



UNIVERSITA' DEGLI STUDI DI  
NAPOLI FEDERICO II

Scuola Politecnica e delle Scienze di Base  
Corso di Laurea Magistrale in Ingegneria Informatica

Elaborato d'esame

## *Software Architecture Design*

Anno 2022/23

Professore:

**Annarita Fasolino**

A cura di:

**Gianluigi Liguori M63001246**

**Mario Sicignano M63001187**

**Agostino Vallefuoco M63001175**

# Indice

<b>1</b>	<b>Introduzione</b>	<b>1</b>
1.1	Presentazione del progetto . . . . .	1
1.2	Organizzazione del lavoro . . . . .	1
1.2.1	Tecnologie, strumenti e linguaggi adottati . . .	2
<b>2</b>	<b>Sprint Planning</b>	<b>5</b>
2.0.1	Primo Sprint Planning . . . . .	5
2.0.2	Secondo Sprint Planning . . . . .	5
2.0.3	Terzo Sprint Planning . . . . .	6
2.0.4	Quarto Sprint Planning . . . . .	6
2.0.5	Quinto Sprint Planning . . . . .	6
<b>3</b>	<b>Specifica dei Requisiti</b>	<b>7</b>
3.1	Descrizione testuale dei requisiti . . . . .	7
3.2	Descrizione generale . . . . .	8
3.3	Descrizione degli attori . . . . .	8
3.4	Obiettivi di alto livello e problemi delle parti di interesse	9
3.5	Regole di business . . . . .	9
<b>4</b>	<b>Requisiti</b>	<b>10</b>

4.1	Identificazione degli attori . . . . .	10
4.2	Contesto del sistema . . . . .	10
4.3	Casi d'uso . . . . .	11
4.4	Descrizione dettagliata dei casi d'uso . . . . .	12
4.5	Diagramma di Sequenza . . . . .	14
4.6	Requisiti non funzionali . . . . .	15
<b>5</b>	<b>Architettura Software</b>	<b>17</b>
5.1	Stile architetturale scelto . . . . .	17
5.2	Pattern architetturale scelto . . . . .	18
5.3	Architettura Software . . . . .	19
5.4	Diagramma dei componenti . . . . .	20
5.5	Diagramma delle classi . . . . .	21
<b>6</b>	<b>Diagrammi di progetto</b>	<b>23</b>
6.1	Sequence Diagram Richiesta dati realtime . . . . .	24
6.2	Sequence Diagram Richiesta dati locali . . . . .	25
6.3	Sequence Diagram Parser . . . . .	25
6.4	Activity Diagram . . . . .	25
<b>7</b>	<b>Implementazione e testing</b>	<b>28</b>
7.1	Integrated Development Environment (IDE) . . . . .	28
7.2	Linguaggio di programmazione e framework python Flask	29
7.3	Building . . . . .	29
7.4	Application realizzate . . . . .	30
7.4.1	Parser . . . . .	30
7.4.2	Server . . . . .	31

7.4.3 Client . . . . .	32
7.5 Testing . . . . .	36
<b>8 Stima costi di sviluppo</b>	<b>40</b>
<b>9 Sviluppi futuri</b>	<b>43</b>
<b>10 Conclusioni</b>	<b>44</b>

# Capitolo 1

## Introduzione

### 1.1 Presentazione del progetto

Il progetto prevede la realizzazione di una web application che ha lo scopo di far visualizzare, in formato tabellare, una serie di dati che vengono raccolti da un dispositivo GPS. Questi dati vengono raccolti all'interno di un file UBX, che deve essere opportunamente analizzato affinché siano visibili. Il sistema che verrà realizzato, dunque, prevederà prima un parsing di questo file per l'analisi dei dati di interesse. In questo modo l'utente potrà interagire con il sistema per la visualizzazione e la memorizzazione di questi dati.

### 1.2 Organizzazione del lavoro

Per la realizzazione del prodotto abbiamo adottato le metodologie agile con un approccio di tipo iterativo ed evolutivo. Scegliendo questa metodologia, di tipo agile, la documentazione e la modellazione del sistema 'e stata raffinata durante l'intero processo di sviluppo. Tuttavia, in questo documento abbiamo riportato solo le versioni già rifinite. In particolare, il processo è stato diviso in tre fasi:

- Inception: si stabiliscono gli obiettivi generali del progetto, si progetta l'architettura del software e si inizia la stesura del product backlog individuando le principali funzionalità del sistema e le loro priorità;
- Sprint: fase iterativa di durata 2 settimane al cui termine si ha a disposizione un prodotto incompleto ma funzionante;

- Sprint planning: all'inizio di ogni sprint, sono state individuate le funzionalità a maggior priorità dal product backlog ed inserite nello sprint backlog.
- Sprint review: alla fine di ogni sprint, è stato esaminato il lavoro svolto nelle settimane precedenti e rimodellato il backlog in funzione dei risultati e delle nuove conoscenze ottenute.
- Release: ultima fase in cui è stato revisionato il progetto, completata la documentazione e i test e generato i file eseguibili per il rilascio al cliente.

Inoltre, nel sistema, il requisito funzionale prioritario è sicuramente quello di visualizzare i dati, in quanto rappresenta la funzionalità principale per l'utente, ma delinea anche la struttura completa dell'architettura che andrà ad avere il sistema.

### 1.2.1 Tecnologie, strumenti e linguaggi adottati

Il team è composto da tre membri. Per gestire la divisione dei task, per la gestione delle riunioni e per la gestione dello sviluppo dell'applicativo, sono state utilizzate le seguenti tecnologie:



- Microsoft Teams: per effettuare i meeting;



- Github: applicazione utilizzata per la gestione dei file relativi al progetto e la loro condivisione;



- Overleaf: per la stesura della documentazione in maniera condivisa.



- Discord: Per meeting del gruppo giornalieri e condivisione delle risorse.

Gli strumenti adottati per la progettazione e lo sviluppo software sono qui di seguito elencati.



**Visual Paradigm**

- Visual Paradigm: è uno strumento di supporto di UML 2, SysML e Business Process Modeling Notation (BPMN) dal Object Management Group (OMG). Oltre al supporto per la modellazione, fornisce funzionalità di generazione di report e di ingegnerizzazione del codice, inclusa la generazione di codice. Può decodificare i diagrammi dal codice e fornire ingegneria di andata e ritorno per vari linguaggi di programmazione;



- Visual Studio Code: è un editor di codice sorgente sviluppato da Microsoft per Windows, Linux e macOS. Include il supporto per debugging, un controllo per Git integrato, Syntax highlighting, IntelliSense, Snippet e refactoring del codice. Sono personalizzabili il tema dell'editor, le scorciatoie da tastiera e le preferenze. È un software libero e gratuito, anche se la versione ufficiale è sotto una licenza proprietaria;



**mongoDB**

- MongoDB: è un DBMS non relazionale, orientato ai documenti. Classificato come un database di tipo NoSQL, MongoDB si allontana dalla struttura tradizionale basata su tabelle dei database relazionali in favore di documenti in stile JSON con schema dinamico, rendendo l'integrazione di dati di alcuni tipi di applicazioni più facile e veloce.

I linguaggi adottati per lo sviluppo dell'intero sistema software sono qui di seguito riportati:



- HTML: linguaggio adottato per lo sviluppo delle interfacce del sistema;



- Python: linguaggio adottato per alcune funzionalità del sistema, quali la comunicazione tra i diversi nodi e l'avvio.



## Capitolo 2

# Sprint Planning

Durante il processo di sviluppo sono stati effettuati x workshop. Per ognuno di essi è stato perfezionato il risultato relativo al task di un workshop precedente ed, di conseguenza, ‘è stata aggiornata la documentazione.

### 2.0.1 Primo Sprint Planning

Nel primo sprint planning ‘e stato valutato quanto richiesto, ricavando i principali requisiti funzionali e non di alto livello che il sistema deve rispettare. Sono stati delineati gli strumenti e le tecnologie da utilizzare e tracciato il processo di sviluppo. Gli obiettivi di questo Sprint:

- Esplorazione dei requisiti tramite modellazione dei seguenti diagrammi:
  - Diagramma dei casi d’uso;
  - prima bozza diagramma architetturale;
  - Diagrammi di sequenza di sistema;
- Prima bozza del parser per la lettura dei dati.

### 2.0.2 Secondo Sprint Planning

Nel secondo sprint planning, gli obiettivi sono stati i seguenti:

- Prima bozza del diagramma delle classi di progetto;
- Perfezionamento del parser;

- Implementazione del Server;
- Modellazione del diagramma di sequenza dettagliato per la funzionalità implementata.

### **2.0.3 Terzo Sprint Planning**

Nel terzo sprint planning, gli obiettivi sono stati i seguenti:

- Implementazione del Client;
- Modellazione dei diagrammi di sequenza dettagliati per le funzionalità implementate;
- Progettazione dei test per le funzionalità e le interfacce implementate;
- Completamento diagramma architetturale;
- Inizio stesura documentazione.

### **2.0.4 Quarto Sprint Planning**

Nel quarto sprint planning, gli obiettivi sono stati i seguenti:

- Completamento dei diagrammi rimanenti;
- Completamento dei test;
- Generazione degli eseguibili.

### **2.0.5 Quinto Sprint Planning**

Nel quinto e ultimo sprint planning, gli obiettivi sono stati i seguenti:

- Completamento stesura documentazione;
- Release su GitHub.

## Capitolo 3

# Specifica dei Requisiti

### 3.1 Descrizione testuale dei requisiti

Si vuole realizzare un sistema per la visualizzazione e memorizzazione di dati coordinate che vengono raccolti tramite un dispositivo GPS montato su un'autovettura. In particolare:

- Il dispositivo raccoglie i dati in un file in formato ubx;
- Il file ubx viene analizzato tramite un parser;
- I dati vengono raccolti all'interno di file JSON;
- L'utente attiva la web application per poter selezionare e visualizzare la tabella con i dati di interesse.

Il sistema dunque viene utilizzato da due figure professionali, ovvero l'utente interessato a visualizzare e memorizzare i dati di interesse e l'amministratore, che ha il compito di avviare il server e il parser. Inoltre l'utente può visualizzare i dati locali, contenenti una visualizzazione su mappa dell'ultimo dato memorizzato. Bisogna andare ad introdurre, quindi, anche un meccanismo di selezione della tabella di interesse (Tabella dati in tempo reale e tabella dati locali). Il sistema deve essere facilmente accessibile, usabile, sicuro e deve essere sempre disponibile.

## 3.2 Descrizione generale

Il sistema si inserisce in un contesto accademico, dove si ha a disposizione un dispositivo GPS montato su un autovettura che raccoglie dati in un file formato ubx che verrà successivamente analizzato dal parser, attivato da un amministratore. L'utente è interessato ad utilizzare il sistema per poter visualizzare i dati di interesse e memorizzarli.

## 3.3 Descrizione degli attori

- Utente: figura professionale capace di utilizzare adeguatamente un'applicazione web. Ha interesse ad utilizzare il prodotto durante il suo consueto orario di lavoro mediante un personal computer. Usufruisce del prodotto dopo la raccolta dei dati da parte del dispositivo GPS e l'avvio del Server, con l'obiettivo di andare a visualizzare e memorizzare i dati di interesse;
- Amministratore: figura professionale capace di utilizzare adeguatamente un'applicazione web. Ha interesse ad utilizzare il prodotto durante il suo consueto orario di lavoro mediante un personal computer. Ha il compito di avviare il Server, e quindi il Parser, per l'analisi del file ubx.

### 3.4 Obiettivi di alto livello e problemi delle parti di interesse

Obiettivi	User Story	Priorità	Complessità	Note
Parsing dei dati	<b>Come:</b> Amministratore <b>Voglio:</b> Fare un parsing del file ubx <b>Perché:</b> Per poter rendere visualizzabili i dati di interesse	Alta	Alta	
Visualizzazione dati in tempo reale	<b>Come:</b> Utente <b>Voglio:</b> Visualizzare i dati in tempo reale <b>Perché:</b> Per poter leggere e memorizzare i dati di interesse	Alta	Alta	
Visualizzazione dati locali	<b>Come:</b> Utente <b>Voglio:</b> Visualizzare i dati locali <b>Perché:</b> Per poter visualizzare e rendere visibile a schermo le ultime coordinate memorizzate	Media	Media	Per questo è necessario che i dati siano stati memorizzati prima su un database locale

Figure 3.1: Tabella Obiettivi

### 3.5 Regole di business

Le regole di business applicate sono le seguenti:

- Regola 1: Solo l'amministratore può attivare il server per il parsing dei dati. Tale regola non è modificabile
- Regola 2: Solo l'utente può attivare il client per visualizzare i dati in tempo reale e locali. Tale regola non è modificabile

# Capitolo 4

## Requisiti

In questa sezione andiamo a descrivere i requisiti funzionali e non identificati in fase di analisi.

### 4.1 Identificazione degli attori

Durante lo sviluppo del sistema sono stati individuati i seguenti attori primari:

- Utente
- Amministratore

### 4.2 Contesto del sistema

Gli attori attraverso un personal computer interagiscono con il sistema. In particolare però, l'amministratore ha il semplice compito di avviare il Server con il parser, per cui interagisce direttamente con il sistema. L'utente, invece, lo fa attraverso un browser che consente di effettuare richieste HTTP per richiedere le interfacce con le quali gli attori possono visualizzare i dati offerti del sistema.

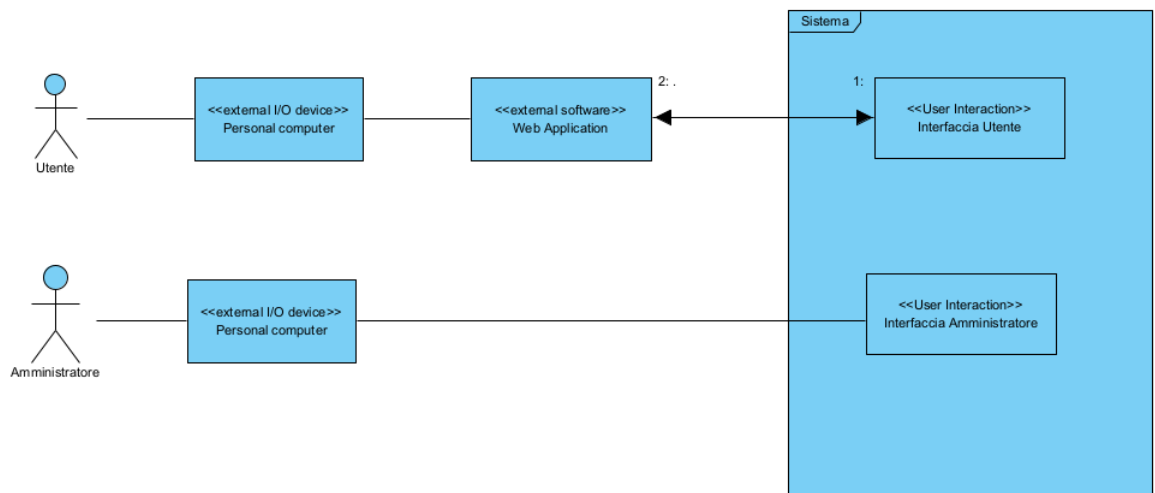


Figure 4.1: Diagramma di contesto

## 4.3 Casi d'uso

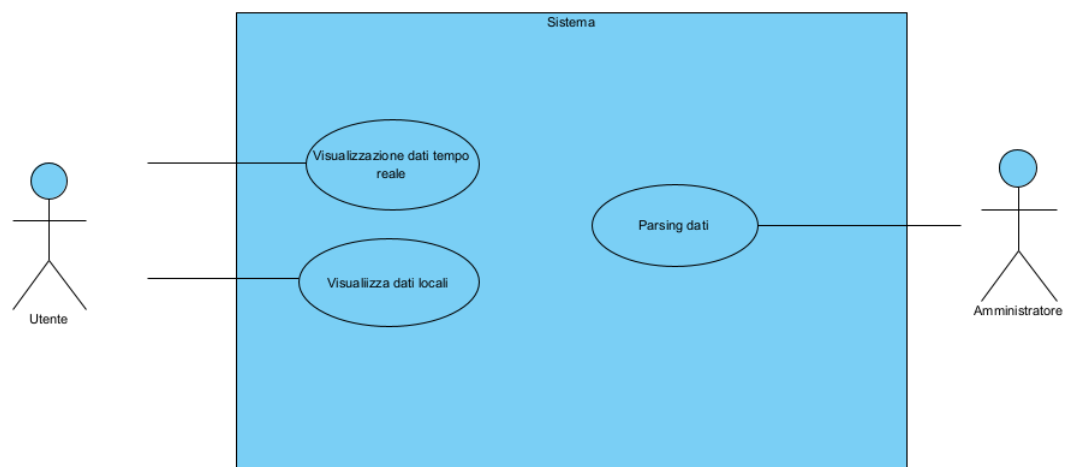


Figure 4.2: Diagramma casi d'uso

Nel diagramma soprastante vengono riportati i casi d'uso del sistema per una facile lettura. I casi d'uso dunque sono:

- Parsing dati
- Visualizzazione dati tempo reale

- Visualizzazione dati locali

## 4.4 Descrizione dettagliata dei casi d'uso

Caso d'uso: Parsing dati
ID: 1
Breve descrizione: L'utente è interessato a fare il parsing dei dati contenuti all'interno del file ubx
Attori Primari: Amministratore
Attori Secondari: Nessuno
Precondizioni: L'amministratore deve avere necessariamente il file ubx
Sequenza degli eventi principali: <ol style="list-style-type: none"> <li>1. L'amministratore attiva il Server</li> <li>2. L'amministratore attiva il parser</li> </ol>
Postcondizione: L'amministratore ha a disposizione i dati in tempo reale
Sequenze degli eventi alternative: Nessuna

Figure 4.3: Tabella caso d'uso Parsing



Caso d'uso: Visualizza dati in tempo reale
ID:2
Breve Descrizione: L'utente è interessato a visualizzare i dati in tempo reale
Attori Primari: L'utente
Attori Secondari: Nessuno
Precondizioni: Il server deve essere attivo con l'esposizione dei dati
Sequenza degli eventi principale: <ol style="list-style-type: none"> <li>1. L'utente apre la homepage del sito web</li> <li>2. L'utente selezione "Dati in tempo reale"</li> </ol>
Postcondizioni: L'utente visualizza e memorizza i dati localmente
Sequenza degli eventi alternative: Nessuna

Figure 4.4: Tabella caso d'uso Visualizza dati tempo reale

Caso d'uso: Visualizza dati locali
ID:3
Breve Descrizione: L'utente è interessato a visualizzare i dati locali
Attori Primari: L'utente
Attori Secondari: Nessuno
Precondizioni: I dati devono essere memorizzati in locale
Sequenza degli eventi principale: <ol style="list-style-type: none"> <li>1. L'utente apre la homepage del sito web</li> <li>2. L'utente selezione "Dati locali"</li> </ol>
Postcondizioni: L'utente visualizza i dati di interesse
Sequenza degli eventi alternative: Nessuna

Figure 4.5: Tabella caso d'uso Visualizza dati locali

## 4.5 Diagramma di Sequenza

I diagrammi di sequenza sono utili per visualizzare le interazioni tra un attore e il sistema. Essi mettono in evidenza una specifica sequenza di azioni in cui tutte le scelte sono già state effettuate. Di seguito sono riportati i diagrammi di sequenza di analisi per:

- Parsing dati
- Visualizzazione dati tempo reale
- Visualizzazione dati locali



Figure 4.6: Diagramma di sequenza Parsing dati

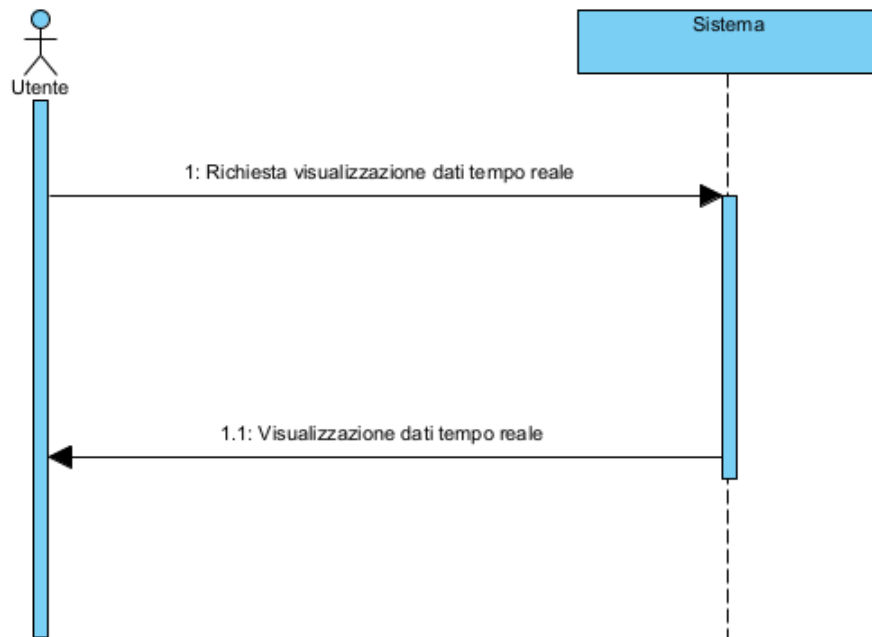


Figure 4.7: Diagramma di sequenza visualizzazione dati tempo reale

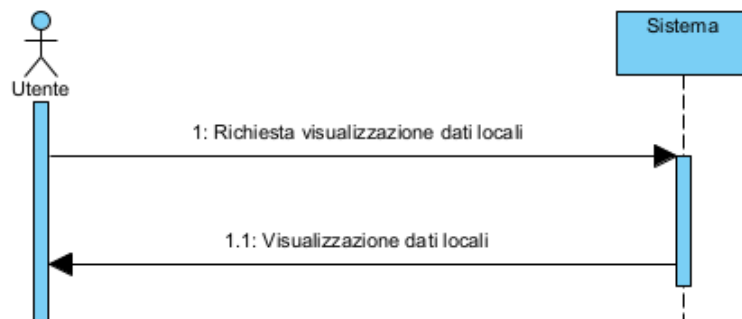


Figure 4.8: Diagramma di sequenza Visualizzazione dati locali

## 4.6 Requisiti non funzionali

Tra i requisiti non funzionali, ovvero quelli che riguardano gli aspetti secondari del sistema, possiamo individuare:

- Usabilità: L'utente deve poter visualizzare le schermate dell'applicativo in maniera chiara, facile e veloce. Le informazioni hanno un'interfaccia user friendly, in quanto deve risultare semplice inserire i dati e le diagnosi per evitare errori;

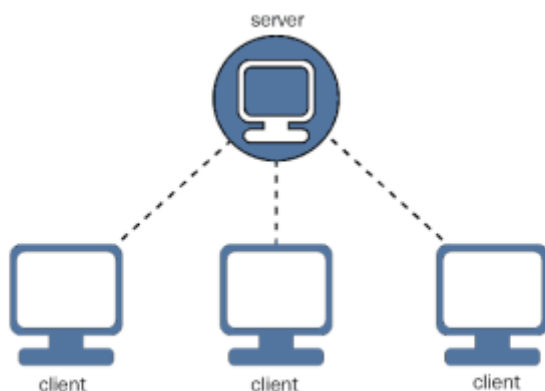
- Manutenibilità: Il software realizzato potrà essere facilmente modificato e migliorato, includendo correzioni miglioramenti e nuove funzionalità.
- Evolvibilità: Il software deve consentire l'aggiunta di nuove feature per espanderne le funzionalità ed adattarsi a diversi domini di utilizzo.

## Capitolo 5

# Architettura Software

In un sistema software è fondamentale avere una visione di come sarà l'architettura di quest'ultimo. Un'architettura software è definita dai suoi componenti, dalle relazioni tra essi e dai principi che ne governano la progettazione e l'evoluzione. Infatti, per descrivere l'architettura software di un sistema, bisogna elencarne le sotto parti costituenti ed illustrarne i rapporti inter funzionali.

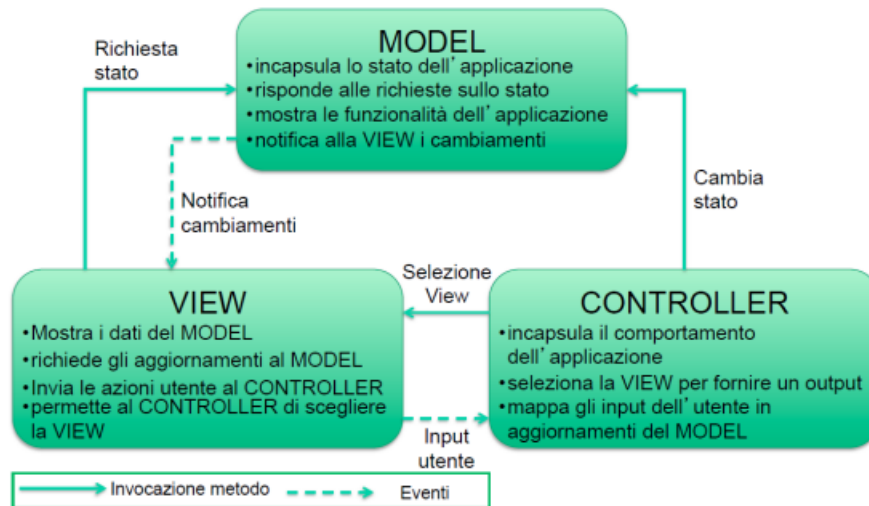
### 5.1 Stile architetturale scelto



Ad un alto livello di dettaglio l'architettura del sistema segue uno stile Client-Server distribuito, in cui sia il client che il server sono in esecuzione su nodi differenti. Nel dettaglio si ha un “thin” client che richiede servizi al “fat” server.

## 5.2 Pattern architetturale scelto

Per la realizzazione del client è stato scelto il pattern architetturale Model-View-Controller. La scelta è ricaduta su di esso vista la necessità di mantenere separata la logica di visualizzazione dalle logiche di business e di accesso ai dati.



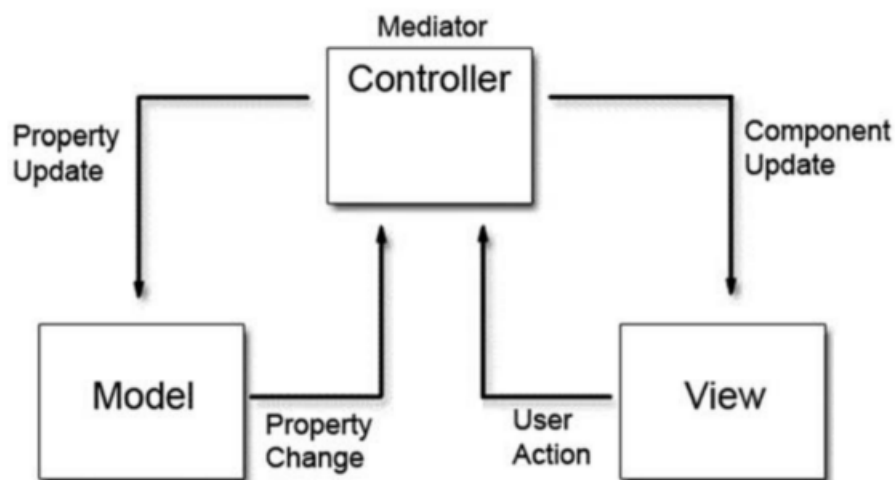
Dalla figura soprastante, ricordiamo i principali ruoli di questi componenti:

- Il Model gestisce i dati e le regole di business per poter interagire con essi. Fornisce alla view ed al controller le funzionalità per l'accesso e l'aggiornamento;
- La View gestisce la logica di presentazione dei dati e permette di implementare diverse viste su di essi;
- Il Controller trasforma gli input utente della view in azioni da eseguire sul model e seleziona le schermate da mostrare tramite la view.

A questa tipologia di MVC detta di tipo push, si affianca una seconda tipologia di tipo pull tipicamente utilizzata nelle soluzioni distribuite. In questa, sequenza di azioni tra i tre componenti è leggermente differente. Infatti:

- La view riceve un input utente e lo invia al controller;

- Il controller seleziona l'azione corrispondente all'input, che dovrà essere eseguita sul model;
- Il model esegue l'azione, si aggiorna e restituisce al controller il nuovo stato;
- Il controller è a sua volta delegato di restituire alla view il nuovo stato del model.



## 5.3 Architettura Software

L'architettura software dell'applicativo GeoGeo usa lo stile architetturale Client-Server e il pattern architetturale MVC per la progettazione del client. Di seguito possiamo vedere il "Diagramma Architetturale":

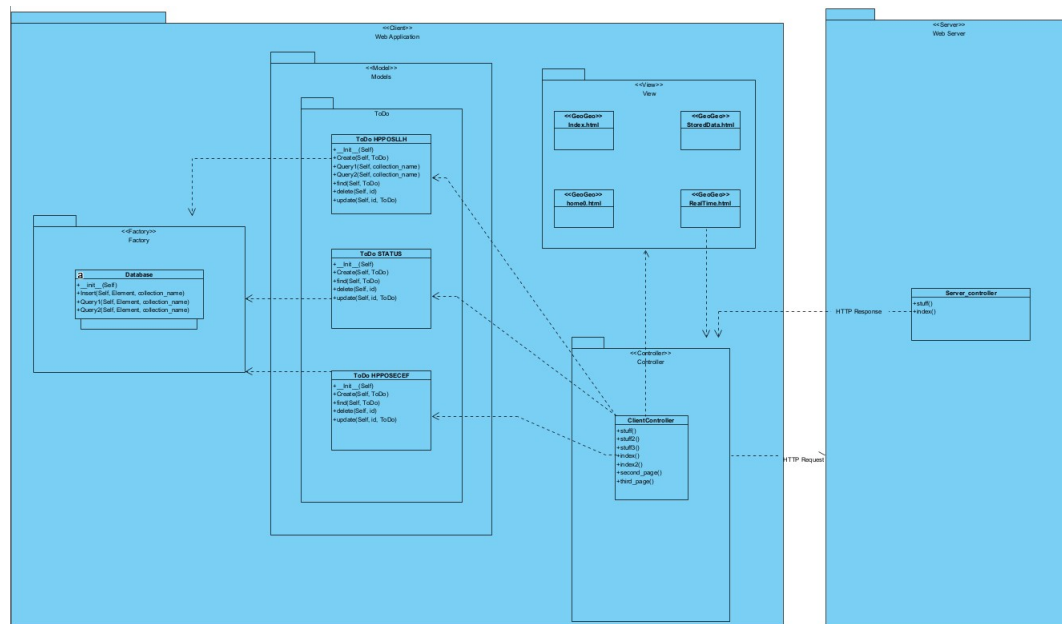


Figure 5.1: Diagramma Architetture

Si noti l'utilizzo del pattern MVC, che ci permette di richiedere dati dal server per la (tramite il controller) per la creazione e costruzione della vista delle tabelle presenti nelle pagine "RealTime.html" e "Stored-Data" della web application "GeoGeo".

## 5.4 Diagramma dei componenti

I diagrammi dei componenti UML rappresentano le relazioni tra singoli componenti di sistema in una vista di progettazione statica. I componenti sono parti modulari di un sistema, che sono indipendenti e possono essere sostituiti da componenti equivalenti. Sono chiusi in sé e incapsulano tutte le strutture complesse desiderate. Per gli elementi incapsulati sono possibili contatti con altri componenti solo tramite interfaccia. Possono mettere a disposizione le proprie interfacce, ma anche ricorrere a interfacce di altri componenti, per avere accesso ad esempio alle loro funzioni o servizi. Allo stesso tempo, in un diagramma dei componenti, le interfacce documentano le relazioni e interdipendenze di un'architettura software. Di seguito riportiamo il diagramma delle componenti:



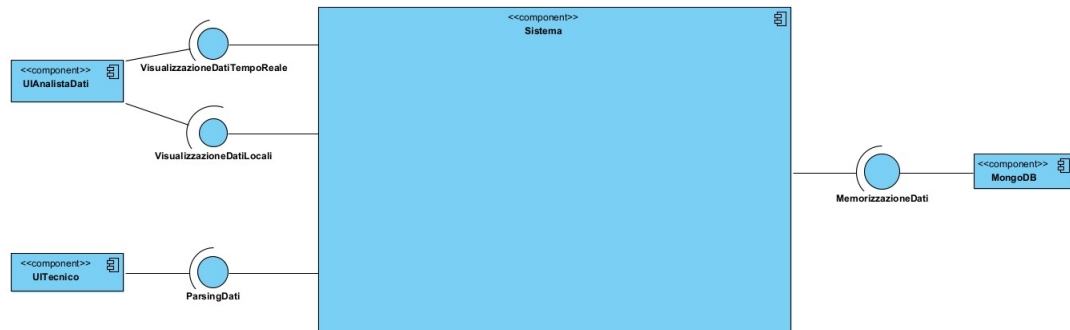


Figure 5.2: Diagramma dei Componenti

## 5.5 Diagramma delle classi

Il Diagramma delle classi descrive il tipo degli oggetti che compongono il sistema e le relazioni statiche esistenti tra loro. Come si può vedere, nel nostro caso l'utente interagisce con le classi "pagina DatiRT" costruita come composizione delle tre classi "Tab NAV STATUS", "Tab NAV HPPOSLLH" e "Tab NAV HAPPOSECEF", mentre la "pagina DatiL" è composizione delle classi "Mappa DatiL" e "Tab DatiL". Entrambe le classi pagine sono costruite dalla web application "GeoGeo" che richiede i dati ad un "server", che a sua volta riceve di dati da un parser, entrambi inizializzati da un "Amministratore".

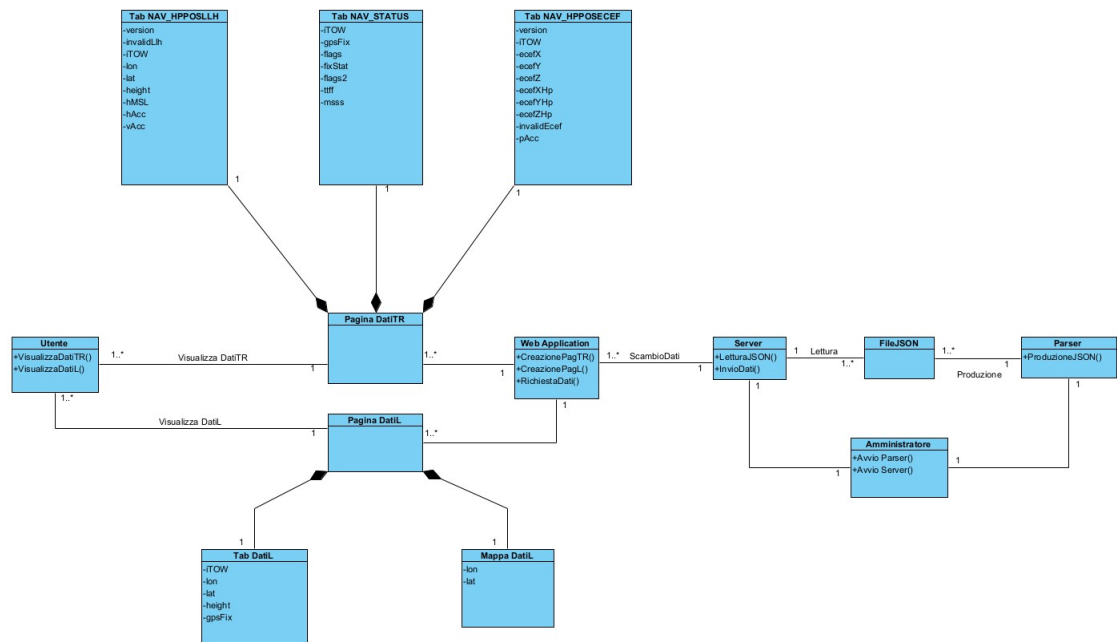


Figure 5.3: Diagramma delle Classi

## Capitolo 6

# Diagrammi di progetto

Questo paragrafo è dedicato ai diagrammi di sequenza raffinati, che includono scelte progettuali più dettagliate, e che descrivono in dettaglio la logica di business. Nel nostro caso sono tre ed i primi due definiscono le interazioni utente client all'atto di richiesta di visualizzazione dei dati, in tempo reale o locali che siano, mentre l'ultimo sequence ci permette di avere una visione su come avviene il parsing e quindi la creazione del file .JSON i cui dati verranno poi organizzati in tabelle ed in una mappa nella quale si aggiorna in tempo reale la posizione del sensore che trasmette i dati. Si può accedere a tali tabelle e alla mappa tramite il sito "GeoGeo" creato appositamente.

## 6.1 Sequence Diagram Richiesta dati real-time

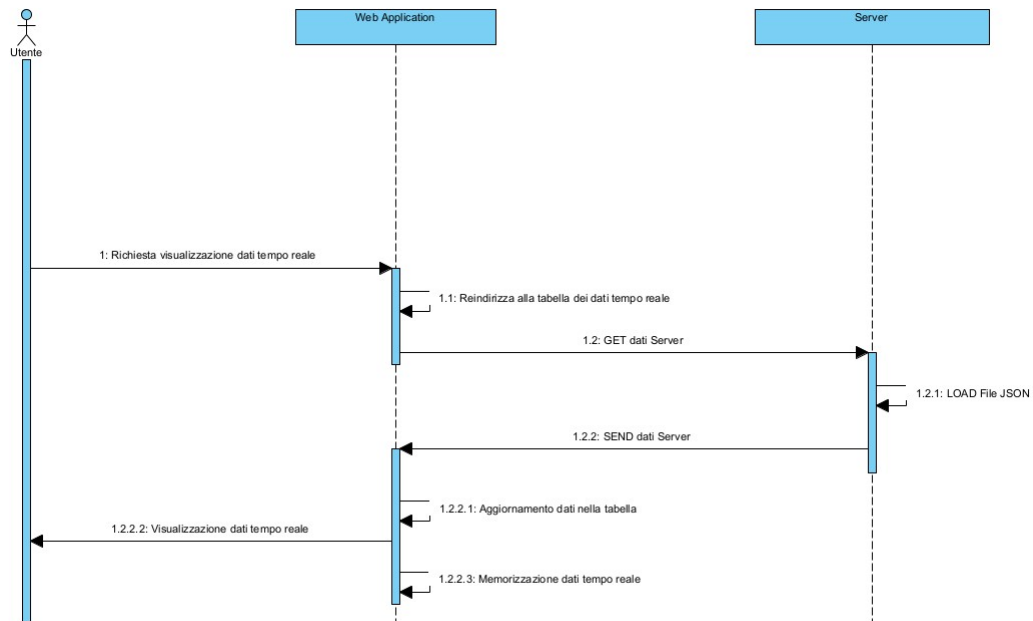


Figure 6.1: Sequence Diagram: Richiesta dati realtime

## 6.2 Sequence Diagram Richiesta dati locali

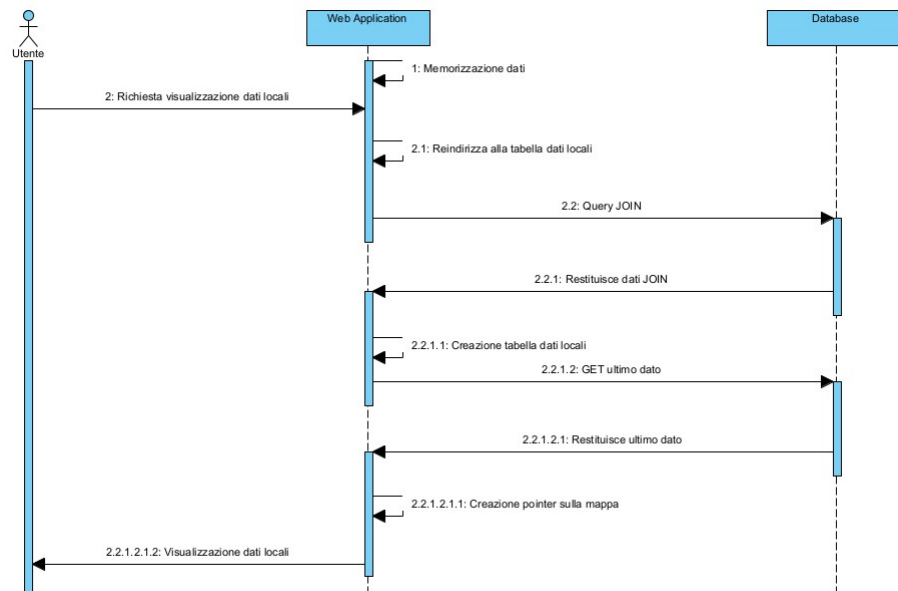


Figure 6.2: Sequence Diagram: Richiesta dati locali

## 6.3 Sequence Diagram Parser

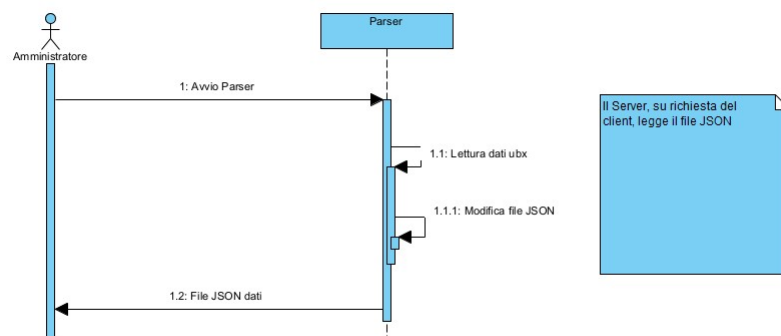


Figure 6.3: Sequence Diagram: Parser

## 6.4 Activity Diagram

Vediamo di seguito gli activity diagram. Essi aiutano a visualizzare i processi che fanno parte dell'applicativo ad un livello più dettagliato.

Mettono in evidenza una serie di azioni che rappresentano le attività del processo. Hanno un nodo iniziale ed un nodo finale per descrivere il punto di partenza e il punto di arrivo, inoltre sono presenti le indicazioni di come evolve il flusso del sistema.

Nel primo diagramma possiamo vedere come avviene la richiesta tramite il client di visualizzazione di dati in tempo reale e di come il server, attraverso le funzioni di ricezione, prelievo e invio, mandi come risposta al client un file json.

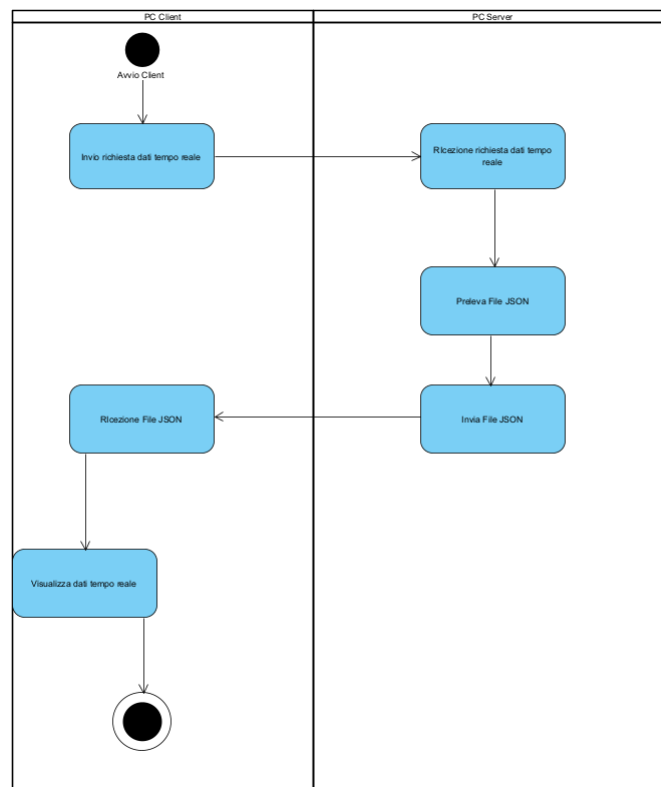


Figure 6.4: Activity Diagram: Richiesta dati tempo reale

Nel secondo diagramma possiamo vedere come avviene l'avvio del server. Tale operazione è composta da due fasi che sono l'avvio del server e l'avvio del parsing.

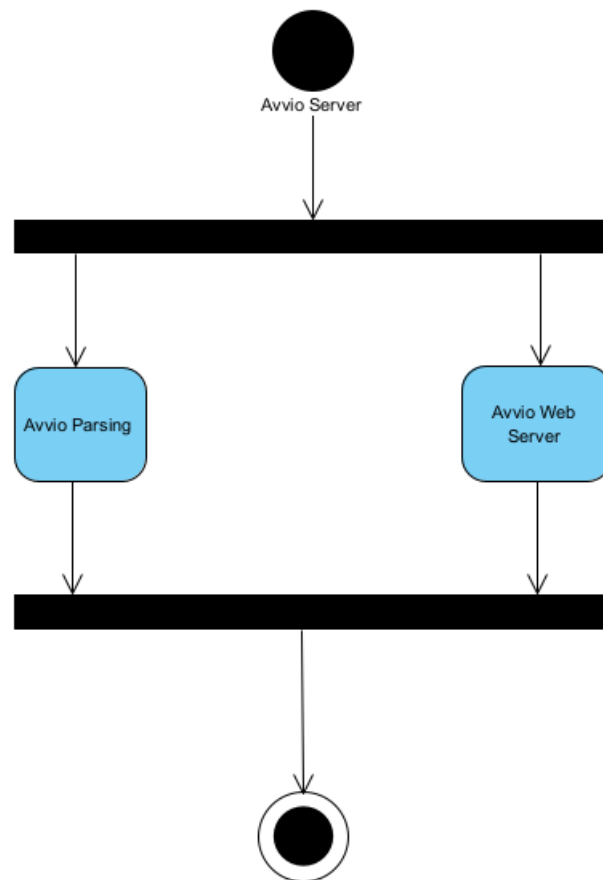


Figure 6.5: Activity Diagram: Avvio server

## Capitolo 7

# Implementazione e testing

Spostiamoci dalla fase di progettazione a quella di implementazione riferendoci agli aspetti fondamentali che hanno caratterizzato la scrittura del codice.

### 7.1 Integrated Development Environment (IDE)

Per supportare l'intera sviluppo abbiamo utilizzato l'IDE Visual Studio Code. Questa scelta è stata fatta poichè tale software offre le seguenti caratteristiche:

- code completion e code navigation
- supporto a più linguaggi di programmazione, nel nostro caso python
- facile integrazione con il framework python Flask
- Integrazione con Github

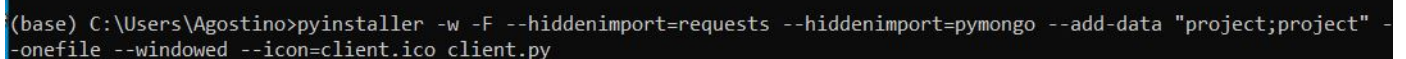


## 7.2 Linguaggio di programmazione e framework python Flask

Come linguaggio di programmazione per lo sviluppo delle applicazioni si è scelto python. In primo luogo la scelta è stata effettuata poichè avevamo bisogno di una libreria che ci permettesse di leggere i file UBX provenienti dal dispositivo che effettua le misurazioni GPS. Successivamente, ricercando altri strumenti adatti allo sviluppo di applicazioni web, abbiamo deciso di utilizzare il framework python Flask. Esso è definito come un Flask è un web framework open source scritto in Python caratterizzato da flessibilità, leggerezza e semplicità d'uso. Fa parte della categoria dei micro-framework per via del suo approccio allo sviluppo minimalista. Questa tecnologia ci ha permesso di sviluppare in tempi rapidi sia il sever che il client.

## 7.3 Building

Per il building dell'applicazione ci siamo serviti della libreria pyinstaller che in ambiente windows, l'ambiente designato dal nostro committente, permette di ottenere un eseguibile in maniera abbastanza rapida. Uno degli esempi di codice utilizzato per fare il building è nella figura sottostante.



```
(base) C:\Users\Agostino>pyinstaller -w -F --hiddenimport=requests --hiddenimport=pymongo --add-data "project;project" -onefile --windowed --icon=client.ico client.py
```

Figure 7.1: Utilizzo di pyinstaller per la build del server

Volendo scendere un po' più nel dettaglio si sono utilizzate anche le librerie python quali:

- requests, per la ricezione delle informazioni dal server
- pyubx2, per la lettura del file UBX prodotto dal sensore collegato al GPS

- pymongo, per lo scambio di dati con il database noSQL mongoDB
- pywebview, per la visualizzazione dell'applicazione in finestra

Si noti che alcune di esse sono esplicitamente specificate nel comando di build a causa di alcuni bug della libreria pyinstaller. In assenza di essi sarebbe stato superfluo includerli.

## 7.4 Application realizzate

Lo sviluppo del nostro sistema è dislocato su più entità fisiche per le quali abbiamo sviluppato chiaramente applicazioni differenti. In particolare per la macchina che riceve le informazioni dal GPS e poi le manda al client su richiesta abbiamo sviluppato due applicativi differenti. Tale scelta permette anche di separare i ruoli e in futuro usare un dispositivo dedicato per il parsing dei dati ed un altro per esporre il servizio. Vediamo nei paragrafi seguenti le tre applicazioni.

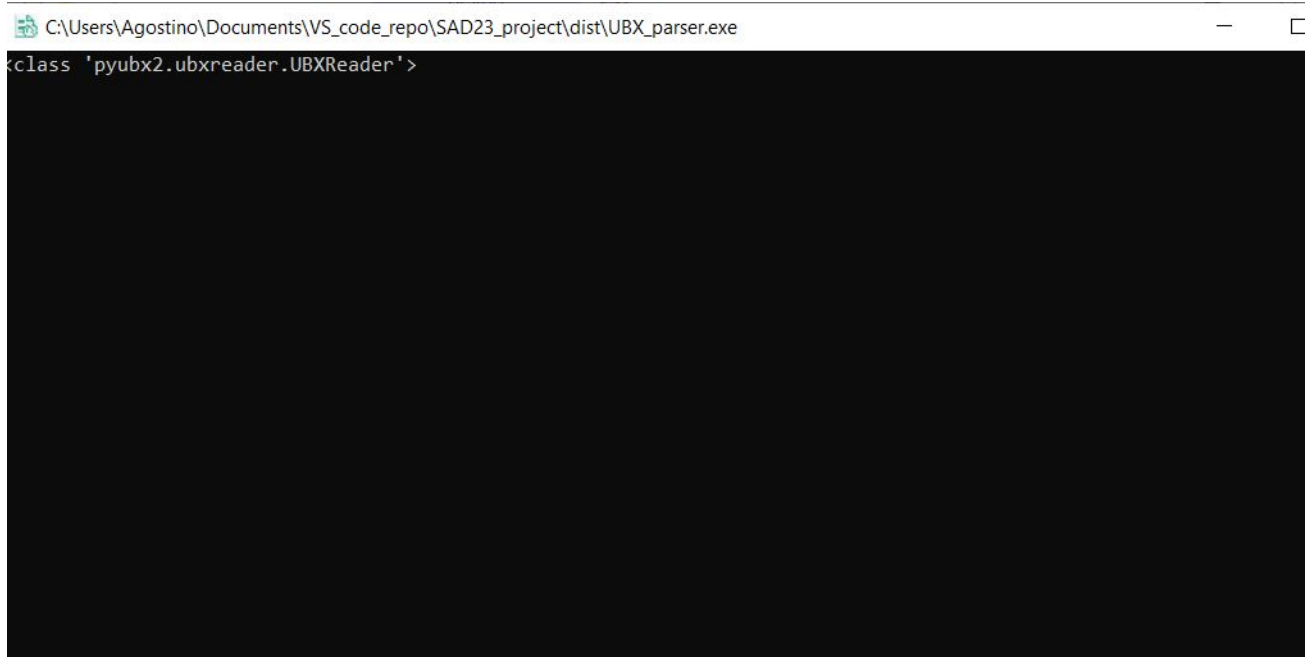
### 7.4.1 Parser

Il parser è stato il primo obiettivo del nostro committente e lo sviluppo dello stesso ha richiesto capire il funzionamento della libreria pyubx2 che ci ha permesso di leggere i suddetti file. Il parser essendo molto vicino ad essere un sistema da installare su dispositivi IOT è stato realizzato con un solo script. Tale script si occupa di aprire il file UBX ed estrarne tre tipi di messaggi che ci erano stati richiesti dal committente. Per ogni messaggio estratto abbiamo fatto riferimento alla documentazione per comprendere quali campi contenesse e per stamparli in maniera corretta. In dettaglio i messaggi a cui eravamo interessati sono:

- NAV-HPPOSECEF, contenente coordinate spaziali
- NAV-HPPOSLLH, contenente latitudine, longitudine, altezza ecc.
- NAV-STATUS, contenente dati sulla validità della posizione delle altre informazioni ricevute

Si noti che NAV-HPPOSECEF non sarà presente negli output in quanto non sono stati ritrovati questi tipi di messaggi nei file UBX che ci sono stati dati come esempio.

Per quanto riguarda l'output del parsing è stato realizzato con dei file json che si prestano ad essere mandati via web. Dai colloqui con il committente è emerso che la memorizzazione sul server dei dati prodotti dal parser non era necessaria a lungo termine, anzi avrebbe finito per saturare la memoria locale dello stesso, Per tale specifica abbiamo deciso di mantenere il numero di entry fissato per il file json e cancellare l'ultima riga non appena se ne aggiungeva un'altra. Riassumendo il file json ha solo 10 entry, mai di più. Nella figura sottostante possiamo vedere l'avvio del prompt del parser.



```
C:\Users\Agostino\Documents\VS_code_repo\SAD23_project\dist\UBX_parser.exe
class 'pyubx2.ubxreader.UBXReader'>
```

Figure 7.2: Parser in esecuzione

### 7.4.2 Server

Il server è stato realizzato con il framework python Flask. Risulta essere molto semplice nella sua struttura poiché ha come unici compiti la visualizzazione di una schermata di conferma di avvio del servizio e inoltre fornisce il metodo get invocabile dal client. Quando viene richiamato questo metodo la server application preleva il file json e lo manda al richiedente senza fare altro, Si tratta di un semplice fornitore

di servizio al quale non sono richiesti particolari oneri. Ecco un esempio del server in esecuzione.



Figure 7.3: Server in esecuzione

### 7.4.3 Client

Il client è di sicuro l'entità più complessa del progetto al quale sono date maggiori responsabilità. Da progetto il client è realizzato tramite il pattern MVC. Il client ha bisogno infatti di ricevere informazioni apposite dal server per i dati in tempo reale e allo stesso tempo memorizzarli in locale per elaborazioni future. Noi ne abbiamo mostrati alcuni esempi nella pagina `StoredData` che effettua delle query con visualizzazione dei dati (mappa e tabella). La prima pagina del client si pone come un semplice benvenuto all'utente e la possiamo vedere in figura 7.4

**Benvenuti sul sito per la visualizzazione dei dati!**

Figure 7.4: Pagina di benvenuto del client

Dalla pagina iniziale è possibile, attraverso il menu apposito in alto, scegliere in quale tab spostarsi come se fosse un sito web. La prima opzione è dati in tempo reale e ci dà la possibilità di fare una richiesta al server ogni 350 millisecondi per ricevere i dati attuali prodotti dal parser. L'intervallo è indicativo ed in sviluppi successivi si potrebbe decidere come settarlo in funzione del dominio di interesse.

## NAV\_HPPOSLLH

version	invalidLlh	ITOW	lon	lat	height	hMSL	hAcc	vAcc
0	0	302069000	14.191137749	40.828832127	84241.3	42966	108.7	205.5
0	0	302068000	14.191137725	40.828832223	84308.6	43033.3	110.2	207.7
0	0	302067000	14.191137739	40.828832313	84389.6	43114.3	112.3	210.3
0	0	302066000	14.191137766	40.82883243	84470.4	43195	114	212.4
0	0	302065000	14.191137803	40.828832541	84547.5	43272.1	115.3	214.1
0	0	302064000	14.191137832	40.828832658	84614.5	43339.1	116.8	216.1
0	0	302063000	14.191137876	40.828832776	84669.7	43394.4	118.5	218.6
0	0	302062000	14.191137901	40.828832952	84728.9	43453.6	120.6	221

## NAV\_STATUS

ITOW	gpsFix	flags	fixStat	flags2	ttff	msss
302069000	3	1111	110	011	30076	300576

Figure 7.5: Pagina per la visualizzazione e la memorizzazione dei dati real time

Come si potrà vedere facendo eseguire il software la tabella si aggiornerà ad intervalli regolari. Qualora non vi siano dati verranno visualizzati dei punti interrogativi. In questa pagina sono presenti tre tabelle di cui abbiamo mostrato uno scorcio solo delle prime due, in realtà la terza rimarrà vuota in quanto, come detto in precedenza, quel tipo di messaggio non ci è stato fornito dal committente. Il codice per il suo funzionamento è comunque stato scritto ma per ovvie ragioni non testato.

L'altra pagina accessibile dal menu è relativa ai dati locali e la possiamo vedere in figura 7.6. Questa pagina presenta due differenti viste che altro non sono che la visualizzazione del risultato di due query sul database.

## Valori gpsFix per istanti di tempo

iTOW	lon	lat	height	gpsFix
302069000	14.191137749	40.828832127	84241.3	3
302053000	14.191138652	40.828837698	84456.3	3
302052000	14.191138974	40.828838746	84281.3	3
302051000	14.191139341	40.828840059	84057.9	3
302049000	14.19114024	40.82884339	83268.9	?
302046000	14.191135469	40.828859939	84278.8	3
302044000	14.191135908	40.828860413	84091.9	3
302042000	14.19113609	40.82886088	83913.9	?
302040000	14.191136398	40.828861202	83772.3	?
302037000	14.191136756	40.828861497	83715.6	3

## Ultima posizione registrata

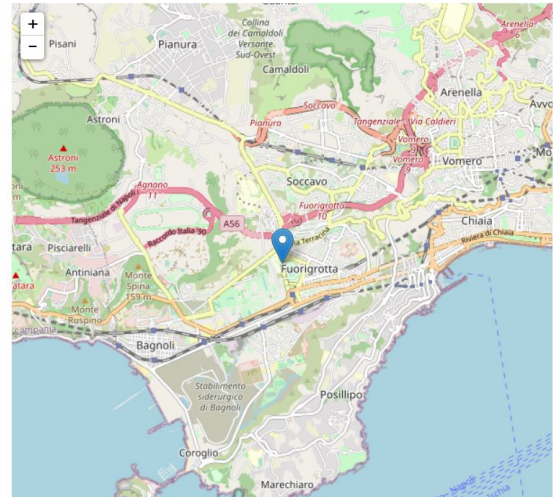
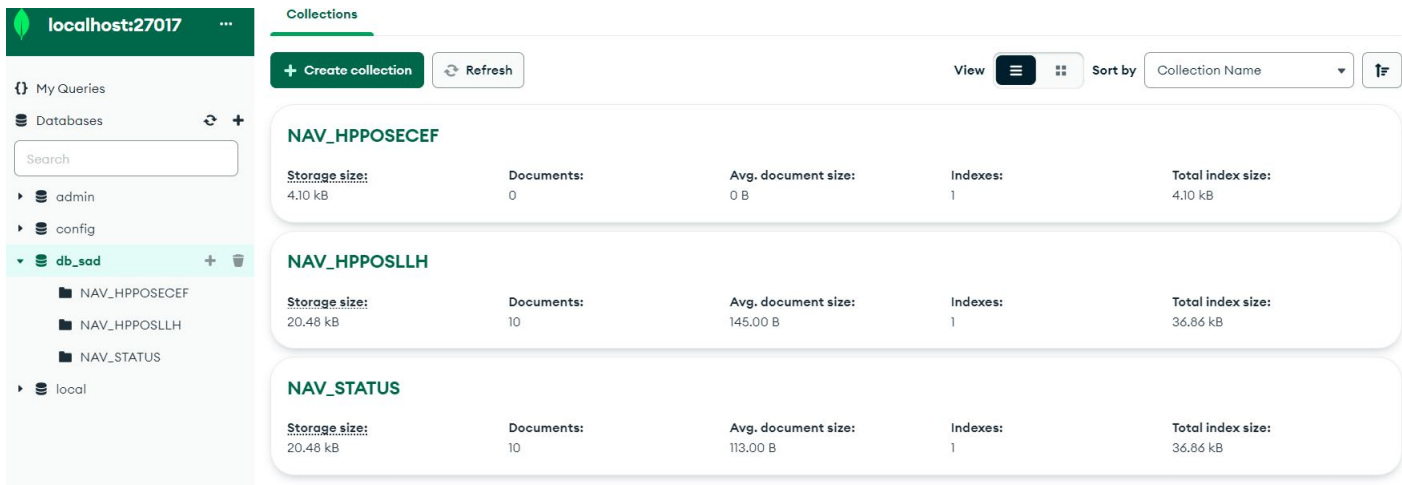


Figure 7.6: Pagina per la visualizzazione dei dati locali

Parlando di database, noi abbiamo optato per un database noSQL poiché più flessibile alla struttura dei dati che possono arrivare, non necessita di primary key ed è adatto alla memorizzazione di grandi quantità di dati strutturati. Nello specifico è un database noSQL di tipo documentale. MongoDB ci ha permesso di creare tre collection diverse, una per ogni tipo di messaggio. Va osservato che quando arrivano i dato l'inserimento è fatto solo se il dato non esiste già. Tale operazione può essere facilmente velocizzata inserendo un indice sul campo iTOW che non è altro che il timestamp.



Collection Name	Storage size	Documents	Avg. document size	Indexes	Total index size
NAV_HPPOSECEF	4.10 kB	0	0 B	1	4.10 kB
NAV_HPPOSLLH	20.48 kB	10	145.00 B	1	36.86 kB
NAV_STATUS	20.48 kB	10	113.00 B	1	36.86 kB

Figure 7.7: Database noSQL mongoDB

Ritornando alla figura 7.6, quindi alla pagina dei dati locali, possiamo vedere che nella tabella sulla sinistra è stata effettuata un'estrazione di alcune caratteristiche provenienti da due tipi di messaggi diversi ovvero STATUS e HPPOSLLH. Il join delle due collection è stato fatto attraverso il campo di timestamp iTOW e, come si può notare dai risultati, è un left join poiché alcune entry non presentano il campo gpsfix.

La seconda modalità di visualizzazione è stata fatta attraverso la query che identifica il valore più recente del campo iTOW e va a selezionare la latitudine e la longitudine. Questi due valori sono poi utilizzati per creare un point di riferimento sulla mappa. La mappa utilizzata è fornita dal servizio OpenStreetMap che essendo open source fornisce grande facilità di integrazione all'interno di pagine html.

## 7.5 Testing

Per quanto riguarda il testing noi ci siamo concentrati su due tipologie. In primo luogo abbiamo i test funzionali svolti in maniera manuale che hanno portato alla costruzione della tabella 7.10. Essi sono dei test fatti per comprendere se i principali funzionalità facessero ciò per il quale erano state pensate.



Test n°	Azione	Risultati attesi	Risultati effettivi	Passed
1	Avvio client	Visualizzazione pagine iniziale client	Visualizzazione pagine iniziale client	SI
2	Pagina Dati in tempo reale	visualizzazione dei dati provenienti dal server, l'aggiornamento della tabella avviene in automatico	La tabella si aggiorna in automatico come previsto	SI
3	Apertura pagina dati locali passando prima per la pagina dati in tempo reale	viene visualizzata la tabella con i dati richiesti e la mappa	viene visualizzata la tabella con i dati richiesti e la mappa	SI
4	Apertura pagina dati locali senza passare prima per la pagina dati in tempo reale	viene comunque visualizzata una mappa vuota	la mappa non viene visualizzata	NO
5	Avvio server	Visualizzazione pagine iniziale server	Visualizzazione pagine iniziale server	SI
6	Più client accedono insieme alla pagina dei dati in tempo reale	La pagina dati in tempo reale viene visualizzata correttamente	entrambi gli utenti visualizzano i dati	SI
7	Il parser manda i dati molto più velocemente	i dati in tempo reale vengono visualizzati tutti	La frequenza di aggiornamento bassa fa perdere qualche dato	NO
8	Avvio parser	Creazione e aggiornamento continuo dei file json	I file json vengono creati ed aggiornati	SI
9	Stop parser	Fine invio dati senza errore	Gli ultimi dati sono correttamente visualizzati in tabella	SI
10	Avvio client senza avvio del server	Il client visualizza i dati tutti come punti interrogativi	Il client visualizza i dati tutti come punti interrogativi (dato sconosciuto)	SI
11	Il server invia un dato del timestamp molto alto per errore	La mappa visualizzerà il dato di timestamp corretto, ultima posizione valida	La mappa visualizza solo il timestamp più grande e segnerà sempre la stessa posizione	NO
12	il parser finisce i dati perché non gli arrivano più	L'applicazione parser non si chiude e rimane in attesa	il parser si chiude e deve essere riavviato	NO

Figure 7.8: Test iniziali

Possiamo notare che alcuni di essi sono più specifici e sono stati fatti negli sprint più avanzati dello sviluppo come nel caso di visualizzazione della mappa. Successivamente abbiamo creato anche dei test automatici con l'ausilio del framework pytest. Essi sono per lo più incentrati a chiarire se i messaggi che arrivano al client sono corretti nella loro forma.

```
def test_stuff():
    response = app.test_client().get('/_stuff')
    lista = response.data.decode('utf-8').split(sep='], ', maxsplit=-1)
    num_type_mes = len(lista)
    assert num_type_mes == 3

def test_stuff2():
    response = app.test_client().get('/_stuff2')
    lista = response.data.decode('utf-8').split(sep='}', ', ', maxsplit=-1)
    search_string = lista[0]
    all_words=['gpsFix', 'height', 'iTOW', 'lat', 'lon']
    found = True
    for word in all_words:
        if not word in search_string.split(''):
            found = False
    assert found

def test_stuff3():
    response = app.test_client().get('/_stuff3')
    lista = response.data.decode('utf-8').split(sep='}', ', ', maxsplit=-1)
    search_string = lista[0]
    all_words=['iTOW', 'lat', 'lon']
    found = True
    for word in all_words:
        if not word in search_string.split(''):
            found = False
    assert found
```

Figure 7.9: Utilizzo di pytest

Scendendo nel dettaglio i test sono tre. Il primo si occupa di verificare se le informazioni che arrivano dal server hanno al loro interno tre messaggi diversi. In questo caso il test fallirà se il dato che arriva ha un numero diverso da tre. Il secondo ed il terzo test si concentrano con l'interazione con la base di dati locale. Essi hanno lo scopo di verificare se il risultato delle query fatte sul database noSQL mongoDB restituiscono oggetti i cui campi sono corrispondenti ai campi previsti che vorremmo visualizzare in tabella e in mappa.

```
(env_sad2) C:\Users\Agostino\Desktop\client_SAD>pytest -v unit_test.py
===== test session starts =====
platform win32 -- Python 3.11.4, pytest-7.3.1, pluggy-1.0.0 -- C:\Users\Agostino\anaconda3\envs\env_sad2\python.exe
cachedir: .pytest_cache
rootdir: C:\Users\Agostino\Desktop\client_SAD
collected 4 items

unit_test.py::test_index PASSED [ 25%]
unit_test.py::test_stuff PASSED [ 50%]
unit_test.py::test_stuff2 PASSED [ 75%]
unit_test.py::test_stuff3 PASSED [100%]

===== warnings summary =====
```

Figure 7.10: Risultati pytest

## Capitolo 8

# Stima costi di sviluppo

Per stimare i costi di sviluppo, può essere necessario conoscere la produttività, ovvero la velocità con cui si producono parti del software e la relativa documentazione. Le misure di produttività richiedono la misura di qualche attributo del software, pertanto si dividono in size-related measures, basate su qualche output del processo, e function-related measures, basate su una stima della funzionalità del software rilasciato. Alla prima categoria appartiene la misurazione delle LOC, alla seconda categoria la misurazione dei Function Points. In particolare, per linguaggi di vecchia generazione, una linea di codice (line of code, LOC) corrispondeva ad un'istruzione; per linguaggi moderni il discorso si complica perché un'istruzione può corrispondere a più linee di codice o una linea può contenere più istruzioni, e perché il riuso dei componenti impone di conteggiare solo alcuni moduli del software. Gli FP invece sono indipendenti dal linguaggio di programmazione adottato e, inoltre, hanno il vantaggio di essere basati su dati che hanno la maggior probabilità di essere noti all'inizio di un progetto, nonché quello di fornire ottime indicazioni per applicazioni di elaborazione dati, che usano linguaggi convenzionali. Tuttavia non hanno un diretto significato fisico, e l'assegnazione dei pesi è soggettiva. Esiste, però, un terzo metodo che ricorre ai casi d'uso. Gli Use Case Points (UCP) sono basati sull'intuizione che più sono complessi i casi d'uso di un sistema, più tempo sarà necessario per lo sviluppo del codice. Quest'ultima tecnica è stata utilizzata per calcolare i costi di sviluppo, attraverso la seguente formula:

$$UCP = UUCP \cdot TFC \cdot EF$$

Dobbiamo quindi andare a definire i singoli fattori. L'UUCP (Unadjusted Use Case Point) 'e dato dalla somma degli UUCW (Unadjusted Use Case Weight, ovvero la somma dei prodotti di ciascun caso d'uso per il peso ad esso assegnato) e UAW (Unadjusted Actor Weight, ovvero la somma dei prodotti di ciascun attore per il rispettivo peso), valutati attraverso gli scenari che descriveremo più avanti e calcolati attraverso le tabelle:

Tipo caso d'uso	Peso	Numero caso d'uso	Prodotto
Semplice	5	0	0
Medio	10	1	10
Complesso	15	2	30
Totale		3	40

Figure 8.1: Tabella UUCW

Tipologia attore	Peso	Numero attori	Prodotto
Semplice	1	2	2
Medio	2	0	0
Complesso	3	0	0
Totale			2

Figure 8.2: Tabella UAW

La valutazione dell'UUCP, dunque, risulta essere 42. Successivamente si è passati a calcolare i due successivi fattori. Definito "impatto" il prodotto della valutazione del fattore per il peso ad esso associato, si calcola il Technical Complexity Factor (TCF). Un altro aggiustamento da fare riguarda la complessità dell'ambiente, detto Environment Factor (EF). Le formule per ottenere questi due fattori sono le seguenti:

$$TCF = 0.6 + (0.1 \cdot TotalFactor)$$

$$EF = 1.4 + (-0.03 \cdot TotalFactor)$$

Dove i Total Factor sono calcolati attraverso le seguenti tabelle, in cui l'impatto della complessità tecnica è valutato attraverso diversi fattori, cui viene dato un punteggio da 0 (irrilevante) a 5 (molto importante).:

Fattore	Descrizione	Peso	Valutazione	Prodotto
T1	Sistema distribuito	2	1	2
T2	Performance	1	4	4
T3	Efficienza per l'utente finale	1	5	5
T4	Complessità di elaborazione interna	1	4	4
T5	Riusabilità del codice	1	5	5
T6	Facile da installare	0.5	5	2.5
T7	Facile da usare	0.5	5	2.5
T8	Portabile	2	3	6
T9	Facile da cambiare	1	3	3
T10	Elaborazione parallela	1	0	0
T11	Caratteristiche di sicurezza	1	1	1
T12	Accesso diretto da terzi	1	0	0
T13	Formazione per utenti	1	4	4
Totale				39

Figure 8.3: Tabella TCF

Fattore	Descrizione	Peso	Valutazione	Prodotto
E1	Familiarità con il processo di sviluppo	1.5	4	4.5
E2	Esperienza di applicazione	0.5	0	0
E3	Esperienza object-oriented del team	1	0	0
E4	Capacità di analisi	0.5	5	2
E5	Motivazione del team	1	5	5
E6	Stabilità dei requisiti	2	4	8
E7	Personale part-time	-1	0	0
E8	Complessità del linguaggio utilizzato	-1	2	-4
Totale				15.5

Figure 8.4: Tabella EF

Dunque, sostituendo i fattori ricavati nella formula, otteniamo un UCP pari a:

$$UCP = UUCP \cdot TFC \cdot EF = 38.8$$

Infine, supponendo un rapporto di 25 ore per Use Case Point si ottiene un quantitativo di ore pari a:

$$ORE = 38.8 \cdot 25 = 970$$

## Capitolo 9

# Sviluppi futuri

In conclusione diamo una panoramica dei possibili sviluppi futuri del progetto appuntando alcune idee che possano espanderne le funzionalità e migliorare alcuni punti critici.

- lato parser, collegamento ad uno stream di dati continuo
- aggiunta di più viste sui dati locali e maggiori possibilità di interazione con il database locale
- modifica degli intervalli di aggiornamento per la ricezione dei dati dal server con possibilità di scelta da parte dell'utente
- aggiornamento continuo delle posizioni sulla mappa per tracciare il percorso effettuato
- integrazione con servizi di mappe più diffusi come google maps

## Capitolo 10

# Conclusioni

Il progetto si pone come un tassello per la risoluzione di un problema specifico, la visualizzazione e gestione dei dati provenienti da un sensore di precisione. Esso può avere molteplici utilizzi che spaziano dall'agricoltura di precisione al monitoraggio sismico del terreno. Le sfide future, come già scritto nella sezione precedente, saranno focalizzate sul ridurre la latenza e sull'implementazione di nuove feature sia funzionali che non funzionali.