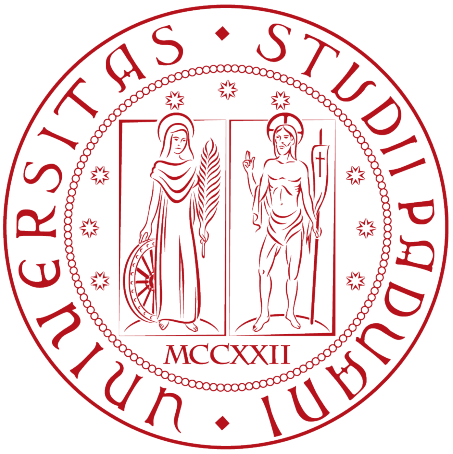
Università degli Studi di Padova

Dipartimento di Matematica "Tullio Levi-Civita"

Corso di Laurea in Informatica



Applicazione Cloud per la riproduzione video on-demand a qualità dinamica

Tesi di laurea

Relatore

Prof. Ombretta Gaggi

Laureando

Mattia Brunello 2009096

Anno Accademico 2022-2023

Sommario

Il presente documento descrive il lavoro svolto durante il periodo di stage, della durata di circa trecento ore, dal laureando Mattia Brunello presso l'azienda AdMaioraStudio S.r.l.
Gli obbiettivi da raggiungere erano molteplici.
In primo luogo lo studio e la documentazione delle varie opzioni per lo sviluppo di un'applicazione cloud, tecniche e tecnologie utilizzate e i loro possibili futuri sviluppi.
In secondo luogo, lo sviluppo e l'analisi di un PoC di una WebApp per la gestione dello streaming di video di prodotti di vari espositori nelle fiere mondiali.
In terzo luogo le conclusioni con la documentazione completa degli artefatti sviluppati e definizione dei possibili casi d'uso futuri.

Ringraziamenti

Innanzitutto, vorrei esprimere la mia gratitudine al Prof. Ombretta Gaggi, relatore della mia tesi, per l'aiuto e il sostegno fornitomi durante la stesura del lavoro.

Desidero ringraziare con affetto i miei genitori e i miei nonni per il sostegno, il grande aiuto e per essermi stati vicini in ogni momento durante gli anni di studio.

Padova, Luglio 2023 Mattia Brunello 2009096

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Introduzione

1.1 L'idea

Nell'attuale panorama delle fiere e degli eventi commerciali, le aziende partecipanti hanno manifestato un crescente interesse nella promozione innovativa dei propri prodotti. Attualmente, essa avviene principalmente attraverso la distribuzione di materiale pubblicitario, come brochure, volantini e cataloghi, lasciando al visitatore il compito di informarsi autonomamente sui prodotti offerti dai vari espositori.
In questa tesi verrà descritto lo sviluppo di una WebApp attraverso la quale gli espositori avranno la possibilità di caricare i propri video relativi ai prodotti esposti, rendendoli successivamente disponibili per la riproduzione on-demand da parte dei visitatori. L'idea rappresenta un passo avanti verso l'innovazione della promozione dei prodotti nelle fiere, perché offre ai partecipanti un'esperienza interattiva e coinvolgente.

1.2 Descrizione dello stage

L'azienda ha manifestato l'esigenza di sviluppare un PoC per la realizzazione di una WebApp che permetta agli espositori di caricare i propri video relativi ai prodotti esposti, e li renda disponibili per la riproduzione on-demand da parte dei visitatori.
L'idea è quella di realizzare un prodotto che possa essere utilizzato in occasione di fiere ed eventi commerciali, in modo da offrire ai partecipanti un'esperienza interattiva e coinvolgente.
L'applicazione è stata sviluppata utilizzando come linguaggio di backend C#, per lo sviluppo del frontend il framework JavaScript React [1] , mentre per la gestione dell'archiviazione e streaming dei video sono stati utilizzati i servizi di Microsoft Azure: Media Service, Account Storage, SQL Server e App Service. [2]
L'obbiettivo principale di questo PoC, è stato quello di studiare e verificare la fattibilità di un prodotto di questo tipo.

[1] https://reactjs.org/
[2] https://azure.microsoft.com/

1.3 L'azienda

Ad Maiora Studio è una software house nata nel 2013 per operare nel campo del mobile e che negli anni si è specializzata anche nello sviluppo di software.
La sua mission è centrata sull'attenzione verso i clienti e lo sviluppo di software moderni, scalabili e progettati ad hoc per soddisfare le loro esigenze.
I prodotti sviluppati da Ad Maiora Studio si basano sulle più recenti tecnologie, integrano componenti e librerie eterogenee e pongono una forte enfasi sull'esperienza utente e l'interfaccia grafica.
L'azienda si distingue per l'approccio continuativo di assistenza ai clienti, garantendo un partner sempre accessibile e in grado di rispondere tempestivamente alle richieste.
Ad Maiora Studio si concentra principalmente su piccole e medie imprese operanti nei settori industriale e dei servizi, incoraggiandole a intraprendere un percorso di modernizzazione iniziando dal software.
L'obiettivo è supportare efficacemente l'adattamento alle mutevoli esigenze di mercato e di business, fornendo strumenti innovativi e personalizzati, che rappresentano il cuore dell'attività di Ad Maiora Studio.
Grazie alla competenza Full Stack del team di sviluppatori, l'azienda è in grado di realizzare ogni tipo di software, coprendo l'intero processo di sviluppo, dalla progettazione all'implementazione.
La qualità delle soluzioni software offerte è sempre un punto focale, al fine di soddisfare appieno le aspettative dei clienti e garantire il massimo risultato.

Capitolo 2

Fondamenti teorici e tecnologie utilizzate

In questo capitolo verranno descritti i fondamenti teorici necessari per la comprensione del lavoro svolto.

2.1 Concetti di video on-demand e streaming

Video on-demand (tradotto come video su richiesta) o VOD, è un sistema che permette di accedere a contenuti multimediali (video, audio, immagini) in qualsiasi momento e in qualsiasi luogo tramite una connessione internet. Contrariamente alla trasmissione televisiva tradizionale, nella quale gli utenti sono limitati da un palinsesto predefinito, il VOD dà la possibilità agli utenti di scegliere quale contenuto guardare e quando usufruirne.
Nel contesto della webapp sviluppata, questo concetto ha una funzione chiave, infatti consente agli utenti di accedere a una vasta gamma di selezione video riguardanti i prodotti esposti nelle fiere mondiali, permettendo di scegliere i video di loro interesse in base alle loro preferenze e necessità, eliminando le limitazioni spazio-temporali delle fiere fisiche.
Nel corso della tesi, verranno analizzate le caratteristiche e le sfide associate all'implementazione del video on-demand nella webapp, quindi le strategie di gestione e organizzazione dei contenuti, nonché la scalabilità e la qualità dello streaming per garantire un'esperienza fluida e coinvolgente per gli utenti.

2.1.1 Tipologie di video on-demand

Esistono diverse tipologie di video on-demand, sotto elencate:

- **Subscription VOD** ovvero i servizi con un canone periodico come ad esempio Netflix, Amazon Prime Video ecc..

- **Transactional VOD** ossia servizi che permettono di acquistare o noleggiare contenuti, come ad esempio Google Play, Apple TV, Chili ecc..

- **Advertising VOD** ossia servizi gratuiti che mostrano annunci pubblicitari durante la riproduzione dei contenuti, come ad esempio Youtube, RaiPlay e Mediaset Play

- **Premium VOD** ovvero la trasmissione di contenuti Premium, come anteprime cinematografiche, eventi sportivi ecc.., proposti da piattaforme come ad esempio Curzon Cinemas

2.1.2 Concetti di streaming video

Lo streaming video è un metodo di trasmissione di dati multimediali, in particolare di video e audio. Esistono due principali categorie di streaming video:

- **Video on-demand** è la trasmissione di contenuti pre-registrati, come ad esempio film, serie TV, documentari ecc.., i quali vengono compressi e memorizzati su un server come file, e vengono trasmessi agli utenti che ne fanno richiesta senza la necessità che il contenuto venga scaricato sul dispositivo dell'utente. Infatti i dati ricevuti dalla richiesta vengono decompressi e riprodotti in tempo reale.
- **Live streaming** è simile alle trasmissioni televisive tradizionali, in cui gli utenti guardano i contenuti in tempo reale. Viene utilizzato per trasmettere eventi in diretta come ad esempio concerti, eventi sportivi ecc.., vengono anch'essi leggermente compressi e memorizzati su un server, ma vengono trasmessi in tempo reale agli utenti che ne fanno richiesta.

2.1.3 Protocolli e tecnologie di streaming video

Per la trasmissione di contenuti multimediali, esistono diversi protocolli e tecnologie, sotto elencati:

- **HTTP Live Streaming** o HLS è un protocollo di streaming sviluppato da Apple nel 2009. Permette la trasmissione in streaming di contenuti multimediali frammentando il contenuto in segmenti di file HTTP e distribuendolo ai dispositivi client utilizzando il medesimo protocollo.
  HLS è adattivo: il client può cambiare la qualità del video in base alla larghezza di banda disponibile senza interrompere la riproduzione. È nativamente compatibile con i dispositivi Apple ma è supportato anche dalla maggior parte dei dispositivi e browser che supportano HTTP, non richiedendo l'installazione di plugin aggiuntivi.

- **Dynamic Adaptive Streaming over HTTP** o DASH è un protocollo di streaming sviluppato dal Moving Picture Experts Group (MPEG), permette la trasmissione di contenuti multimediali attraverso il protocollo HTTP.
  DASH suddivide il contenuto in segmenti e li trasmette ai dispositivi client tramite HTTP, permettendo un adattamento dinamico della qualità del video in base alla larghezza di banda disponibile. Offre una vasta gamma di scelta del formato video e codec, permettendo di scegliere il formato più adatto per il dispositivo client.

- **Real Time Messaging Protocol** o RTMP è un protocollo di streaming sviluppato da Adobe nel 2012, permette la trasmissione di contenuti multimediali in tempo reale, divide il contenuto in pacchetti e li trasmette ai dispositivi client tramite TCP o UDP, consente una comunicazione bidirezionale tra il server e il dispositivo client utilizzando un flusso continuo.
  RTMP è un protocollo di streaming non adattivo, ovvero non permette di cambiare la qualità del video in base alla larghezza di banda disponibile, ma permette di trasmettere contenuti in tempo reale con una bassa latenza.
  Dal 2020, con la deprecazione di Adobe Flash Player, RTMP è stato sostituito da protocolli di streaming adattivi come HLS e DASH.

2.2 I problemi dello streaming

2.2.1 Latenza

La latenza è il tempo di ritardo tra l'invio di un pacchetto e la ricezione di una risposta, è un problema comune nello streaming, in quanto può causare ritardi nella riproduzione del video. Può essere causata da diversi fattori, come ad esempio la velocità delle connessione, la distanza tra il server e il dispositivo client, la compressione del video e la capacità di elaborazione del dispositivo client. Per ridurre la latenza si utilizzano protocolli efficienti in base alla tipologia del contenuto.

2.2.2 Qualità del video

Un aspetto importante dello streaming video è la qualità del video, essa dipende da diversi fattori, i due principali sono la compressione e la codifica del video.

- **La compressione** è un processo che riduce la dimensione del file video, rimuovendo le informazioni ridondanti o non necessarie, permettendo una trasmissione più rapida ed efficiente del video. Può essere di due tipi: lossless e lossy.
  La compressione lossless è un processo che riduce la dimensione del file video senza perdita di qualità, mentre la compressione lossy è un processo che riduce la dimensione del file video con una leggera perdita di qualità.

- **La codifica** è un processo che converte il video da un formato a un altro formato, consentendo la trasmissione e riproduzione dei video su diversi dispositivi e piattaforme. Esistono diversi formati video, i più comuni sono: H.264, H.265, VP9 e AV1.

2.2.3 Sicurezza

La sicurezza è un aspetto importante dello streaming video, in quanto i contenuti multimediali possono essere facilmente copiati e distribuiti senza autorizzazione. Per proteggerlo esistono diversi metodi, i più comuni sono: Digital Rights Management (DRM) e Watermarking.
Il DRM è un metodo di protezione da copie non autorizzate, impostando una chiave di

protezione, che viene utilizzata per decodificare i contenuti multimediali.
Il Watermarking è un altro metodo di protezione che permette di proteggere i contenuti con un watermark, ovvero un segno distintivo (come ad esempio un logo), che viene utilizzato per identificare il proprietario.

2.2.4 Gestione del carico del server

I server di streaming possono ricevere un elevato numero di richieste da tutto il mondo, questo può causare dei problemi di distribuzione del contenuto alle richieste più distanti geograficamente dal server, in quanto i pacchetti impiegano più tempo a raggiungere il dispositivo client e quindi causano ritardi nella riproduzione del video.
Inoltre per eventi particolarmente popolari come ad esempio eventi sportivi, il numero di richieste può aumentare notevolmente, causando un sovraccarico del server e quindi ritardi nella riproduzione del video, in quanto la banda disponibile è limitata e viene distribuita tra tutti i dispositivi client.
Per risolvere questi problemi si utilizzano dei servizi di Content Delivery Network (CDN), ovvero una rete di server presenti in tutto il mondo, che permette di distribuire i contenuti multimediali ai dispositivi client più vicini geograficamente, riducendo la latenza e il carico dei vari server.
Mentre per la gestione del carico del server, si utilizzano dei servizi di Load Balancing, ovvero dei servizi di bilanciamento di carico, come il bilanciamento di carico di rete (NLB) o il bilanciamento di carico applicativo (ALB), che permettono di distribuire il traffico più efficientemente tra i server, riducendone il carico.
Inoltre per gestire i picchi di richieste, si utilizzano infrastrutture server scalabili basate su cloud, come ad esempio Azure Media Services di Microsoft, che permette di scalare automaticamente le risorse in base alle esigenze del traffico di richieste, mantenendo il servizio sempre disponibile e riducendo i costi di gestione.

2.3 Tecnologie utilizzate

Per la realizzazione sono state utilizzate le seguenti tecnologie, divise in due categorie: tecnologie lato backend e tecnologie lato frontend.

2.3.1 Tecnologie backend

C#

C# è un linguaggio di programmazione orientato agli oggetti, sviluppato da Microsoft nel 2000, è un linguaggio di programmazione multiparadigma: supporta i paradigmi di programmazione procedurale, funzionale, generica, orientata agli oggetti e asincrona.
È un linguaggio di programmazione fortemente tipizzato, in quanto ogni variabile deve essere dichiarata con un tipo specifico, e supporta la programmazione generica, in quanto permette di creare classi e metodi generici, che possono essere utilizzati con tipi diversi.
È spesso utilizzato per lo sviluppo di applicazioni web, desktop e mobile, in quanto permette di creare applicazioni efficienti e affidabili.
Ha un ampio supporto per il framework .NET, il quale offre una vasta gamma di librerie

e strumenti per lo sviluppo di applicazioni backend e non solo, inoltre, il framework .NET include ASP.NET, un framework utilizzato per lo sviluppo di applicazioni web, che consente la creazione di API RESTful con facilità, grazie anche alla gestione avanzata delle richieste HTTP.

Entity Framework Core

Entity Framework Core è un ORM (Object Relational Mapper) open source, sviluppato da Microsoft, che permette di mappare le entità del database con le classi del codice, in modo tale da permettere agli sviluppatori di utilizzare le classi per interagire con il database, senza scrivere query SQL, offre una mappatura ORM che traduce in modo automatico le operazioni di lettura e scrittura degli oggetti nell'applicazione in istruzioni SQL eseguite sul database sottostante.
Le principali caratteristiche di Entity Framework Core sono:

- **Mappatura delle entità**: consente di definire classi che rappresentano le tabelle del database e di specificare le relazioni tra di esse in modo automatico;
- **Querying**: permette di eseguire query sul database utilizzando LINQ (Language Integrated Query), un'estensione del linguaggio C# che permette di scrivere query SQL in modo dichiarativo, ovvero scrivendo il risultato che si vuole ottenere e non come ottenerlo;
- **Migrazioni**: permette di creare delle migrazioni del database, ovvero consente di gestire le modifiche e l'aggiornamento automatico dello schema del database nel corso del tempo senza dover scrivere manualmente del codice SQL per ogni modifica;
- **Supporto per database multipli**: supporta una vasta gamma di database relazionali, inclusi SQL Server, MySQL, PostgreSQL, SQLite, ecc. permettendo di scrivere un unico codice che può funzionare con i diversi provider di database senza dover modificare il codice sorgente.

In conclusione, l'utilizzo del framework Entity Framework Core semplifica lo sviluppo di applicazione che utilizzano un database relazionale, in quanto permette di scrivere meno codice e di gestire in modo automatico le operazioni di lettura e scrittura sul database.

ASP.NET Core

ASP.NET Core è un framework open source, sviluppato da Microsoft, che permette di creare applicazioni web e API RESTful, utilizzando il linguaggio C#. Le caratteristiche principali includono:

- **Cross-platform**: permette di creare applicazioni web e API RESTful che sono compatibili con Windows, Linux e macOS;
- **Modularità**: è basato su un'architettura modulare che consente di includere solo i componenti necessari per l'applicazione, offrendo un'ampia flessibilità e consentendo di ridurre le dimensioni della distribuzione dell'applicazione;

- **Performance**: grazie alla sua architettura leggera e ottimizzata, offre prestazioni elevate e scalabilità;
- **Supporto per i protocolli HTTP**: permette di creare API RESTful che supportano i protocolli HTTP, come ad esempio HTTP/2 e WebSockets consentendo una comunicazione bidirezionale tra client e server;
- **Estensibilità**: offre un'ampia gamma di estensioni e librerie di terze parti che permettono di aggiungere funzionalità personalizzate all'applicazione.
- **Middleware**: utilizza il concetto di middleware per gestire le richieste HTTP. I middleware possono essere utilizzati per eseguire operazioni comuni come l'autenticazione, l'autorizzazione, la memorizzazione nella cache e la registrazione delle richieste, semplificando lo sviluppo delle applicazioni web.

Tusdotnet

Tusdotnet è una libreria open source per ASP.NET Core, che implementa il protocollo TUS, semplificando l'implementazione di esso fornendo un middleware che gestisce la logica di upload e il mantenimento di stato dell'upload. Grazie a esso è possibile gestire l'upload di file di grandi dimensioni in modo efficiente, in quanto permette di riprendere l'upload da dove si era interrotto, senza dover ricaricare l'intero file. Le principali caratteristiche di Tusdotnet sono:

- **Resumable upload**: permette di riprendere l'upload di un file da dove si era interrotto, senza dover ricaricare l'intero file;
- **Supporto per i file di grandi dimensioni**: permette di gestire l'upload di file di grandi dimensioni consentendo di superare i limiti imposti dalle API o dal server di hosting;
- **Supporto per i metadata**: permette di aggiungere dei metadata ai file caricati, in modo tale da poterli utilizzare per aggiungere informazioni aggiuntive al file;
- **Estensibilità**: offre un'architettura estensibile che permette di personalizzare ed estendere il comportamento dell'upload secondo le esigenze dell'applicazione.

Automapper

Automapper è una libreria open-source per .NET che semplifica la mappatura degli oggetti (object mapping) tra classi diverse. È progettato per ridurre il codice ripetitivo e la complessità associati alla mappatura degli oggetti, consentendo agli sviluppatori di gestire in modo efficiente la trasformazione dei dati da un tipo di oggetto a un altro. Le principali caratteristiche di Automapper sono:

- **Mappatura convenzionale**: Automapper supporta la mappatura convenzionale degli oggetti, il che significa che può associare automaticamente le proprietà degli oggetti sorgente alle proprietà corrispondenti degli oggetti di destinazione, a meno che non sia specificato diversamente.
- **Configurazione semplice**: offre un'API semplice e intuitiva per la configurazione delle mappe. È possibile definire le regole di mappatura utilizzando metodi fluenti o attributi personalizzati.

- **Mappatura bidirezionale**: supporta la mappatura bidirezionale, consentendo di trasformare un oggetto sorgente in un oggetto di destinazione e viceversa. Ciò semplifica l'aggiornamento dei dati dell'oggetto di destinazione in base alle modifiche apportate all'oggetto sorgente.
- **Supporto per relazioni complesse**: supporta la mappatura di relazioni complesse tra oggetti. È possibile gestire la mappatura di proprietà nidificate, collezioni di oggetti e relazioni uno-a-molti o molti-a-molti.

2.3.2 Tecnologie frontend

React

RReact è una libreria JavaScript per la creazione di interfacce utente. È stata sviluppata da Facebook e viene utilizzata per la creazione di UI per applicazioni web e mobile. React è basato su sei principi di progettazione:

- **Componenti**: Le componenti sono elementi dell'interfaccia utente che possono essere riutilizzate in diverse parti dell'applicazione, una componente può essere una piccola porzione di pagina o un elemento complesso e autonomo, consentono di creare interfacce utente modulari e riutilizzabili, in modo tale da rendere il codice più semplice da scrivere, leggere e mantenere.

- **State**: Lo stato è un oggetto JavaScript che contiene i dati che vengono utilizzati dai componenti dell'applicazione, è immutabile, quindi non può essere modificato direttamente; per modificarlo, è necessario utilizzare il metodo `setState()` che viene fornito da React; quando lo stato viene modificato, viene aggiornato automaticamente il DOM.

- **DOM**: Il DOM (Document Object Model) è una rappresentazione virtuale degli elementi della pagina, quando avvengono cambiamenti nello stato dell'applicazione, React aggiorna automaticamente il DOM in modo efficiente e successivamente aggiorna solo le parti della pagina che sono state modificate, così facendo React rende l'applicazione più veloce e reattiva senza dover ricaricare l'intera pagina.

- **Route**: Le route vengono utilizzate per gestire la navigazione e la visualizzazione delle diverse pagine dell'applicazione, consentono di definire le corrispondenze tra gli url specifici e i componenti che devono essere visualizzati quando viene richiesto un url specifico. Per gestire il routing, React utilizza una libreria esterna chiamata React Router, la quale fornisce diverse componenti che consentono di definire le route e di stabilire le corrispondenze tra gli url e i componenti.

- **Context**: Il Context è un meccanismo che consente di condividere dati specifici con tutti i componenti figli di un componente padre, evitando di dover passare manualmente le props attraverso i livelli intermedi, è composto da due parti: il Provider e il Consumer; il primo è responsabile di definire il contesto e di fornire i dati, mentre il secondo accede ai dati forniti dal Provider.

- **Props**: Le props (abbreviazione di proprietà) sono oggetti JavaScript immutabili che vengono utilizzati per passare dati da un componente padre a un componente figlio in modo unidirezionale. Il passaggio di dati tra la componente padre e la componente figlio avviene tramite gli attributi di quest'ultimo, mentre il passaggio di dati tra la componente figlio e la componente padre avviene tramite le funzioni callback.

React Player

React Player è una libreria open-source per React, un framework JavaScript per lo sviluppo di interfacce utente. React Player fornisce un componente React che semplifica l'integrazione di player multimediali come video e audio nelle applicazioni React, permette di incorporare facilmente file multimediali da varie fonti, gestirne la riproduzione e controllare il comportamento del player.
Le principali caratteristiche di React Player sono:

- Supporto per diversi formati video: supporta una vasta gamma di formati video, tra cui MP4, WebM, Ogg, FLV, HLS, DASH e YouTube;
- Controllo della riproduzione: permette di controllare la riproduzione del video, consentendo di riprodurre, mettere in pausa, saltare avanti e indietro, regolare il volume e il tempo di riproduzione;
- Eventi di riproduzione: permette di gestire gli eventi di riproduzione, consentendo di eseguire operazioni personalizzate quando avvengono determinati eventi, come ad esempio la riproduzione, la messa in pausa, il caricamento, l'errore ecc..;
- Personalizzazione: permette di personalizzare il player, consentendo di aggiungere controlli personalizzati, di personalizzare l'aspetto del player e di aggiungere funzionalità personalizzate.
- Gestione del buffering: permette di gestire il buffering del video, consentendo di visualizzare un indicatore di buffering personalizzato e di gestire gli eventi di buffering.

React Player semplifica notevolmente l'integrazione di file multimediali nelle applicazioni React, offrendo un'API semplice e intuitiva per il controllo della riproduzione. È una scelta popolare tra gli sviluppatori React per incorporare e gestire la riproduzione di video e audio in modo efficace e flessibile.

Hls.js

hls.js è una libreria JavaScript open-source che consente la riproduzione di file video utilizzando il protocollo di streaming HTTP Live Streaming (HLS), è un protocollo di streaming sviluppato da Apple per la distribuzione di contenuti multimediali su Internet.
hls.js è progettato per funzionare nel browser utilizzando la tecnologia HTML5 e JavaScript senza richiedere plugin aggiuntivi. Supporta il caricamento e la riproduzione di file video HLS in formato MPEG-TS (Transport Stream) suddivisi in segmenti, consentendo la riproduzione continua e adattiva dei contenuti video.
Le principali caratteristiche di hls.js sono:

- Riproduzione HLS: consente di riprodurre file video HLS direttamente nel browser utilizzando la tecnologia HTML5 e JavaScript, senza dipendere da plugin o tecnologie aggiuntive.
- Riproduzione adattiva: supporta lo streaming adattivo, che consente al lettore di selezionare automaticamente il giusto bitrate in base alla connessione di rete e alle capacità del dispositivo. Ciò assicura una riproduzione fluida senza interruzioni dovute a problemi di connessione.
- **Supporto per i file di segmenti MPEG-TS**: hls.js supporta il formato di file MPEG-TS, che è comunemente utilizzato con HLS. I file video sono suddivisi in segmenti che vengono scaricati e riprodotti in sequenza per garantire una riproduzione continua.
- **Caricamento dinamico dei segmenti**: hls.js carica in modo dinamico i segmenti video in base alla riproduzione corrente. Questo approccio consente di gestire file video di grandi dimensioni in modo efficiente, riducendo i tempi di buffering e consentendo una riproduzione fluida.
- **Controllo della riproduzione**: La libreria offre controlli per avviare, mettere in pausa, riprendere, avanzare e riavvolgere la riproduzione del video. È possibile anche gestire eventi come la fine del video o gli errori di riproduzione.

Hls.js è ampiamente utilizzato per la riproduzione di video HLS nel browser e offre un modo efficace e affidabile per gestire lo streaming di contenuti video attraverso HLS. Grazie alla sua natura open-source, la libreria è in costante sviluppo e supporta una vasta gamma di funzionalità per migliorare l'esperienza di riproduzione dei video HLS.

MUI

MUI (Material-UI) è una libreria open-source per la creazione di interfacce utente reattive e basate sul design dei Material UI di Google. MUI è basata sul framework React e offre un set di componenti React riutilizzabili, stili predefiniti e strumenti per la creazione di applicazioni web moderne e attraenti.
Le principali caratteristiche di MUI sono:

- **Componenti reattivi**: MUI fornisce una vasta gamma di componenti React, come pulsanti, modali, form, tabelle, navigazione e molto altro ancora. Questi componenti sono progettati per adattarsi e rispondere alle diverse dimensioni dello schermo, garantendo un'esperienza utente coerente su dispositivi desktop e mobili.
- **Design basato su Material UI**: MUI segue le linee guida di Material Design di Google, offrendo uno stile di design moderno e pulito. I componenti MUI presentano animazioni fluide, icone Material Design e un'estetica visuale coerente.
- **Personalizzazione e temi**: MUI consente di personalizzare i componenti e gli stili secondo le esigenze specifiche dell'applicazione. È possibile modificare i colori, le tipografie, gli spazi e altri aspetti dei componenti utilizzando i temi di MUI o sovrascrivendo le classi di stile.
- **Supporto per responsive design**: MUI facilita la creazione di layout reattivi e adattabili. È possibile definire comportamenti diversi dei componenti in base alle dimensioni dello schermo, migliorando l'esperienza utente su dispositivi di diverse dimensioni.

- **Gestione dello stato globale**: MUI offre supporto integrato per la gestione dello stato globale delle applicazioni utilizzando il contesto di React. Ciò consente di condividere dati tra i componenti senza dover passare manualmente le proprietà da un componente all'altro.
- **Strumenti per lo sviluppo**: MUI fornisce strumenti di sviluppo come MUI Icons, che offre una vasta gamma di icone Material Design, e MUI Lab, che contiene componenti sperimentali e di nuova generazione per estendere le funzionalità di MUI.

MUI è ampiamente utilizzata nella comunità di sviluppatori React per creare interfacce utente moderne e attraenti. Grazie alla sua flessibilità, personalizzazione e conformità con il design Material UI, MUI semplifica la creazione di applicazioni web di alta qualità e dall'aspetto professionale.

2.3.3 Tus.js

Tus.js è una libreria JavaScript open-source che implementa il protocollo TUS (Tus Upload Protocol) per la gestione di upload di file resumabili su server. Il protocollo TUS consente di caricare file di grandi dimensioni in modo affidabile e riprendere l'upload da dove si è interrotto, senza dover ricominciare da capo. Tus.js semplifica l'implementazione del protocollo TUS nel browser, fornendo una API per la gestione dell'upload dei file e la comunicazione con il server che supporta il protocollo TUS. Con Tus.js, è possibile gestire l'upload di file di grandi dimensioni in modo efficiente, garantendo la continuità dell'upload anche in caso di interruzioni di connessione o altri problemi. Le principali caratteristiche di Tus.js sono:

- **Gestione dell'upload resumabile**: Tus.js implementa il protocollo TUS, che consente di interrompere e riprendere l'upload dei file da dove si è interrotto, senza dover ricominciare da capo. Ciò rende l'upload di file di grandi dimensioni affidabile e consente agli utenti di riprendere l'upload da dove si sono interrotti.
- **Supporto per file di grandi dimensioni**: Tus.js può gestire file di grandi dimensioni suddividendoli in segmenti più piccoli e caricandoli in modo incrementale. Questo permette di caricare file di dimensioni significative senza sovraccaricare la memoria del browser.
- **Gestione delle interruzioni di connessione**: Tus.js è in grado di gestire interruzioni di connessione durante l'upload dei file. In caso di perdita di connessione o altri problemi di rete, Tus.js può riprendere l'upload dal punto esatto in cui si è interrotto una volta ripristinata la connessione.
- **Eventi di upload**: La libreria offre una serie di eventi che possono essere utilizzati per monitorare lo stato dell'upload, come l'avanzamento dell'upload, le interruzioni, i completamenti e gli errori. Questi eventi consentono di implementare logiche personalizzate in base allo stato dell'upload.
- **Personalizzazione**: Tus.js offre opzioni di personalizzazione per regolare il comportamento dell'upload, come la dimensione dei segmenti, i tempi limite e altro ancora. Questo consente di adattare l'upload alle esigenze specifiche dell'applicazione.

Tus.js è una scelta popolare per gli sviluppatori che necessitano di gestire l'upload di file di grandi dimensioni in modo affidabile e resumabile. Con Tus.js, è possibile semplificare l'implementazione di questa funzionalità nel browser, garantendo un'esperienza utente migliore durante l'upload dei file.

2.3.4 Tecnologie esterne

Azure

Azure è una piattaforma di cloud computing, sviluppata da Microsoft nel 2010, permette di creare, testare, distribuire e gestire applicazioni e servizi tramite i data center di Microsoft. Offre una ampia varieta di servizi, come servizi di elaborazione, servizi di archiviazione e database, servizi di rete, servizi di intelligenza artificiale e servizi di sicurezza.
Il vantaggio principale di Azure è la scalabilità, infatti permette di scalare le risorse di elaborazione e di rete in base alle esigenze dell'applicazione, inoltre permette di ridurre i costi di gestione, in quanto ci sono diversi piani di pagamento, che permettono di pagare solo le risorse utilizzate.

Azure Media Services

Azure Media Services è un servizio di cloud computing fornito da Microsoft Azure per la visualizzazione e la distribuzione di contenuti multimediali su Internet. È progettato per fornire un'infrastruttura scalabile e verificabile per la codifica, l'archiviazione, la protezione e la distribuzione di video, audio e altri contenuti multimediali in varie forme e dispositivi. Le principali caratteristiche di Azure Media Services sono:

- **Codifica**: consente di convertire i file multimediali in diversi formati e bit-rate per adattarsi a diverse piattaforme e velocità di connessione. Supporta la codifica e la transcodifica di video in formati come H.264, VP9, MPEG-2 e audio in formati come AAC, MP3, Dolby Digital.
- **Archiviazione**: permette di archiviare i file multimediali in modo sicuro e affidabile attraverso l'utilizzo di Azure Storage Account, un servizio di archiviazione di dati non strutturati come file, immagini e video; utilizza dei container per archiviare i file multimediali, che sono delle cartelle che contengono i file multimediali e i relativi metadati.
- **Distribuzione**: offre funzionalità di streaming per la distribuzione di contenuti multimediali in tempo reale o on-demand. Supporta lo streaming adattivo per garantire una riproduzione ottimale su diverse velocità di connessione e dispositivi.
- **Protezione dei contenuti**: include funzionalità di protezione dei contenuti per proteggere i file multimediali da copie non autorizzate e accessi non autorizzati. Supporta la crittografia dei contenuti, la gestione dei diritti digitali (DRM) e la firma URL per garantire la sicurezza dei contenuti.
- **Analisi dei contenuti**: offre funzionalità di analisi dei contenuti per estrarre informazioni utili dai file multimediali. Supporta la generazione di miniature, la trascrizione automatica, il riconoscimento facciale e la generazione di sottotitoli.

L'utilizzo di Azure Media Service offre vari vantaggi rispetto a un'infrastruttura di streaming video tradizionale, come la scalabilità, la sicurezza e la facilità di gestione. Grazie alla sua natura basata su cloud, Azure Media Services consente di ridurre i costi di gestione e di semplificare la distribuzione di contenuti multimediali su Internet.

Azure App Sevice

Azure App Service è un servizio di cloud computing offerto da Microsoft Azure che consente agli sviluppatori di creare, distribuire e gestire applicazioni web e mobili in modo rapido e scalabile. Azure App Service fornisce un ambiente di esecuzione completamente gestito per le applicazioni, eliminando la necessità di gestire l'infrastruttura sottostante.

Azure SQL Server

Azure SQL Server è un servizio di database relazionale completamente gestito che offre funzionalità di database SQL. Offre una serie di vantaggi rispetto a un Server SQL tradizionale, come la facilità di gestione, la scalabilità e sicurezza.
Il database non deve essere gestito in quanto Azure si occupa della gestione dell'infrastruttura, aggiornamenti delle patch di sicurezza e delle operazioni di manutenzione del server, consentendo agli sviluppatori di concentrarsi sulla logica di business e sull'applicazione dati.
Inoltre permette di scalare sia orizzontalmente che verticalmente il database, in maniera automatica, in modo tale da adattarsi alle esigenze dell'applicazione.
Un altro vantaggio è la sicurezza, in quanto integra funzionalità di sicurezza avanzate, come la crittografia dei dati in transito e a riposo, la protezione da minacce e la gestione degli accessi.
È stato utilizzato per la gestione dei dati dell'applicazione, integrandosi con il backend dell'applicazione attraverso il context contenuto nel layer Data.

2.3.5 Strumenti utilizzati

Visual Studio

Visual Studio è un IDE (Integrated Development Environment) sviluppato da Microsoft, permette di sviluppare applicazioni per Windows, Android, iOS e web, in modo efficiente.

Visual Studio Code

Visual Studio Code è un editor di codice sorgente sviluppato da Microsoft, permette di sviluppare applicazioni per Windows, Android, iOS e web, in modo efficiente.

Capitolo 3

Descrizione dello stage

In questo capitolo verrà descritto lo stage, paretendo dalla descrizione di esso, gli obbiettivi, la pianificazione, i risultati attesi ecc..

3.1 Obiettivi

Seguono gli obbietti identificati nel piano di lavoro.

3.1.1 Notazione

Gli obbiettivi sono cosi identificati:

- **O** per gli obbiettivi obbligatori;
- **D** per gli obbiettivi desiderabili;
- **F** per gli obbiettivi facoltativi.

3.1.2 Obbiettivi obbligatori

Obbiettivi obbligatori:

- **O1** Realizzazione del POC;
- **O2** Sviluppo Unit test backend;
- **O3** Analisi dei costi;
- **O4** Stesura della documentazione.

3.1.3 Obbiettivi desiderabili

Obbiettivi desiderabili:

- **D1** Analisi approfondita e comparazione di diverse architetture e sul loro impatto sui costi;
- **D2** Analisi dei competitor per osservare la gestione dei problemi di carico elevato.

3.1.4 Obbiettivi facoltativi

Obbiettivi facoltativi:

- **F1** Analisi dei sviluppi futuri;
- **F2** Realizzazione di un'interfaccia grafica, funzionale e conforme agli standard qualitativi di un'applicazione moderna.

3.2 Pianificazione

Durata in ore	Descrizione dell'attività
80	**Formazione sulle tecnologie**
30	*Studio delle basi dello stack tecnologico utilizzato dall'azienda e dei metodi di sviluppo*
50	*Studio e comparazione dei servizi offerti da Azure per realizzare la web app. Analisi di pro e contro dei vari servizi.*
180	**Sviluppo**
40	*Realizzazione della struttura di base della web app considerando le analisi di cui sopra.*
100	*Implementazione di front-end, back-end e DB*
80	*Test, ottimizzazione, refactoring. Realizzazione di unit test.*
40	**Documentazione e Demo**
25	*Collaudo*
10	*Stesura documentazione finale*
1	*Incontro di presentazione della piattaforma con gli stakeholders*
2	*Live demo di tutto il lavoro di stage*
Totale ore	**300**

Tabella 3.1: Tabella di ripartizione delle ore

3.3 Prodotti attesi

- **POC**
  Realizzazione di un POC funzionante che dimostri la fattibilità dell'idea proposta.
- **Unit test**
  Realizzazione completa degli unit test per il backend.
- **Analisi dei costi**
  Analisi dei costi simulando vari scenari di utilizzo della piattaforma.
- **Documentazione**
  Documentazione completa di tutte le fasi di sviluppo e analisi.
  1. Lettura e documentazione
     Documentare le varie opzioni disponibili per lo sviluppo di un'applicazione cloud, tecniche e tecnologie utilizzate e dei loro sviluppi futuri.

2. Sviluppo e analisi
   Preparazione dell'ambiente di sviluppo, sviluppo del POC e delle sue componenti.
3. Conclusioni
   Documentazione completa degli artefatti sviluppati e definizione dei possibili casi d'uso.

3.4 Risorse messe a disposizione

Per lo svolgimento dello stage sono state messe a disposizione le seguenti risorse:

- **Account Azure DevOps**
  Account Azure DevOps per la gestione del progetto, delle sue attività e del versionamento del codice.
- **Account Azure**
  Account Azure per la gestione delle risorse cloud.

3.5 Processo sviluppo prodotto

Il processo di sviluppo seguirà la pianificazione descritta nella sezione 3.2 e sarà suddiviso nelle seguenti fasi:

1. **Formazione**
   Studio delle tecnologie e dei metodi di sviluppo utilizzati dall'azienda.
2. **Analisi**
   Analisi delle tecnologie e dei servizi offerti da Azure per la realizzazione del POC.
3. **Sviluppo**
   Sviluppo del POC e delle sue componenti.
4. **Collaudo**
   Collaudo del POC e documentazione finale.

Capitolo 4

Progettazione

La progettazione del sistema è un passo fondamentale per lo sviluppo di un sistema software.
In questo capitolo verranno descritte le scelte progettuali effettuate per la progettazione del sistema, in particolare per lo sviluppo del database, il frontend, il backend e l'integrazione con Azure Media Services.

4.1 Progettazione del database

La progettazione del database è stata effettuata sulla base dei requisiti funzionali e non funzionali individuati in fase di analisi.
Il database è stato progettato per memorizzare i dati relativi agli eventi, video e utenti. Per la sua progettazione è stato utilizzato il framework Entity Framework Core, che permette di definire il modello del database utilizzando classi e proprietà direttamente dal codice. Non è stato necessario scrivere codice SQL per la creazione del database, ma è stato sufficiente definire le classi e le proprietà dei modelli e successivamente eseguire le migrazioni per aggiornare il database.

4.1.1 Modello del database

Il database è composto da tre tabelle: Event, Video e User.
La tabella Event contiene i dati relativi agli eventi, come l'id, il titolo, l'id dell'user che l'ha creato, la data di inizio e di fine e l'URL alla thumbnail; la tabella Video contiene i dati relativi ai video, come l'id, il titolo, l'URL alla thumbnail, l'URL al video e l'id dell'evento a cui appartiene; la tabella User contiene i dati relativi agli utenti, come l'id e il nome;

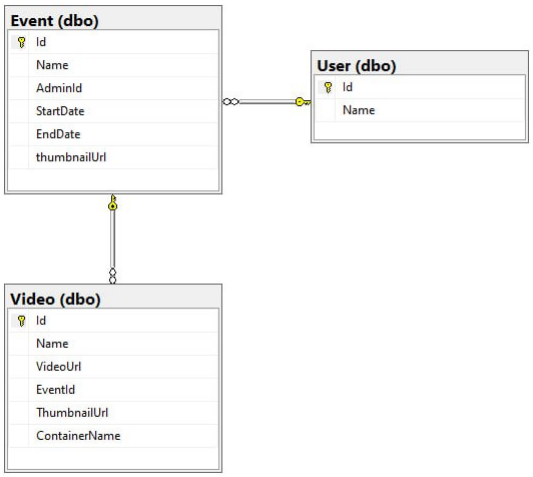


Figura 4.1: Diagramma del database

4.2 Progettazione del frontend

La progettazione del frontend è sviluppata sulla base dell'utilizzo di React come libreria principale per lo sviluppo dell'interfaccia utente.

4.2.1 Architettura del frontend

Il frontend è sviluppato utilizzando un template disposto dall'azienda, che utilizza il design pattern Container-Presenter, è composto da due componenti principali: il Container e il Presenter. Il primo è responsabile della gestione dello stato dell'applicazione, dell'interazione con i dati e della logica di business, si occupa di recuperare i dati, gestire gli eventi, effettuare chiamate API e gestire lo stato globale dell'applicazione; il secondo invece, è responsabile dell'aspetto visuale e dell'interfaccia utente, riceve i dati e le funzioni dai Container e si occupa di renderizzare l'interfaccia utente in base ai dati ricevuti.
La comunicazione tra i due avviene tramite le props, in quanto il Container passa i dati al Presenter tramite props, mentre il Presenter invia le informazioni tramite callback fornite dal Container.

4.2.2 Diagrammi dell'architettura del frontend

Diagramma dell'architettura

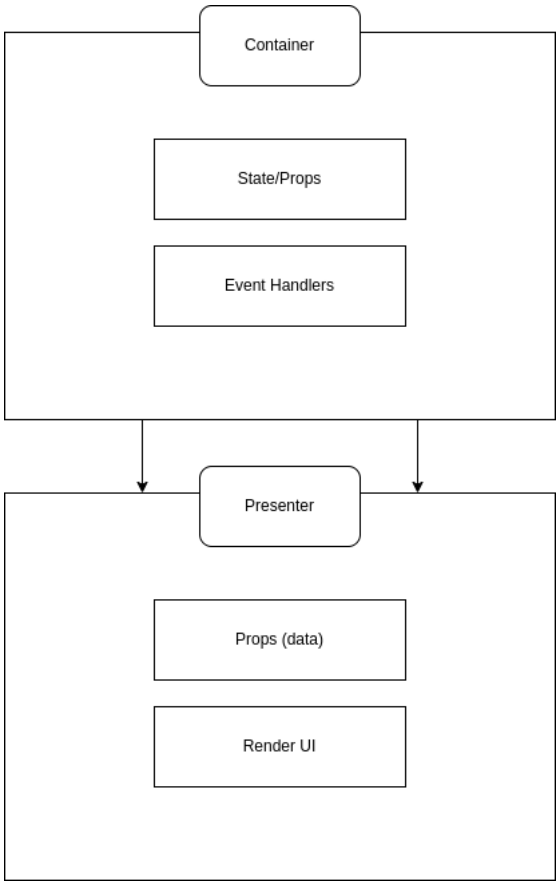


Figura 4.2: Diagramma design pattern Container-Presenter

4.3 Progettazione del backend

La progettazione del backend è sviluppata sulla base dell'utilizzo del linguaggio di programmazione C# e del framework ASP.NET Core.

4.3.1 Architettura del backend

Il backend è sviluppato utilizzando un template disposto dall'azienda, che utilizza una separazione delle componenti in layer distinti, dove ognuno ha un compito specifico. Il template è composto da tre layer fondamentali: API, Core e Data; il primo è responsabile della comunicazione con il frontend, il secondo è responsabile della gestione dello stato dell'applicazione e il terzo è responsabile della comunicazione con il database.

API

È responsabile della comunicazione con il frontend, è composto da due parti: i controller e i DTO. Il primo ha il compito di gestire le richieste HTTP provenienti dal frontend e coordinare le azioni richieste per soddisfarle: quando un controller riceve una richiesta, estrae i dati necessari dalla richiesta e interagisce con i layer Core per fornire una risposta al Client. Il secondo ha il compito di definire la struttura dei dati che vengono trasferiti tra il frontend e il backend durante la chiamata, in modo da consentire una comunicazione standardizzata e senza ambiguità. I DTO possono includere solo i campi necessari per soddisfare una certa richiesta, così facendo, riducono il trasferimento di dati inutili e rendono la comunicazione più veloce; oltre a ciò, sono utilizzati anche per la validazione dei dati, garantendo che i dati ricevuti dal frontend siano validi.

Core

Il layer Core è responsabile della gestione dello stato dell'applicazione. Riceve le richieste dal layer API e le elabora, interagendo con il layer Data per ottenere i dati necessari; è composto da due parti: i models e i service. I models rappresentano la struttura dei dati che vengono utilizzati dall'applicazione per effettuare le operazioni richieste; i service, invece, gestiscono la logica dell'applicazione, contengono i metodi che vengono chiamati dai controller, che eseguono operazioni con i servizi esterni e con il layer Data.

Data

Il layer Data è responsabile dell'accesso ai dati, è composto da tre parti: Context, Entity e Provider. Il primo ha il compito di gestire la connessione con il database, definisce la struttura del database e fornisce i metodi per accedervi; il secondo ha il compito di definire la struttura dei dati che vengono salvati nel database; il terzo ha il compito di gestire la comunicazione con il database, fornisce i metodi per accedere ai dati per effettuare le operazioni di lettura e scrittura.

4.3.2 Diagrammi dell'architettura del backend

Diagramma dell'architettura

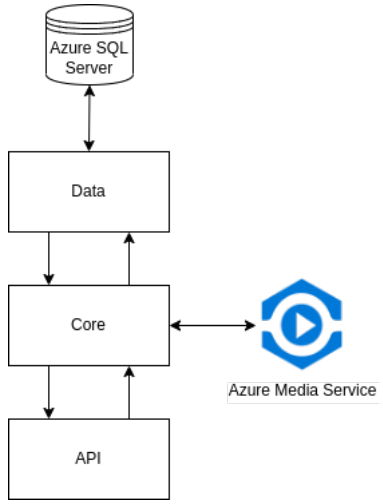


Figura 4.3: Diagramma architettura del backend

4.4 Integrazione con Azure

Azure è una piattaforma cloud proprietaria di Microsoft che offre vari servizi. La WebApp è integrata con Azure per la gestione del database, gestione dei video e per la distribuzione dell'applicazione, sono utilizzati i seguenti servizi: SQL Server, Azure Media Service e Azure App Service.

4.4.1 Azure SQL Server

Il progetto integra un database SQL ospitato su Azure SQL Server, che permette di gestire il database in maniera semplice e veloce, senza dover gestire l'infrastruttura. Il collegamento al database è gestito dal layer Data del backend, che utilizza Entity Framework Core per la gestione del database.

4.4.2 Azure Media Service

Il progetto utilizza Azure Media Service per la gestione dei video, in particolare per la codifica, l'archiviazione e la distribuzione dei video. Il collegamento ad Azure Media Service è gestito dal layer Core del backend, che utilizza il package NuGet Microsoft.Azure.Management.Media per la gestione dei video.

Codifica

Per lo sviluppo di questo PoC è stato deciso di codificare i video in H.264 e utilizzando il preset Adaptive Streaming, che permette di codificare il video in diversi formati e

risoluzioni, in modo da adattarsi alla qualità della rete e al dispositivo utilizzato. Sono state scelte queste impostazioni che permettono di ottenere un video di risoluzione massima pari a 1080p.

Archiviazione

Quando un video termina la codifica, viene archiviato in un container di Azure Media Service, che permette il salvataggio dei vari file video, della thumbnail, dei metadati e del manifest.

Distribuzione

Una volta archiviato, il video viene distribuito tramite Streaming Endpoint, che creano un Url al manifest del video e all'immagine di thumbnail, che vengono poi salvati nel database tramite apposite API.

Azure App Service

Sia il modulo FrontEnd che il modulo del Backend vengono distribuiti attraverso Azure App Service, vengono inizializzatiti tramite l'apposita funzione di Visual Studio 2022.

Capitolo 5

Implementazione

L'implementazione del PoC è una parte fondamentale del progetto di stage, in quanto permette di verificare la fattibilità del prodotto e di valutare la bontà delle scelte progettuali.
In questo capitolo verranno descritte le principali scelte implementative, sia per quanto riguarda il frontend che il backend.

5.1 Struttura del progetto

La struttura del progetto è stata organizzata in modo da separare il frontend dal backend, in modo da poterli sviluppare in modo indipendente. Inoltre, è stato scelto di utilizzare una repository su Azure DevOps per il versioning del codice, in modo da poter avere una storicizzazione delle modifiche.

5.2 Backend

Il backend è stato sviluppato utilizzando il linguaggio di programmazione C# e il framework ASP.NET Core, è basato sul template fornito dall'azienda che prevede l'utilizzo di un'architettura a layer.
Come ambiente di sviluppo è stato utilizzato Visual Studio 2022.

5.2.1 Architettura a layer

L'architettura a layer è una tipologia di architettura software che prevede la suddivisione del codice in diversi livelli, ognuno dei quali ha un compito ben preciso, e che comunica con gli altri livelli attraverso interfacce. Questo permette di avere un codice più modulare e mantenibile, in quanto ogni livello ha un compito ben preciso e non si occupa di altro.
Il backend è composto da un unico progetto "AdMaioraStreamingPOC", che contiene i vari layer suddivisi in cartelle, ognuna delle quali contiene le soluzioni relative al layer. I layer sono: API, Core e Data.

API

Il layer API è il livello più esterno dell'applicazione, si occupa di gestire le richieste HTTP in arrivo dal frontend. È divisa in due soluzioni: Controller e DTO.

- **Controllers**:sono le classi che si occupano di gestire le richieste HTTP in arrivo dal frontend, utilizzano il framework ASP.NET Core e la libreria AutoMapper per la conversione dei DTO in entità e viceversa; inoltre, utilizzano la dependency injection per iniettare le dipendenze necessarie dalle classi di Service presenti nel layer Core, l'uso di questa tecnica rende i controller indipendenti dalle implementazioni concrete delle dipendenze.
  Per ogni entità del database è stato creato un controller, che contiene metodi CRUD e non, per gestire le richieste HTTP in modo completo. Quando un controller riceve una richiesta HTTP, utilizza i metodi presenti nel Service per gestire la richiesta, e restituisce al frontend il risultato della richiesta in formato JSON.

- **DTO**: sono classi che contengono la struttura dei dati che vengono scambiati tra il frontend e il backend. Per ogni entità del database sono stati creati tre DTO: uno per la creazione, uno per l'aggiornamento e uno per la visualizzazione; così facendo, si evita di scambiare dati inutili tra frontend e backend, e si evita di dover gestire la serializzazione e deserializzazione di dati inutili. Inoltre, utilizzando i DTO, si evita di dover esporre le entità del database al frontend, favorendo la separazione tra frontend e backend.

Core

Il layer Core è il livello intermedio dell'applicazione, e si occupa di gestire la logica di business dell'applicazione. È diviso in due soluzioni: Model e Services.

- **Model**: sono classi che rappresentano le entità nel database, il loro scopo è quello di mappare le entità del database in classi C#, in modo da poterle utilizzare all'interno della logica del codice. Per ogni entità del database è stato creato un Model, che contiene le proprietà dell'entità.

- **Services**: sono classi che si occupano di gestire la logica di business dell'applicazione, utilizzano i Model per accedere ai dati nel database, e li restituiscono al layer API. Per ogni entità del database è stato creato un Service, che contiene metodi CRUD e non, per gestire la logica di business dell'applicazione. Fornisce i metodi al layer API attraverso interfacce, e a sua volta, vengono iniettate le dipendenze necessarie del Provider presente nel layer Data, le credenziali per l'accesso ad Azure e della configurazione per il server TUS.io. Il suo scopo principale è quello di fornire un'interfaccia tra il layer API e il layer Data. Inoltre si occupa anche della comunicazione con i servizi esterni, infatti gestisce l'upload dei video su Azure Media Services, tramite le sue API dedicate; comunica con l'istanza TUS.io.

Data

Il layer Data è il livello più interno dell'applicazione, e si occupa di gestire l'accesso ai dati nel database. È diviso in tre soluzioni: Context, Providers e Entity.

- **Context**: è la classe che si occupa di gestire l'accesso ai dati nel database, che quando viene inizializzato, gli viene iniettata la configurazione per il collegamento e definisce tre proprietà di tipo DbSet, una per ogni entità del database, che consentono di effettuare operazione di query e manipolazione dei dati.

- **Providers**: sono classi che si occupano di gestire l'accesso ai dati nel database, utilizzando il Context per accedervi, e li restituiscono al layer Core convertendoli in Model tramite AutoMapper. È stata creata una classe per ogni entità del database, e contengono i metodi CRUD per gestire l'accesso ai dati nel database. Lo scopo principale dei providers è quindi quello di fornire un'interfaccia tra il layer Core e il database.

- **Entity**: sono classi che rappresentano le entità nel database, il loro scopo è quello di mappare le entità del database in classi C#, in modo da poterle utilizzare all'interno della logica del codice. Per ogni entità del database è stato creato un Entity, che contiene le proprietà dell'entità. Queste classi sono utilizzate dal Context per accedere ai dati nel database. Oltre a ciò, vengono utilizzate per definire le proprietà, tramite il framework Entity Framework Core e il suo sistema di migrazioni: quando viene modificata una proprietà di un Entity, viene eseguita una migrazione che modifica la struttura del database in base alle modifiche apportate.

5.3 Frontend

Il frontend è stato sviluppato utilizzando il framework React e la libreria grafica Material-UI.
Il frontend è diviso in tre cartelle principali: Components, Pages e Services, utilizzando il template fornito dall'azienda che prevede l'utilizzo del template Container-Presenter.
È stato utilizzato Visual Studio code come ambiente di sviluppo, e Node.js come runtime per l'esecuzione del codice.

5.3.1 Architettura

L'architettura del frontend è divisa in tre cartelle: Pages, Common e Context.

Pages

La cartella Pages contiene le pagine dell'applicazione, ovvero le pagine che vengono visualizzate dall'utente. Ogni pagina è divisa a sua volta in tre componenti: Context, Page e Components.

- **Context**: svolge il ruolo di fornire un meccanismo per l'accesso e la condivisione dei dati tra i componenti, in modo da evitare di dover passare i dati da un componente all'altro. Per ogni pagina è stato creato un Context, che contiene un provider che contiene lo stato e le funzioni per l'accesso ai dati; ogni componente avvolto dal provider può accedere ai dati e alle funzioni per modificarli. L'accesso ai dati avviene tramite l'utilizzo del context di FetchContext, che contiene le route per effettuare le chiamate al backend.

- **Page**: Ogni pagina contiene un componente PageContent, che renderizza i componenti della pagina. Successivamente essi vengono avvolti dal Provider della pagina specifica che permette l'accesso al Context della pagina, in modo tale che possano accedere ai dati e alle funzioni per modificarli.

- **Components**: sono le porzioni di codice che vengono renderizzate all'interno della pagina.

Common

La cartella Common contiene i componenti che vengono utilizzati in tutte le pagine, è composto da tre file:

- **Layout**:definisce la struttura di ogni pagina, composta a sinistra da un menù laterale, sopra dal titolo della pagina corrente e al centro dal contenuto della pagina passato come props. Il menù laterale è composto da una lista di MenuItems, che vengono renderizzati grazie all'utilizzo del context di NavContext.

- **Route**: permette di navigare tra le pagine in base al percorso specificato contenuto nel MenuItems.

- **DialogCustom**: è un componente che permette di visualizzare un dialog, che contiene un titolo, un contenuto e due bottoni, uno per confermare e uno per annullare. Viene utilizzato in tutte le pagine per confermare le azioni che possono modificare i dati nel database. Il contenuto viene passato tramite props, in modo da poterlo personalizzare in base alla pagina. L'utilizzo di questo componente permette di evitare di dover creare un dialog pressoché identico in tutte le pagine.

Context

La cartella Context, contiene i file di contesto, è composta da tre file: FetchContext, NavContext e ToastContext.

- **FetchContext**: contiene le route per effettuare le chiamate al backend, è composto da una funzione "Compile Route" che permette di comporre la route in base ai parametri passati, e da una funzione "Fetch" che permette di effettuare la chiamata al backend. Questo permette di evitare di dover ripetere le route in ogni componente.

- **NavContext**: contiene le route per effettuare la navigazione tra le pagine, è composto da vari metodi per per la navigazione, per la gestione del menù laterale, per la gestione del titolo della pagina e per la gestione del breadcrumb.

- **ToastContext**: contiene le funzioni per mostrare i toast, composto da quattro funzioni: "Success", "Error", "Warning" e "Info", che permettono di mostrare un toast con il messaggio passato come parametro.

5.3.2 Interfaccia grafica

Pagina Home

La pagina Home è la pagina principale dell'applicazione, contiene una grid di tutti gli eventi disponibili, e un menù laterale che permette la navigazione tra le pagine.

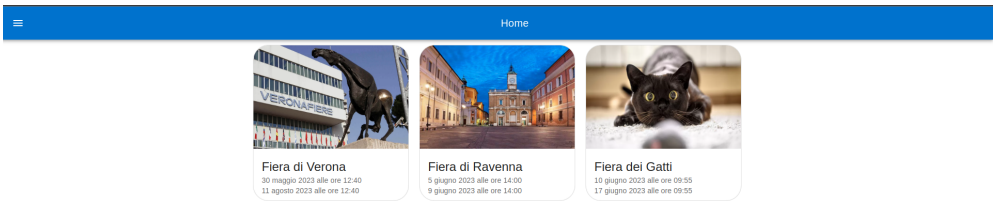


Figura 5.1: Pagina Home

Pagina Eventi

Quando si clicca su un evento nella pagina Home, si viene reindirizzati alla pagina dell'evento, che contiene una grid con tutti i video dell'evento.

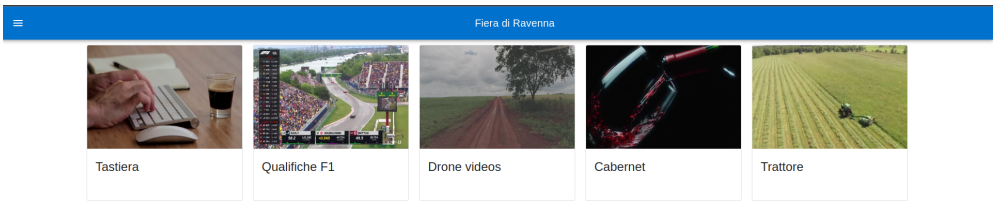


Figura 5.2: Pagina Eventi

Pagina Video

Quando si clicca su un video nella pagina Eventi, si viene reindirizzati alla pagina del video, che contiene il player del video e i pulsanti per la gestione.

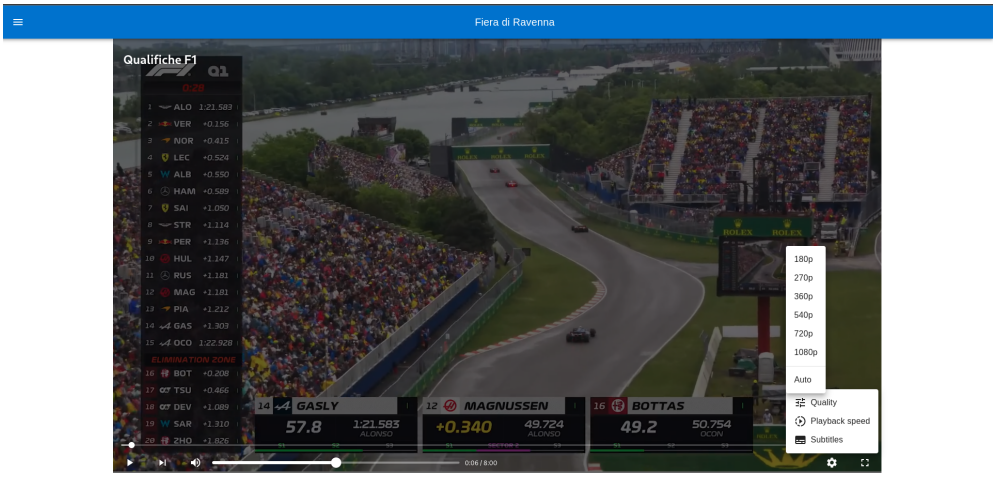


Figura 5.3: Pagina Video

Pagina Gestione

Le pagine di gestione sono tre, una per gli eventi, una per i video e una per gli utenti. Ogni pagina contiene una tabella con tutti gli elementi, ogni riga contiene due pulsanti, uno per modificare l'elemento e uno per eliminarlo. Inoltre è presente anche un pulsante per aggiungere un nuovo elemento attraverso un dialog.

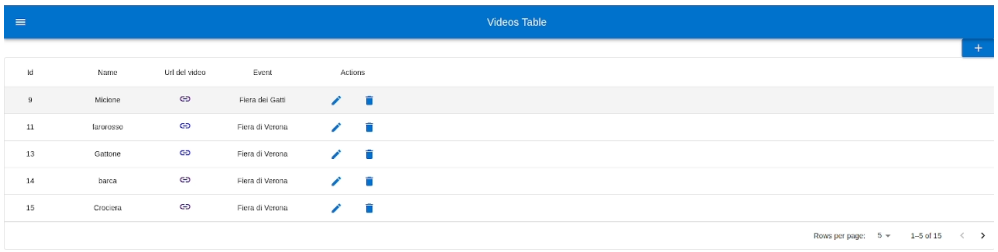


Figura 5.4: Pagina Gestione

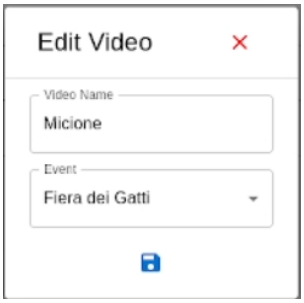


Figura 5.5: Pagina Gestione

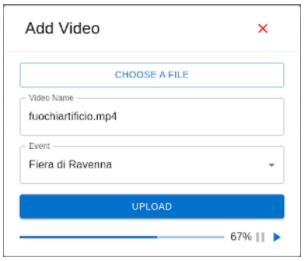


Figura 5.6: Pagina Gestione

5.4 Integrazione con Azure

Per l'integrazione con i servizi di Azure, sono state utilizzate le API ufficiali fornite da Microsoft, che permettono di gestire i servizi di Azure in modo semplice e veloce.
La connessione al database è stata gestita mediante l'utilizzo del framework Entity Framework Core; quando il programma viene inizializzato, nel backend al livello del provider viene iniettata la configurazione per il collegamento al database, che contiene le credenziali per l'accesso al database. Per quanto riguarda il collegamento con Azure Media Service, e vengono utilizzate nel backend nel layer Service per comunicare con le API proprietarie.

5.5 Integrazione con TUS.io

Per la gestione dell'upload dei video dal frontend al backend, è stato utilizzato il servizio TUS.io, che permette di gestire l'upload di file di grandi dimensioni e di gestirne lo stato. Quando viene inizializzato il programma, viene creata un'istanza del server TUS.io, che permette al frontend tramite le API di accedere ad esso e di gestire l'upload dei file. Quando un file viene caricato sul server TUS.io, il frontend invia una chiamata al backend con l'id TUS del file caricato e poi viene passato ai metodi del Service, che si occupa di recuperare il file e successivamente di caricarlo su Azure Media Services.

Capitolo 6

Testing e validazione

In questo capitolo verranno descritte le attività di testing e validazione svolte sul POC.

6.1 Descrizione delle attività di testing

Per quanto riguarda le attività di testing, sono state richieste dall'azienda solo quelle riguardanti al modulo del backend.
Le attività di testing sono state svolte seguendo le best practice fornite da Microsoft, che prevedono la nomenclatura dei test secondo il seguente schema:
NomeMetodo_Scenario_RisultatoAtteso in questo modo è possibile capire subito, cosa si sta testando e quale risultato ci si aspetta senza dover leggere il codice del test.
Inoltre, per ogni metodo testato, è stata utilizzata la metodologia AAA, che prevede la suddivisione del test in tre parti:

- **Arrange**: in questa fase si prepara lo stato iniziale necessario per eseguire il test. Può includere la creazione di oggetti, l'inizializzazione delle variabili, l'impostazione di condizioni specifiche, l'istanziazione delle dipendenze o l'impostazione di un ambiente controllato. L'obiettivo dell'Arrange è creare l'ambiente di test coerente e riproducibile.
- **Act**: in questa fase si esegue l'azione o il comportamento che si desidera testare. Può essere l'invocazione di un metodo, l'interazione con un'interfaccia utente, l'invio di richieste a un server, ecc. L'obiettivo dell'Act è stimolare il sistema in modo che si verifichi l'evento che si vuole testare.
- **Assert**: in questa fase si verifica se il comportamento del sistema durante l'azione corrisponde a quello previsto. Si stabiliscono delle asserzioni, cioè si specificano le condizioni che devono essere verificate per considerare il test superato. Se le asserzioni hanno successo, il test viene considerato valido, altrimenti viene segnalato un errore. L'obiettivo dell'Assert è controllare che l'output o lo stato del sistema sia conforme alle aspettative.

6.1.1 Unit testing

I test di unità, sono stati svolti utilizzando il framework NUnit, che permette di eseguire test di unità per applicazioni .NET.

Per eseguire i test di unità, è stato necessario creare un progetto di test, che permettesse di testare le funzionalità del progetto di backend. Per fare ciò, è stato necessario aggiungere il progetto di backend come riferimento al progetto di test. In questo modo, è stato possibile testare le funzionalità del progetto di backend, utilizzando i metodi pubblici delle classi del progetto di backend.
Il progetto di test è diviso in tre cartelle, una per ogni layer del progetto di backend. Ogni cartella contiene i test delle classi del layer corrispondente.

Provider

Per effettuare i test sul layer dei provider, è stato necessario creare un mock del database utilizzato dal progetto di backend. Per fare ciò, Entity Framework mette a disposizione un package NuGet chiamato *EntityFrameworkCore.InMemory*, che permette di creare un database in memoria, ovvero viene creato un database temporaneo in memoria, che viene popolato con dei dati fittizi che vengono utilizzati per i test. In questo modo, è possibile testare le funzionalità del layer dei provider, senza dover utilizzare un database reale. Inoltre viene passato al costruttore del provider, un oggetto di tipo *DbContextOptions*, che permette di testare i metodi del provider, utilizzando il database in memoria.

Service

Per effettuare i test sul layer Core, è stato necessario anche qui creare un mock del database come nel layer dei provider, successivamente è stata instanziata la classe del Provider che si vuole testare, passando al costruttore un oggetto di tipo *DbContextOptions*, che permette di utilizzare i metodi del provider, utilizzando il database in memoria.In seguito è stata instanziata la classe del Service utilizzando come costruttori i provider precedentemente creati.

Controller

I test sul layer dei controller non sono stati svolti in quanto non potendo effettuare il mock dei servizi esterni, non è stato possibile instanziare le classi Services necessarie per testare i controller.

6.2 Analisi dei risultati

Le attività di testing sono state svolte in maniera parziale in quanto sia per mancanza di tempo che per mancanza di conoscenze, non è stato possibile testare tutte le funzionalità del backend, in particolare i controller e i metodi dei Service che utilizzano i servizi esterni. Sicuramente in futuro, sarà necessario effettuare dei test anche su queste funzionalità, in modo da garantire un prodotto di qualità.

Capitolo 7

Analisi dei costi

In questo capitolo verrà descritta l'analisi dei costi effettuata sul progetto.

7.1 Introduzione

L'analisi dei costi è un'attività fondamentale per la realizzazione di un progetto, in quanto permette di stimare il costo totale del progetto e di conseguenza il budget necessario per la sua realizzazione e mantenimento.
Verranno descritti i principali costi di mantenimento della WebApp, in particolare verranno analizzati i costi di hosting, di distribuzione e caricamento dei video, che essendo basata sull'utilizzo del infrastruttura di Azure, sono i costi principali.

7.1.1 Costi di hosting

I costi di hosting sono i costi necessari per mantenere la WebApp online, in particolare sono i costi per l'hosting del backend e del frontend. Durante lo sviluppo del PoC, è stato utilizzato un piano di distribuzione gratuito, che permette di distribuire l'applicazione senza costi aggiuntivi con delle limitazioni sulle operazioni che possono essere effettuate. In caso di produzione dell'applicazione, è necessario considerare dei costi per l'hosting, in quanto il piano gratuito non è adatto per l'hosting di un'applicazione in produzione. Azure offre diversi piani di hosting, che differiscono per le risorse disponibili e per il costo. Il piano piu adatto per una WebApp di questo tipo in produzione è il piano Premium, che offre un numero illimitato di App, 250GB di spazio di archiviazione, fino a 30 istanze e la possibilità di utilizzare un dominio personalizzato e l'autoscale ad un costo di 0.184€/h, ovvero 4.416€/giorno.

7.1.2 Costi di codifica

I costi di codifica sono i costi necessari per effettuare la codifica dei video caricati nei vari formati, sono riportati nella seguente tabella //TODO: link alla tabella. Per lo sviluppo del PoC è stata utilizzata la codifica in HD, idealmente per l'app in produzione si utilizzerà la codifica in 4K visto l'evoluzione della disponibilità di rete e dell'aumento delle risoluzioni dei dispositivi sempre più accessibili.

7.1.3 Costi di trasferimento

Azure Media Service prevede un costo fisso per ogni GB trasferito tra le stesse zone di distribuzione, ovvero che se il file richiesto si trova in Europa e anche il richiedente avrà un costo di 0.010€/GB mentre se si trovano in zone differenti i costi aumentano come riportato nella seguente tabella //TODO: link alla tabella

7.1.4 Costi di distribuzione

I costi di distribuzione sono i costi necessari per la distribuzione in streaming dei contenuti multimediali. Durante lo sviluppo del PoC è stato utilizzato lo Standard Streaming Endpoint, che ha un Bandwidth di 600Mbps a un costo di 1.9088€/giorno, ovvero 59.172€/mese, che è il piano più basso disponibile, permette una distribuzione a circa 171 utenti contemporaneamente con un utilizzo medio di 3.5Mbps per utente. È limitante in quanto, una volta saturato il Bandwidth, non permette la scalabilità automatica, che causerebbe buffering ai client, e bisognerebbe passare ad un piano superiore.

7.2 Consuntivo preventivato

In caso di produzione dell'applicazione, è necessario stimare il numero di utenti che utilizzeranno l'applicazione per poter scegliere i piani più adatti e avere una stima dei costi circa accurata.
È stato realizzato un modello matematico che stima in base ai dati messi in input come numero di utenti iscritti, percentuale di utenti attivi giornalmente, durata media di visione per utente, orari e ampiezza del picco, qualità video richiesta, il costo medio giornaliero dell'applicazione, che prevede i costi di codifica, trasferimento e distribuzione, vengono riportati tre scenari. Per congruenza dei dati, ogni scenario prevede gli stessi dati in input tranne che per il numero di iscritti:

Oggetto	Valore
Percentuale utenti iscritti	10%
Durata media visualizzazioni in secondi	300s
Percentuale richieste HD	50%
Percentuale richieste FHD	50%
Banda media HD	4Mbps
Banda media FHD	7Mbps
Costo per singolo Premium Unit Streaming	0.17€
Costo per GB trasferito	0.010€

Tabella 7.1: Tabella di dati in input

Caso 10000 utenti iscritti

Per il primo scenario, si è scelto di considerare 10000 utenti iscritti, che è un numero ragionevole per una WebApp di questo tipo all'inizio della sua vita.

Oggetto	Valore
Costo App Service Giornaliero	4.416€
GB usati giornalmente	201 GB
Costo di trasferimento	2.014€
Costo totale di Premium Unit Streaming	4.17€
Totale giornaliero	10.60€
Totale mensile	328.45€
Totale annuale	3.867.23€

Tabella 7.2: Tabella costi con 10000 utenti iscritti

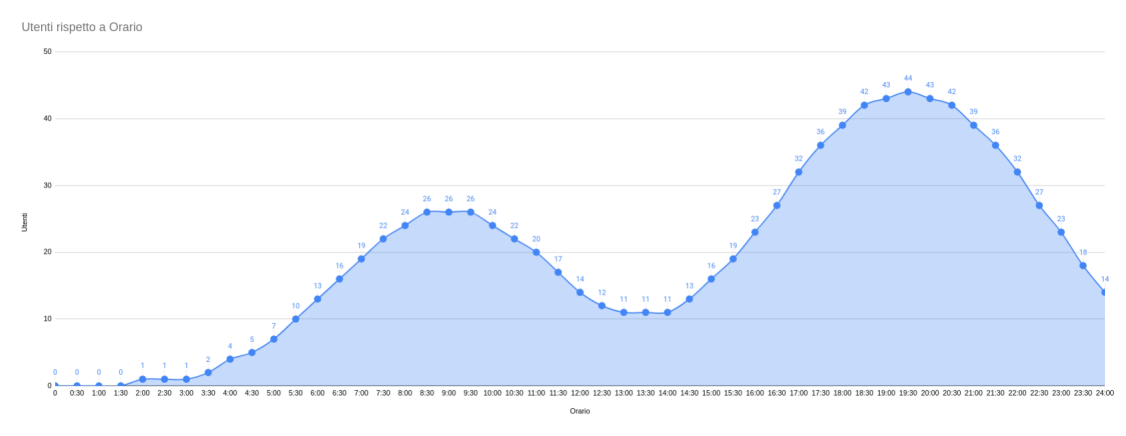


Figura 7.1: Grafico utenti/orario con 10000 utenti iscritti

Caso 100000 utenti iscritti

Per il secondo scenario, si è scelto di considerare 100000 utenti iscritti, che è un buon numero di utenti per una WebApp di questo tipo in caso iniziasse a crescere.

Oggetto	Valore
Costo App Service Giornaliero	4.416€
GB usati giornalmente	2014 GB
Costo di trasferimento	20.142€
Costo totale di Premium Unit Streaming	7.02€
Totale giornaliero	31.57€
Totale mensile	978.80€
Totale annuale	11524.64€

Tabella 7.3: Tabella costi con 100000 utenti iscritti

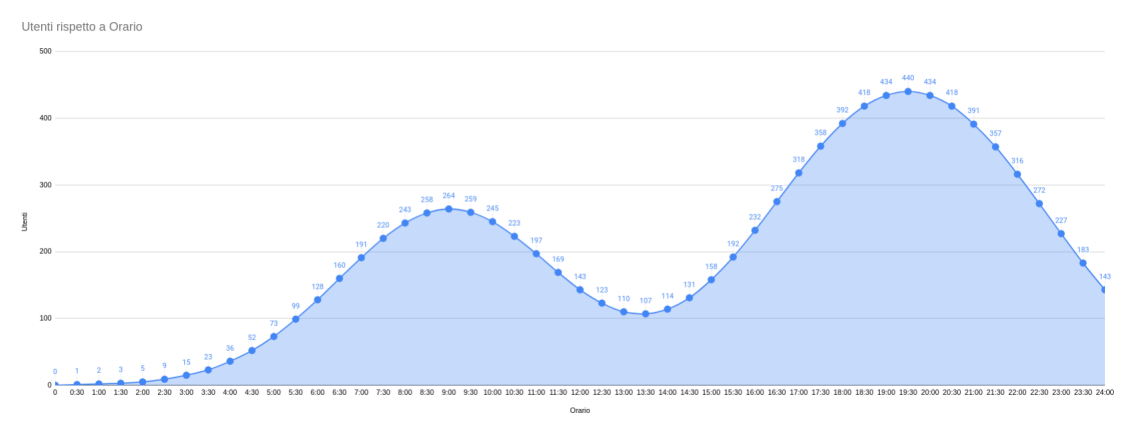


Figura 7.2: Grafico utenti/orario con 100000 utenti iscritti

Caso 1000000 utenti iscritti

Per il terzo scenario, si è scelto di considerare 1000000 utenti iscritti, che è un numero importante di utenti in caso la WebApp avesse un grande successo.

Oggetto	Valore
Costo App Service Giornaliero	4.416€
GB usati giornalmente	20142 GB
Costo di trasferimento	201.416€
Costo totale di Premium Unit Streaming	11.13€
Totale giornaliero	216.96€
Totale mensile	6725.85€
Totale annuale	79191.41€

Tabella 7.4: Tabella costi con 1000000 utenti iscritti

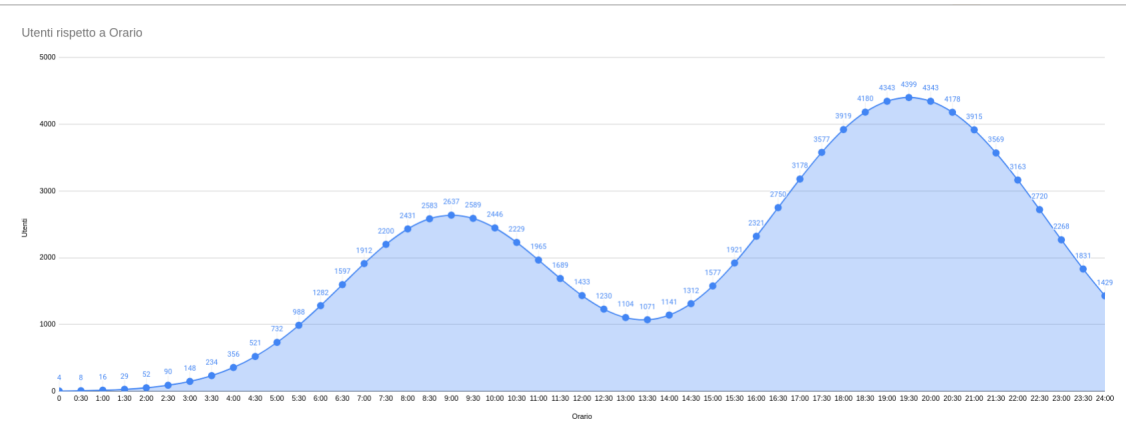


Figura 7.3: Grafico utenti/orario con 1000000 utenti iscritti

Capitolo 8

Conclusioni

8.1 Raggiungimento degli obiettivi

La realizzazione del POC ha permesso di raggiungere la maggiore parte degli obiettivi prefissati, in particolare lo studio per la fattibilità del progetto, la realizzazione di un prototipo funzionante e la realizzazione di un'architettura scalabile.

8.2 Conoscenze acquisite

8.3 Valutazione personale

Appendice A

Appendice A

Citazione

Autore della citazione

Bibliografia