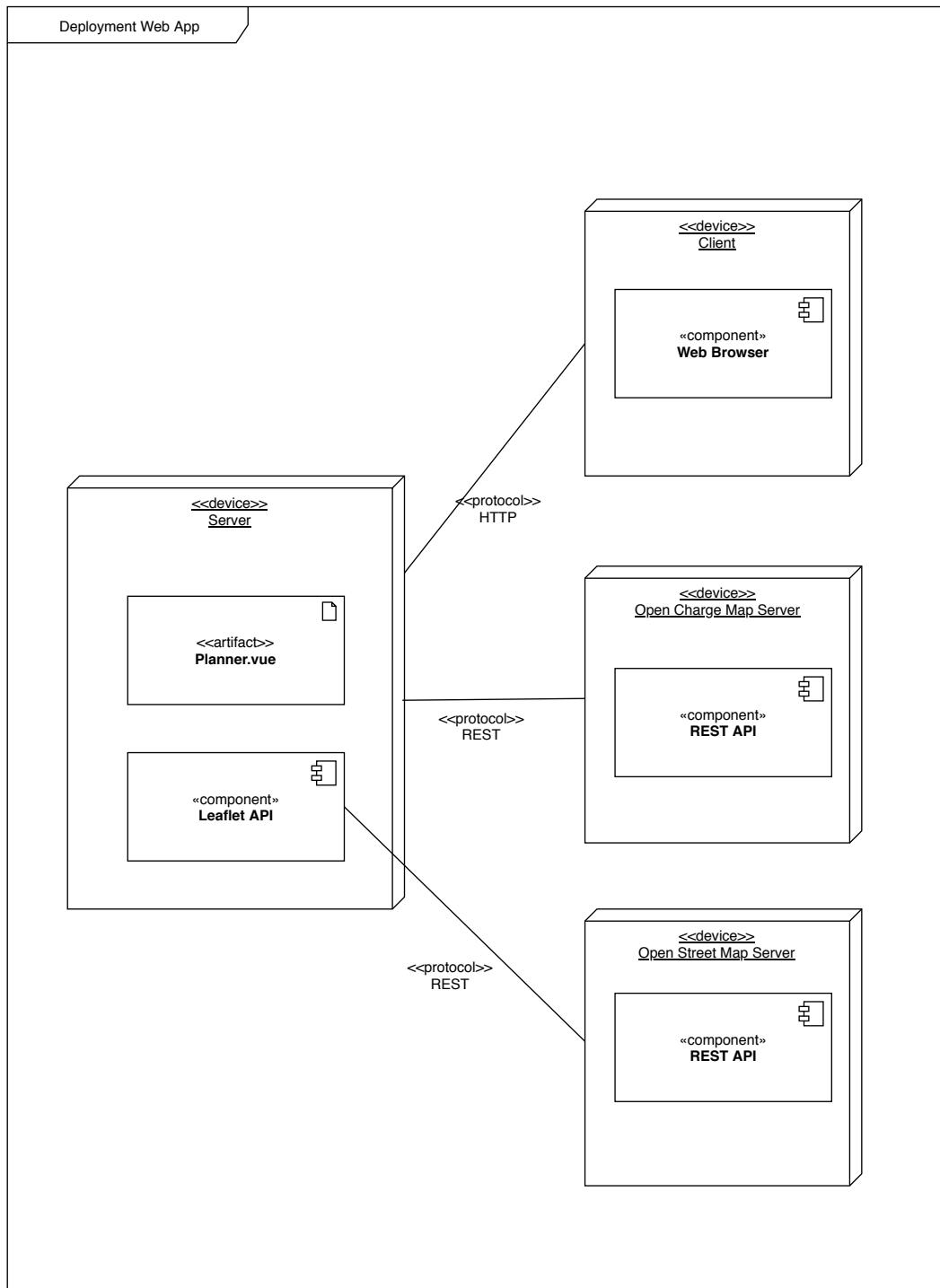


Figura 7.1: Template Deployment Diagram

Per la parte puramente grafica invece è presente un CSS che inserisce nel sito gli elementi grafici visualizzati dall'utente che utilizza il software.^[5]

7.3 Progettazione dei Test

Abbiamo provveduto alla progettazione dei test relativi a questa iterazione. Sappiamo che la nostra pagina web disporrà di campi in cui l'utente dovrà inserire dei valori e di pulsanti che potrà cliccare. Ci serve quindi un caso di test per questi elementi per assicurarsi che i valori inseriti vengano effettivamente ricevuti e che il click dei pulsanti chiami le relative funzioni: main() se viene cliccato il tasto "Search" e reset() se viene cliccato il tasto "Reset".^[6]

7.4 Template Sequence Diagram

Il diagramma di sequenza mostra come a partire da un attore, nel nostro caso un utente (User), il software evolve al fine di restituire all'utente un risultato che in questo caso specifico è rappresentato dalla visualizzazione su mappa del percorso con le fermate alle postazioni di ricarica.

Un utente inserisce l'url nel proprio Browser Web e questi restituirà l'Homepage del sito. Una volta che l'utente inserisce i dati richiesti questi vengono inviati all'algoritmo scritto in Javascript, che a sua volta invierà una richiesta all'API Leaflet che avrà il compito di interrogare il database delle mappe di Open Street Map e restituire all'algoritmo il percorso dal punto di partenza a quello di destinazione. L'algoritmo prenderà queste mappe e calcolerà quando è meglio fermarsi per ricaricare il veicolo elettrico, scegliendo tra le possibili colonnine sul percorso. Dopodichè restituirà l'intero percorso dal punto di partenza a quello di destinazione all'utente con le eventuali fermate da effettuare.^[7]

[5] Elisabeth Robson e Eric Freeman. *Head First HTML and CSS*. inglese. Set. 2012. 768 pp. ISBN: 9780596159900.

[6] Danny Goodman, Michael Morrison, Paul Novitski e Tia Gustaff Rayl. *JavaScript Bible*. Inglese. 1996. 1224 pp. ISBN: 9788126529100.

[7] Leonard Richardson e Sam Ruby. *RESTful Web Services*. Inglese. Mag. 2007. 454 pp. ISBN: 9780596529260.

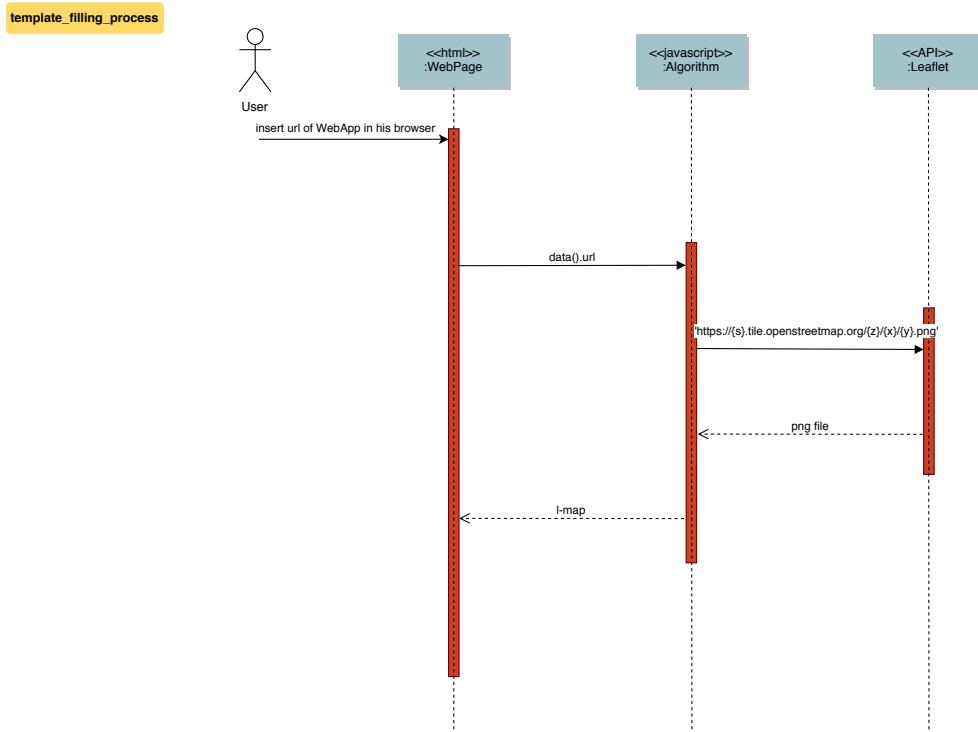


Figura 7.2: Template Sequence Diagram

7.5 Risultato dei Test

Per concludere questa iterazione presentiamo i risultati dei test che avevamo in precedenza definito. Come si può vedere dalla figura è stata eseguita una suite contenente 4 test che hanno avuto tutti esito positivo.

```

Algorithm.spec.js --era
Algoithm.spec.js — era
1: bash
~/Desktop/Progetto Scandurra/repository_progetto_info3b/src$ npm run test -- -u -t="Template Test"
Desktop/Progetto Scandurra/repository_progetto_info3b/src
planner@0.1.0 test
> jest --u --t=Template Test
PASS tests/unit/Template.spec.js

Test Suites: 2 skipped, 1 passed, 1 of 3 total
Tests:       12 skipped, 4 passed, 16 total
Snapshots:   0 total
Time:        3.144s
Ran all test suites with tests matching "Template Test".
~/Desktop/Progetto Scandurra/repository_progetto_info3b/src$ 
  
```

The screenshot shows the VS Code interface with the terminal tab active. The command `npm run test -- -u -t="Template Test"` was run, and the output shows 12 skipped tests and 4 passed tests, totaling 16 tests. The time taken for the test run was 3.144s.

Figura 7.3: Test iterazione 2

Parte III

Iterazione 3

Capitolo 8

Algoritmo

8.1 Algoritmo e Flowchart

In questo capitolo descriveremo i passi dell'algoritmo sviluppato. Essendo l'interazione complessa si è scelto di utilizzare un diagramma di flusso per dare una visione generale di più immediata comprensione. Il flowchart completo di tutto l'algoritmo è mostrato in Figura 8.1

8.1.1 Analisi di complessità

Procediamo ora all'analisi di complessità dell'algoritmo. Per cercare di analizzarlo risulta utile avere un'idea più chiara e di alto livello su ciò che l'algoritmo effettivamente fa; una spiegazione testuale può quindi risultare utile. L'esecuzione inizia richiedendo a OpenStreetMap il percorso fra il punto di partenza e il punto di arrivo inseriti dall'utente. L'API restituisce tutti i percorsi che è riuscita a trovare in ordine ottimale, ossia in ordine crescente di tempo di percorrenza. E' importante notare, per il proseguo della spiegazione, che i percorsi vengono forniti come una serie di punti adiacenti. A questo punto subentra l'algoritmo vero e proprio che analizza i percorsi nell'ordine in cui gli sono stati forniti, attraverso la funzione *getStopCircleCenterPoints*, nel seguente modo: partendo dal punto iniziale si sommano le distanze (che vanno calcolate), fra i punti che costituiscono il percorso fino a raggiungere una distanza maggiore dell'autonomia dichiarata dall'utente. A questo punto si torna indietro di un punto e lo si aggiunge all'array di punti dove sarà necessario fermarsi. Alla fine di quest'analisi si ha quindi la lista delle coordinate nei pressi delle quali è necessario fermarsi. Viene quindi chiamata la funzione *obtainChargersForThisRoute* la quale, insieme alla funzione *getSmallestElementsIndex*, trova il charger più vicino. La ricerca del charger avviene nel seguente modo: viene definito un cerchio di raggio 1 centrato nel punto trovato in precedenza e si cerca in quest'area il charger più vicino. Nel caso non venisse trovato

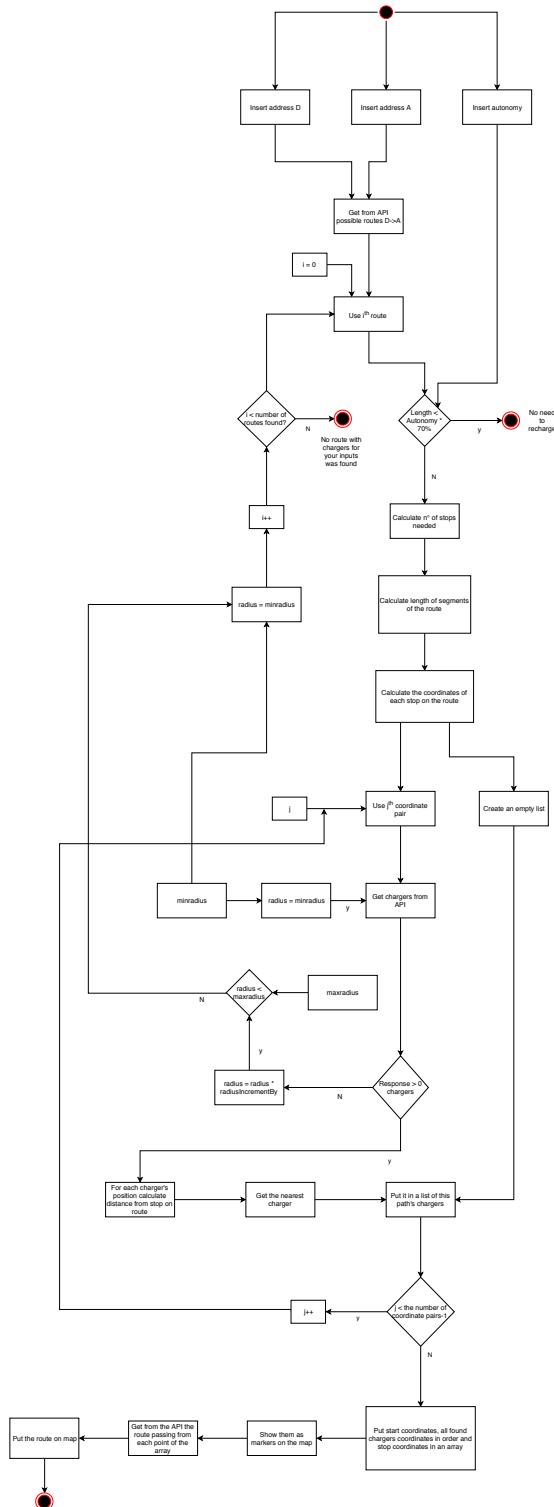


Figura 8.1: Flowchart Algorithm

alcun charger si estende incrementalmente il raggio del cerchio fino ad un massimo di

$$maxRadius = \frac{1 - batteryPercentageUsage}{2} * Autonomia$$

Le coordinate del charger trovato vengono inserite nel percorso e questo viene conseguentemente modificato per includere il passaggio per la colonnina di ricarica. Nel caso in cui non sia possibile trovare un charger per anche solo uno dei punti in cui è necessario fermarsi, il percorso viene dichiarato invalido e l'algoritmo procede ad esaminare il successivo risultato fornитогli dall'API. E' quindi possibile notare che l'algoritmo è costituito da una prima parte di programmazione dinamica, in cui si calcola la somma delle distanze fra tutti i punti successivi, e una seconda parte di stampo greedy, in cui si cerca la soluzione (charger) ottima più vicina, allargando mano a mano l'insieme dei candidati non ancora esplorati.^[8]

E' ora possibile analizzare la complessità dell'algoritmo: la parte dinamica si basa sulla somma degli n segmenti che costituiscono il percorso, e può quindi essere portata a termine in tempo $O(n)$. La seconda parte greedy viene eseguita nel seguente modo: l'algoritmo riceve la lista dei charger presenti nell'area e calcola la distanza di ciascuno di essi dal centro del cerchio, quindi seleziona il charger con distanza minore. Dato m il numero di charger restituiti dall'API per quell'area, l'operazione è semplicemente il calcolo della distanza e viene quindi espletata in tempo $O(m)$. Siccome il numero di charger restituiti è generalmente piccolo, l'operazione può essere eseguita in un tempo costante $\Theta(1)$. Si nota inoltre che la funzione per trovare i charger viene chiamata un numero costante di volte (molto inferiore a n), quindi l'intera operazione è $\Theta(1)$. Da ciò ne deriva che il tempo totale di esecuzione dell'algoritmo è dato solo dalla parte di calcolo delle somme delle distanze ed è perciò $O(n)$.

8.1.2 Pseudocodice

Segue lo pseudocodice.

```

1 /**
2  * Import libraries.
3 */
4 import leaflet // map generation and manipulation
5 import LMap, LTileLayer from vue2-leaflet // map generation and manipulation
6 import vue-places // address coordinate translation
7 import leaflet-routing-machine // route finding engine
8 import leaflet-control-geocoder // markers and description for points in map
9 import axios // API GET to fetch charger data

```

^[8] Alan A. Bertossi e Alberto Montresor. *Algoritmi e Strutture di Dati*. 2000. 402 pp.

```

1 /**
2  * Declare and initialize global variables.
3 */
4 data() {
5   travelData <- {
6     startPoint <- NIL,
7     arrivalPoint <- NIL,
8     carAutonomy <- NIL
9   },
10  startPointCoordinates <- {},
11  arrivalPointCoordinates <- {},
12  zoom <- 6,
13  center <- {lat: 41.8719, lng: 12.5674},
14  url <- 'https://{{s}}.tile.openstreetmap.org/{{z}}/{{x}}/{{y}}.png',
15  departureMarker <- {},
16  arrivalMarker <- {},
17  travelPath <- NIL,
18  finalTravelPath <- NIL,
19  configValues <- {
20    radiusIncrementBy <- 2,
21    batteryPercentageUsage <- 0.7, //70%
22    minRadius <- 1 // 1 km
23  },
24  markersLayers <- new layerGroup[]
25 }

```

```

1 /**
2  * On search button submittion, run main() calculations.
3 */
4 async main() { // runs all the webapp calculations.
5   let validity <- $refs['traveldatavalidateform'].validate() // validates the
       input of the user.
6   if (validity) { // if valid, the route calculations proceed, otherwise, a
       warning is returned.
7     Control.Geocoder.latLng( // set routing markers
8       startPointCoordinates,
9       startPointCoordinates
10    )
11    travelPath <- buildTravelPathStartToEnd() // build travel path from
       beginning to end and assign it to travelPath
12    travelPath.addTo(map) // add it to map
13    travelPath.on('routingerror', handleRoutingError) // if route wasn't
       found handle it using handleRoutingError()

```

```

14     travelPath.on('routesfound', await handleRoutesFound) // if route was
15     found handle it using handleRoutesFound()
16 } else {
17     $notify.warning('Form data not valid') // Form data inserted aren't valid.
18 }

1 /**
2 * If route was found do the handling and next calculations.
3 */
4 async handleRoutesFound(event) {
5     travelPath <- event.routes // get returned routes from OpenStreetMap API
6     and assign them to travelPath
7     let calculationResults <- await getPathChargerCoordinates() // await
8     charger coordinates from OpenChargeMap API
9     displayResults(calculationResults, map) // display results on map calling
10    displayResults() method.
11 }

1 /**
2 * Obtains chargers for route being considered
3 */
4 async obtainChargersForThisRoute(numberOfStopsArg, stopCircleCenterPointsArg) {
5     let maxRadius <- travelData.carAutonomy * ((1 - configValues.
6     batteryPercentageUsage) / 2)
7     let radius <- configValues.minRadius // Set to minimum the initial radius
8     of the circle area where to search for chargers of this stop.
9     let thisPointsIndex <- 0 // Consider the first charger. Will be incremented
10    in next inner iterations.
11    let finalChargersData <- new Array[]
12    while (thisPointsIndex < numberOfStopsArg && radius <= maxRadius) { // // // Inner while loop. Iterate while all chargers needed were not yet found and
13    radius is still smaller or equal to the maxRadius and not enough request
14    failures happened.
15        let innerChargersData <- new Array[], distancesFromStopPoint <- new
16        Array[] // distancesFromStopPoint: Empty (for now) array that will hold the
17        distance of each charger found in the circle search area to the center
18        coordinates of this stop point's. innerChargersData: new empty array.
19        let urlForApiRequest <- buildURL(stopCircleCenterPointsArg[
20        thisPointsIndex], radius) // Build url for api request.
21        innerChargersData <- await getChargerData(urlForApiRequest)
22        if (innerChargersData == undefined || innerChargersData.length == 0) {
23            radius <- radius * configValues.radiusIncrementBy // If the
24            response was undefined or contained no chargers in the circle area searched
25            , increment the circle's radius and continue to the next iteration.
26

```

```

15     continue
16 }
17 for (let j <- 0; j < innerChargersData.length; j++) { // If the
  response contained at least one charger, fill the distancesFromStopPoint
  array with the squares of the distance (of each charger found) form this
  stop's circle area's center.
18   distancesFromStopPoint.push(getDistanceFromLatLonInKm(
  stopCircleCenterPointsArg[thisPointsIndex][0], stopCircleCenterPointsArg[
  thisPointsIndex][1], innerChargersData[j].AddressInfo.Latitude,
  innerChargersData[j].AddressInfo.Longitude))
19 }
20 let minDistanceIndex <- getSmallestElementsIndex(distancesFromStopPoint
) // Find the index of the charger nearest to the center of the search
  circle.
21 finalChargersData.push(innerChargersData[minDistanceIndex])
22 radius <- configValues.minRadius // Reset the radius back to minimum.
23 thisPointsIndex++ // Increment the index to search for the next stop's
  chargers.
24 }
25 return finalChargersData
26 }

1 /**
2 * Calculates distances from point to point of the route
3 * polyline being considered. Adds each distance to an array.
4 * Returns the array containing all distances, from point 0 to
5 * point 1, from point 1 to point 2, and like this till the last
6 * point.
7 */
8 calculateDistancesFromRouteCoordinates(thisRouteArg) {
9   let segmentDistancesResult <- new Array[]
10  for (let coordIndex <- 1; coordIndex < thisRouteArg.coordinates.length;
  coordIndex++) {
11    let thisCoordCoupleDistanceVar <- getDistanceFromLatLonInKm(
12      thisRouteArg.coordinates[coordIndex - 1].lat,
13      thisRouteArg.coordinates[coordIndex - 1].lng,
14      thisRouteArg.coordinates[coordIndex].lat,
15      thisRouteArg.coordinates[coordIndex].lng)
16    segmentDistancesResult.push(thisCoordCoupleDistanceVar)
17  }
18  return segmentDistancesResult
19 }

1 /**

```

```

2 * From the segment distances calculated before and the number of
3 * stops needed that was found, this method finds the coordinates
4 * of the stops in the initial route. Those coordinates will
5 * become the centers of the circles where to search for
6 * chargers.
7 */
8 getStopCircleCenterPoints(thisRouteArg, segmentDistancesArg) {
9     let stopCircleCenterPointsResult <- new Array[]
10    let distanceDriven <- 0, leftOver <- 0, distanceDifference <- 0,
11    lastPartRatio <- 0
12    for (let thisSegmentDistanceIndex <- 0; thisSegmentDistanceIndex <
13        segmentDistancesArg.length; thisSegmentDistanceIndex++) {
14        if ((distanceDriven + segmentDistancesArg[thisSegmentDistanceIndex]) <
15            (travelData.carAutonomy * configValues.batteryPercentageUsage)) {
16            distanceDriven <- distanceDriven + segmentDistancesArg[
17            thisSegmentDistanceIndex]
18        } else {
19            distanceDifference <- (travelData.carAutonomy * configValues.
20            batteryPercentageUsage) - distanceDriven
21            lastPartRatio <- distanceDifference / segmentDistancesArg[
22            thisSegmentDistanceIndex]
23            let latOfInsidePoint <- thisRouteArg.coordinates[
24            thisSegmentDistanceIndex].lat + ((thisRouteArg.coordinates[
25            thisSegmentDistanceIndex].lat - thisRouteArg.coordinates[
26            thisSegmentDistanceIndex + 1].lat) * lastPartRatio)
27            let lngOfInsidePoint <- thisRouteArg.coordinates[
28            thisSegmentDistanceIndex].lng + ((thisRouteArg.coordinates[
29            thisSegmentDistanceIndex].lng - thisRouteArg.coordinates[
30            thisSegmentDistanceIndex + 1].lng) * lastPartRatio)
31            stopCircleCenterPointsResult.push([latOfInsidePoint,
32            lngOfInsidePoint])
33            leftOver <- segmentDistancesArg[thisSegmentDistanceIndex] -
34            distanceDifference
35            if (leftOver <= travelData.carAutonomy * configValues.
36            batteryPercentageUsage) {distanceDriven <- leftOver
37            } else {
38                while (leftOver >= (travelData.carAutonomy * configValues.
39                batteryPercentageUsage)) {
40                    distanceDifference <- leftOver - (travelData.carAutonomy *
41                    configValues.batteryPercentageUsage)
42                    lastPartRatio <- (travelData.carAutonomy * configValues.
43                    batteryPercentageUsage) / leftOver
44                    latOfInsidePoint <- latOfInsidePoint + ((latOfInsidePoint -

```

```

        thisRouteArg.coordinates[thisSegmentDistanceIndex + 1].lat) *
    lastPartRatio)
27            lngOfInsidePoint <- lngOfInsidePoint + ((lngOfInsidePoint -
    thisRouteArg.coordinates[thisSegmentDistanceIndex + 1].lng) *
    lastPartRatio)
28            stopCircleCenterPointsResult.push([latOfInsidePoint,
    lngOfInsidePoint])
29            leftOver <- leftOver - (travelData.carAutonomy *
    configValues.batteryPercentageUsage)
30        }
31        distanceDriven <- leftOver
32    }
33}
34
35    return stopCircleCenterPointsResult
36}

1 /**
2 * Method to get the index of the smallest integer of an array.
3 * It will be used to return the index of the charger
4 * nearest to the center point of the searching circle.
5 */
6 getSmallestElementsIndex(arrayOfElements) {
7     if (arrayOfElements.length == 0) {return NIL}
8     let smallestElement <- arrayOfElements[0]
9     let smallestElementsIndex <- 0
10    for (let i <- 1 i < arrayOfElements.length i++) {
11        if(smallestElement > arrayOfElements[i]) {smallestElement <-
arrayOfElements[i] smallestElementsIndex <- i}
12    }
13    return smallestElementsIndex
14}

1 /**
2 * After having done all the calculations and obtained a result
3 * on the charger locations where to stop for recharging,
4 * this method displays the results on the map inside the page.
5 */
6 displayResults(calculationDataArg) {
7     let finalChargersData <- calculationDataArg[1]
8     if (!calculationDataArg[0]) {
9         $notify.warning('It wasn\'t possible to find an itinerary with chargers
') // If itineraryFoundBool <- false, No itinerary found
10        return false // Failure

```

```

11     } else {
12         for (let k <- 0; k < finalChargersData.length; k++) {
13             setChargerMarker([finalChargersData[k].AddressInfo.Latitude,
14                               finalChargersData[k].AddressInfo.Longitude])
15                 .bindTooltip(buildStringForTooltip(finalChargersData, k),
16                             setMarkerAttributes)
17                 .addTo(map)
18                 .openTooltip()
19             } // Add found charger markers to markersLayers group.
20             let finalChargersCoordinates <- new Array[]
21             finalChargersCoordinates.push([
22                 startPointCoordinates.lat,
23                 startPointCoordinates.lng
24             ]) // Add to finalChargersData start coordinates
25             finalChargersData.forEach(function(elementOfData) {
26                 finalChargersCoordinates.push([elementOfData.AddressInfo.Latitude,
27                     elementOfData.AddressInfo.Longitude])
28             })
29             finalChargersCoordinates.push([
30                 arrivalPointCoordinates.lat, arrivalPointCoordinates.lng
31             ]) // and end point coordinates, needed to compute whole itinerary.
32             finalTravelPath <- buildTravelPath(finalChargersCoordinates) // The
33             path/itinerary computed on finalPathPointCoordinates points and to be
34             showed on the map.
35             finalTravelPath.addTo(map)
36             if (finalChargersCoordinates.length == 2) {
37                 $notify.success('No recharge needed') // Notify user no recharge
38                 needed for the trip.
39                 return true // Success
40             } else {
41                 return true // Success. Itinerary with chargers found.
42             }
43         }
44     }

```

8.1.3 Diagrammi di attività

Il diagramma di sequenza è mostrato in Figura 8.2 questo diagramma mostra come le varie componenti interagiscono tra di loro e i dati che si scambiano.

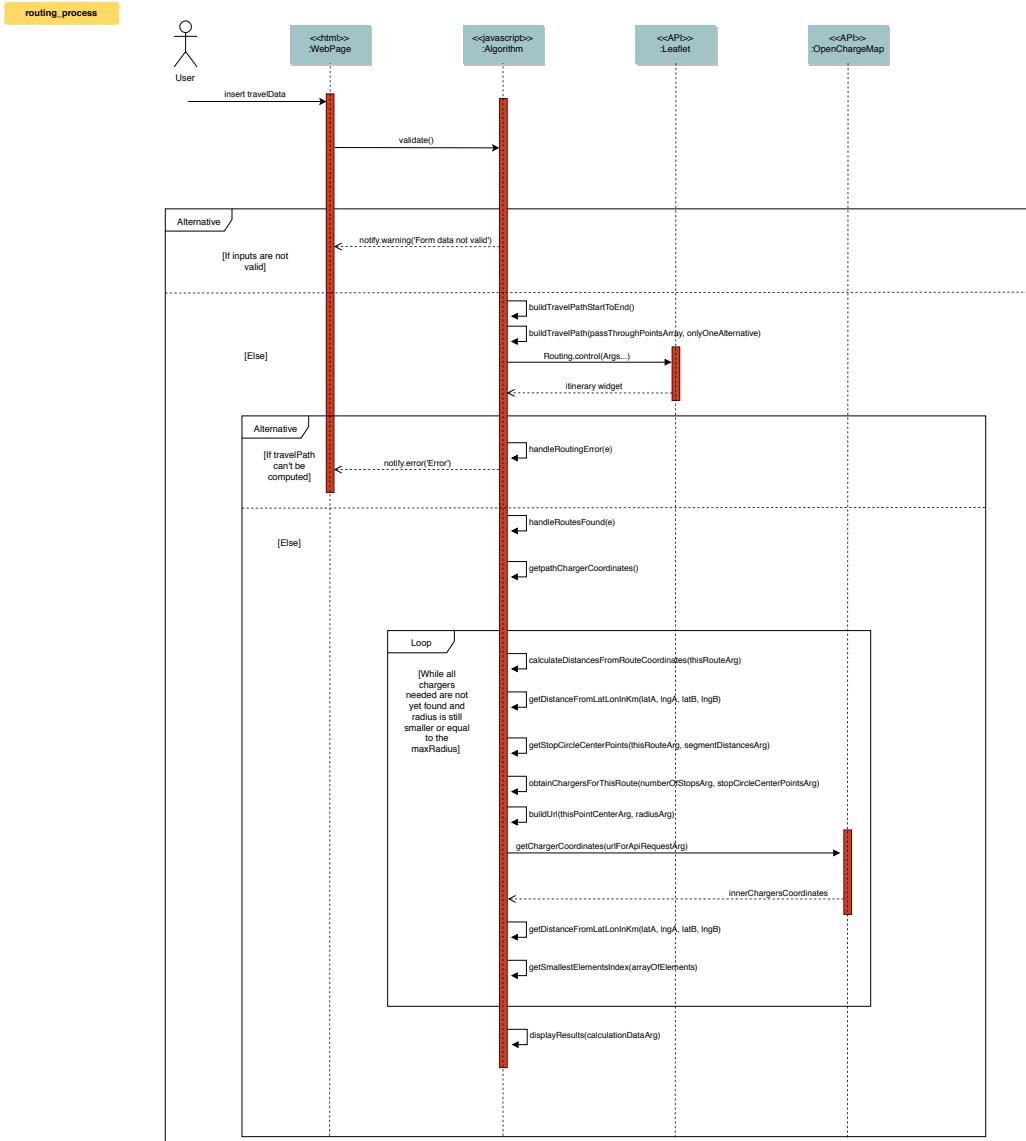


Figura 8.2: Algorithm Sequence Diagram

Nella Figura 8.3 sono mostrati, più in dettaglio, i metodi e le relative chiamate che l'algoritmo utilizza per la sua computazione.

8.2 Progettazione dei Test

L'algoritmo che abbiamo progettato è costituito da diverse funzioni. Per verificarne il comportamento testeremo queste funzioni controllando che i parametri passati e i valori restituiti coincidano con quanto ci aspettiamo. In particolare testeremo le funzioni riportate in tabella 8.1 .

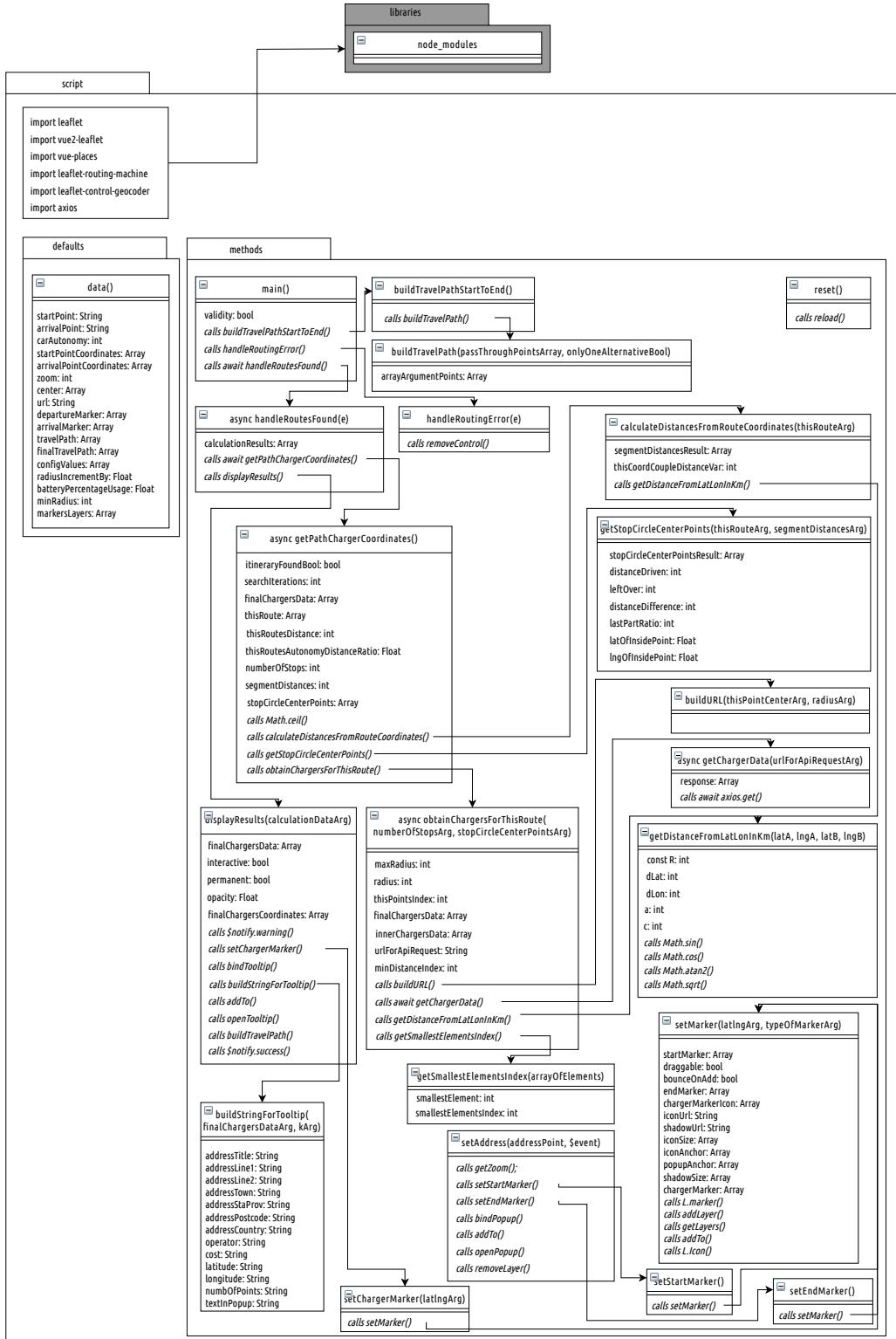


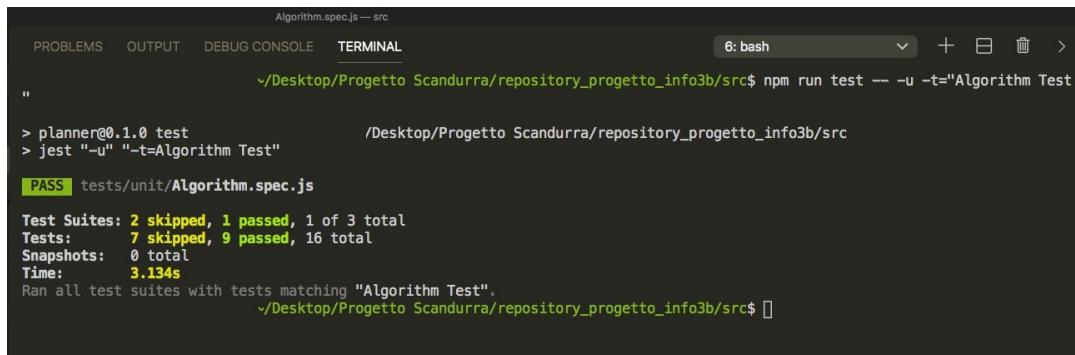
Figura 8.3: Methods

Funzione	Descrizione	Esito test
setMarker	aggiunge un Marker alla lista	Passato
setChargerMarker	aggiunge un Marker per un charger	Passato
setStartMarker	aggiunge il Marker per il punto di partenza	Passato
setEndMarker	aggiunge il Marker per il punto di arrivo	Passato
buildStringForTooltip	costruisce la stringa da visualizzare a schermo riportante le informazioni circa il punto di ricarica	Passato
getSmallestElementsIndex	ottiene l'indice del charger più vicino al punto selezionato	Passato
buildUrl	costruisce l'URL per effettuare la richiesta all'API di Open Charge Map	Passato
getStopCircleCenterPoints	calcola il punto lungo il tragitto in cui è necessario fermarsi per la ricarica	Passato
getDistanceFromLatLonInKm	calcola la distanza fra due punti	Passato

Tabella 8.1: Funzioni implementate nella suite di test

8.3 Risultato dei Test

Il risultato dei test eseguiti è riportato nella figura qui sotto. E' stata eseguita una suite composta di nove test che hanno ricevuto esito positivo.



```

Algorithm.spec.js — src
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
6: bash
~/Desktop/Progetto Scandurra/repository_progetto_info3b/src$ npm run test -- -u -t="Algorithm Test"
"
> planner@0.1.0 test
> jest "-u" "-t=Algorithm Test"
PASS  tests/unit/Algorithm.spec.js
Test Suites: 2 skipped, 1 passed, 1 of 3 total
Tests:    7 skipped, 9 passed, 16 total
Snapshots: 0 total
Time:    3.134s
Ran all test suites with tests matching "Algorithm Test".
~/Desktop/Progetto Scandurra/repository_progetto_info3b/src$ 

```

Figura 8.4: Test iterazione 3

E' quindi possibile passare all'ultima iterazione del nostro processo AMDD.

Parte IV

Iterazione 4

Capitolo 9

Visualizzazione Percorso

In questa ultima sezione ci occupiamo di progettare la parte del software che permetterà la visualizzazione del percorso (calcolato in precedenza), sulla mappa, con le eventuali fermate da effettuare.

9.1 Progettazione dei Test

Sappiamo che la funzione che si occuperà di disegnare il tragitto dovrà prendere in input le coordinate dei punti di sosta, creare i marker sulla mappa, tracciare il percorso e restituire un valore booleano. Sappiamo inoltre che all'interno di questa funzione ci saranno diversi *if statements*, quindi ci assicuriamo di testare le varie casistiche che si svilupperanno in fase di esecuzione.

9.2 Pseudocodice

Di seguito è riportato in pseudocodice la parte di visualizzazione del percorso sulla mappa.^[9]

```
1 /**
2  * After having done all the calculations and obtained a result
3  * on the charger locations where to stop for recharging,
4  * this method displays the results on the map inside the page.
5 */
6 displayResults(calculationDataArg) {
7     let finalChargersData <- calculationDataArg[1]
8     if (!calculationDataArg[0]) {
```

^[9] Frank Zammetti. *Practical Javascript, Dom Scripting and Ajax Projects*. Inglese. Apr. 2007. 546 pp. ISBN: 9781590598160.

```

9      $notify.warning('It wasn\'t possible to find an itinerary with chargers
') // If itineraryFoundBool <- false, No itinerary found
10     return false // Failure
11 } else {
12     for (let k <- 0; k < finalChargersData.length; k++) {
13         setChargerMarker([finalChargersData[k].AddressInfo.Latitude,
14             finalChargersData[k].AddressInfo.Longitude])
15         .bindTooltip(buildStringForTooltip(finalChargersData, k),
16             setMarkerAttributes)
17         .addTo(map)
18         .openTooltip()
19     } // Add found charger markers to markersLayers group.
20     let finalChargersCoordinates <- new Array[]
21     finalChargersCoordinates.push([
22         startPointCoordinates.lat,
23         startPointCoordinates.lng
24     ]) // Add to finalChargersData start coordinates
25     finalChargersData.forEach(function(elementOfData) {
26         finalChargersCoordinates.push([elementOfData.AddressInfo.Latitude,
27             elementOfData.AddressInfo.Longitude])
28     })
29     finalChargersCoordinates.push([
30         arrivalPointCoordinates.lat, arrivalPointCoordinates.lng
31     ]) // and end point coordinates, needed to compute whole itinerary.
32     finalTravelPath <- buildTravelPath(finalChargersCoordinates) // The
33     path/itinerary computed on finalPathPointCoordinates points and to be
34     showed on the map.
35     finalTravelPath.addTo(map)
36     if (finalChargersCoordinates.length == 2) {
37         $notify.success('No recharge needed') // Notify user no recharge
38         needed for the trip.
39         return true // Success
40     } else {
41         return true // Success. Itinerary with chargers found.
42     }
43 }

```

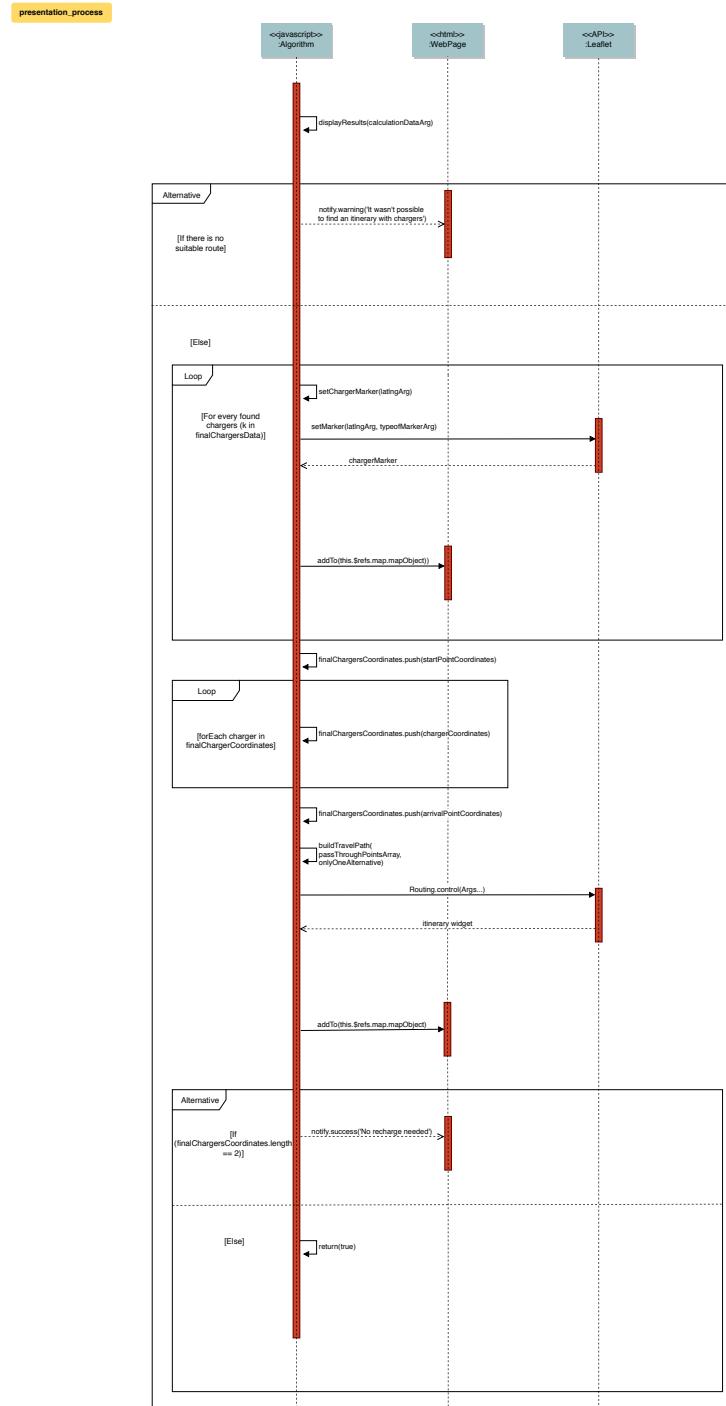
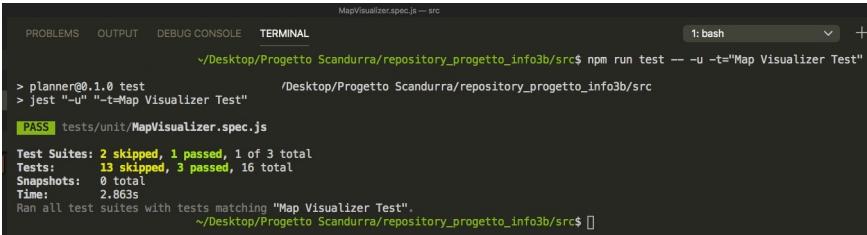


Figura 9.1: Template Deployment Diagram

9.3 Risultato dei Test

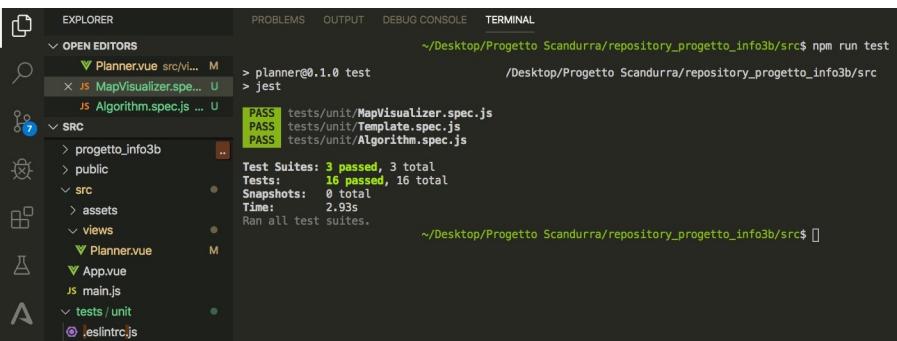
Il risultato dei test per questa iterazione è mostrato in figura. La suite composta da 3 test ha ricevuto esito positivo.



```
MapVisualizer.spec.js -- src
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
1: bash + +
~/Desktop/Progetto Scandurra/repository_progetto_info3b/src$ npm run test -- -u -t="Map Visualizer Test"
> planner@0.1.0 test
> jest "-u" "-t=Map Visualizer Test"
PASS tests/unit/MapVisualizer.spec.js
Test Suites: 2 skipped, 1 passed, 1 of 3 total
Tests:    13 skipped, 3 passed, 16 total
Snapshots: 0 total
Time:    2.863s
Ran all test suites with tests matching "Map Visualizer Test".
~/Desktop/Progetto Scandurra/repository_progetto_info3b/src$
```

Figura 9.2: Test iterazione 3

Presentiamo infine anche il risultato di tutte le suite di test per tutte le iterazioni eseguite contemporaneamente.



```
EXPLORER PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
~/Desktop/Progetto Scandurra/repository_progetto_info3b/src$ npm run test
> planner@0.1.0 test
> jest
PASS tests/unit/MapVisualizer.spec.js
PASS tests/unit/Template.spec.js
PASS tests/unit/Algorithm.spec.js
Test Suites: 3 passed, 3 total
Tests:    16 passed, 16 total
Snapshots: 0 total
Time:    2.93s
Ran all test suites.
~/Desktop/Progetto Scandurra/repository_progetto_info3b/src$
```

Figura 9.3: Test completo

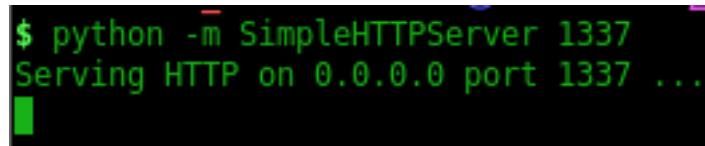
Capitolo 10

Manuale Utente

10.1 Installazione

Per l'esecuzione del software sviluppato è necessario aprire l'applicazione in locale tramite le istruzioni di seguito riportate.

```
1 // LANCIARE IL SITO (VERSIONE DI PRODUZIONE/BUILD)
2
3 // Cambiare directory andando nella root dei file
4 // del progetto build. In VueJS e la cartella /dist
5 $ cd /indirizzo/alla/directory/dist
6
7 // Installare un virtual server di HTML & JavaScript
8 $ sudo apt-get install http.server
9
10 // Lanciare il server con Python 2.*
11 $ python -m SimpleHTTPServer 1337
12
13 // Lanciare il server con Python 3.*
14 $ python -m http.server 1337
15
16 // o in alternativa si puo usare http-server di NodeJS
17 $ npm install http-server -g
18 $ http-server /indirizzo/alla/directory/dist -p 1337
19
20 // Sfogliare la pagina nell'indirizzo
21 $ firefox localhost:1337/
```



```
$ python -m SimpleHTTPServer 1337
Serving HTTP on 0.0.0.0 port 1337 ...
```

Figura 10.1: Virtual server in esecuzione

Questa modalità di visualizzazione del sito presuppone che il sito sia già compilato per versione di produzione.

Se invece si volesse lanciare il sito dal codice sorgente, seguire le seguenti istruzioni.

```
1 // LANCIARE IL SITO (VERSIONE DI DEVELOPMENT/SERVE)
2
3 // Cambiare directory andando nella root dei file
4 // del progetto build. In VueJS è la cartella /src
5 $ cd /indirizzo/alla/directory/src
6
7 // Installare un gestore di packages di JavaScript
8 $ sudo apt-get install yarn
9 // oppure in alternativa si può usare npm
10 $ sudo apt-get install nodejs npm
11
12 // Installare tutte le packages necessarie per il progetto.
13 $ yarn
14 // oppure
15 $ yarn add
16 // o
17 $ npm install
18
19 // Lanciare il sito mediante il comando
20 $ yarn serve
21 // oppure
22 $ npm run serve
23
24 // Sfogliare la pagina nell'indirizzo
25 $ firefox localhost:8080/
```

Il building si può fare con i seguenti comandi.

```
1 // BUILDING DEL SITO (VERSIONE DI PRODUZIONE/BUILD)
2 $ yarn build
3 // oppure tramite
4 $ npm run build
```

```
yarn run v1.21.1
warning package.json: "dependencies" has dependency "@vue/test-utils" with ran-
ith version "1.0.0-beta.29"
$ vue-cli-service serve
  INFO  Starting development server...
98% after emitting CopyPlugin DONE  Compiled successfully in 7767ms5:45:17 PM

App running at:
- Local: http://localhost:8080/
- Network: http://172.16.129.40:8080/

Note that the development build is not optimized.
To create a production build, run yarn build.
```

Figura 10.2: "yarn serve" sito versione development in esecuzione

Una volta che le istruzioni sopra citate sono state eseguite verrà aperta una porta in locale nel proprio pc (ad.esempio <http://localhost:8080/>) con una schermata come riportato nella Figura 10.2.^[10]

Si procede aprendo un qualsiasi browser web e incollando l'indirizzo trovato nella barra di ricerca e premendo invio, il browser restituirà la Homepage della WebApp sviluppata.

^[10] Brad Dayley. *Node.js, MongoDB, and AngularJS Web Development*. Inglese. Giu. 2014. 647 pp. ISBN: 9780321995780.

```
warning
asset size limit: The following asset(s) exceed the recommended size limit (244 KiB).
This can impact web performance.
Assets:
  css/chunk-vendors.df948891.css (254 KiB)
  js/chunk-vendors.624a449e.js (1.63 MiB)

warning
entrypoint size limit: The following entrypoint(s) combined asset size exceeds the recommended limit (244 KiB). This can impact web performance.
Entrypoints:
  app (1.89 MiB)
    css/chunk-vendors.df948891.css
    js/chunk-vendors.624a449e.js
    css/app.0fe05ebc.css
    js/app.876da472.js

warning
webpack performance recommendations:
You can limit the size of your bundles by using import() or require.ensure to lazy load some parts of your application.
For more info visit https://webpack.js.org/guides/code-splitting/
      File          Size        Gzipped
dist/js/chunk-vendors.624a449e.js     1667.53 KiB   384.02 KiB
dist/js/app.876da472.js                14.47 KiB    4.72 KiB
dist/css/chunk-vendors.df948891.css   254.47 KiB   45.32 KiB
dist/css/app.0fe05ebc.css              0.50 KiB    0.29 KiB

Images and other types of assets omitted.

DONE Build complete. The dist directory is ready to be deployed.
INFO Check out deployment instructions at https://cli.vuejs.org/guide/deployment.html
Done in 28.05s.
```

Figura 10.3: "yarn build" compilazione del sito versione di produzione

10.2 Utilizzo

In questa sezione verrà mostrato l'utilizzo dell'applicazione web sviluppata. All'apertura della pagina web verrà mostrata una schermata contenente le caselle da riempire con i dati richiesti ed una mappa per la successiva visualizzazione. Dopo l'inserimento dell'indirizzo sul proprio browser web verrà mostrato a schermo una pagina come nella Figura 10.4

La schermata è composta da 3 campi:

1. **Departure Address:** in questo campo deve essere inserita la città di partenza con l'eventuale indirizzo di partenza;
2. **Arrival Address:** in questo campo deve essere inserita la città di destinazione con l'eventuale indirizzo di destinazione;
3. **Car Autonomy:** in questo campo va inserita l'autonomia del proprio veicolo elettrico.

Il sistema è in grado di offrire dei suggerimenti riguardo l'inserimento delle destinazioni iniziali e finali e fare un check dei valori inseriti prima dell'invio al sistema. In particolare nel campo autonomia non sono ammessi valori negativi ma soltanto valori compresi tra 1 e 1000 km.

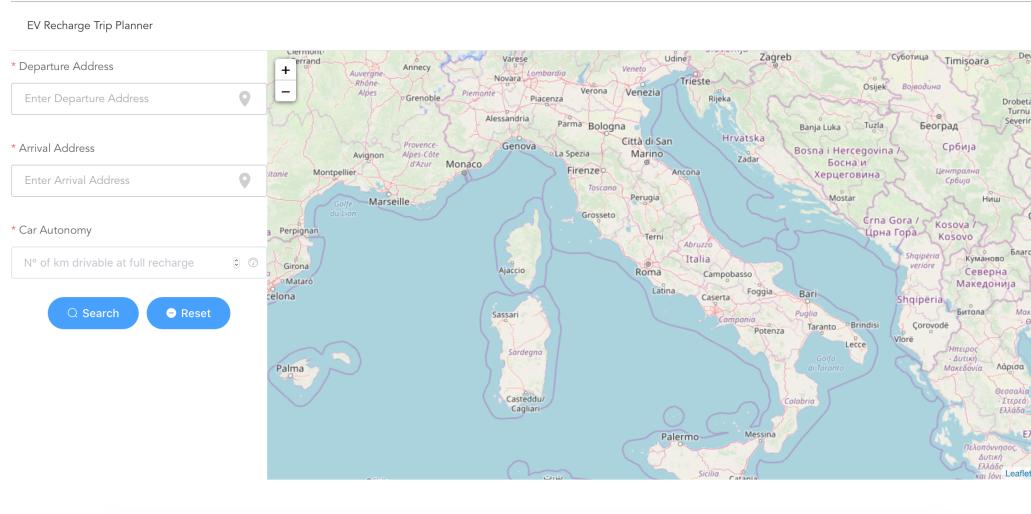


Figura 10.4: Schermata Homepage

Una volta inseriti nella schermata i dati richiesti, l'utente dovrà premere il pulsante Search come mostrato in Figura 10.5.

Il sistema risponderà con una schermata come in Figura 10.6 dove viene mostrato sulla mappa il punto iniziale e finale con il percorso da effettuare e le eventuali fermate.

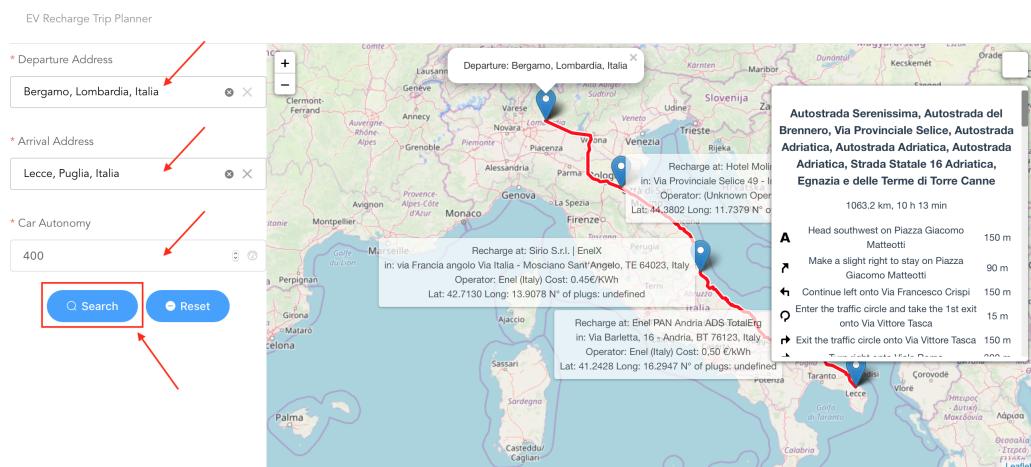


Figura 10.5: Schermata Esempio Planner

In particolare vengono mostrate sulla mappa le fermate da effettuare con il nome della colonnina di ricarica, la via in cui essa è situata, l'operatore erogatore del servizio, il costo della colonnina e il numero di prese di corrente disponibili.

Inoltre compare una finestra a tendina interattiva dove sono mostrate le indicazioni dettagliate da seguire per raggiungere il posto desiderato. La tendina è scorribile in lunghezza e posizionando la freccia su una specifica indicazione è possibile vedere sulla mappa tramite un cerchio blu dove corrisponde l'indicazione sulla mappa.

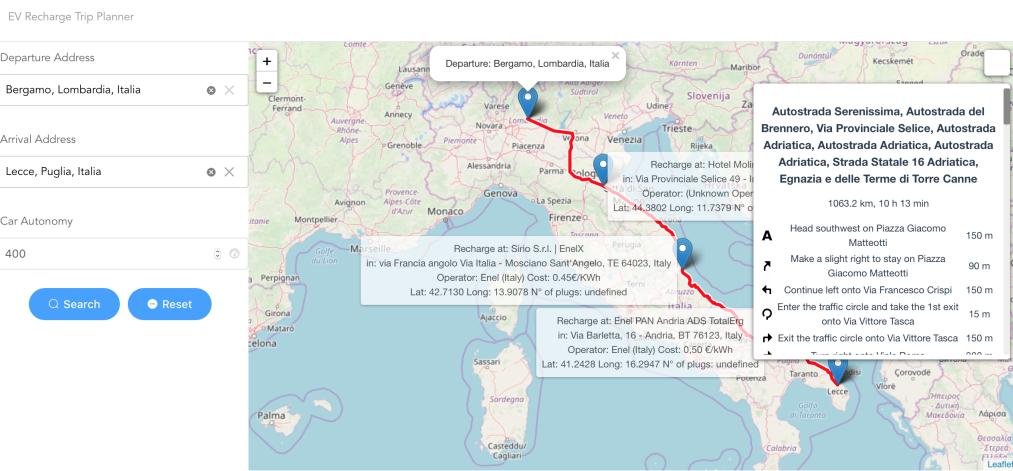


Figura 10.6: Schermata Esempio Planner 2

Una volta che l'utente ha visualizzato un determinato percorso esso potrà cancellare quello appena cercato ed effettuarne uno nuovo tramite il pulsante Reset come mostrato in Figura 10.7, la schermata che si presenterà è identica alla Homepage del programma.

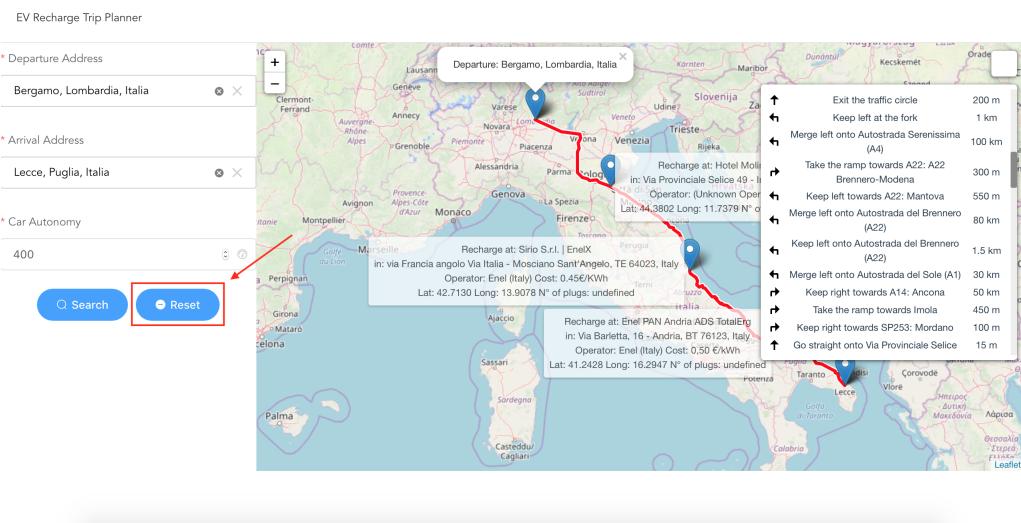


Figura 10.7: Schermata Esempio Planner 3

Capitolo 11

Toolchain

11.1 Tools

Per realizzare il nostro progetto ci siamo avvalsi dei seguenti tool:

1. termux: terminale per mobile
2. python: linguaggio di programmazione usato per "runnare" il server virtuale
3. http-server: server virtuale per runnare siti web
4. Visual Studio: IDE di sviluppo general purpose
5. SublimeText: text editor per sviluppo general purpose
6. pkg: package manager per terminale mobile
7. pip: python package manager
8. Mozilla Firefox: Web Browser
9. Safari: Web Browser
10. Chromium : Web Browser
11. Konsole: terminale per ambiente GNU/Linux
12. tmux: multiplexer di finestre di terminale
13. gige: software per GUI git, gestione branch e versioning
14. inspector: web browser debugging tool
15. evince: pdf document viewer
16. inkscape: editore di immagini e documenti raster
17. Git Cola: GUI software per la gestione dei branch e versioning git
18. Mocha: libreria per VueJS testing
19. Synaptic Package Manager: package manager per sistemi GNU/Linux

20. Jest: libreria per VueJs testing
21. ESLint: analisi statica del codice
22. Mega: servizio di archiviazione e backup online
23. Brackets: IDE per sviluppo HTML e CSS
24. Wireshark: software per catturare ed analizzare pacchetti scambiati durante le interrogazioni alle API
25. Github: sito di hosting di repository
26. WebPack: module bundler e minimizer per applicazioni JavaScript
27. SourceTree: client di gestione versioning con GUI git
28. Eclipse: IDE per la programmazione in Java
29. WebStorm: IDE per programma in VueJS Framework di Javascript
30. NodeJS: gestore di moduli e librerie per sviluppo di siti web
31. Yarn: package manager
32. Bitbucket: servizio web per la gestione di repository git
33. Overleaf: servizio web per la creazione e archiviazione di documenti L^AT_EX. Consente inoltre la collaborazione di più utenti sullo stesso documento in tempo reale
34. Draw.io: servizio Web per la creazione di grafici fra cui anche i diagrammi UML
35. Telegram: app di messaggistica

11.2 Librerie

Per la realizzazione del nostro progetto ci siamo avvalsi delle seguenti librerie:

1. Leaflet
2. Vue2 leaflet
3. vue places
4. leaflet routing machine
5. leaflet control geocoder
6. axios

Bibliografia

1. Len Bass, Paul Clements e Rick Kazman. *Software Architecture in Practice, 3rd Edition.* Inglese. ISBN: 9332502307.
2. Mauro Ferrari e Giovanni Pighizzini. *Dai fondamenti agli oggetti: Corso di programmazione java.* 2008. 646 pp. ISBN: 9788871924489.
3. P Bill Scott e Theresa Neil. *Designing Web Interfaces: Principles and Patterns for Rich Interactions.* Inglese. Gen. 2009. 334 pp. ISBN: 9780596516253.
4. Jeremy Sydik. *Design Accessible Web Sites.* Inglese. Nov. 2007. 328 pp. ISBN: 9781934356029.
5. Elisabeth Robson e Eric Freeman. *Head First HTML and CSS.* Inglese. Set. 2012. 768 pp. ISBN: 9780596159900.
6. Danny Goodman, Michael Morrison, Paul Novitski e Tia Gustaff Rayl. *JavaScript Bible.* Inglese. 1996. 1224 pp. ISBN: 9788126529100.
7. Leonard Richardson e Sam Ruby. *RESTful Web Services.* Inglese. Mag. 2007. 454 pp. ISBN: 9780596529260.
8. Alan A. Bertossi e Alberto Montresor. *Algoritmi e Strutture di Dati.* 2000. 402 pp.
9. Frank Zammetti. *Practical Javascript, Dom Scripting and Ajax Projects.* Inglese. Apr. 2007. 546 pp. ISBN: 9781590598160.
10. Brad Dayley. *Node.js, MongoDB, and AngularJS Web Development.* Inglese. Giu. 2014. 647 pp. ISBN: 9780321995780.

