

Università degli Studi di Camerino

Scuola di Scienze e Tecnologie
Corso di Laurea in Informatica (Classe L-31)



**Sviluppo di un Sistema per la realizzazione di
Tour Virtuali**

Laureando:

Fabio Virgili

Matricola: 095868

Relatore:

Prof. Fausto Marcantoni

Correlatore:

Dott. Massimiliano Sampaolo

Abstract

Questo lavoro si pone come obiettivo quello di esporre i passaggi nello sviluppo di un sistema per la realizzazione di tour virtuali, composto da un lato front-end per l'aggiunta e la modifica degli elementi dei tour utilizzando un'interfaccia utente e un lato back-end per l'elaborazione e la gestione delle informazioni in arrivo dal front-end e il mantenimento dei dati dei tour in un database.

Le tecnologie che vengono utilizzate sono: ASP.NET Core MVC, JavaScript, Bootstrap, A-Frame per il lato front-end, ASP.NET Core e SQLITE per il lato back-end.

A tutte le persone che hanno sempre
creduto in me: la mia famiglia e i miei
amici

Indice

Abstract	3
1. Introduzione	10
1.1 Motivazioni	10
1.2 Obiettivi	10
1.3 Contesto	11
1.4 Struttura della tesi.....	12
2. Tecnologie utilizzate	13
2.1 ASP.NET Core.....	13
2.2 Pattern Model-View-Controller (MVC)	13
2.3 ASP.NET Core MVC.....	14
2.4 HTML	1
2.5 Bootstrap	1
2.6 ORM	1
2.7 Entity Framwork Core	1
2.8 LINQ.....	1
2.9 A-Frame	1
2.9.1 Entity-Component-System.....	1
2.9.2 Three.js e WebGL	1
3. Progettazione	2
3.1 Architettura Client-Server.....	2
3.2 Cosa sono le REST API?	2
3.3 Requisiti	2
3.3.1 Requisiti delle WebAPI.....	2
3.3.2 Requisiti del Client Amministrativo	1
3.4 Struttura del database	2
4. Sviluppo	8
4.1 Sviluppo delle REST API	8
4.2 Sviluppo del client amministrativo	2

5.	Conclusioni	2
6.	Riferimenti sitografici.....	3
7.	Ringraziamenti.....	4

Indice delle Figure

Figura 1: una scena del client utente con l'elemento spot.....	11
Figura 2: una scena del client utente con punto di informazione e pannello descrittivo.....	12
Figura 3: schema del pattern MVC.....	14
Figura 4: schema dell'utilizzo del componente ViewModel.....	15
Figura 5: schema entità relazionale del database.....	2
Figura 6: schermata di postman che mostra una richiesta GET con relativa risposta.	2
Figura 7: interfaccia del client amministrativo.....	2
Figura 8: uno dei form per l'aggiunta di un elemento nella scena (in questo caso il punto di informazione).....	1

1. Introduzione

1.1 Motivazioni

Negli ultimi anni il mercato della realtà virtuale e della realtà aumentata è in costante crescita e sempre più aziende iniziano ad investire su queste tecnologie. I potenziali utilizzi del VR in un futuro più o meno prossimo saranno innumerevoli e troverà applicazione in molteplici settori anche molto diversi tra loro. Questo ha portato molte aziende ad interessarsi allo sviluppo di applicazioni VR e alla sperimentazione di nuove tecnologie che ne facilitino l'utilizzo e lo sviluppo.

Mi è stato assegnato questo lavoro di tesi dall'azienda dove ho svolto il tirocinio universitario, perché interessata allo studio delle potenzialità del framework WebVR A-Frame e dello sviluppo di un'applicazione webVR per l'esplorazione dei tour virtuali. L'idea era quella di creare un client utente per dare la possibilità di scegliere un tour tra quelli disponibili ed iniziare l'esplorazione. Risultava perciò necessario sviluppare anche un lato back-end per la gestione di tutte le informazioni utili al client per la visualizzazione ed esplorazione dei tour virtuali. Inoltre, è sorta l'esigenza di creare un client amministrativo per poter inserire nel database nuovi tour o modificare quelli esistenti attraverso un pannello contenente degli appositi form.

1.2 Obiettivi

All'interno del progetto mi è stato assegnato il compito di sviluppare il lato back-end e il client amministrativo, mentre una mia compagna di corso si è occupata del client utente. L'obiettivo di questo lavoro di tesi è quindi quello di spiegare i passaggi nello sviluppo di questo sistema per la creazione e modifica di tour virtuali e la gestione del suo lato back-end

Nello specifico gli argomenti trattati sono:

- La creazione di un server WeI per l'esposizione di end-point per l'invio e la ricezione di dati.
- Lo sviluppo di una semplice interfaccia di tipo amministrativo utilizzabile per l'inserimento dei dati nel database e la creazione e modifica dei tour virtuali disponibili per l'esplorazione.
- Struttura e realizzazione del database.

1.3 Contesto

Ai fini della comprensione del lavoro svolto e delle motivazioni che mi hanno portato a fare delle specifiche scelte durante lo sviluppo, è importante dare delle definizioni di come i tour sono creati all'interno del client utente.

Nel client utente, la prima pagina visualizzata, è un menù dove vengono inseriti tutti i tour presenti nel database e pronti per l'esplorazione. Un tour è composto da un titolo e un'immagine visibili nel menù, e da un insieme di scene che vengono caricate una volta scelto un tour.

Ogni Scena che compone il Tour è formata da almeno un elemento Sky che corrisponde ad un'immagine a 360 gradi e da diversi elementi opzionali: lo spot, il punto di informazione, il pannello descrittivo.

L'elemento Spot è il punto che permette il passaggio da una scena all'altra, il punto di informazione permette di cambiare la visibilità del pannello informativo, il pannello informativo è un pannello descrittivo contenente del testo scorrevole attraverso delle frecce.

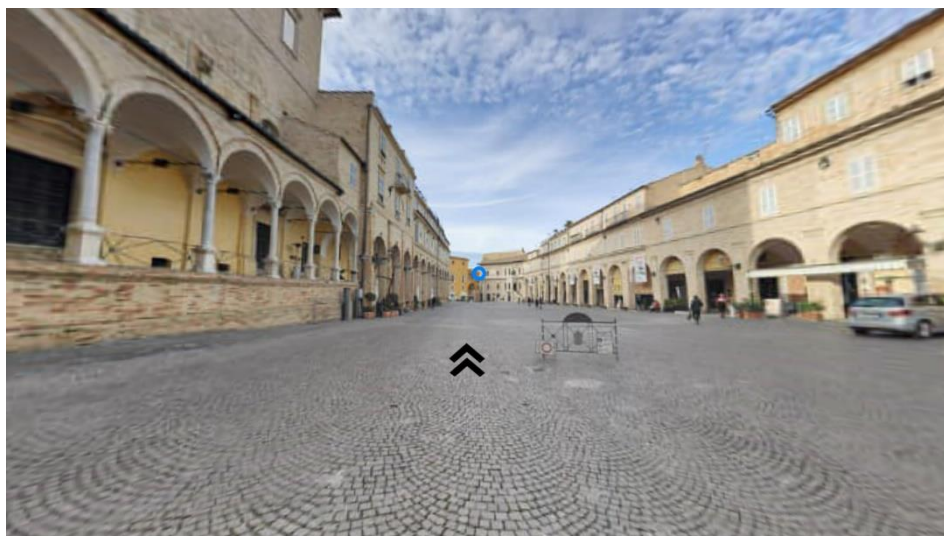


Figura 1: una scena del client utente con l'elemento spot.



Figura 2: una scena del client utente con punto di informazione e pannello descrittivo.

1.4 Struttura della tesi

Questo lavoro di tesi è strutturato in una serie di paragrafi e sottoparagrafi divisi in otto capitoli. Il primo capitolo è quello introduttivo, che comprende le motivazioni, gli obiettivi e il contesto del lavoro. Nel secondo capitolo è presente l'elenco delle tecnologie utilizzate nel progetto con relativa spiegazione. Il terzo capitolo elenca i requisiti che sono stati analizzati per poter sviluppare il progetto, mentre il quarto comprende la parte di progettazione del lavoro. Il quinto capitolo corrisponde allo sviluppo del progetto, il sesto alle conclusioni e il settimo all'elenco dei siti presi come riferimento per descrivere le parti teoriche del lavoro. Infine l'ottavo capitolo contiene i ringraziamenti finali.

2. Tecnologie utilizzate

2.1 ASP.NET Core

ASP.NET Core è un framework Web open-source sviluppato da Microsoft che decide di rivedere completamente il suo approccio nello sviluppo per Internet presentando una soluzione riscritta da zero che includeva i progetti precedenti: ASP.NET MVC e API Web ASP.NET. Microsoft si è avvalsa dei suggerimenti e del supporto della sua vasta community di sviluppatori in modo da individuare le esigenze e le preferenze dei developer di terze parti che fanno riferimento alla sua piattaforma.

ASP.NET Core è un framework Web modulare che funziona sia su .NET Framework completo che su Windows e su .NET Core multiplatforma, dunque è possibile sfruttarlo ovunque indipendente dal sistema operativo o dalla piattaforma software scelta per iniziare lo sviluppo della propria applicazione o servizio. In aggiunta è possibile aggiungere funzionalità tramite l'installazione di Pacchetti NuGet.

Le applicazioni ASP.NET Core supportano il controllo delle versioni affiancate, con cui diverse applicazioni, in esecuzione sulla stessa macchina, possono scegliere come target versioni diverse di ASP.NET Core.

ASP.NET Core è indicato sia per lo sviluppo di pagine web, sia per lo sviluppo di Api Rest.

2.2 Pattern Model-View-Controller (MVC)

Model-View-Controller (MVC) è un pattern utilizzato in programmazione per dividere il codice in blocchi dalle funzionalità ben distinte.

Secondo questo framework, la costruzione di un'applicazione web si basa sulla composizione di tre ruoli distinti:

- **Model:** rappresenta il modello dei dati, ovvero l'unione dei servizi che regolano l'accesso ai dati, la definizione delle entità e delle relazioni tra esse.
- **View:** permettono di presentare i dati in HTML, in modo che possano essere visualizzati nel browser.
- **Controller:** hanno il ruolo di coordinatori, ovvero ricevono i dati dagli utenti e sfruttano le classi del Model per ottenere i dati richiesti.

L'idea fondamentale del pattern è che nell'applicazione, la View è il componente dedicato solamente alla visualizzazione dell'interfaccia utente. Questa comunica con il Model che contiene tutte le informazioni che la View deve visualizzare. Il modello non ha alcuna informazione su come i dati vengono mostrati all'utente finale.

Il Controller si occupa di gestire le richieste in arrivo dall'utente (di solito attraverso le View) e li attua andando a modificare lo stato degli altri due componenti.

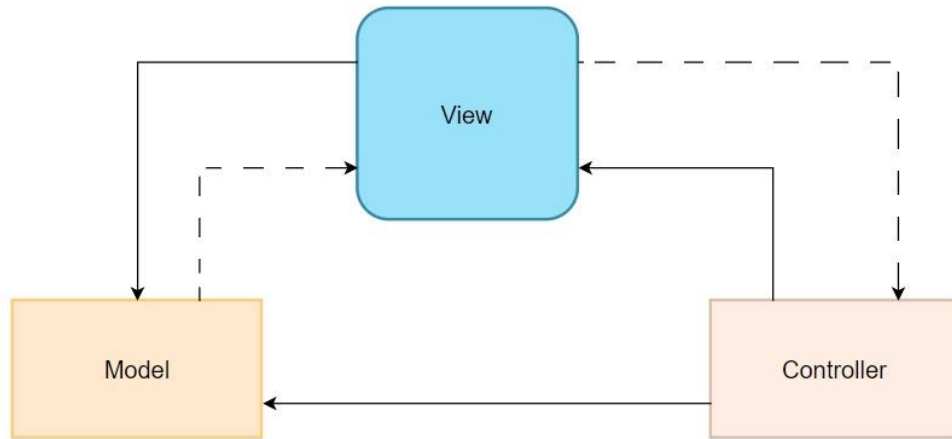


Figura 3: schema del pattern MVC.

2.3 ASP.NET Core MVC

Il framework ASP.NET Core MVC segue la divisione dettata dal pattern MVC, andando a dividere il codice dell'applicazione in tre blocchi distinti: il Model, le View ed i Controller.

Le View sono i componenti che hanno la responsabilità di presentare in HTML i dati che vengono forniti dal controller. Non dovrebbero contenere al loro interno alcun codice per l'accesso al database, o di qualunque attività che richiede comunicazione con l'esterno dell'applicazione.

In ASP.NET Core MVC le View oltre al codice HTML possono contenere, a supporto della presentazione dei dati, anche codice C#. Questo permette allo sviluppatore di poter utilizzare, ad esempio, dei cicli for per scorrere una lista o controlli if-else per nascondere o mostrare una sezione della pagina. Il codice viene inserito tramite il simbolo @.

Un Controller è una semplice classe C# che deriva dalla classe Controller e che viene creata all'interno della directory /Controllers. I metodi pubblici creati all'interno di un Controller si chiamano action. I Controller utilizzano il middleware di routing e dei servizi di ASP.NET Core MVC per richiamare un metodo quando viene richiamata una vista che di solito ha il suo stesso nome.

Il Model è il componente MVC dedicato alla logica per l'accesso ai dati, che si occupa cioè di fare da tramite tra l'applicazione e il database sottostante.

Il Model inoltre non soltanto fornisce fisicamente l'accesso ai dati, ma si preoccupa anche di creare il necessario livello di astrazione tra il formato in cui i dati sono memorizzati alla fonte e il formato in cui i livelli di Controller e View si aspettano di riceverli.

Le classi contenute nel Model sono usate, per esempio, dai Controller che le sfruttano per ottenere i dati che dovranno essere presentati dalle View. Per rendere i singoli componenti riutilizzabili è buona scelta organizzare il Model dividendolo in piccole classi altamente specializzate. Il pattern MVC non fornisce alcuna istruzione su quale tecnologia utilizzare all'interno di un Model, lasciando completa libertà agli sviluppatori.

Per il passaggio dei dati da Model a View può essere usato un ulteriore elemento chiamato ViewModel: classi che hanno la funzione di contenere al proprio interno delle proprietà. Sono classi prive di logica che vengono passate alle View in modo che i dati al loro interno siano presentati in HTML. Le classi ViewModel contengono solo i dati necessari alla creazione della View a cui vengono passati.

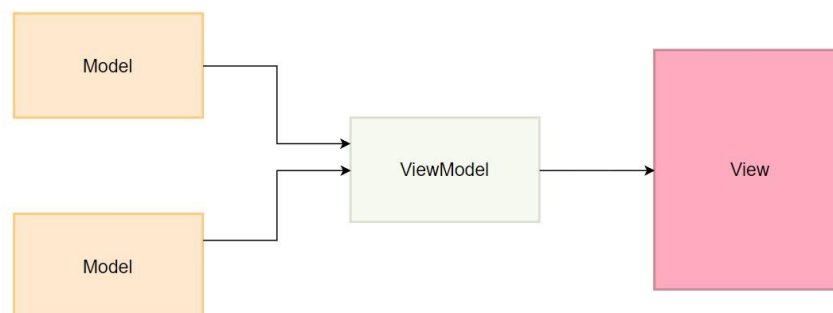


Figura 4: schema dell'utilizzo del componente ViewModel.

Una delle convenzioni implementate da Microsoft in ASP.NET MVC segue l'approccio "Convention over Configuration" che prevede di default che alcuni aspetti nello sviluppo siano implicitamente configurati dal framework stesso.

In ASP.NET MVC, ogni sottocartella di Views viene automaticamente collegata all'elemento controller che presenta il suo stesso nome. Ad esempio, una sottocartella "Home" della cartella "Views" contenente diverse Views, sarebbe automaticamente collegata al controller "HomeController" contenuto nella cartella Controllers.

2.4 HTML

HTML (HyperText Markup Language) è un linguaggio a marcatori per ipertesti. È definito come un linguaggio di markup utilizzato per strutturare e impaginare documenti ipertestuali di una pagina web. HTML rappresenta lo scheletro di una pagina web, e ne stabilisce la struttura attraverso dei componenti chiamati tag.

2.5 Bootstrap

Bootstrap è un framework open-source per lo sviluppo di siti e pagine web. È un framework multiplatforma poiché compatibile con le ultime versioni di tutti i principali browser. Supporta il responsive web design, ovvero la regolazione dinamica del layout degli elementi di una pagina web in base alle caratteristiche del dispositivo con la quale la si visualizza.

2.6 ORM

Un ORM (Object Relational Mapper) è una tecnologia che consente la manipolazione (creazione, aggiornamento, eliminazione) dei dati situati in database relazionale con i linguaggi di programmazione orientata agli oggetti come C#. Un ORM si occupa di convertire in SQL le query fortemente tipizzate, inoltre legge i risultati trovati nel database e li converte in oggetti prima di restituirli.

I vantaggi dati dall'utilizzo di un ORM ad uno sviluppatore sono molteplici, oltre a permettere l'utilizzo di dati come oggetti ed a rendere il codice più facile da riutilizzare e aggiornare, permette di ridurre la dimensione. L'utilizzo di ORM protegge anche dagli attacchi di code injection (inserimento ed esecuzione di stringhe di codice) andando a filtrare i dati.

2.7 Entity Framework Core

Entity Framework Core è un ORM open-source, cross-platform ed è stato costruito da Microsoft su ADO.NET che usa a basso livello per interagire con il database.

Nel realizzare il modello con Entity Framework si possono seguire due approcci:

- database-first: nel caso in cui lo sviluppatore ha già a disposizione un suo database, EFC semplifica la creazione delle classi del modello concettuale usando il comando `dotnet ef dbcontext scaffold`. In questo modo infatti, è possibile fare un reverse engineering del database e lasciare che sia il framework ad analizzare le tabelle e creare per ognuna di esse una rispettiva classe.

- code-first: in questo caso è lo sviluppatore che si prende gli oneri di scrivere le classi che compongono il modello concettuale e in seguito, sarà il framework a generare lo schema del database grazie alle migration.

A prescindere dall'approccio utilizzato, Entity Framework ha bisogno di un componente che derivi dalla classe DbContext in cui sia realizzato il mapping tra il modello a oggetti e il modello relazionale.

Questo mapping avviene attraverso un'interfaccia fluida, e deve essere effettuato tra:

- classi di entità: ovvero le classi che hanno come corrispettivo nel database una tabella con chiave primaria.
- owned types: classi che non hanno un riscontro nel database e che servono solo per mantenere dei dati.
- relazioni tra classi di entità: seguono le normali relazioni di un database relazionale (uno a uno, uno a molti, molti a molti).

2.8 LINQ

LINQ (Language-Integrated Query) è il nome di un set di tecnologie basate sull'integrazione delle funzionalità di query direttamente nel linguaggio C#.

In LINQ, la query è considerata un costrutto del linguaggio di prima classe, come lo sono i metodi, le classi e gli eventi. È infatti possibile scrivere query su insiemi di oggetti fortemente tipizzati usando parole chiave del linguaggio e gli operatori comuni. Poiché è possibile formare query nel linguaggio di programmazione stesso, non è necessario usare un altro linguaggio di query incorporato sotto forma di valori letterali stringa che il compilatore non è in grado di riconoscere o verificare.

Sono disponibili tre tecnologie LINQ separate: LINQ to DataSet, LINQ to SQL e LINQ to Entities. LINQ to DataSet offre una query più completa e ottimizzata sul DataSet e LINQ to SQL consente di eseguire query direttamente sugli schemi di database SQL Server e LINQ to Entities consente di eseguire query su un Entity Data Model.

È possibile quindi scrivere query LINQ in C# per database SQL Server, documenti XML, set di dati ADO.NET e qualsiasi raccolta di oggetti che supporta IEnumerable o l'interfaccia IEnumerable<T> generica. Il supporto per LINQ viene anche offerto da terze parti per numerosi servizi Web e altre implementazioni di database. Il trasferimento di dati da tabelle SQL in oggetti in memoria è un'operazione spesso laboriosa e soggetta a errori. Il provider LINQ implementato da LINQ to DataSet e LINQ to SQL converte i dati di origine in raccolte di oggetti basati su IEnumerable. Il programmatore visualizza sempre i dati sotto forma di una raccolta IEnumerable, sia quando si esegue una query che quando si effettua un aggiornamento. Per la scrittura di query da eseguire su tali raccolte, viene fornito il supporto per IntelliSense

Per uno sviluppatore che scrive query, la parte integrata nel linguaggio più visibile di LINQ è l'espressione di query. Le espressioni di query vengono scritte con una sintassi di query dichiarativa. Tramite la sintassi di query è possibile eseguire operazioni di filtro, ordinamento e raggruppamento sulle origini dati usando una quantità minima di codice.

2.9 A-Frame

A-Frame è un framework utilizzato per la creazione di tour virtuali basato sul pattern entity-component-system. Rende possibile la visualizzazione di elementi 3D in realtà virtuale usando degli elementi personalizzati di HTML che utilizzano a livello inferiore Three.js e WebGL.

2.9.1 Entity-Component-System

Entity-Component-System è il pattern architetturale su cui è basato A-Frame, usato soprattutto nello sviluppo dei videogiochi e divide gli elementi in tre parti:

- Entità: un elemento generico caratterizzato solo da un ID, il suo ruolo è quello di contenitore dei componenti. Sono i componenti a dare una vera identità all'entità che nella sua forma base non ha caratteristiche né comportamenti.
- Componenti: un insieme di dati che caratterizzano il comportamento o l'aspetto di un elemento e come esso interagisce con il mondo.
- Sistema: esegue delle azioni globali, permette di eseguire delle azioni su tutte le entità in possesso di un determinato componente.

2.9.2 Three.js e WebGL

WebGL (web graphics library) è un set di API che permette di utilizzare alcune funzioni del linguaggio JavaScript per produrre elementi di grafica 3D renderizzati e visualizzati dal browser. WebGL lavora con elementi basilari come punti, linee e triangoli e richiede un lavoro con questi elementi per sviluppare cose più complesse.

Three.js è un'API cross-browser basata su WebGL che permette la creazione e la visualizzazione di elementi e animazioni in 3D come parte di un sito web, semplificando lo sviluppo rispetto a WebGL. L'utilizzo di Three.js permette di evitare l'uso di applicazioni esterne al browser.

3. Progettazione

Il lavoro che mi è stato assegnato si basa su un'architettura client-server e presenta diverse parti:

- La creazione del database per mantenere i dati relativi ai tour virtuali.
- Lo sviluppo e l'esposizione di WebAPI, nello specifico REST API.
- La creazione di un'interfaccia di tipo amministrativo per il caricamento e la modifica delle informazioni nel database.

3.1 Architettura Client-Server

Un sistema basato sull'architettura client-server è un sistema composto da due parti: un server che mette a disposizione dei servizi o delle risorse tramite la rete e un client che accede a questi servizi o risorse.

Un client non è molto complesso, di solito si limita ad operare come interfaccia verso il server. Un client può contattare un server se ne conosce l'indirizzo, e richiede il servizio o le risorse da lui condivise tramite l'invio di appositi messaggi.

Il server è il componente che ha al suo interno la logica di sistema, gestisce il modo in cui le risorse vengono rilasciate e ne regola l'accesso da parte del client.

Un client e un server comunicano attraverso un protocollo di comunicazione che può essere in chiaro o crittografato, l'utilizzo di un protocollo permette che le due parti possano comunicare tra loro anche se sono state sviluppate indipendentemente e con linguaggi diversi.

3.2 Cosa sono le REST API?

Una REST API è un'application programming interface che utilizza le richieste HTTP (GET, POST, PUT, DELETE,...) per compiere delle azioni sui dati definiti dal modello. Un'API permette a due software di comunicare tra loro in maniera trasparente anche se sono sviluppati con linguaggi di programmazione diversi e girano su sistemi operativi eterogenei, senza che l'uno abbia informazioni sull'altro. Il principale vantaggio dell'usare una REST API per il passaggio dei dati è il fatto che richiede solamente l'utilizzo di HTTP e per questo è di facile utilizzo.

3.3 Requisiti

3.3.1 Requisiti delle WebAPI

Lo scopo principale nello sviluppo delle WebAPI è l'esposizione degli endpoint per l'invio e la ricezione di informazioni riguardanti gli elementi che compongono i tour.

Nello specifico, per il client utente mi è stato richiesto di esporre gli endpoint per l'invio dei dati utili all'applicazione web per la visualizzazione dei tour in realtà virtuale:

- GET /api/tour: il client utente utilizza questo URI per ricevere le informazioni riguardante posizione e titolo di tutti i tour presenti in memoria per caricare il menu di selezione iniziale.
- GET /api/tour/{idTour}: viene usato al click di uno specifico tour e serve a ricevere le informazioni di tutti gli elementi all'interno di quello specifico tour.
- GET /api/tour/{idTour}/{idScene}: utilizzato per prendere le informazioni di una specifica scena del tour selezionato dall'utente. Viene usato dal client per ricevere i dati utili all'inizializzazione delle scene in realtà virtuale.

Per il client amministrativo, che si occupa sia di aggiungere che di modificare i dati nel database, erano richieste le operazioni CRUD per i seguenti URI:

- /api/tour
- /api/scene
- /api/spots
- /api/infos
- /api/panels
- /api/sky

3.3.2 Requisiti del Client Amministrativo

La funzione principale del Client Amministrativo è dare la possibilità all'utente di creare e modificare i tour virtuali inserendo i dati attraverso l'interfaccia grafica nel database.

Con "creazione di un tour" si intende l'inserimento nel database di tutte le informazioni che permettono la sua visualizzazione all'interno del client utente.

I requisiti per la creazione di un tour sono:

- Avere un nome univoco per poter essere caricato nel menu.
- Avere delle coordinate e un titolo per essere posizionato correttamente e visibile all'utente nel menu.

I requisiti per la creazione di ogni scena che contiene un tour sono:

- Avere un nome univoco e almeno l'elemento sky al suo interno.
- La prima scena di ogni Tour deve avere il flag *visible = true*.

Ogni scena può avere al suo interno degli elementi opzionali, quali:

- Lo Spot, che permette il passaggio da una scena a quella successiva.
- Il Pannello, che permette di visualizzare un testo descrittivo all'interno della scena.
- L'elemento info, che attiva o disattiva la visibilità di un pannello.

3.4 Struttura del database

Una volta ricevute le specifiche relative agli elementi che compongono un tour, sono state analizzate le possibili entità che avrebbero composto il database cercando di rendere il tutto più snello possibile per velocizzare poi lo sviluppo del server Web API.

Sono stati identificati tutti gli attributi che compongono le entità e quindi create le varie tabelle del database e le relazioni tra esse.

L'analisi ha portato alla creazione del seguente schema entità relazionale:

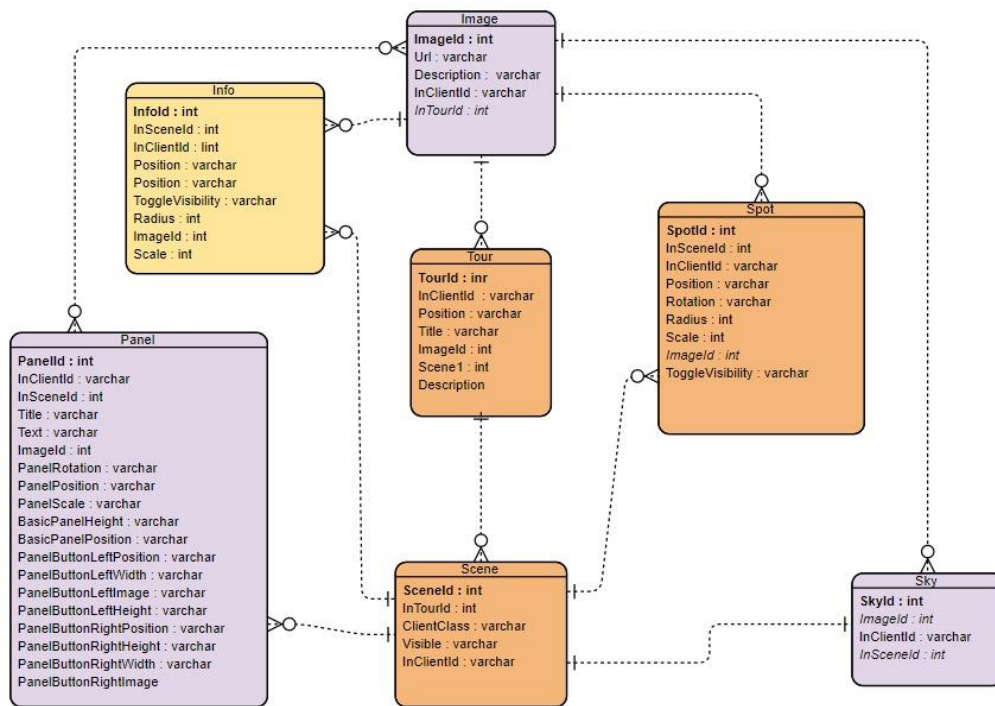


Figura 5: schema entità relazionale del database.

Tabella Tour

Nome Campo	Tipo	Descrizione
TourId	Int	Chiave Primaria

InClientId	Varchar	Id del tour utilizzato come indice univoco del client
Position	Varchar	Posizione del tour all'interno del menù di scelta
Title	Varchar	Titolo del Tour
ImageId	Varchar	Chiave esterna della tabella "Image". Utilizzata per la visualizzazione dell'immagine del tour all'interno del menù
Scene1	Varchar	Chiave esterna della tabella "Scene". Utilizzata dal front-end per poter cercare la prima scena di ogni tuor in caso di selezione.

Tabella Scene

Nome Campo	Tipo	Descrizione
SceneId	Int	Chiave Primaria
InTourId	Int	Chiave Esterna della tabella Tour
ClientClass	Varchar	Classe della scena utilizzata per capire quale scena viene visualizzata al click dell'elemento spot (solo nel client utente)
Visible	Varchar	Flag per indicare la visibilità della scena
InClientId	Int	Id del tour indicato come indice univoco nel client

Tabella Info

Nome Campo	Tipo	Descrizione
------------	------	-------------

InfoId	Int	Chiave Primaria
InSceneId	Int	Chiave esterna della tabella Scene
InClientId	Varchar	Id dell'elemento info utilizzato come indice univoco nel client
Position	Varchar	Coordinate dell'elemento info all'interno della scena
Rotation	Varchar	Coordinate per l'orientamento del pannello info all'interno della scena
Radius	Int	Raggio del cerchio dell'elemento Info
ToggleVisibility	Varchar	Id dell'elemento panel a cui l'elemento info modifica la visibilità
ImageId	Varchar	Chiave esterna della tabella Image
Scale	Int	Grandezza dell'elemento info

Tabella Sky

Nome Campo	Tipo	Descrizione
SkyId	Int	Chiave primaria della tabella Sky
ImageId	Varchar	Chiave Esterna della tabella Image
InSceneId	Int	Chiave esterna della tabella Scene
InClientId	Varchar	Id dell'elemento Sky utilizzato come indice univoco nel client

Tabella Spot

Nome Campo	Tipo	Descrizione
SpotId	Int	Chiave primaria
InClientId	Varchar	Id dell'elemento Spot utilizzato come indice univoco nel client
InSceneId	Int	Chiave esterna della tabella Scene
Position	Varchar	Coordinate dell'elemento Spot all'interno della scena
Rotation	Varchar	Coordinate per l'orientamento dell'elemento Spot all'interno della scena
ToggleVisibility	Varchar	Usato per cambiare scena, indica quale scena è quella di partenza e quale quella di arrivo
Radius	Int	Raggio del cerchio dell'elemento spot
Scale	Varchar	Grandezza dell'elemento spot
ImageId	Int	Chiave esterna della tabella Image

Tabella Panel

Nome Campo	Tipo	Descrizione
PanelId	Int	Chiave primaria
InSceneId	Int	Chiave esterna della tabella Scene

InClientId	Varchar	Id dell'elemento Panel utilizzato come indice univoco nel client
Title	Varchar	Titolo del pannello
Text	Varchar	Il testo informativo contenuto nel pannello
PanelPosition	Varchar	Coordinate dell'elemento Panel all'interno della scena
PanelRotation	Varchar	Coordinate per l'orientamento dell'elemento Panel all'interno della scena
PanelScale	Varchar	Grandezza dell'elemento panel
BasicPanelPosition	Varchar	Posizione del pannello contenente il titolo all'interno della scena
BasicPanelHeight	Varchar	Altezza del pannello contenente il titolo all'interno della scena
PanelButtonLeftPosition	Varchar	Posizione del bottone di sinistra all'interno della scena
PanelButtonLeftHeight	Varchar	Altezza del bottone di sinistra all'interno della scena
PanelButtonLeftWidth	Varchar	Larghezza del bottone di sinistra all'interno della scena
PanelButtonLeftImage	Varchar	Immagine del bottone di sinistra all'interno della scena
PanelButtonRightPosition	Varchar	Posizione del bottone di destra all'interno della scena

PanelButtonRightHeight	Varchar	Altezza del bottone di destra all'interno della scena
PanelButtonRightWidth	Varchar	Larghezza del bottone di destra all'interno della scena
PanelButtonRightImage	Varchar	Immagine del bottone di destra all'interno della scena

Tabella Image

Nome Campo	Tipo	Descrizione
ImageId	Int	Chiave Primaria
Description	Varchar	Descrizione opzionale dell'immagine
Url	Varchar	URL relativo all'immagine
InClientId	Varchar	Id dell'immagine utilizzato come indice univoco nel client
InTourId	Int	Chiave esterna della tabella Tour

4. Sviluppo

4.1 Sviluppo delle REST API

Terminata la stesura del modello Entità relazionale del database si è scelto di popolarlo manualmente con dei record di prova per creare un tour di test utile al client utente.

Avendo già creato il database, si è scelto di utilizzare l'approccio database first, ovvero sono state create le classi che hanno composto il modello a oggetti (o modello concettuale) del progetto. Queste classi sono: Tour, Scene, Spot, Panel, Sky, Info ed Image e corrispondono, appunto, alle entità che si trovano nel database.

Successivamente è stata creata la classe `TourDbContext` derivata dalla classe `"DbContext"`. La funzione di questa classe è quella di esporre tutti i tipi che si vogliono includere nel modello inserendo la proprietà `DbSet` per ogni tipo al suo interno e creare il mapping tra il modello relazionale e il modello a oggetti.

Il mapping viene eseguito all'interno del metodo `"onModelCreating"` tramite interfaccia fluida.

```
modelBuilder.Entity<Tour>(entity =>
    {
        entity.ToTable("Tour");
        entity.HasKey(e => e.TourId);
        entity.Property(e =>
e.InClientId).HasColumnName("InClientId");
    }
}
```

Anche Entity Framework Core, come ASP.NET Core, segue l'approccio convention over configuration. Questo permette di semplificare lo sviluppo e il mapping tra il modello relazionale e il modello a oggetti rendendo automatiche alcune funzioni.

Per il mapping tra una classe e un'entità si usa il comando `entity.ToTable([columnName]);` ma questa operazione non è necessaria se la classe nel modello concettuale ha lo stesso nome dell'entità nel modello relazionale.

Per il mapping tra la chiave primaria di un'entità e la sua relativa proprietà, viene usato il metodo `entity.HasKey([Property]);` che viene effettuato automaticamente se una proprietà di una classe è denominata `Id` o con il nome della classe seguita da `"id"`.

Per il mapping tra una proprietà e la sua rispettiva colonna si usa `entity.property([Property]).HasColumnName([ColumnName]);` ma questa operazione non è necessaria se la colonna ha lo stesso nome della proprietà.

In seguito, per ogni entità del database di cui bisognava esporre i dati, sono stati creati i controller che, attraverso i metodi, si sono occupati di rispondere alle richieste GET POST PUT DELETE. Il formato con il quale vengono gestite e inviate le risorse è il JSON.

Di seguito l'esempio di un metodo GET.

```
[HttpGet]
[Route("api/tour/{id}/image")]
public ActionResult<List<Tour>> GetTourImage(int id)
{
    var db = new TourDbContext();
    var image = from i in db.Image join t in db.Tour on i.InTourId
equals t.TourId
where t.TourId == id
select i;
    if (image == null)
    {
        return NotFound("image not found");
    }
    return Ok(image);
}
```

Sono stati successivamente effettuati dei test tramite l'applicazione Postman, per verificare la funzionalità dei metodi. Postman è un'API development tool usato per costruire, testare e modificare le API. La sua funzione principale è quella di poter fare vari tipi di richieste HTTP (GET, POST, PUT, PATCH,...), salvare lo stato delle richieste e i loro risultati per una consultazione futura.

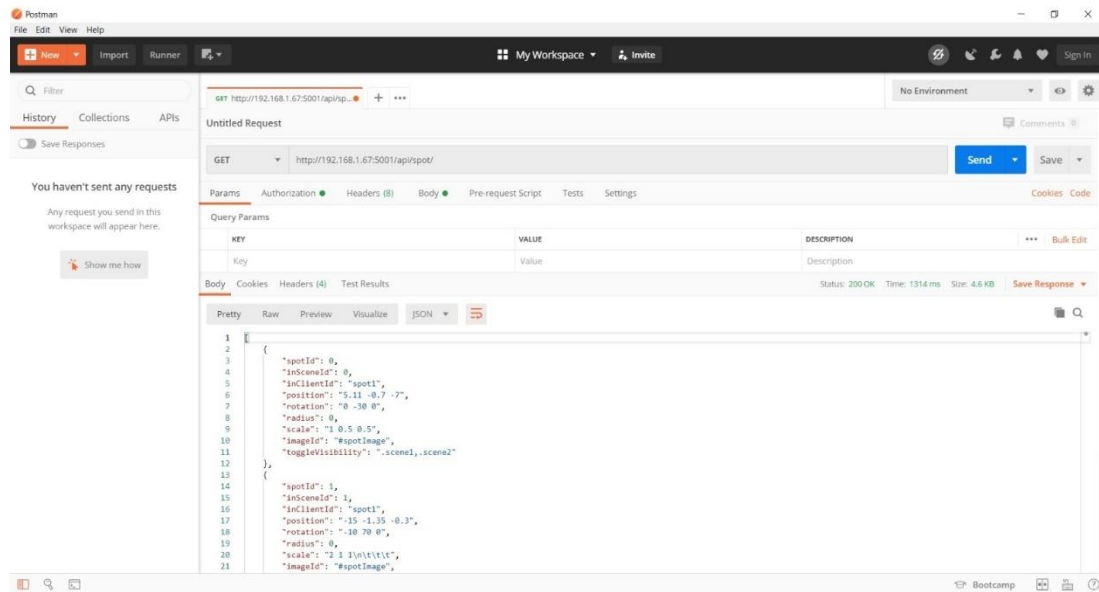


Figura 6: schermata di postman che mostra una richiesta GET con relativa risposta.

4.2 Sviluppo del client amministrativo

Il pannello di amministrazione è stato sviluppato in ASP.NET Core seguendo il pattern MVC.

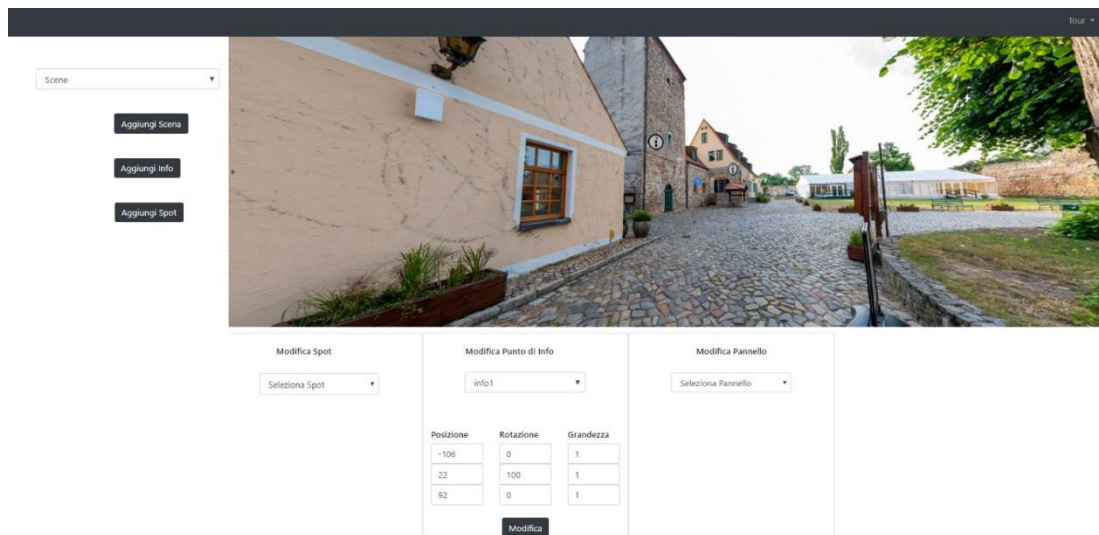


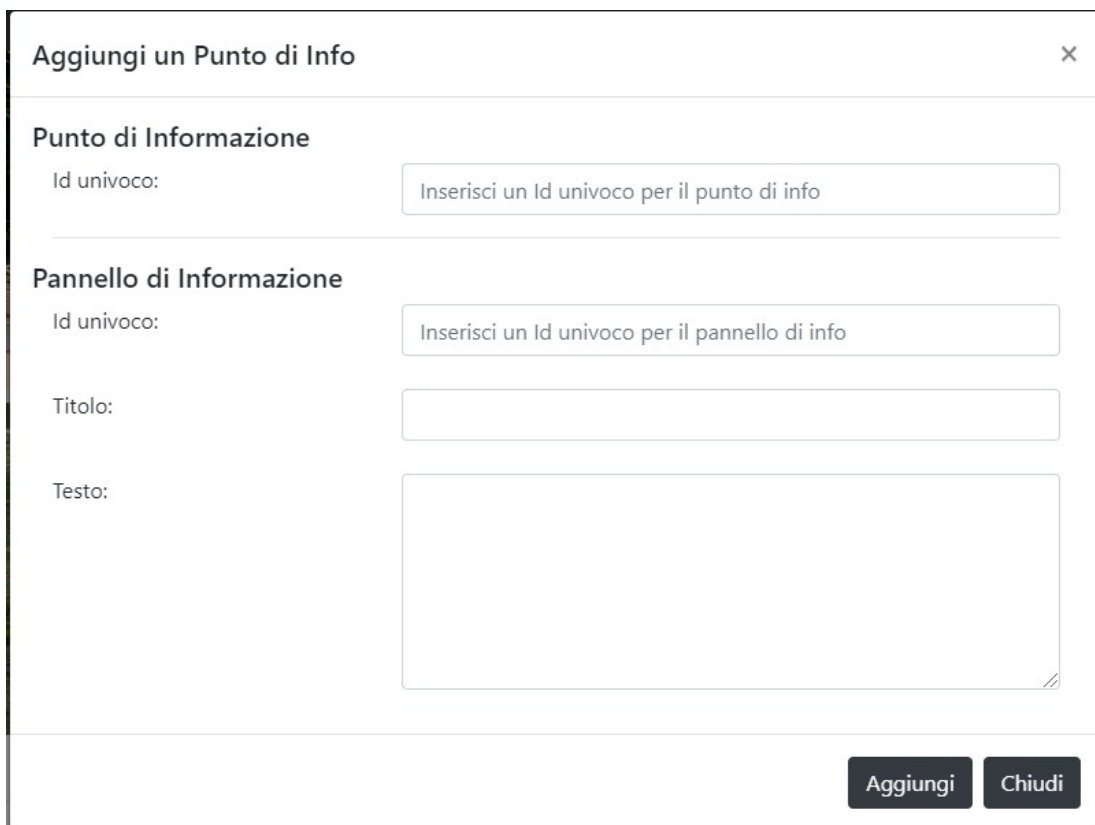
Figura 7: interfaccia del client amministrativo.

Il client amministrativo è composto di una sola View che permette la creazione di nuovi tour o la modifica di tour già presenti nel database.

La View presenta:

- Una barra superiore per la scelta di un tour esistente o la creazione di un nuovo tour.
- Un pannello laterale composto da bottoni per l'aggiunta di scene, spot o punti di informazione e da un elemento dropdown per la selezione di una delle scene esistenti.
- Un pannello inferiore per la scelta degli elementi presenti nella scena selezionata.
- La scena selezionata come scena VR di A-Frame con tutti gli elementi presenti visibili.

Ogni volta che viene aggiunto un elemento alla scena attraverso il pannello laterale, apparirà un form da compilare con le informazioni necessarie al database.



The image shows a modal window titled "Aggiungi un Punto di Info" with a close button (X) in the top right corner. The form is divided into two main sections:

- Punto di Informazione:** Contains a label "Id univoco:" followed by a text input field with the placeholder text "Inserisci un Id univoco per il punto di info".
- Pannello di Informazione:** Contains three fields:
 - "Id univoco:" followed by a text input field with the placeholder text "Inserisci un Id univoco per il pannello di info".
 - "Titolo:" followed by a text input field.
 - "Testo:" followed by a large text area with a small diagonal icon in the bottom right corner.

At the bottom right of the form, there are two buttons: "Aggiungi" and "Chiudi".

Figura 8: uno dei form per l'aggiunta di un elemento nella scena (in questo caso il punto di informazione).

La selezione di un elemento della scena attraverso il pannello inferiore comporterà la visualizzazione di un pannello di modifica della posizione, rotazione e grandezza di

quell'elemento. Il click del bottone “modifica” relativo a quell'elemento porterà alla modifica in tempo reale dei parametri impostati per quell'elemento.

La scena VR di A-Frame è incorporata all'interno della View grazie al parametro *embedded* posto all'interno del tag `<a-scene>`. Questo permette di eliminare lo *style: fullscreen* del css della scena che perciò può essere ridimensionata.

Il corretto funzionamento e costruzione della scena VR in A-Frame, avviene per mezzo di una serie di componenti:

- *loadAsset.js*: ha la funzione di richiamare l'endpoint `/tour/image` del tour selezionato dall'utente, in modo tale da ricevere l'URL di tutte le immagini che esso contiene e poter renderizzarle prima del caricamento della scena.
- *init-scene.js*: si occupa delle chiamate iniziali al server WebApi per l'inizializzazione degli elementi della scena selezionata attraverso la funzione *createScene()*;
- *split()*: utilizzato per dividere il testo contenuto all'interno di un pannello descrittivo in sottostringhe di 180 caratteri. Senza questo componente il testo coprirebbe un'area maggiore della grandezza del pannello.
- *component.js*: prende i dati in input dall'utente e, in base al form compilato, si occupa di creare un nuovo elemento nella scena VR. Ha anche il compito di far apparire i pannelli relativi alla modifica di rotazione, posizione e grandezza degli elementi spot, info e pannello descrittivo e modificare quindi i parametri dello specifico elemento selezionato.

All'avvio dell'applicazione, il Controller HomeController si occupa di instanziare le classi Services: TourService, SceneService, InfoService, SpotService PanelService che hanno il compito di recuperare i dati delle relative entità nel database e creare una lista di suddetti elementi. Il Controller poi, creerà una classe HomeViewModel che conterrà la lista di tutti gli elementi recuperati dai Services da passare alla View.

Nel momento in cui l'utente fa click sul bottone modifica relativo ad uno specifico elemento per poter cambiare i suoi parametri di posizione, rotazione e grandezza, verrà scatenata la funzione *add()*; dell'elemento selezionato. Questa funzione si occupa di prendere i valori all'interno del form, creare un json e invocare una chiamata PUT alle WebAPI, in modo da aggiornare i parametri dell'elemento selezionato dall'utente nel database.

Nel momento in cui l'utente fa click sul bottone per la creazione di un nuovo elemento da inserire nella scena, vengono raccolti i dati inseriti dall'utente e invocata una chiamata POST al server API per la creazione di un nuovo elemento che sarà presente sia nel database che nella scena VR.

5. Conclusioni

L'obiettivo prefissato è stato raggiunto: il lavoro svolto ha dato una solida base nello sviluppo del client utente e nel conseguente studio del framework A-Frame. Lo studio delle tecnologie e lo sviluppo dell'applicazione hanno permesso l'acquisizione personale di nuovi concetti e abilità, permettendomi di avere una miglior conoscenza di ASP.NET Core e dell'architettura MVC. Lavorare in un ambiente aziendale è stato stimolante ed importante per la comprensione degli argomenti e delle dinamiche aziendali.

Lo studio basato sull'integrazione di A-Frame con codice HTML delle Views ha avuto successo ed è stata riscontrata massima compatibilità tra i componenti, permettendo la visualizzazione dei tour all'interno della pagina HTML e la modifica degli elementi presenti nelle scene grazie all'uso dei form.

L'applicazione è ovviamente solo ad uno stato embrionale: essendo un lavoro finalizzato allo studio del framework, non sono state implementate alcune funzionalità importanti per un'applicazione web come ad esempio un metodo di autenticazione o l'integrazione del tool swagger per documentare la struttura delle API.

È importante dire che parte delle decisioni prese durante la fase di progettazione e sviluppo del progetto, sono state delle scelte obbligate dalle tante limitazioni date dall'utilizzo del framework A-Frame. Prima fra tutte la necessità di utilizzare JavaScript per le funzioni riguardanti la scena VR.

6. Riferimenti sitografici

1. <https://www.wikipedia.org/>
2. <https://aframe.io/>
3. <https://developer.mozilla.org/it/>
4. <https://docs.microsoft.com/it-it/aspnet/>
5. <https://getbootstrap.com/>

7. Ringraziamenti

Questo spazio lo dedico a tutti coloro che mi hanno supportato durante il mio intero percorso formativo universitario.

Ringrazio in primis mio padre e mia madre, e la loro infinita pazienza, per avermi dato la possibilità di arrivare alla fine di questo percorso.

Ringrazio i miei amici e colleghi universitari, Giuseppe, Jonathan, Matteo e Sara, per ogni momento di studio e non, passato insieme.

Ringrazio il team dell'azienda DevQ srl in cui ho svolto il tirocinio formativo, grazie la quale ho ampliato le mie conoscenze e imparato molto sull'ambiente aziendale.

Ringrazio infine Alice per il sostegno che mi ha dato durante tutto questo periodo.