

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

| | | |
|----------|--|-----------|
| 1 | Introduzione | 1 |
| 1.1 | L'idea | 1 |
| 1.2 | Descrizione dello stage | 1 |
| 1.3 | L'azienda | 2 |
| 2 | Fondamenti teorici e tecnologie utilizzate | 3 |
| 2.1 | Concetti di video on-demand e streaming | 3 |
| 2.1.1 | Tipologie di video on-demand | 3 |
| 2.1.2 | Concetti di streaming video | 4 |
| 2.1.3 | Protocolli e tecnologie di streaming video | 4 |
| 2.2 | I problemi dello streaming | 5 |
| 2.2.1 | Latenza | 5 |
| 2.2.2 | Qualità del video | 5 |
| 2.2.3 | Sicurezza | 5 |
| 2.2.4 | Gestione del carico del server | 6 |
| 2.3 | Tecnologie utilizzate | 6 |
| 2.3.1 | React | 6 |
| 2.3.2 | C# | 7 |
| 2.3.3 | Azure | 7 |
| 3 | Analisi dei requisiti | 8 |
| 3.1 | Individuazione e specifica dei requisiti funzionali e non funzionali | 8 |
| 3.1.1 | Requisiti funzionali | 8 |
| 3.1.2 | Requisiti non funzionali | 9 |
| 3.2 | Casi d'uso | 9 |
| 4 | Progettazione | 10 |
| 4.1 | Progettazione del database | 10 |
| 4.1.1 | Modello del database | 11 |
| 4.2 | Progettazione del frontend | 11 |
| 4.2.1 | Principi di progettazione di React | 11 |
| 4.2.2 | Architettura del frontend | 12 |
| 4.2.3 | Diagrammi dell'architettura del frontend | 13 |
| 4.3 | Progettazione del backend | 14 |
| 4.3.1 | Architettura del backend | 14 |
| 4.3.2 | Diagrammi dell'architettura del backend | 15 |
| 4.4 | Integrazione con Azure | 15 |
| 4.4.1 | Azure SQL Server | 15 |

| | | |
|----------|---|-----------|
| 4.4.2 | Azure Media Service | 16 |
| 4.4.3 | Azure App Service | 16 |
| 5 | Implementazione | 18 |
| 5.1 | Struttura del progetto | 18 |
| 5.2 | Backend | 18 |
| 5.2.1 | Architettura a layer | 18 |
| 5.3 | Frontend | 20 |
| 5.3.1 | Architettura | 20 |
| 5.4 | Integrazione con Azure | 22 |
| 5.5 | Integrazione con TUS.io | 22 |
| 6 | Testing e validazione | 23 |
| 6.1 | Descrizione delle attività di testing | 23 |
| 6.1.1 | Unit testing | 24 |
| 6.2 | Analisi dei risultati | 24 |
| 7 | Conclusioni | 25 |
| 7.1 | Consuntivo finale | 25 |
| 7.2 | Raggiungimento degli obiettivi | 25 |
| 7.3 | Conoscenze acquisite | 25 |
| 7.4 | Valutazione personale | 25 |
| A | Appendice A | 26 |
| | Bibliografia | 28 |

Elenco delle figure

| | | |
|-----|--|--------------------|
| 4.1 | Diagramma del database | 11 |
| 4.2 | Diagramma design pattern Container-Presenter | 13 |
| 4.3 | Diagramma architettura del backend | 15 |

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 e Account Storage. ²

L'obiettivo 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 connessioni, 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:

2.3.1 React

React è una libreria JavaScript open source, sviluppata da Meta nel 2013, permette la creazione di componenti riutilizzabili e la gestione dello stato dell'applicazione in modo efficiente. Scompone l'interfaccia utente in piccoli componenti modulari, ciascuno responsabile di una specifica parte dell'interfaccia, garantendo una maggior facilità di sviluppo e manutenibilità del codice.

Una caratteristica distintiva di React è la sua efficacia di rendering, infatti, grazie al virtual DOM (Dynamic Object Model), permette di aggiornare solo le parti dell'interfaccia che vengono modificate, senza dover ricaricare l'intera pagina, garantendo una maggiore efficienza e velocità di rendering.

È spesso utilizzato insieme ad altre librerie come React Router, che permette di gestire le rotte dell'applicazione.

2.3.2 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.

2.3.3 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 varietà 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.

Capitolo 3

Analisi dei requisiti

In questo capitolo verranno descritti i requisiti individuati per lo sviluppo del POC.

3.1 Individuazione e specifica dei requisiti funzionali e non funzionali

I requisiti individuati sono stati classificati in base alla loro tipologia, funzionali o non funzionali, e in base alla loro priorità, obbligatori, desiderabili o opzionali.

I requisiti funzionali sono stati individuati in base alle funzionalità che il sistema deve offrire, mentre i requisiti non funzionali sono stati individuati in base alle caratteristiche che il sistema deve avere.

I requisiti obbligatori sono quelli che il sistema deve assolutamente soddisfare, i requisiti desiderabili sono quelli che il sistema dovrebbe soddisfare, mentre i requisiti opzionali sono quelli che il sistema potrebbe soddisfare.

3.1.1 Requisiti funzionali

- **RF1:** l'utente deve poter creare un nuovo evento;
- **RF2:** l'utente deve poter modificare un evento esistente;
- **RF3:** l'utente deve poter eliminare un evento esistente;
- **RF4:** l'utente deve poter visualizzare la tabella degli eventi;
- **RF5:** l'utente deve poter caricare un video e associarlo ad un evento;
- **RF6:** l'utente deve poter visualizzare la tabella dei video;
- **RF7:** l'utente deve poter modificare un video esistente;
- **RF8:** l'utente deve poter eliminare un video esistente;
- **RF9:** l'utente deve poter visualizzare la lista dei video associati ad un evento;
- **RF10:** l'utente deve poter aggiungere un utente al sistema;

- **RF11:** l'utente deve poter modificare un utente esistente;
- **RF12:** l'utente deve poter eliminare un utente esistente;

3.1.2 Requisiti non funzionali

3.2 Casi d'uso

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;

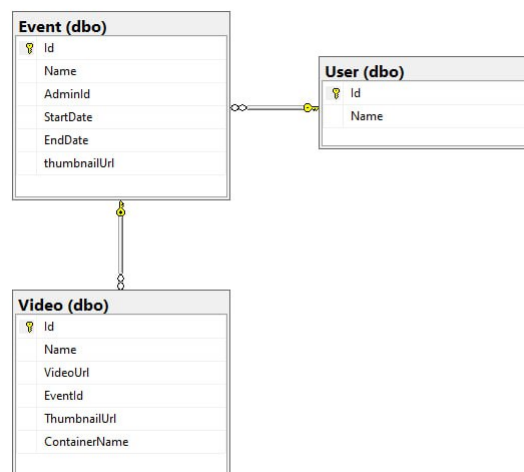


Figura 4.1: Diagramma del database

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 Principi di progettazione di React

React 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, State, DOM, Route, Context e Props.

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.

4.2.2 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.

4.2.3 Diagrammi dell'architettura del frontend

Diagramma dell'architettura

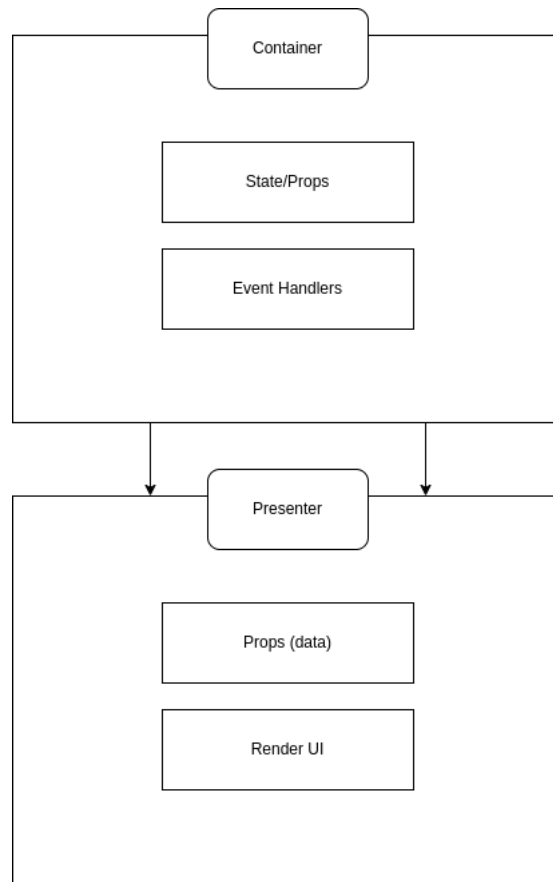


Figura 4.2: Diagramma design pattern Container-Presenter

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

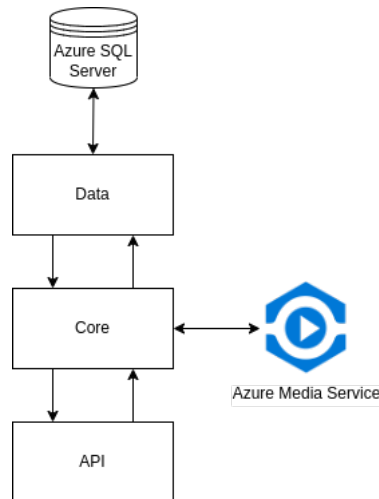


Figura 4.3: Diagramma architettura del backend

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

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.

4.4.2 Azure Media Service

Azure Media Service è un servizio che permette la gestione e la distribuzione di contenuti multimediali su diverse piattaforme e dispositivi.

Offre varie funzionalità, ma per gli scopi di questa applicazione è stato utilizzato per la codifica, archiviazione dei video e la distribuzione dei video.

Codifica

La codifica è un processo che permette di convertire un video in un formato compatibile con la maggior parte dei dispositivi e delle piattaforme.

Azure Media Service permette di codificare i video in diversi formati e risoluzioni, in modo da rendere disponibile la visione del video su qualsiasi dispositivo e in qualsiasi condizione di rete.

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

Una volta codificato, il video viene salvato in Azure Storage Account, un servizio di archiviazione di dati non strutturati come file, immagini e video.

Ogni video viene salvato in un container, che è una cartella che contiene i video codificati in varie risoluzioni, il manifest e la thumbnail generata automaticamente da Azure Media Service.

Distribuzione

Una volta archiviato, il video viene distribuito attraverso degli Streaming Endpoint, che consentono di creare dei punti di accesso per lo streaming di contenuti multimediali in diretta o on-demand, offre la possibilità di distribuire i video attraverso vari protocolli di streaming; per questo PoC è stato utilizzato il protocollo HLS.

L'utilizzo di Streaming Endpoint offre vari vantaggi, il più importante è la scalabilità, in quanto permette di scalare automaticamente il numero di istanze in base al numero di richieste, in modo da garantire fluidità e continuità dello streaming senza interruzioni. La distribuzione avviene attraverso un URL, generato da Azure Media Service, che punta al manifest del video, che viene salvato nel database dell'applicazione insieme agli altri dati del video.

4.4.3 Azure App Service

È un servizio di Azure che permette di distribuire e gestire applicazioni web e API senza dover gestire l'infrastruttura.

Il deploy dell'applicazione avviene attraverso l'interfaccia di Visual Studio 2022, che permette di pubblicare l'applicazione direttamente su Azure App Service.

Permette di scalare automaticamente il numero di istanze in base al numero di richieste,

in modo da garantire fluidità e continuità dell'applicazione senza interruzioni. Inoltre permette di gestire il dominio dell'applicazione, in modo da poter utilizzare un dominio personalizzato.

In generale, l'integrazione con Azure permette di ridurre i costi di gestione e di manutenzione dell'infrastruttura, in quanto è tutto gestito in automatico, permettendo di concentrarsi sullo sviluppo dell'applicazione più che sulla gestione dell'infrastruttura.

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: I 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: I 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: I 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: I 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: Il 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: I 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: Le 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: Il 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: I 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 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.

5.4 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'istanziamento 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 istanziata 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 istanziata 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 istanziare 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

Conclusioni

7.1 Consuntivo finale

7.2 Raggiungimento degli obiettivi

7.3 Conoscenze acquisite

7.4 Valutazione personale

Appendice A

Appendice A

Citazione

Autore della citazione

Bibliografia