Dipartimenti di Ingegneria Elettrica
e delle Tecnologie dell'Informazione
Universita' degli Studi di Napoli "Federico II"

Anno Accademico 2018/2019
Secondo Semestre

Sistemi Multimediali
IoT Monitoring Museum

Prof.	Piccialli Francesco

N46003287	d'Andrea Fabio
N46003511	Daniele Dario
N46003206	Di Chiara Guido
N46003359	Fordellone Domenico

Indice

Capitolo 1

Introduzione

Il seguente elaborato mira a presentare un progetto di Sistemi Multimediali, intitolato **Iot Monitoring Museum**, il cui obiettivo è stato lo sviluppo di un'applicazione desktop utile per la visualizzazione di alcune statistiche relative ai visitatori del *Museo Archeologico Nazionale di Napoli (MANN)*.
È possibile suddividere il processo di sviluppo del progetto in due fasi principali:

- **Sviluppo dei modelli 3D del museo**
- **Sviluppo delle applicazioni**

Tali fasi possono essere suddivise a loro volta in ulteriori step, ampiamente descritti nel documento.

1.1 Specifiche

È stato richiesto di sviluppare l'applicativo utilizzando il software **Unity 3D** e di permettere all'utente di:

- Caricare un insieme di dati relativi ai percorsi effettuati dai visitatori. Il dataset, assunto come materiale fornito, è un file di testo caratterizzato dalla seguente struttura:

  LABELS,ore:minuti:secondi,giorno/mese/anno ora:minuti

  dove:

  – Il primo campo rappresenta il percorso di un visitatore, espresso come sequenza di *label*.

Ciascuna label è associata ad una centralina bluetooth. Tali centraline sono state disseminate all'interno del museo al fine di rilevare, in maniera non invasiva, il passaggio dei visitatori dotati di dispositivi bluetooth attivi.

  - Il secondo campo rappresenta la durata della permanenza della visita.
  - Il terzo campo rappresenta la data di ingresso nel museo.

- Visualizzare i percorsi effettuati dai visitatori secondo diversi livelli di aggregazione, in un ambiente 2D o 3D che rispetti fedelmente la planimetria del museo.

1.2 Scelte progettuali

Sulla base delle specifiche richieste, si è scelto di permettere all'utente di caricare più dataset, intesi come file testuali differenti, e di visualizzare i percorsi in un ambiente 3D offrendo la possibilità di:

- Visualizzare i percorsi in tre modalità diferenti:
  - **Modalità statica:** ogni percorso è mostrato come un insieme di archi che collegano ogni centralina alla successiva;
  - **Modalità animata:** ogni percorso è mostrato dinamicamente attraverso il movimento di una sfera;
  - **Modalità ibrida:** ogni percorso è mostrato come un insieme di linee tracciate nel corso del tempo;

  In tutti i casi, ad ogni percorso è associato un colore differente.
  Tuttavia, si è data la possibilità all'utente di inserire un dataset contenente un campo numerico aggiuntivo. Tale campo potrebbe rappresentare la **classe** di un visitatore, assegnata da un eventuale esperto di dominio. In queste condizioni, ad ogni classe di visitatori è associato un colore differente.
  Si è scelto poi di aggregare i visitatori caratterizzati dallo stesso percorso, associando loro uno spessore delle linee maggiore, nelle modalità statica e ibrida, ed una dimensione della sfera maggiore, nella modalità dinamica.

- Visualizzare i percorsi relativi ad un intervallo temporale compreso tra due date, con una granularità spinta fino all'ora di ingresso.

- Visualizzare i percorsi che includono una o più centraline.
- Visualizzare i percorsi dei visitatori il cui tempo di permanenza del museo è compreso tra un limite inferiore ed un limite superiore, con una granularità spinta fino al minuto.

È ovviamente possibile combinare i diversi filtri di visualizzazione.
Oltre all'**applicazione Front-end** dedicata alla visualizzazione dei dati, fruibile dal direttore del museo, è stata sviluppata un'**applicazione Back-end** che consente ad eventuali tecnici di gestire le centraline presenti nel museo. In particolare le funzionalità implementate permettono di:

- Aggiungere una nuova centralina;
- Modificare la posizione di una centralina esistente;
- Rimuovere una centralina;

1.3 Strumenti utilizzati

Si presentano di seguito gli strumenti utilizzati per la realizzazione del progetto.

1.3.1 Unity 3D

Unity 3D è un motore grafico cross-plattform sviluppato da Unity Technologies che permette lo sviluppo di applicazioni 3D o 2D interattive.

Un **Motore Grafico 3D** (3D Game Engine) è un sistema software utilizzato per la creazione e lo sviluppo di applicazioni 3D, in quanto permette la rappresentazione di un ambiente virtuale 3D su uno schermo 2D.
Un Game Engine fornisce agli sviluppatori un framework completo di strumenti grafici per la realizzazione del proprio applicativo. Le funzionalità di base tipicamente includono un motore di rendering (renderer) per grafica 2D e 3D, un motore fisico (rilevatore di collisioni), suono, scripting, animazioni, intelligenza artificiale, networking e gestione della memoria.
In definitiva, i vantaggi di utilizzare un 3D Game Engine sono: maggior numero di funzionalità disponibili, minor tempo impiegato nello sviluppo e nel testing e, di conseguenza, maggior tempo da poter dedicare al progetto. Tuttavia alcuni svantaggi risultano essere: non avere il pieno controllo sull'implementazione delle funzionalità utilizzate, un elevato carico di elaborazione nel caso in cui venissero aggiunte nuove funzionalità, non poter

utilizzare librerie esterne.
Unity 3D è uno dei più diffusi Game Engine. Un'applicazione creata mediante Unity 3D è definita come un insieme di *assets*, ciascuno dei quali rappresenta un elemento dell'applicativo. L'editor di Unity è cosituito da un'insieme di finestre, tra cui:

- **Inspector Window:** permette di impostare le proprietà degli asset.
- **Scene Window:** permette di progettare le singole scene dell'applicazione, intese come insieme di oggetti 3D e/o 2D.
- **Game Window**: permette di visualizzare un'anteprima della scena in esecuzione.
- **Hierarchy Window:** permette la gestione dei diversi oggetti presenti scena corrente.
- **Project Window:** permette la gestione delle directory componenti il progetto.

1.3.2 Sketchup

SketchUp è un software di modellazione 3D versatile, potente e nel contempo semplice da imparare e da utilizzare.

La **modellazione 3D** è il processo atto a definire una forma tridimensionale in uno spazio virtuale. I modelli 3D rappresentano un oggetto fisico attraverso una collezione di punti nello spazio 3D.

SketchUp permette la creazione di forme bidimensionali e tridimensionali, fornendo al disegnatore uno strumento intuitivo e veloce, in grado di assisterlo dal punto di vista grafico e di consentirgli un'esplorazione dinamica e creativa degli oggetti.
Durante la fase di modellazione il disegnatore può contare su strumenti particolarmente efficaci come il **push/pull**, che permette di realizzare figure tridimensionali attraverso l'estrusione di forme bidimensionali.

Capitolo 2

Sviluppo dei modelli 3D

La prima fase di sviluppo è stata fondamentale al fine di porre le basi per la successiva implementazione delle applicazioni in Unity 3D. In particolare, sulla base delle planimetrie messe a disposizione dal MANN stesso, si è creato per ogni piano del museo un modello 3D, utilizzando Sketchup.
Il tool ha permesso la modellazione di quatto oggetti tridimensionali, i importati successivamente in Unity 3D come *GameObject* distinti, ciascuno dei quali costituito da un certo numero di *Meshes*. Nello specifico, per ogni piano del museo è stato ripetuto il seguente procedimento.

Dalla planimetria al modello 3D

Una volta creato un nuovo progetto in Sketchup, è stata importata l'immagine della planimetria relativa ad un piano. Successivamente, tramite gli appositi *Shape Tools* forniti dal software, è stata ricreata la mappa del piano seguendo come linee guida i confini dell'immagine sottostante.

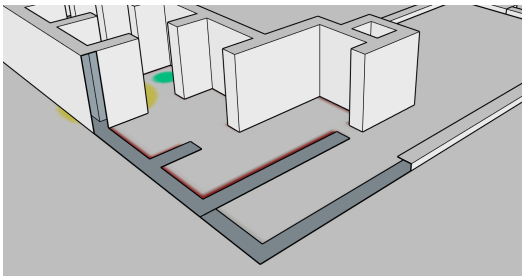

Figura 2.1: Planimetria 2D (zona grigia)

Terminata la planimetria 2D, tramite lo strumento di *estrusione*, si è ottenuto il modello 3D del piano del museo.

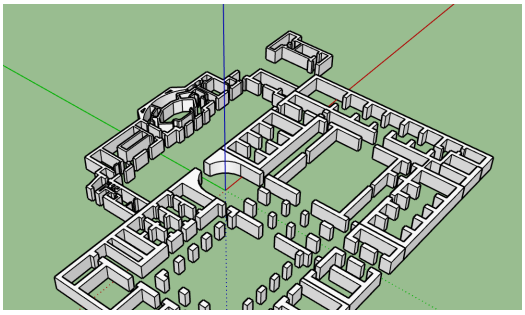

Figura 2.2: Modello 3D

Al fine di evidenziare le diverse aeree del museo, essendo queste distinte per colore, è stata necessaria un'ulteriore azione prima dell'esportazione dell'oggetto 3D. Per ogni piano sono stati definiti i contorni delle sale, in modo da ottenere un oggetto composto da un insieme di *mesh* differenti. In questo modo è stato possibile assegnare in Unity un *Material* diverso a ciascuna area e associare ad ognuna di esse il relativo colore, come indicato nella planimetria fornita.
Terminato il processo, si è proseguito con l'esportazione del modello tridimensionale in formato *.obj*.

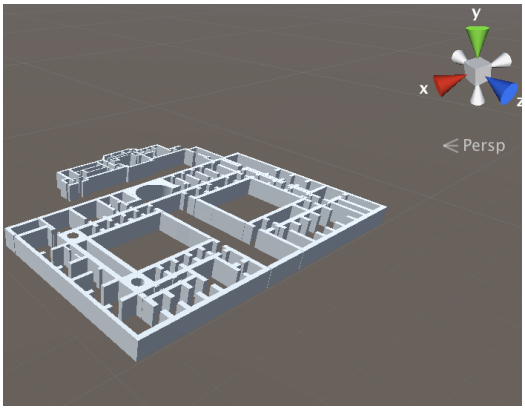

Figura 2.3: Oggetto importato in Unity

Capitolo 3

Sviluppo delle applicazioni

La seconda fase di sviluppo ha previsto la progettazione e l'implementazione di due applicazioni differenti, Front-end e Back-end, utilizzando Unity 3D. Le due applicazioni comunicano indirettamente tra loro tramite un web server FTP disponibile all'indirizzo:

ftp://files.000webhost.com/public_html/UnitsRepository.txt.

In particolare, l'applicazione Back-end carica sul server un file di testo formattato, in cui è riportata la posizione aggiornata di ogni centralina. Viceversa, l'applicazione Front-end scarica dal server il file e aggiorna di conseguenza la posizione ed il numero delle centraline.
Di seguito sono riportati i dettagli di entrambe le applicazioni.

3.1 Applicazione Front-end

L'applicazione Front-end è dedicata alla visualizzazione dei percorsi e prevede come utente finale il direttore stesso del museo.

3.1.1 Piani e Centraline

Dopo aver importato i modelli 3D dei **piani** del museo, sono stati assegnati loro *Material* di colore differente, in modo da ricreare fedelmente la planimetria fornita dalle specifiche.
Tramite l'asset *Text Mesh Pro*, in grado di garantire migliori qualità grafiche, sono state associate a ciascun piano alcune informazioni testuali, tra cui informazioni relative al numero del piano e l'ingresso del museo.
Durante l'esecuzione dell'applicativo sono inoltre mostrate statistiche aggiornate sul numero di visitatori per ciascun piano, calcolate sulla base della

combinazione di filtri selezionati dall'utente.

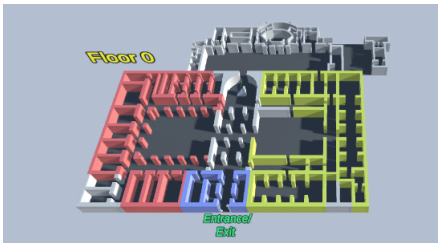

Figura 3.1: Modello 3D del Piano 0

Per rappresentare le **centraline** è stato invece definito un *Prefab*, a partire dall'oggetto primitivo *Capsule*. A tempo di esecuzione, le centraline sono create e disposte in base alle informazioni contenute nel file di setup caricato on-line dall'applicazione Back-end.
Ad ogni centralina è associato il proprio nome e, durante la visualizzazione, il numero di visitatori passati per la determinata centralina. Tali informazioni sono visualizzabili tramite "click" sulla centralina stessa.
Infine, sono stati realizzati dei modelli di **scale** utilizzando il plug-in *Pro Builder*. Su queste sono indicati i piani raggiungibili dai visitatori.

3.1.2 Interfaccia Utente

Per permettere all'utente di interagire con l'applicazione e impostare i vari filtri di visualizzazione è stata sviluppata una apposita interfaccia grafica utente (**GUI, Graphic User Interface**).

Menu principale

Nel *Menu Principale* è stata data la possibilità all'utente di caricare uno o più dataset, in formato *.txt*. Caricati i file, è possibile avviare la visualizzazione dei dati.
Il caricamento dei file è stato realizzato utilizzando l'asset **FileBrowser**, disponibile gratuitamente nell'Assets Store. Tale asset fornisce un Prefab composto da diversi elementi UI e da alcuni script in grado di renderli interattivi. Sono state effettuate alcune modifiche agli script al fine di integrare i componenti necessari nel progetto.
Sono inoltre, presenti le sezioni:

- **Impostazioni:** permette di modificare alcune impostazioni di base, tra cui: qualità video, volume audio , modalità schermo intero ed inoltre si da la possibilità di cambiare i parametri per la connessione FTP.
- **Crediti:** presenta alcune informazioni relative agli sviluppatori del progetto.
- **Selezione lingua:** permette di cambiare la lingua del Menu Principale tra Italiano ed Inglese.

Avviata la visualizzazione, l'utente è quindi portato in una nuova scena contenete i quattro piani del museo.

Figura 3.2: Menu Principale

Menu Utente

All'interno della scena principale è presente un *Menu Utente* attraverso cui è possibile selezionare i vari filtri che rendono possibile l'attività di *monitoring* del museo.
In particolare, mediante tale menu è possibile impostare:

- **Modalità di visualizzazione:** si dà all'utente la possibilità di visualizzare i dati secondo tre modalità differenti:
  - **Modalità statica**: si visualizzano archi corrispondenti ai percorsi dei visitatori;
  - **Modalità animata**: si visualizzano sfere in movimento rappresentanti i visitatori stessi;

  - **Modalità ibrida**: si visualizzano linee ed archi che si compongono nel tempo corrispondenti ai percorsi dei visitatori;

- **Prospettiva di visualizzazione:** si dà all'utente la possibilità di scegliere tra disposizioni dei piano differenti: orizzontale, verticale ed obliqua.
  La prospettiva verticale è disponibile esclusivamente in modalità animata.

- **Filtro per data:** si dà all'utente la possibilità di visualizzare i percorsi: compresi tra due date, a partire da una data in poi e fino ad una data specificata.
  Le date indicate si riferiscono alle date di ingresso dei visitatori all'interno del museo.
  La gestione dell'inserimento delle date da parte dell'utente è stata realizzata utilizzando l'asset **DateTimePicker**. Tale asset fornisce un Prefab composto da diversi selettori e da uno script in grado di renderlo interattivo. Nello specifico, si è scelto di gestire esclusivamente i campi anno, mese, giorno ed ora, eliminando ogni riferimento relativo a minuti e secondi.

- **Filtro per centraline:** si dà all'utente la possibilità di visualizzare i percorsi che includono una o più centraline, specificandone il nome in un apposito campo.

- **Filtro per durata:** si dà all'utente la possibilità di visualizzare i percorsi la cui durata è: compresa tra durata minima e massima, superiore ad una durata minima ed inferiore ad una durata massima.

- **Filtro di campionamento:** si dà all'utente la possibilità di estrarre dal dataset inserito un campione casuale di dimensioni pari al 20% del totale.

- **Generazione di statistiche**: si dà all'utente la possibilità di generare un report statistico generale a partire dai dataset caricati e dai filtri selezionati dall'utente.
  In particolare, le statistiche generate sono le seguenti:

  - Numero di visitatori totali;
  - Lista delle centraline presenti;
  - Numero di visitatori per mese;
  - Numero di visitatori per piano;

– Numero di visitatori per *classe*, se specificata.

Le statistiche relative al numero di visitatori totali ed alla lista delle centraline presenti fanno riferimento all'intero dataset ed esulano dai filtri selezionati dall'utente.

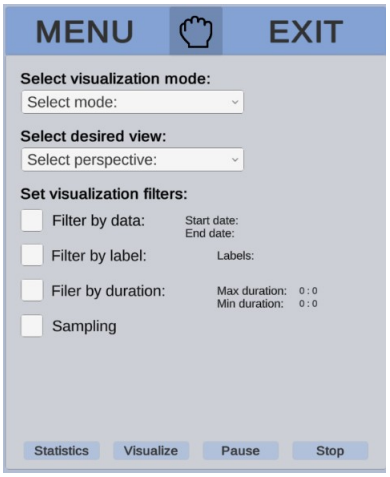

Figura 3.3: Menu Utente

3.1.3 Camera RTS

Per permettere all'utente di esplorare la scena principale si è deciso di importare nel progetto l'asset *RTS Camera* (Real-time strategy Camera), disponibile gratuitamente nell'*Assets Store* di Unity. Dopo aver configurato opportunamente il prefab fornito dall'asset e dopo aver apportato alcune modifiche agli script, è stato reso possibile all'utente di spostarsi all'interno della scena tramite la tastiera e di modificare la propria visuale tramite il mouse.
Per facilitare il corretto posizionamento della camera è stato inserito un apposito bottone *ResetCamera* in grado di ripristinare la visuale di default.

3.1.4 Modalità di visualizzazione

L'applicazione permette di visualizzare i dati forniti in ingresso secondo tre modalità principali: **statica**, **animata** ed **ibrida**.

Modalità statica

La modalità di visualizzazione statica prevede la visualizzazione dei percorsi dei visitatori come un insieme di archi che collegano ogni centralina alla successiva.
In particolare, per la rappresentazione delle linee è stato utilizzato un componente nativo di Unity 3D chiamato **LineRenderer**. Tale componente riceve in ingresso un insieme di vertici, oggetti della classe *Vector3*, ed effettua il tracciamento delle rette congiungenti tali vertici. È inoltre possibile impostare alcuni parametri tra cui colore e spessore delle linee.
Il calcolo dei singoli vertici viene effettuato applicando la funzione matematica nota come **curva di Bezier quadratica**:

$$B(t) = (1-t)^2 P_0 + 2t(1-t)P_1 + t^2 P_2, \quad t \in [0,1] \tag{3.1}$$

Tale funzione permette di tracciare la curva rappresentata in figura (fig.3.4).

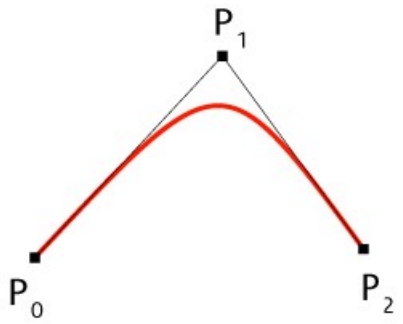

Figura 3.4: Curva di Bezier Quadratica

Modalità animata

La modalità di visualizzazione animata prevede il tracciamento dei percorsi da parte di oggetti sferici, ognuno dei quali rappresenta uno o più visitatori. Tale funzionalità è stata implementata utilizzando le risorse di intelligenza artificiale messe a disposizione dal *package* **UnityEngine.AI** e dall'API *NavMeshComponents*, disponibile gratuitamente nell'*Assets Store*.
Nello specifico, i componenti utilizzati sono:

- **NavMeshSurface:** permette di definire le superfici *navigabili* dagli agenti;
- **NavMeshModifier:** permette di definire le superfici *non navigabili* dagli agenti;

Il primo passo per la realizzazione della modalità animata è stato quindi definire, per ogni piano, le superfici percorribili, rappresentate dal pavimento, e di quelle non percorribili, rappresentate dalle mura.
Il *baking* della **NavMesh**, ovvero la creazione delle superfici navigabili e non viene effettuato a tempo di esecuzione. Tale operazione è rimandata in quanto è prevista la visualizzazione dei piani del museo secondo diverse prospettive.
All'atto della visualizzazione vengono creati i diversi **NavMeshAgent**, agenti intelligenti, ai quali è fornito in ingresso il percorso relativo al proprio visitatore. Ogni agente è in grado di calcolare attraverso algoritmi di *path finding* il tragitto migliore per raggiungere la prossima centralina, evitando le superfici non percorribili e gli altri agenti.

Modalità ibrida

La modalità di visualizzazione ibrida prevede il tracciamento dei percorsi in maniera dinamica, rappresentati come un insieme di linee ed archi che si compongono nel tempo.
Per realizzare tale funzionalità, sono state ancora una volta impiegate risorse di intelligenza artificiale. Preliminarmente sono stati disseminati all'interno dei piani del museo un insieme di **waypoints**.
In questo modo è possibile tracciare, a tempo di esecuzione, il percorso effettuato dagli agenti e rappresentarlo graficamente attraverso il componente LineRenderer. In particolare, ogni qual volta un agente si avvicina ad uno dei suddetti waypoints ne viene rilevata la posizione. Ogni volta che un agente raggiunge la centralina successiva, viene tracciato il percorso congiungente i waypoints rilevati durante il tragitto. Il passaggio di un agente da un piano all'altro è invece indicato dalla presenza di un arco, anche in questo caso tracciato mediante l'equazione della curva quadratica di Bezier.
In questo modo è stato possibile tracciare linee che, rispetto alla modalità statica, rispettano più fedelmente i reali percorsi effettuati dai visitatori. Inoltre, grazie alla presenza dei waypoints e all'impiego dell'intelligenza artificiale, tale funzionalità risulta indipendente dalla specifica configurazione delle centraline del museo.

3.1.5 Scripts

Per implementare la logica dell'applicazione e garantire le funzionalità richieste dalle specifiche sono stati implementati diversi script, ognuno dei quali è stato assegnato ad un oggetto presente in scena.
Di seguito sono descritti i principali script realizzati.

- **GameController:** script chiave della scena principale, coordina le diverse funzionalità offerte dell'applicazione. Tra le diverse funzioni di tale script vale la pena menzionare:

  - **LoadSetup:** funzione dedicata alla comunicazione con l'applicazione Back-end. In particolare, presenta due override:

    1. *Con parametri:* richiamata se è disponibile connessione Internet. Si occupa di scaricare il file di setup contente le posizioni delle centraline.
       Instaura una connessione FTP con un server remoto situato ad una certa URL, specificando determinate credenziali di accesso. Tutti i parametri della connessione possono essere impostati dall'utente, tramite l'apposita sezione nel pannello *Impostazioni* del Menu Principale.
       Durante il download del file di setup ne viene creata una copia locale, in modo da permettere all'utente di usufruire dell'applicazione anche in assenza di connessione.
    2. *Senza parametri:* richiamata se non è disponibile connessione Internet. Effettua il caricamento dell'ultimo file di setup scaricato dal repository, se presente.
       Nel caso in cui non sia presente alcun file locale, l'utente è riportato al menu principale.

Listing 3.1: Estratto di LoadSetup(string p): download dal Repository

```
1  string[] ftpParametrs = File.ReadAllLines(ftpPath)
       ;
2
3  FtpWebRequest request= (FtpWebRequest)WebRequest.
       Create(ftpParametrs[0]);
4  request.Method= WebRequestMethods.Ftp.DownloadFile
       ;
5
6  request.Credentials= new NetworkCredential(
       ftpParametrs[1], ftpParametrs[2]);
7
8  FtpWebResponse response= (FtpWebResponse)request.
       GetResponse();
9
10 Stream responseStream= response.GetResponseStream
       ();
11 StreamReader reader = new StreamReader(
       responseStream);
```

- **LoadDataset:** funzione dedicata al caricamento di uno o più dataset forniti dall'utente.
  Effettua quindi il *preprocessing* dei file testuali, memorizzando le informazioni di interesse in strutture dati utilizzate per le successive elaborazioni.
  È necessario sottolineare che tale funzione supporta il preprocessing di un formato di dateset alternativo, caratterizzato dalla presenza di un ulteriore campo numerico che si riferisce alla "classe" del visitatore. Tuttavia, è necessario che l'intero dataset, inteso come l'insieme dei file caricati, presenti lo stesso formato. In caso contrario tutti i file saranno analizzati ignorando il campo classe.
- **CheckLabels:** si occupa di garantire la coerenza tra tra dataset e file di setup, verificando se le centraline presenti nel dataset, caricato dall'utente, sono tutte e sole quelle presenti nel file di setup.
- **UpdateLabelsInfo:** si occupa di valutare le statistiche relative alle centraline, sulla base dei filtri selezionati dall'utente.
- **UpdateFloorsInfo:** si occupa di valutare le statistiche relative ai piani, sulla base dei filtri selezionati dall'utente.
- **CreatePath:** metodo dedicato alla realizzazione della visualizzazione statica dei percorsi.
  A partire dai filtri selezionati dall'utente, crea gli oggetti necessari per il tracciamento dei percorsi, richiamando le funzionalità dello script *DrawPath*. Gestisce inoltre l'aggregazione dei dati per assicurare una visualizzazione più agevole.
- **CreateUsers**: funzione analoga alla precedente, dedicata alla realizzazione della visualizzazione animata dei percorsi.
  Crea gli oggetti necessari per il tracciamento dei percorsi, richiamando le funzionalità dello script *UserController*.
- **CreateUsersDrawing:** funzione analoga alle precedenti, dedicata alla realizzazione della visualizzazione ibrida dei percorsi.
  Crea gli oggetti necessari per il tracciamento dei percorsi, richiamando le funzionalità dello script *UserDrawing*.
- **Visualize**: si occupa della gestione dei filtri di visualizzazione impostati dall'utente a tempo di esecuzione, richiamando in base ad essi la funzione corretta.
- **StopVisualize, PauseVisualie, ResumeVisualize**: si occupano rispettivamente di gestire il termine, la pausa e la ripresa della

visualizzazione.
I metodi *PauseVisualize* e *ResumeVisualize* sono disponibili esclusivamente per la modalità animata.

- **GenerateStatistics**: funzione dedita alla generazione di un report complessivo, sulla base delle informazioni presenti nei dataset forniti dall'utente.
  In particolare viene prodotto un file testuale reperibile nella directory: *C:/Users/nomeUtente/AppData/LocalLow/DDFG/IoT Monitoring Museum/Resources*.

- **Drawpath:** script dedicato al tracciamento dei percorsi statici.
  Tale script è assegnato ad ognuno degli oggetti istanziati dal *GameController*. Dopo aver ricevuto in ingresso il percorso effettuato dal relativo visitatore, per ogni tratto congiungente due centraline:
  - Viene istanziato un componente **LineRenderer**;
  - Vengono impostati alcuni parametri, tra cui colore e spessore delle linee.
    Se all'interno del dataset è specificata la classe di appartenenza del visitatore, ad ognuna di esse è associato un colore differente. In caso contrario, è associato un colore ad ogni visitatore.
    Lo spessore delle linee tracciate è proporzionale al numero di visitatori che hanno effettuato il determinato percorso.
  - Viene effettuato il calcolo della curva attraverso il metodo *CalculateCuadraticBezierPoint*.
  - Viene fornito al LineRenderer l'insieme di punti risultanti.

- **UserController:** script responsabile del movimento degli agenti nella modalità animata.
  Tale script è assegnato ad ognuno degli oggetti istanziati dal *GameController*. Dopo aver ricevuto in ingresso il percorso effettuato dal relativo visitatore, si fa carico di impostare continuamente la prossima destinazione di un agente. In particolare:
  - Gli spostamenti tra centraline appartenenti allo stesso piano sono effettuati attraverso il componente **NavMeshAgent**. In questo modo ogni agente è in grado di raggiungere la propria destinazione evitando le superfici non navigabili e gli altri agenti.
  - Gli spostamenti tra centraline appartenenti a piani diversi sono invece gestiti in modo differente. Nel momento in cui un agente

deve trasferirsi in un altro piano si reca alla scala specifica del proprio piano, raggiunge la scala specifica del piano di destinazione attraverso la funzione nativa *MoveTowards* ed infine raggiunge la centralina di interesse.
Al fine di nascondere all'utente lo spostamento dei visitatori da un piano all'altro del museo, durante il trasferimento viene assegnato all'oggetto un *layer* non visibile alla camera presente in scena.

- **UserDrawing:** script dedicato al tracciamento dei percorsi nel tempo. Il funzionamento dello script è del tutto analogo al precedente, sebbene presenti alcune differenze. Durante il movimento, se un agente si trova nei pressi di uno degli oggetti indicati come *waypoints* viene memorizzata la posizione del determinato waypoint. Raggiunta una centralina di destinazione, l'insieme dei punti associati ai diversi wayponts rilevati dall'agente viene fornito in ingresso ad un oggetto di tipo LineRenderer, in grado di tracciare il percorso descritto. Il movimento degli agenti è nascosto all'utente durante l'intero percorso.

- **LabelScript:** script associato ad ogni centralina, consente attraverso le funzioni *AddVistors* e *GetVisitors* rispettivamente di ottenere informazioni statistiche riguardanti una determinata centralina.

- **FloorDataScript:** script associato ad ogni piano, consente attraverso le funzioni *SetFloorInfo*, *getVisitors* ed *AddVisitors* di ottenere informazioni statistiche riguardanti un determinato piano.

- **RayastingPopUp:** script associato alla *Main Camera*, consente di visualizzare statistiche dinamiche relative a visitatori e centraline. In particolare, tale script sfrutta la funzione *Raycast* del motore fisico di Unity e fa in modo che venga visualizzato un pop-up contenente:

  - Per le centraline, il numero di visitatori il cui percorso include la centralina selezionata;
  - Per i visitatori, il relativo percorso e il numero di visitatori associati alla singola sfera.
    Tale funzionalità è disponibile esclusivamente in modalità animata.

  Il raycast viene attivato alla pressione del tasto sinistro del mouse su una centralina o un utente.

- **FloorMovement:** script dedicato alla gestione della prospettiva di visualizzazione.

A seconda dell'opzione selezionata dall'utente, permette di spostare i piani in alcune posizioni prefissate, in modo da garantire tre diverse prospettive: orizzontale, verticale ed obliqua. Lo spostamento è visibile dinamicamente a tempo di esecuzione.
Terminato il movimento dei piani viene effettuato il baking della NavMesh, utilizzata nelle modalità animata e ibrida.

Listing 3.2: Spostamento del piano 0 nelle coordinate indicate

```
target0 = new Vector3(0, 0, 0);
float step = 0;
Floor0.transform.position = Vector3.MoveTowards(Floor0
    .transform.position, target0, step);

void Update()
{
    if (visualization == 1) //Orizzontal View
    {
        targetm1 = new Vector3(0, 0, -20);
        target0 = new Vector3(0, 0, 0);
        target1 = new Vector3(-30, 0, 0);
        target2 = new Vector3(-30, -0.2f, -22);
    }
    else if (visualization == 3) //Vertical View
    {
        targetm1 = new Vector3(0, -15, 0);
        target0 = new Vector3(0, 0, 0);
        target1 = new Vector3(0, 15, 0);
        target2 = new Vector3(0, 30, 0);
    }
    else if (visualization == 2) //Obliqual View
    {
        targetm1 = new Vector3(0, -5, 15);
        target0 = new Vector3(0, 0, 0);
        target1 = new Vector3(0, 5, -15);
        target2 = new Vector3(0, 10, -30);
    }

    float step = speed * Time.deltaTime;

    if (change != visualization)
    {
        viewDropdown.interactable = false;
        surface.enabled = false;

        Floorm1.transform.position = Vector3.
            MoveTowards(Floorm1.transform.position,
            targetm1, step);
```

```
37          Floor0.transform.position = Vector3.
                MoveTowards(Floor0.transform.position,
                target0, step);
38          Floor1.transform.position = Vector3.
                MoveTowards(Floor1.transform.position,
                target1, step);
39          Floor2.transform.position = Vector3.
                MoveTowards(Floor2.transform.position,
                target2, step);
40
41          if (Vector3.Distance(Floorm1.transform.
                position, targetm1) < r && Vector3.Distance
                (Floor0.transform.position, target0) < r &&
                 Vector3.Distance(Floor1.transform.position
                , target1) < r && Vector3.Distance(Floor2.
                transform.position, target2) < r)
42          {
43              surface.BuildNavMesh();
44              surface.enabled = true;
45              change = visualization;
46              viewDropdown.interactable = true;
47          }
48      }
49  }
```

- **FileManager:** script dedito alla gestione del caricamento di uno o più dataset da parte dell'utente. Utilizza le seguenti funzioni:
  - **UpdateText:** verifica se il path che riceve in ingresso è una stringa non vuota e in tal caso richiama due funzioni: *Add()* e *CheckElement()*;
  - **Add:** aggiunge un item al content del componente grafico ScrollView e su di esso aggiunge un Listener che richiama la funzione *OnItemSelected()* passando l'indice dell'elemento selezionato ricavato tramite la funzione *FindIndex()*;
  - **RemoveSelected:** rimuove un file selezionato dalla *ScrollView*;
  - **ClearOption:** elimina tutti i file dalla *ScrollView*;
  - **FindIndex**: ricerca l'indice del file selezionato;
  - **AddPath:** funzione statica che ritorna un vettore di stringhe precedentemente creato;
  - **OnItemSelected:** si occupa di aggiornare di volta in volta l'elemento selezionato dalla *ScrollView*;

- **CheckElement:** si occupa della gestione di alcuni bottoni che abilita/disabilita in base alla veridicità di alcune condizioni;

- **DraggingManager:** script in grado di realizzare lo spostamento del Menu Utente, entro i limiti consentiti dallo schermo.

Listing 3.3: Drag sulle direzioni X ed Y riassegnando la posizione

```
1 var posx = Input.mousePosition.x;
2 var posy = Input.mousePosition.y;
3
4 var positionx = Mathf.Clamp(posx, 0.01f, Screen.width)
      ;
5 var positiony = Mathf.Clamp(posy, 0.01f, Screen.height
      );
6
7 panel.transform.position = new Vector2(positionx,
      positiony);
```

3.2 Applicazione Back-end

L'applicazione Back-end è dedicata alla gestione delle centraline presenti all'interno del museo e prevede come utente finale un addetto alla gestione del placement delle centraline all'interno dei piani.

3.2.1 Interfaccia Utente

Per permettere all'utente di interagire con l'applicazione e gestire le centraline del museo è stata sviluppata una apposita interfaccia grafica utente (**GUI, Graphic User Interface**).

Menu Principale

Anche in questo caso, nel *Menu Principale* sono presenti le sezione relative ad *Impostazioni* e *Crediti*. È ovviamente possibile avviare la scena principale, contenete i quattro piani del museo.

Figura 3.5: Menu Principale Back-end

Menu Utente

All'interno della scena principale è presente un *Menu Utente* attraverso cui è possibile:

- **Aggiungere una centralina:** viene fornita all'utente la possibilità di aggiungere una nuova centralina, specificandone il nome e il piano di appartenenza.
  La centralina viene quindi aggiunta al centro del piano selezionato e l'utente è di trascinarla nella posizione desiderata attraverso il movimento del cursore del mouse e delle frecce direzionali.

- **Spostare una centralina:** viene fornita all'utente la possibilità di selezionare una centralina esistente e spostarla all'interno del piano di appartenenza.

- **Rimuovere una centralina:** viene fornita all'utente la possibilità di rimuovere una centralina esistente.

- **Salvare lo stato corrente:** se disponibile connessione Internet, vengono memorizzate le posizioni correnti delle centraline e rese disponibili all'applicazione Front-end.
  In caso contrario è visualizzato un messaggio di errore. Tuttavia, per preservare le modifiche apportate dall'utente alla disposizione delle centraline, viene creato un file di testo locale contenente la loro posizione

nella scena. In questo modo, anche in assenza di connessione, è possibile lavorare con l'applicativo per poi caricare le modifiche apportate quando la connessione Internet sarà disponibile.

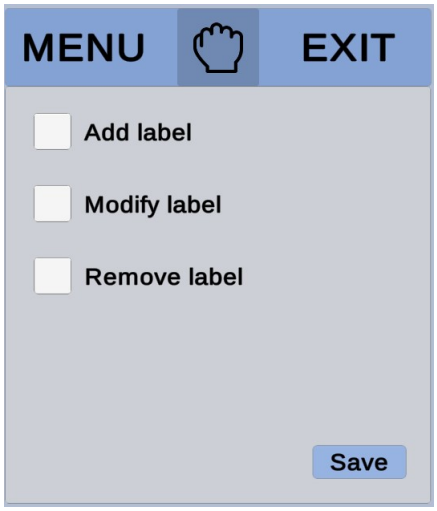

Figura 3.6: Menu Utente Back-end

3.2.2 Scripts

Per realizzare le funzionalità descritte sono stati implementati i seguenti script.

- **UnitsManagerController:** script chiave della scena principale, coordina le diverse funzionalità offerte dall'applicazione. Inoltre, garantisce la coerenza tra interfaccia utente e strutture dati, mantenendo una lista aggiornata delle centraline presenti in scena.

  Tra le diverse funzioni di tale script vale la pena menzionare:

  - **LoadSetUp:** funzione richiamata all'avvio della scena principale. Si occupa di caricare in scena, se disponibile, l'ultima configurazione di centraline salvata.
  - **SavaData**: funzione dedicata alla comunicazione con l'applicazione Front-end.
    Genera un file in formato *.txt* in cui sono memorizzate le posizioni correnti delle centraline presenti nel museo ed alcune informazioni

ad esse associati.
Successivamente il file è caricato tramite connessione FTP su un server remoto situato ad una certa URL, specificando determinate credenziali di accesso. Anche in questo caso tutti i parametri della connessione possono essere impostati dall'utente.

– **CreateUnit, MoveUnit, DestroyUnit**: si occupano rispettivamente di gestire la creazione, la modifica e la rimozione di ogni centralina.

Listing 3.4: Estratto di UnitsManagerController: upload FTP sul repository

```
1  try
2  {
3      FtpWebRequest request = (FtpWebRequest)WebRequest.
           Create(ftpParametrs[0]);
4          request.Method = WebRequestMethods.Ftp.
               UploadFile;
5
6
7      request.Credentials = new NetworkCredential(
           ftpParametrs[1], ftpParametrs[2]);
8
9      byte[] fileContents;
10     using (StreamReader sourceStream = new
           StreamReader(pathPreviousSetup))
11     {
12         fileContents = Encoding.UTF8.GetBytes(
               sourceStream.ReadToEnd());
13     }
14
15     request.ContentLength = fileContents.Length;
16
17     using (Stream requestStream = request.
           GetRequestStream())
18     {
19         requestStream.Write(fileContents, 0,
               fileContents.Length);
20     }
21
22     using (FtpWebResponse response = (FtpWebResponse)
           request.GetResponse())
23     {
24     }
25
26     popSavWithConn.SetActive(true);
27
```

```
28 }
29 catch (UriFormatException e)
30 {
31 popUpURI.SetActive(true);
32
33 }
34 catch (WebException e)
35 {
36     popUpConnectionPanel.SetActive(true);
37     mainPanel.SetActive(false);
38 }catch(Exception e)
39 {
40     popUpConnectionPanel.SetActive(true);
41     mainPanel.SetActive(false);
42 }
```

- **UnitController:** script associato ad ogni centralina, si occupa di gestire il posizionamento della stessa.
  In particolare, viene effettuato un controllo sulla posizione scelta dall'utente, verificando se è lecita e compatibile con i campi selezionati. In caso contrario viene visualizzato un messaggio di errore e la centralina è riportata al centro del piano specificato.

Capitolo 4

Diagrammi UML

I diagrammi UML forniscono una visione delle classi e dei componenti che intervengono nella creazione di un sistema e ciò che il sistema permette all'utente di fare.
Sono di seguito riportati alcuni diagrammi UML relativi agli applicativi di Front-end e Back-end, realizzati al fine di:

- Mostrare le funzionalità offerte dal sistema;
- Mostrare le modalità di interazione fra utente e sistema;
- Mostrare la topologia del sistema distribuito;
- Mostrare in maniera semplificata il funzionamento interno del sistema e l'interazione fra i due applicativi;

4.1 Use Case Diagrams

Gli Use Case Diagram (diagrammi dei casi d'uso) sono diagrammi UML dedicati alla descrizione delle funzioni o servizi offerti da un sistema, così come sono percepiti e utilizzati dagli attori che interagiscono con il sistema stesso.
Di seguito sono riportati i diagrammi dei casi d'uso per le applicazioni di *Front-end* e *Back-end.*

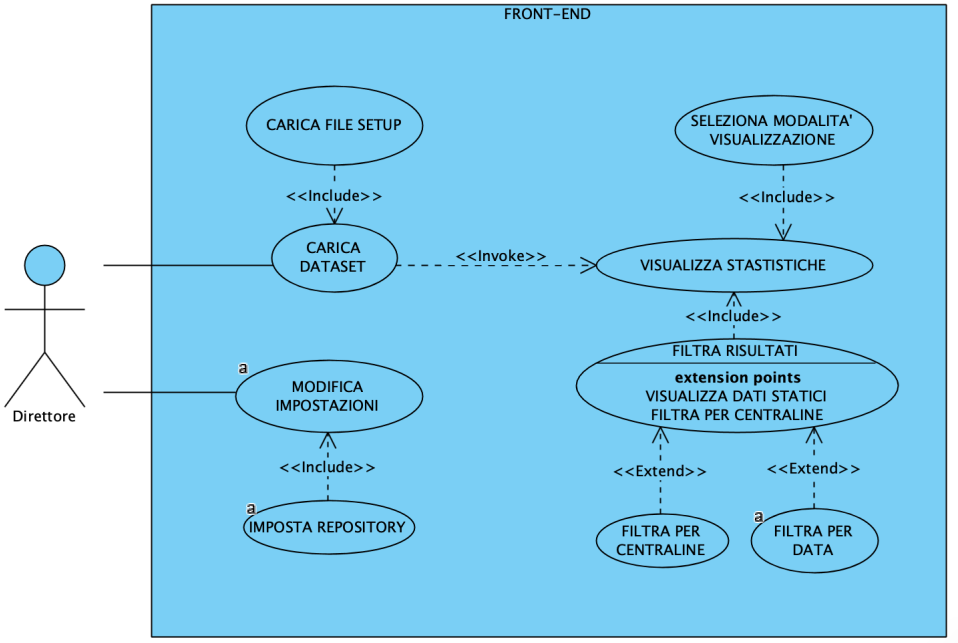

Figura 4.1: Use Case Diagram: Front-end

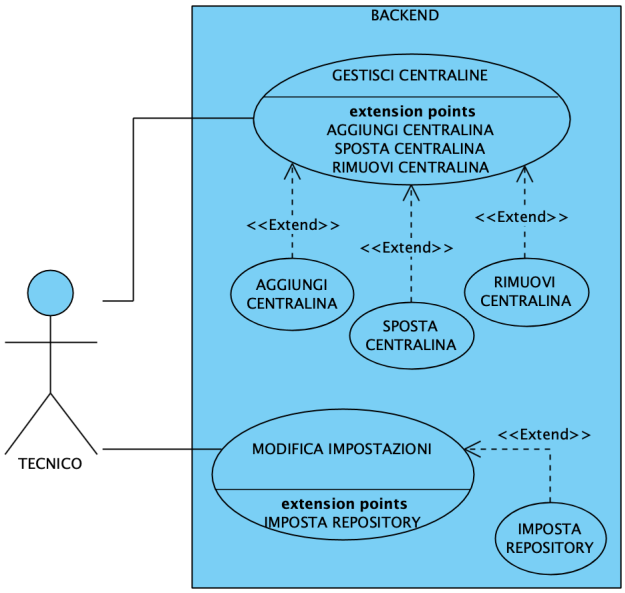

Figura 4.2: Use Case Diagram: Back-end

4.2 Deployment Diagram

Il Deployment Diagram (diagramma di dispiegamento) è un diagramma UML di tipo statico che descrive un sistema in termini di risorse hardware, dette nodi, e relazioni fra di esse.
Nello specifico si riporta di seguito la disposizione del sistema:

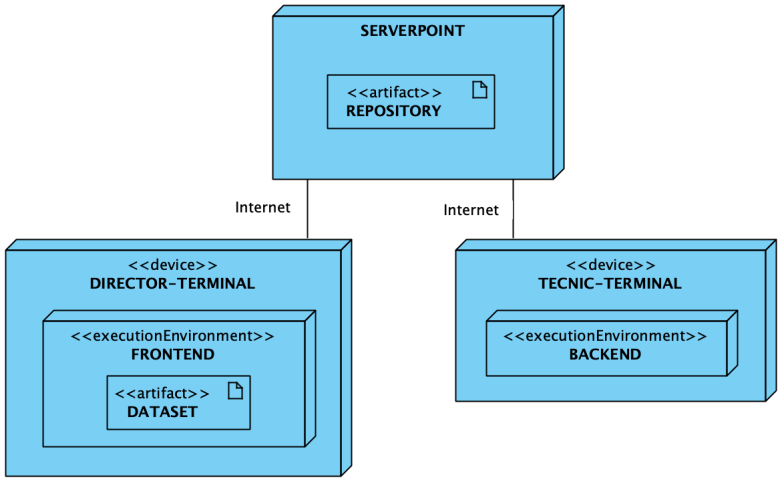

Figura 4.3: Deployment Diagram

4.2.1 Comunicazione tra Back-End e Front-End

Tramite l'applicazione di Back-end, è possibile modificare la disposizione delle centraline, nonchè aggiungerle e rimuoverle. Tutte le modifiche si ripercuoto su un file .txt che è caricato dal Back-end su un hosting online alla pressione del tasto *Salva* presente nella UI. *In caso di connessione Internet assente il Back-end non può comunicare con il repository online e le modifiche non si possono ripercuotere sul Front-end.* Tuttavia le modifiche apportare non sono perse poichè salvate in locale.
Al primo avvio del Front-end, il file di setup necessario per la corretta disposizione delle centraline è assente. È quindi necessario disporre di connessione Internet, in modo da poter scaricare tale file, precedentemente caricato dal Back-end, dal repository online. Per i successivi utilizzi del Front-end, l'applicazione tenta sempre di scaricare eventuali versioni aggiornate del file di setup dal server online ma, in caso di connessione Internet assente, viene utilizzato l'ultimo file scaricato e salvato in memoria.

4.3 Sequence Diagrams

Un Sequence Diagram è un diagramma UML apartenente al gruppo degli *Interaction Diagrams*, utilizzato per descrivere uno scenario.
Uno scenario è una determinata sequenza di azioni in cui tutte le scelte sono state già effettuate. Di conseguenza non compaiono nel diagramma costrutti condizionali, né altri tipi di flussi alternativi. Tipicamente ad ogni Use Case Diagram è associato uno o più Sequence Diagram.
Dalla versione 2 dell'UML è stata introdotta la possibilità di indicare nello stesso diagramma anche sequenze di eventi alternative, attraverso i costrutti *Loop*, *Alt*, *Opt*.
Il Sequence Diagram descrive le relazioni che intercorrono, in termini di messaggi, tra Attori, Oggetti di business, Oggetti o Entità del sistema che si sta rappresentando. Nel seguito sono riportati alcuni semplici sequence diagram di analisi, che descrivono le interazioni fra i principali componenti del sistema in specifici scenari di utilizzo.

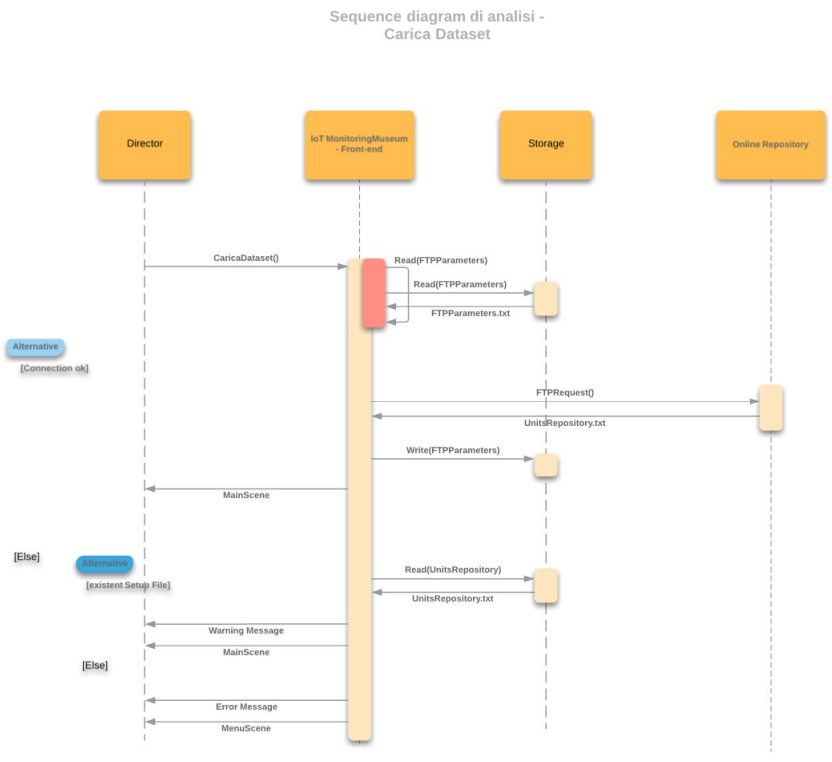

Figura 4.4: Sequence Diagram : Carica Dataset

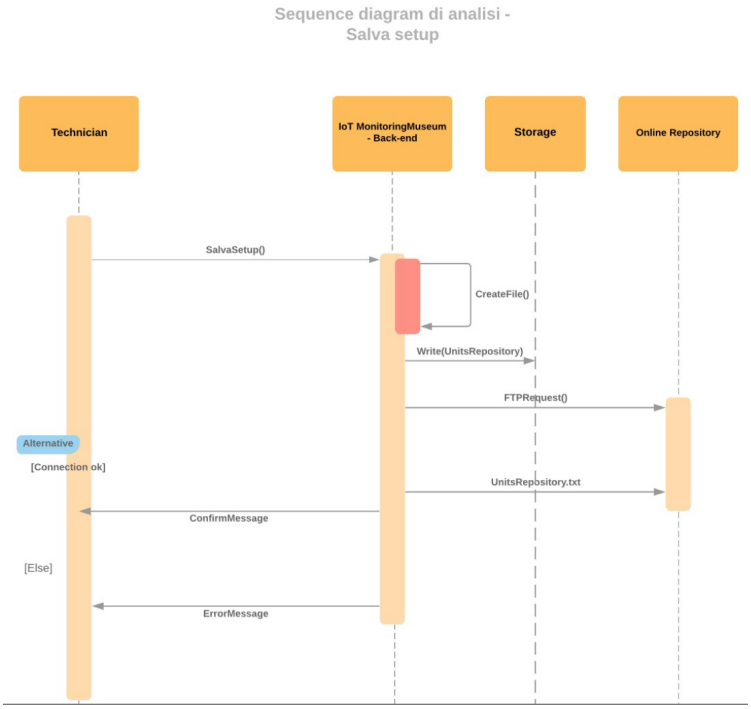

Figura 4.5: Sequence Diagram : Salva Setup