UNIVERSITÀ DEGLI STUDI
DI MODENA E REGGIO EMILIA

Dipartimento di Scienze Fisiche, Informatiche e Matematiche

Personalizzazione di Contenuti Audiovisivi Attraverso Dati da Social Network

Alessandro Bedini

Tesi di Laurea

Relatore:
Prof. Riccardo Martoglia

Anno Accademico 2019/2020

RINGRAZIAMENTI

Ringrazio l'Ing. Riccardo Martoglia per la continua disponibilità e assistenza.
Un ringraziamento speciale va alla mia famiglia, alla mia morosa e ai miei amici che in questi anni mi hanno sempre sostenuto e supportato.
Ringrazio, inoltre, il gruppo di lavoro che mi ha seguito durante il tirocinio.

PAROLE CHIAVE

Video Personalizzati

Angular

TypeScript

Amazon Web Services

Facebook

Indice

Elenco delle figure

Introduzione

In un mondo sempre più mobile, social e multicanale, cambiano le esigenze ed i comportamenti degli utenti, mentre si creano nuove opportunità di interazione e relazione con i brand. La Customer Experience (CX) è indubbiamente oggi un elemento cardine del business, dal quale dipende il successo delle aziende.

A supporto ci sono fortunatamente piattaforme e tecnologie digitali sempre più innovative, in grado di garantire risultati tangibili [1].

Le aspettative dei clienti sono sempre più elevate: ognuno di loro ha desideri, aspettative ed esigenze che vanno costantemente ascoltate e risolte per rendere ogni cliente il tuo migliore cliente, offrendogli i prodotti ed i servizi che desidera davvero. Non coinvolgere i clienti in maniera adeguata significa metterne a rischio la fiducia e perdere l'opportunità di costruire relazioni di valore essenziali per la crescita del proprio business.

Uno dei maggiori strumenti di interazione con l'utente è sicuramente quello del video personalizzato. Permette di offrire un contenuto piacevole e capace di stimolare curiosità da parte del destinatario, che, al contrario di una e-mail contenente solo informazioni, potrebbe decidere di prenderne visione più volentieri.

Un discorso molto simile avviene allo stesso tempo sui social network, dove le persone condividono continuamente contenuti di vario genere ma, come testimonia Cisco nel suo Annual Internet Report [2], i video continueranno a costituire una gran parte del traffico internet generato.

Per questo motivo Doxee, un'azienda modenese che eroga servizi di Customer Communication Management, Digital Archiving, Conservazione Sostitutiva a norma di legge e che ha reso possibile la realizzazione di questo progetto, mira a integrare uno dei suoi prodotti per la personalizzazione video con uno strumento altrettanto importante come quello dei social network.

La tesi seguente è strutturata in quattro capitoli: i primi due sono di carattere introduttivo, i restanti due, invece, entrano nel dettaglio del progetto.
Nel capitolo 1 sarà possibile comprendere a pieno l'obiettivo di questo studio e del perché di alcune scelte di base.
Il capitolo 2 offrirà un'introduzione a tutte le tecnologie utilizzate dando una piccola motivazione della scelta, che verrà spiegata approfonditamente nei capitoli successivi.
Nel capitolo 3 viene analizzata la parte di progettazione: la suddivisione del lavoro in storie implementative e la definizione dell'architettura.
Il capitolo 4 descriverà la parte di sviluppo, spiegando e mostrando le funzioni principali del codice, scendendo nella parte più pratica del progetto.

Parte I

Il caso di studio

Capitolo 1

Analisi dei requisiti

In questo capitolo viene descritto l'obiettivo finale del progetto, l'approccio utilizzato per portarlo a termine e alcune decisioni prese a discapito di altre.

1.1 Obiettivo

Non vi è alcun dubbio quanto la capacità di dare ai clienti un'esperienza d'uso di un prodotto o servizio che si sollevi dal coro passi per l'utilizzo di video come strumento di interazione. Sono un mezzo di comunicazione polivalente e pertinente a tutti i livelli, hanno, inoltre, il potere di incrementare la popolarità di un sito internet e incoraggiano l'interattività e l'engagement dell'utente con tale sito [3].
Un'uguale, se non maggiore, importanza l'assumono i social network, in cui le aziende, grandi o piccole che siano, devono essere presenti. Essi rappresentano un canale di comunicazione fondamentale e offrono opportunità di crescita e visibilità, permettono di velocizzare la diffusione delle informazioni e di migliorare il posizionamento del proprio sito internet mettendo le aziende in contatto con migliaia di potenziali clienti aiutandole a costruire relazioni [4]. Uno dei principali strumenti dei social network è senza alcun dubbio il social sharing. Questo fenomeno ci porta ad essere continuamente coinvolti nella fruizione di contenuti multimediali di qualsiasi tipo, realizzati sia da soggetti privati che da aziende. I contenuti di qualità che spingono la condivisione sono l'obiettivo di strategie di content marketing di molte aziende che intendono ottenere maggiore visibilità e ampliare la propria community [5].

Ed è proprio a questo proposito che il progetto della tesi consiste nella realizzazione di un **proof of concept** per la personalizzazione di contenuto audiovisivo in tempo reale, con dati provenienti da social login e con possibilità di condivisione.

1.2 Metodi adottati

Negli ultimi anni l'utilizzo dei social network è diventato parte integrante delle nostre giornate, è possibile trovarne di tutti i generi, da quelli destinati alla condivisione di sole foto o video, a quelli in cui è possibile condividere alcuni pensieri, e altri che includono entrambe le cose. Molti di questi mettono a disposizione degli strumenti chiamati API, Application Programming Interface, che permettono un'interazione facilitata tra programmi e web service. Facebook, uno dei social network tra i più eterogenei in circolazione per quanto riguarda la tipologia di contenuto che è possibile trovare, come foto, video e semplici frasi fino ad un servizio di messaggistica privata, mette a disposizione gratuitamente le proprie API, permettendo a privati e aziende di usarle per accedere ai dati disponibili in modo programmatico.
È stato quindi scelto di sviluppare un'applicazione web che permetta la creazione di un video personalizzato, integrandola col social login di Facebook per garantire agli utenti l'accesso mediante il proprio account e inserire le foto presenti sul loro profilo. Al termine del processo viene data la possibilità di condividere il proprio risultato tramite il social media sopracitato o tramite e-mail.

Capitolo 2

Tecnologie utilizzate

In questo capitolo vengono introdotte le principali tecnologie utilizzate e verrà spiegato in breve il perché di queste scelte. Si vedrà, inoltre, il modo in cui verranno ottenuti i dati degli utenti provenienti da Facebook.

2.1 Angular

Angular è un framework open-source utilizzato per lo sviluppo di applicazioni web con licenza MIT e sviluppato principalmente da Google [6]. Utilizza TypeScript come linguaggio di programmazione, a differenza del suo predecessore AngularJS che utilizza JavaScript. Uno dei motivi principali per cui è stato scelto come framework per la realizzazione della web app è la sua velocità, dovuta al fatto che permette una maggiore elaborazione lato client facendo eseguire le applicazioni interamente dal browser dell'utente dopo essere state scaricate dal web server. Questo permette di evitare un continuo scambio di informazioni tra client e server. Inoltre, il codice generato da Angular è cross platform [7] e supportato da tutti i principali browser moderni.

2.2 TypeScript

Come accennato prima, per il linguaggio di programmazione del front-end è stato utilizzato TypeScript, un linguaggio open-source sviluppato da Microsoft che compila in JavaScript permettendo anche la definizione statica del tipo delle

variabili [8]. Nasce a causa del crescente bisogno di un linguaggio front-end per applicazioni JavaScript su larga scala e dalla necessità di sicurezza e robustezza [9].

2.3 Amazon Web Services

Amazon Web Services, noto come AWS, è la piattaforma cloud più completa ed utilizzata del mondo. Offre più di 200 servizi completi da data center a livello globale [10] tra i quali è possibile trovare infrastrutture per il calcolo, l'archiviazione e i database fino alle nuove tecnologie come machine learning, intelligenza artificiale, data lake, analytics e Internet delle cose. AWS è progettato per essere un ambiente di cloud computing flessibile, sicuro e veloce.
Di seguito verranno descritti i servizi offerti da AWS che sono stati utilizzati per la realizzazione del progetto.

2.3.1 CloudFront

CloudFront, uno dei servizi che mette a disposizione AWS, è una rete per la distribuzione rapida di contenuti o CDN (Content Delivery Network). Permette la distribuzione di contenuti a livello globale agli utenti con latenza minima, una velocità di trasferimento elevata, il tutto offrendo funzionalità di sicurezza avanzate quali la crittografia a livello di campo e il supporto HTTPS [11]. La rete CloudFront ha oltre 225 punti di presenza (PoPs) interconnessi mediante la dorsale AWS, una rete privata basata su una rete parallela in fibra metropolitana, globale e completamente ridondante di 100GbE. Questo servizio mappa automaticamente le condizioni della rete e instrada in modo intelligente il traffico dell'utente verso la edge location AWS più performante. È stato scelto come CDN per l'applicazione web perché facile e intuitivo da utilizzare, inoltre è già integrato con altri servizi offerti da Amazon Web Services necessari allo scopo del progetto.

2.3.2 Simple Storage Service

Amazon Simple Storage Service (Amazon S3) è un servizio di storage di oggetti che offre scalabilità, disponibilità dei dati, sicurezza e prestazioni all'avanguardia [12]. Permette l'archiviazione e la protezione di una qualsiasi quantità di dati per una vasta gamma di casi d'uso. Dispone di caratteristiche di gestione semplici da utilizzare che consentono di organizzare i dati e di configurare controlli di accesso ottimizzati, utili, come nel nostro caso, per vietare l'accesso ai contenuti se non tramite CloudFront. S3 è progettato per garantire la durabilità dei dati creando e memorizzando automaticamente copie di tutti gli oggetti su più sistemi, rendendo i dati disponibili in qualsiasi momento e proteggendoli da guasti, errori e minacce. Abbiamo quindi usato questo servizio di AWS come storage per i file statici dell'applicazione web, in modo da poter essere prelevati da CloudFront e inviati come risposta agli utenti che li richiedono.

2.3.3 Lambda

AWS Lambda è un servizio di elaborazione serverless che permette di eseguire codice automaticamente senza dover effettuare il provisioning né gestire un'infrastruttura [13]. È sufficiente scrivere il codice nel linguaggio che si preferisce e caricarlo in Lambda come file ZIP. Molto utile per gestire richieste API dal Web o creare propri servizi di back-end. Il codice inserito prende il nome di "funzione lambda" e una volta creata è sempre pronta per essere attivata. Le funzioni sono inoltre stateless, prive di affinità con l'infrastruttura sottostante, quindi in grado di essere eseguite più di una alla volta in modo parallelo.

2.3.4 API Gateway

Amazon API Gateway è un servizio completamente gestito che semplifica la creazione, la manutenzione, il monitoraggio e la protezione delle API. Fungono da

“porta di entrata” per consentire l’accesso delle applicazioni ai dati, alla logica aziendale o alle funzionalità dei servizi back-end.
Offre due tipi di API:

- *API RESTful*: API HTTP per carichi di lavoro serverless e backend http e API REST per funzionalità proxy e di gestione delle API riunite in un’unica soluzione
- *API WEBSOCKET*: Per applicazioni di comunicazione bidirezionale in tempo reale.

Per il nostro progetto si sono rese necessarie come tramite tra il front-end e il back-end.

2.4 Doxee Pvideo

Pvideo è un prodotto Doxee, azienda modenese orientata allo sviluppo di servizi IaaS, PaaS, SaaS, dedicato alla creazione e alla distribuzione di video personalizzati ed interattivi [14]. Consente la creazione di video unici, composti da scene selezionate in base ai dati di ogni singolo destinatario, in cui è possibile inserire testi e banner personalizzati, immagini, una voce personalizzata grazie al text-to-speech e alla libreria audio. Grazie alla funzionalità User-Directed-Storytelling l’utente finale potrà scegliere il percorso di narrazione e di navigazione di suo gradimento, l’interattività, infatti, è una delle principali caratteristiche di Doxee Pvideo, la quale contribuisce a renderlo uno strumento di comunicazione bi-direzionale.

2.5 SendGrid

SendGrid è un servizio di distribuzione e-mail cloud-based di vario tipo [15], include e-mail di notifica spedizione, richieste d’amicizia, conferma di registrazione, email di newsletter. Offre anche, se richiesto, un servizio di tracking che monitora se le e-mail vengono aperte, rimbalzate o inserite tra le e-mail di spam.

Pensando agli sviluppatori mette a disposizione una sua API da poter integrare nel proprio codice, cosa che si è rivelata molto utile all'interno del progetto.

2.6 Facebook API Graph

L'API Graph rappresenta il metodo principale tramite cui le applicazioni possono interagire con i vari contenuti di Facebook. Si basa sul protocollo HTTP e può essere usata per effettuare query sui dati, pubblicare nuove storie, gestire le pubblicità, caricare foto e molte altre operazioni [16].
Prende il nome dall'idea che Facebook sia un grafico rappresentante tutte le informazioni contenute al suo interno e composto da:

- *node:* Gli oggetti individuali, come un utente una foto, una pagina o un commento.
- *edge:* Le connessioni tra i gruppi di oggetti e l'oggetto singolo.
- *field:* I dati che contiene un oggetto, come la data di compleanno di un utente o il nome di una pagina.

2.6.1 Facebook App

Per potere utilizzare questa API, occorre essere in possesso di un account Facebook da sviluppatore tramite il quale creare un'applicazione Facebook con la quale verrà identificata sul social media la tua app, che sia per il web o per smartphone. Le applicazioni dispongono di un app id che deve essere utilizzato per importare il Source Development Kit di Facebook all'interno del codice dell'applicativo. Una volta importato sarà possibile utilizzare l'API Graph.

2.6.2 Facebook Login

L'accesso alle informazioni dell'API Graph è consentito solo se in possesso di un token d'accesso:

- Permette di accedere alle informazioni sull'utente senza la richiesta della relativa password.
- Permette a Facebook di identificare l'app, l'utente che la sta utilizzando e il tipo di dati per cui l'utente ha concesso l'autorizzazione di accesso.

Quasi tutti gli endpoint dell'API Graph richiedono un token d'accesso conforme al protocollo OAuth2.0, il quale consente alle entità come utenti e pagine di autorizzare i token.

Come è possibile vedere nella figura 2.1, quando un utente effettua l'accesso ad un'applicazione tramite l'interfaccia di Facebook gli verrà chiesto se continuare e garantire l'accesso alle proprie informazioni personali necessarie.

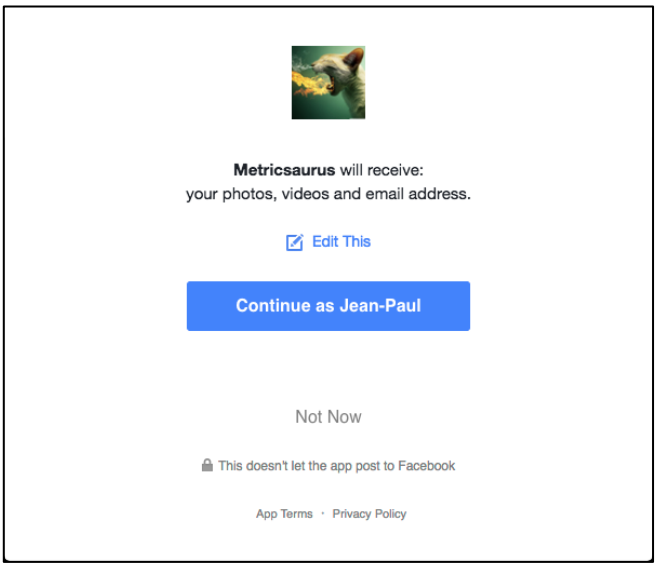

Figura 2.1 – Richiesta accesso ad informazioni dell'utente

https://developers.facebook.com/docs/graph-api/using-graph-api/

In caso l'utente acconsenta a continuare il token dell'app verrà sostituito col token dell'utente utilizzabile per accedere alle sue informazioni.

Per semplificare agli sviluppatori i vari reindirizzamenti di OAuth2.0, Facebook mette a disposizione un prodotto chiamato Facebook Login che prende a carico tutta le gestione dei token. Permette inoltre di creare account e accedere velocemente alla tua applicazione da diverse piattaforme.

2.6.3 Lettura

Le operazioni di lettura iniziano quasi sempre con un *node*. I nodi sono singoli oggetti con ID unici e per leggere i dati al suo interno occorre effettuare una query rispetto al suo specifico ID. */me* è un endpoint speciale che si traduce nell'*user_id* dell'utente il cui token d'accesso viene attualmente usato per effettuare le chiamate API, come abbiamo visto nel capitolo 2.6.2. Quindi, per leggere il nodo Utente che è attualmente connesso all'applicazione occorre effettuare una query del tipo:

Query effettuata in JavaScript

```
FB.api(
  '/me',
  'GET',
  {"fields":"id,name"},
  function(response) {
      // Insert your code here
  }
);
```

Questa richiesta esegue *function* con la variabile *response* contenente i campi utente inseriti nel valore *fields* e ritornati nel formato JSON per impostazione predefinita:

```
{
  "name": "Your Name",
  "id": "your-user-id"
}
```

I nodi presentano anche dei segmenti, che consistono in raccolte di altri nodi a essi collegati. Per leggerne uno occorre effettuare una query su un endpoint che parte da un ID di un nodo interessato e appendere ad esso il segmento che si desidera leggere. Per leggere, ad esempio, tutti i post di un utente occorrerà effettuare una chiamata API all'endpoint */me/feed* che, se in possesso dell'autorizzazione per gli *user_posts* ritornerà:

```
{
  "data": [
    { "created_time": "2017-12-08T01:08:57+0000",
      "message": "Love this puzzle. One of my favorite puzzles",
      "id": "post-id"},
    { "created_time": "2017-12-07T20:06:14+0000",
      "message": "You need to add grape as a flavor.",
      "id": "post-id"}
    ]
}
```

Parte II

Risoluzione

Capitolo 3

Progettazione

Nei capitoli precedenti si è visto quali tecnologie sono state necessarie alla realizzazione del progetto. Nel seguente, invece, verrà spiegato come tali strumenti saranno utilizzati e sarà possibile osservare la suddivisione del lavoro per il raggiungimento dell'obiettivo.

3.1 User Story

Le User Story, in ambito di sviluppo del software e di product management, sono un modo informale di descrivere le funzionalità di un software mediante un linguaggio naturale e comprensibile a chiunque [17]. Sono, infatti, descritte dalla prospettiva dell'utente finale o di un utente che dovrà effettivamente utilizzare il software e permettono al team di sviluppo di avere una sorta di documentazione e di comprendere meglio quello che sarà il prodotto finale.
Una volta finita l'analisi del funzionamento dell'API Graph di Facebook abbiamo definito una base di partenza mediante le User Story. Abbiamo pensato di partire da una possibile schermata d'arrivo, in cui l'utente può arrivare tramite un url pubblico e avere un'anteprima di quello che è possibile fare con l'applicazione web.

Title	Description	Acceptance criteria
Web app landing	As a user I want to ge to the page through a public url and see a sample of what the app can do So that i can decide if go on or not	• A web page publicy reachable • Custom domain • Web hosting on AWS • A video that show the user an output of the app

Figura 3.1 – Web app landing user story

Se l'utente dovesse poi decidere di utilizzare l'app, viene messo a disposizione un bottone per il login tramite Facebook.

Facebook login	As a user
	
	I want to make the login with Facebook
	
	So that i can use my data to personalize my experience
		• Implementation of the Facebook login with
		  • email of Facebook's account
		  • password of Facebook's account

Figura 3.2 – Facebook login user story

Quando l'utente effettua il login diamo la possibilità di creare e personalizzare il proprio video tramite un'interfaccia che lo guida passo dopo passo.

Creation interface	As a user
	
	I want to choose what to insert in the Pvideo
	
	So that i can get a personalized one
		• An interface that allow the user to personalize the Pvideo
		  • select some photos/posts
		  • select some friends
		  • add some phrase

Figura 3.3 – Creation interface user story

Alla fine del processo di creazione l'utente verrà portato all'ultima schermata dell'applicazione in cui potrà avere un'anteprima del prodotto appena ultimato.

Pvideo preview	As a user I want to see my Pvideo at the end of the process So that a can see the result	• A pURL to the Pvideo is displayed at the end of the process • A preview of the Pvideo embedded in the Web page

Figura 3.4 – Pvideo preview user story

In questa schermata mettiamo a disposizione due metodi per poter condividere il proprio risultato con chi preferisce, scegliendo tra la condivisione tramite e-mail o tramite Facebook.

Sharing with Facebook	As a user I want the possibility to share my Pvideo on Facebook So that my friends can see it	• Implementation of the Sharing Facebook plugin • Video player embedded in the facebook's post • Button for sharing via facebook's post
Sharing with email	As a user I want the possibility to share my Pvideo via email So that my friends can see it	• Button for sharing via email

Figura 3.5 – Sharing with Facebook e Sharing with email user story

3.2 Architettura

Il passo successivo alla creazione delle User Story è stata l'ideazione di una ipotetica architettura. Amazon Web Services, in questa parte del progetto, si è rivelato essenziale in quanto mette a disposizione diversi servizi che sono stati utilizzati. Primo fra tutti S3, un servizio di storage che abbiamo deciso di utilizzare come hosting dei file statici del progetto. Per poter, invece, distribuire a tutti gli utenti il contenuto di S3 abbiamo utilizzato CloudFront, un altro servizio offerto da AWS, che funge da Content Delivery Network, permettendoci così di gestire una lista di accessi ai file di S3 vietandone il loro accesso diretto per questioni di sicurezza e per garantire una migliore esperienza d'uso. Tramite CloudFront è stato possibile anche assegnare un dominio personalizzato all'applicazione rendendola più facile da riconoscere e ricordare.

Doxee Pvideo, il prodotto aziendale che il progetto mira ad integrare con i social network, permette la creazione di video personalizzati sulla base di dati passati come parametri. Per riuscire a fare ciò, un ruolo importante lo assume la Doxee Platform, una piattaforma interna all'azienda che prende in input un file XML contenente i dati da inserire nel Pvideo e li carica in un template preparato in precedenza. I dati vengono inviati tramite una chiamata API REST ad un endpoint che, per questioni di sicurezza, è protetto da password. Per evitare di inserire le credenziali all'interno del codice dell'applicazione web, impedendo così che venissero usate da eventuali utenti malintenzionati, abbiamo fatto uso di AWS Lambda, un servizio di elaborazione che permette di eseguire delle "funzioni

lambda" senza gestire alcun tipo di server. Queste funzioni possono essere invocate tramite una chiamata API REST ad un API Gateway AWS che abbiamo collegato a loro, permettendo così al front-end di inviare i dati che l'utente vuole inserire nel video personalizzato. Sono state create in tutto tre funzioni lambda:

- *lambda-create*: Una funzione che prende in ingresso un file JSON inviato dall'applicazione web e lo trasforma in un file XML compatibile con la Doxee Platform per poi inviarglielo tramite una chiamata API REST. Successivamente aspetta l'url del Pvideo e lo restituisce al front-end.
- *lambda-email*: Una funzione che prende in ingresso un file JSON contenente le informazioni dell'e-mail che l'utente vuole mandare e le inoltra all'endpoint di SendGrid tramite una chiamata API REST e le dovute credenziali.
- *lambda-proxy*: Una funzione che agisce da proxy tra il server in cui è contenuto il Pvideo e il browser dell'utente in modo da aggiungere all'HTML del video i meta tag di cui necessita Facebook per lo share delle pagine sul suo sito.

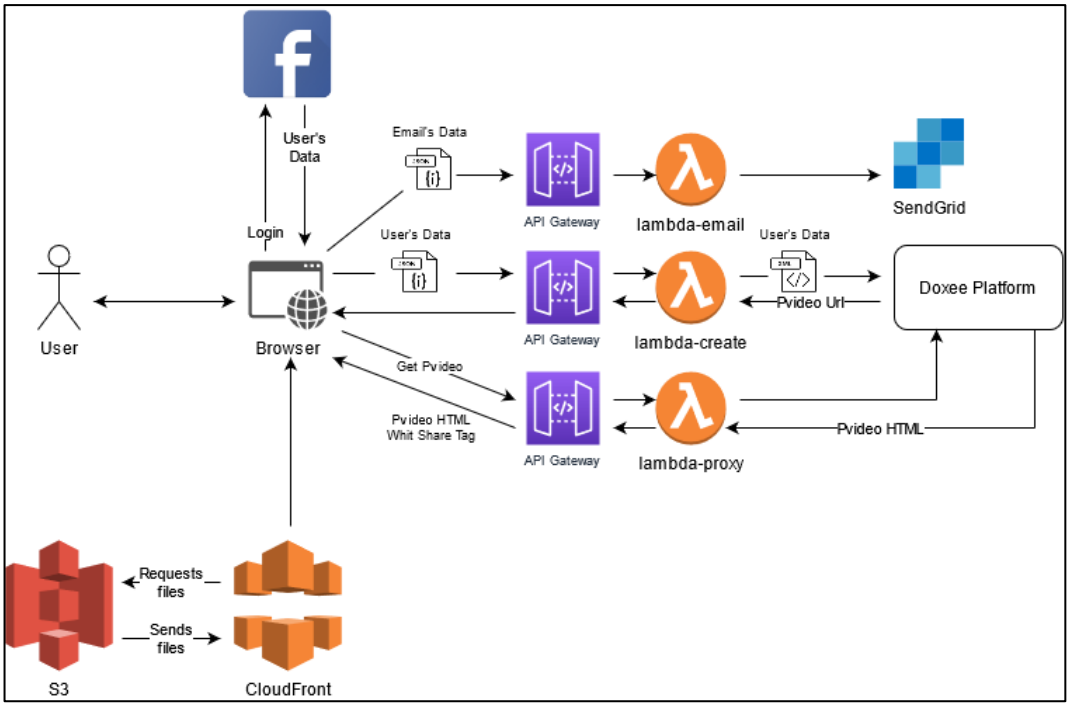

Figura 3.6 – Diagramma dell'architettura

3.3 Storie implementative

L'ultimo passo di progettazione è stata la definizione delle User Story, ossia la loro trasformazione in compiti precisi e dettagliati, chiamati storie implementative, in modo tale che chiunque, all'interno del progetto, potesse prenderli in carico e portarli a termine senza ulteriori spiegazioni.

Come prima storia implementativa è stata scelta la creazione dei wireframe dell'applicazione, che consistono in una prima rappresentazione abbozzata di quello che sarà il sito web.

Story	Activities	Acceptance criteria
Wireframe	Creation of a wireframe for: • Homepage • Creation page • Preview page	• wireframe for every Web App's pages avaiable

Figura 3.7 – Storia implementativa "Wireframe"

Ai wireframe seguono poi i mockup, una realizzazione a scopo illustrativo di quello che sarà il prodotto finale. Nel nostro caso l'aspetto di come l'applicazione apparirà all'utente.

Mockup	Creation of a mockup for: • Homepage • Creation page • Preview page	• mockup for every Web App's pages avaiable

Figura 3.8 – Storia implementativa "Mockup"

Una volta realizzati i Mockup, e di conseguenza in possesso di una idea precisa per il design del front-end, risulta possibile procedere con l'implementazione vera e propria delle pagine HTML della web app.

HTML Implementation: Homepage	Implementation of the html code for the Homepage	• an Hompage visible in browser
HTML Implementation: Creation page	Implementation of the html code for the Creation page	• a Creation page visible in browser
HTML Implementation: Preview page	Implementation of the html code for the Preview page	• a Preview page visible in browser
HTML Implementation: Privacy Policy page	Implementation of the html code for the Privacy Policy page	• a Privacy Policy page visible in browser
HTML Implementation: User's Pvideo list page	Implementation of the html code for the User's Pvideo list page	• a User's Pvideo list page visible in browser

Figura 3.9 – Storie implementative "HTML Implementation"

All'implementazione dell'HTML segue la realizzazione delle funzioni che gestiscono la parte di creazione del video da parte dell'utente: le frasi e le foto che vuole inserire, la loro memorizzazione, il loro invio e i cambiamenti a livello visivo all'interno della presentazione della scena per offrire all'utilizzatore del software un'idea del risultato.

Creation page: functions	Implementation of the functions that allow the user to choose what to insert in his Pvideo: • select a photo from Facebook • add a text	• a JSON output is produced with every information that the user want to insert in the Pvideo

Figura 3.10 – Storia implementativa "Creation page: functions"

Per poter poi integrare il tutto con Facebook abbiamo inserito anche le sue storie implementative. La prima riguardava la creazione di una Facebook App, attraverso un account sviluppatore, per aver accesso all'App ID da poter utilizzare nella seconda storia. La quale consisteva nell'implementazione di Facebook Login all'interno dell'applicativo, in modo da permettere all'utente di effettuare l'accesso al sito tramite il social media e poter inserire le proprie foto nel video.

Facebook App	Prerequisite: • Facebook Developer account Creation of a Facebook app in development mode and set to Consumer type	• a Facebook app is created and we have access to the App ID
Facebook login	Prerequisite: • Facebook Javascript SDK integration Implementation of the Facebook Login on the button of the Homepage.	• the login button open a Facebook window that ask the user email, password and the permission for what the Web App need • after a successful login the user's profile picture is showed and a logout button appeared

Figura 3.11 – Storie implementative "Facebook App" e "Facebook login"

Finta la parte front-end del progetto, occorreva creare il template del Pvideo su cui gli utenti avrebbero inserito i propri dati personali. Andava quindi creata la base di quello che sarebbe stato il video finale, con tanto di animazioni di foto e scritte, e definizione della struttura del file da cui il video avrebbe reperito le informazioni da inserire. Per fare tutto ciò, Doxee dispone di un plugin per il software di creazione di animazioni digitali Adobe Animate, con il quale è possibile creare un progetto compatibile col prodotto Doxee Pvideo e inserirlo all'interno della loro piattaforma collegato ad un endpoint che accetta in input il file contenente le informazioni da inserire nel video. Una volta gestita tutta la catena di creazione ed ottenuto un primo risultato abbiamo provveduto ad inserirlo nella homepage del sito per fornire una breve presentazione delle capacità della web app.

Sample Pvideo	Creation of a sample Pvideo of the Web app's output	• a Pvideo output of the Web app • the Pvideo inserted in the Homepage

Figura 3.12 – Storia implementativa "Sample Pvideo"

Per ospitare e distribuire poi l'applicazione agli utenti, abbiamo dovuto creare un Bucket S3 in cui salvare tutti i file riguardanti il front-end e usare un dominio CloudFront per la distribuzione.

Setup S3 and CloudFront	Prerequisite: • ticket to create a new doxee.com subdomain	• the new doxee subdomain can be used as custom domain for pvideos • the new subdomain has a context root that points to the s3 web app origin • public access to S3 without cloudfront is forbidden • acl for bucket access are defined

Figura 3.13 – Storia implementativa "Setup S3 and CloudFront"

L’attenzione si è successivamente spostata sulla parte back-end che prevedeva la creazione delle tre funzioni lambda che abbiamo visto nel capitolo 3.2

Lambda create	Implementation of a lambda that authenticate with the doxee platform and sends the JSON file to the endpoint	• a lambda that recive the JSON file from the web app and sends it to the Doxee platform

Figura 3.14 – Storia implementativa “Lambda create”

Le ultime aggiunte per poter terminare il progetto sono state le opzioni di condivisione, tramite social sharing e e-mail, nella pagina di preview del Pvideo, e l’incorporazione del video finale nella stessa pagina, di cui l’url è ottenuto come riposta dalla funzione lambda *create*.

Pvideo preview	the embedded of the final result on the Preview page	• the Pvideo created by the doxee platform is showed in the Preview page
Facebook sharing	Integration of the Facebook sharing plugin, a "Share" button on the Preview page and the modification of the main code of the Pvideo to automate the thumbnail's and the sharing message's creation	• a Share button is created and show a window where the user can modify the sharing's view on Facebook
Email sharing	Implementation of a email sharing feature with SendGrid	• When the user use the email sharing feature, SendGrid sends the emails to share the Pvideo content

Figura 3.15 – Storie implementative “Pvideo preview”, “Facebook sharing” e “Email sharing”

Capitolo 4

Sviluppo

In questo capitolo, dopo aver appreso la progettazione nel precedente, verrà analizzato lo sviluppo della suddetta. In particolare, verranno spiegate eventuali decisioni prese durante il percorso, alcune scelte di stile e le parti più significative del codice.

4.1 Wireframe

I wireframe sono stati realizzati mediante Figma, un editor di grafica vettoriale e strumento di prototipizzazione. Abbiamo pensato di dare all'applicazione un aspetto molto semplice ma efficace e intuitivo, in modo che l'utente possa capire come muoversi facilmente all'interno di essa.
Nella homepage è possibile trovare il titolo della web app, un video dimostrativo, un paragrafo esplicativo delle funzionalità del sito e un pulsante per poter effettuare il login tramite Facebook.

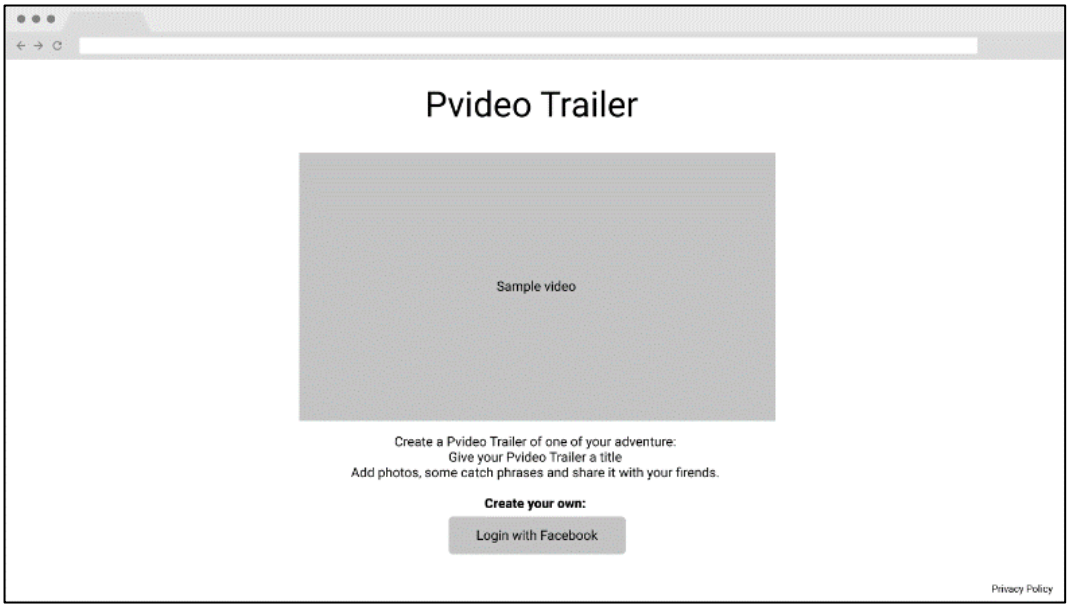

Figura 4.1 – Wireframe della homepage

Successivamente è possibile trovare le pagine per la personalizzazione del video. In ognuna di queste è stata inserita: una barra di navigazione, contenente alcuni link e uno spazio per la foto del profilo dell'utente, una barra di progressione per spostarsi tra le varie scene, vedere quali sono state inserite e quali mancano e, infine, tre bottoni, uno per tornare alla scena precedente, uno per saltare la scena corrente e l'ultimo per inserirla e passare alla scena successiva.

In base alla scena in cui ci si trova saranno presenti delle opzioni di personalizzazione specifici, e una sua raffigurazione statica per permettere all'utente di avere riscontro diretto delle sue personalizzazioni. Nelle scene del titolo e di intermezzo tra le foto sarà possibile inserire rispettivamente titolo e frasi direttamente sull'anteprima della scena.

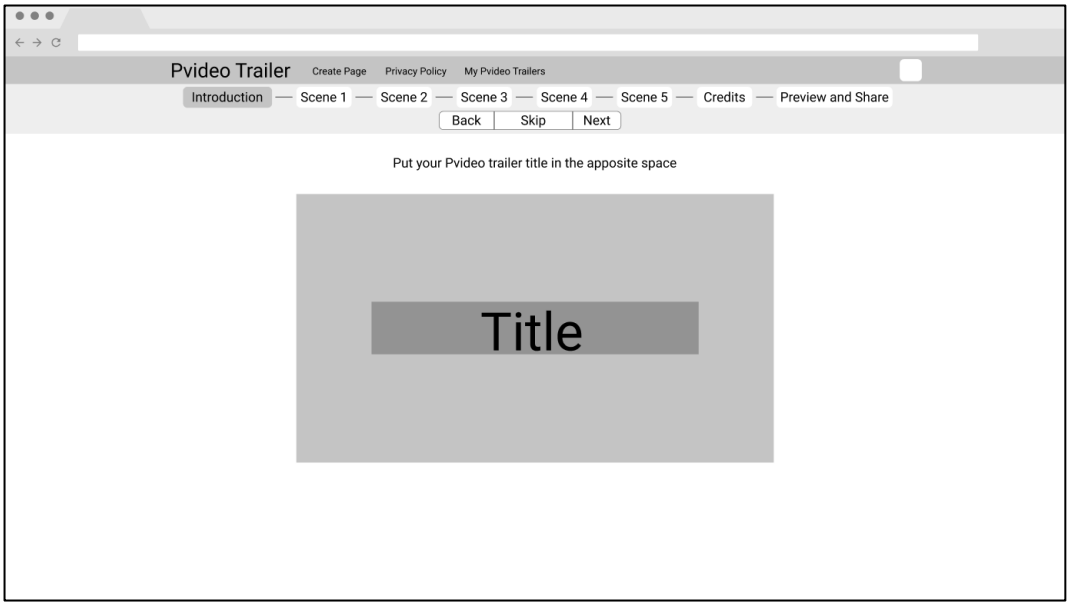

Figura 4.2 – Wireframe della scena "Introduzione"

Avviene la stessa cosa anche nelle scene delle foto, in cui si aggiunge una griglia contenente le immagini dell'utente da cui è possibile scegliere quale inserire. Quella selezionata comparirà nell'anteprima.

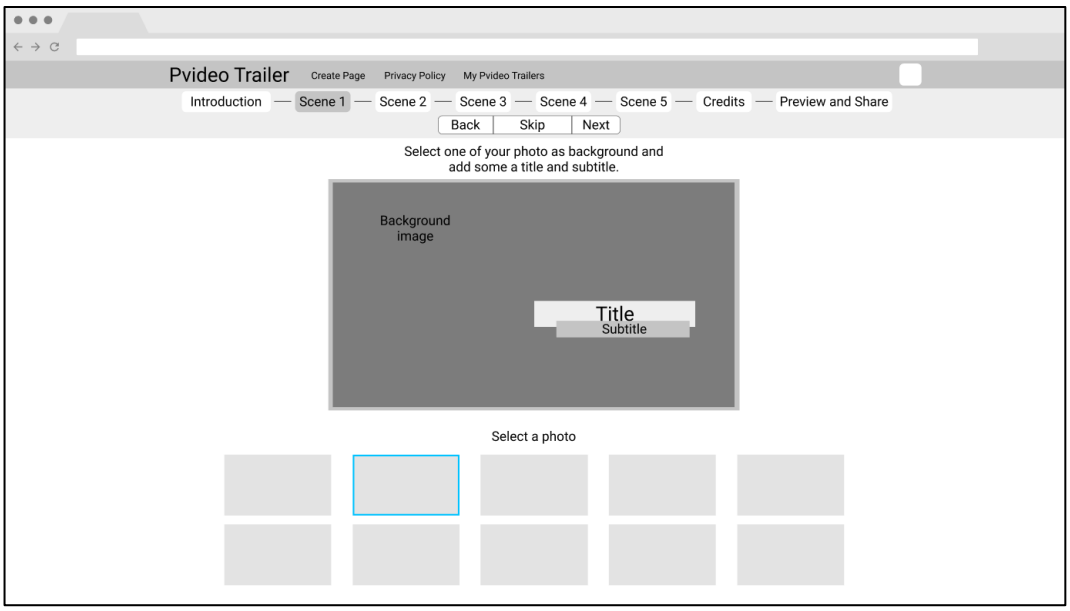

Figura 4.3 – Wireframe della "Scena 1"

La pagina finale, chiamata "Preview and Share", è la sezione in cui l'utente si ritroverà una volta finita la personalizzazione del video e in cui potrà osservare il risultato finale. Sotto di essa sono presenti tre pulsanti: il primo per la condivisione

su Facebook, il secondo per la condivisione via e-mail e il terzo offre la possibilità di creare un nuovo Pvideo.

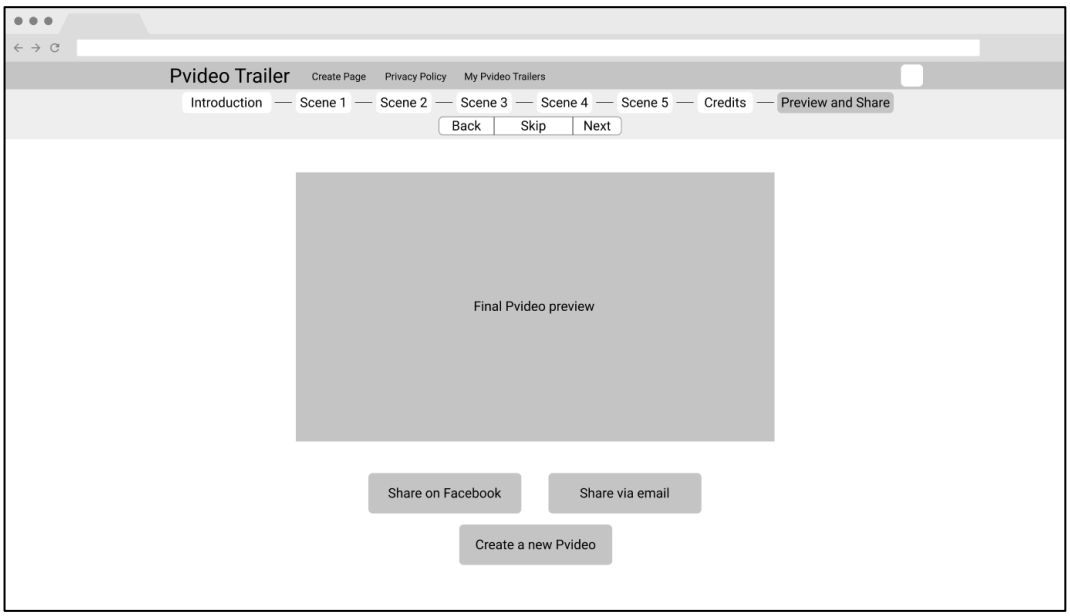

Figura 4.4 – Wireframe della pagina "Preview and Share"

È stata prevista anche una pagina che permettesse all'utente di capire se qualcosa, durante la creazione del video, fosse andato storto, dandogli la possibilità di creare un video dall'inizio o riprovare la creazione del video in questione.

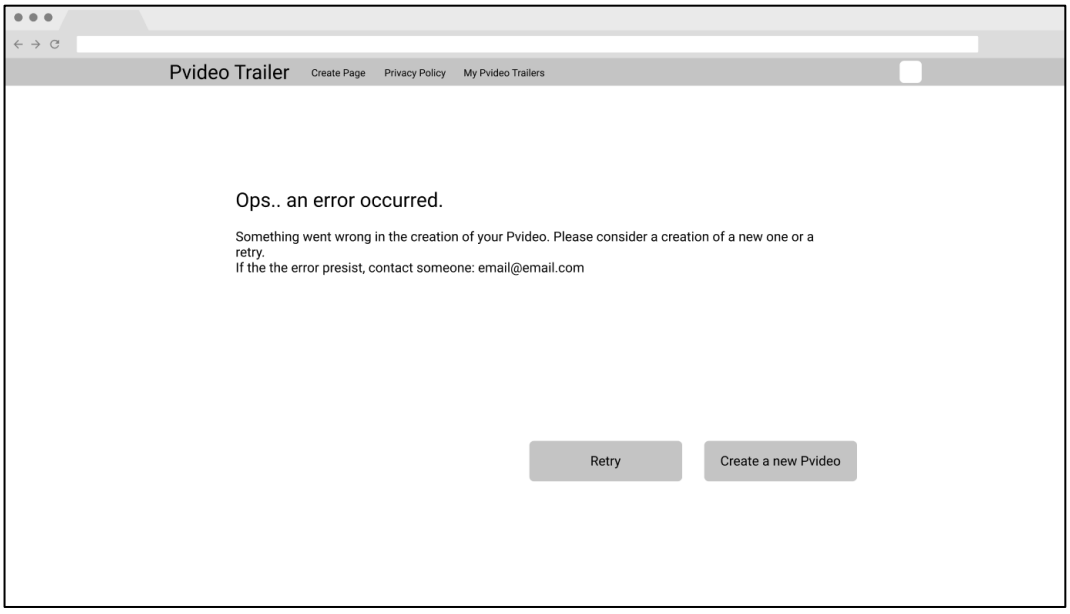

Figura 4.5 – Wireframe della pagina di errore

4.2 Mockup

Per i mockup, anch'essi realizzati tramite il software Figma, sono stati privilegiati i colori e gli elementi delle pagine già caratteristici della ditta Doxee. La barra di navigazione presenta il logo della ditta bianco su sfondo arancione, le scritte principali, come il titolo dell'applicazione, sono arancioni e i pulsanti, a seconda del ruolo che hanno, si presentano con bordi arrotondati e uno sfondo azzurro (primari) o sfondo bianco con bordo azzurro (secondari). La barra di progressione permette di far capire all'utente la scena in cui si trova, colorandone il titolo in arancione e aggiungendo un contorno quadrato con angoli arrotondati dello stesso colore.

Le scene su cui non è ancora stato sono grigie, quelle inserite arancioni e quelle saltate grigio chiaro.

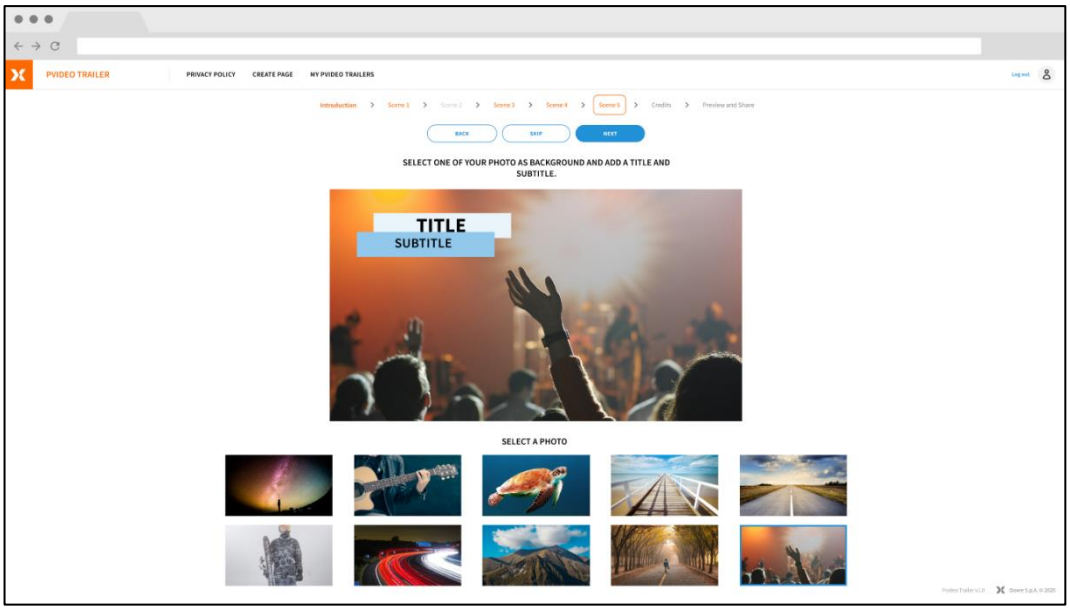

Figura 4.6 – Mockup della scena 5

Per il pulsante nella homepage destinato al login tramite Facebook è stato usato, invece, il classico blu del social network abbinato al suo logo, posto sulla sinistra, e una forma più squadrata ma sempre con angoli arrotondati.

Figura 4.7 – Mockup pulsante di login

I tre pulsanti nella pagina di preview seguono lo stesso stile di quello di login. Questo perché è presente anche quello per la condivisione tramite Facebook, che, per coerenza, condivide il design con il bottone di login, e per omogeneità anche gli altri due mantengono lo stesso stile ma con colori differenti. Quello per la condivisione tramite e-mail presenta uno sfondo arancione e quello per la creazione di un nuovo video uno sfondo bianco con bordo arancione.

Figura 4.8 – Mockup pulsanti della pagina di preview

4.3 Front-end

Come framework per il front-end è stato scelto Angular, ottimo per realizzare siti di piccole dimensioni. Questo perché, grazie alla sua funzione di routing, permette di concentrare tutto il sito in un'unica pagina in cui sono contenute tutte le sezioni dell'applicazione, che vengono caricate dal browser a seconda dei collegamenti interni che l'utente ha deciso di seguire. Tutto ciò permette al sito di essere molto più fluido e di non aver bisogno di ulteriori caricamenti dopo quello iniziale.
Per poter usufruire del Source Development Kit di Facebook occorre importarlo tramite uno script utilizzando l'app ID. Il modo migliore per essere fatto è in maniera asincrona in modo da evitare che la pagina blocchi il suo caricamento nell'attesa che l'SDK venga caricato.

```
1. <script>
2.     window.fbAsyncInit = function() {
3.       FB.init({
4.         appId            : 'appID',
5.         autoLogAppEvents : true,
6.         xfbml            : true,
7.         status           : true,
```

```
8.          cookie           : true,
9.          version          : 'v9.0'
10.        });
11.      };
12.    </script>
13.    <script async defer
14.      src="https://connect.facebook.net/en_US/sdk.js"></scri
   pt>
15.    </script>
16.
17.
```

Grazie ad esso è possibile configurare il login tramite Facebook e successivamente ottenere le foto presenti sul profilo del social network dell'utente grazie ad una richiesta di tipo GET all'API Graph.

```
1. getImages() {
2.     FB.api(
3.       '/me/photos/uploaded?fields=images.limit(10){source}',
4.     (response: any) => {
5.       if(localStorage.getItem('photo_1'))
6.         document.getElementById('table').style.setProperty('
   --img', 'url('+localStorage.getItem('photo_1')+')');
7.       for(let index = 1; index<=10; index++) {
8.         document.getElementById('photo_'+index.toString())
9. .setAttribute('src', response.data[index].images[0].source);
10.          console.log('photo: ', response.data[index]
11.  .images[0].source);
12.        }
13.      });
14.  }
```

Per gestire tutte le scelte di personalizzazione dell'utente, esse vengono salvate come variabili all'interno del local storage del browser, in modo da poter essere reperite in caso di ritorno su una scena già personalizzata.

```
1. data: any =
2.   {
3.     "name": localStorage.getItem('userName'),
4.     "userID": localStorage.getItem('userID'),
5.     "title": localStorage.getItem('title'),
6.     "enable_1": localStorage.getItem('Introduction'),
7.     "enable_2": localStorage.getItem('Scene1'),
8.     "photo_1": localStorage.getItem('photo_1'),
9.     "title_1": localStorage.getItem('title_1'),
10.      "subtitle_1": localStorage.getItem('subtitle_1'),
11.      "enable_3": localStorage.getItem('Scene2'),
12.      "description_1": localStorage.getItem('description_1')
   ,
13.      "enable_4": localStorage.getItem('Scene3'),
14.      "photo_2": localStorage.getItem('photo_2'),
```

```
15.       "title_2": localStorage.getItem('title_2'),
16.       "subtitle_2": localStorage.getItem('subtitle_2'),
17.       "enable_5": localStorage.getItem('Scene4'),
18.       "description_2": localStorage.getItem('description_2')
    ,
19.       "enable_6": localStorage.getItem('Scene5'),
20.       "photo_3": localStorage.getItem('photo_3'),
21.       "title_3": localStorage.getItem('title_3'),
22.       "subtitle_3": localStorage.getItem('subtitle_3'),
23.       "enable_7": localStorage.getItem('Credits'),
24.       "director": localStorage.getItem('director'),
25.       "propic": localStorage.getItem('propic'),
26.       "actor_1": localStorage.getItem('actor_1'),
27.       "actor_2": localStorage.getItem('actor_2'),
28.       "actor_3": localStorage.getItem('actor_3')
29.     }
```

Inoltre, vengono usate per la creazione del JSON da inviare all'endpoint tramite una richiesta di tipo POST.

```
1. sendPostRequest(data: any): Observable<String> {
2.     const headers = { 'content-type': 'application/json'}
3.     const body=JSON.stringify(data);
4.     console.log(body);
5.     return this.httpClient.post<String>(this.urlApi, data, {
    headers, responseType: 'text' as 'json' });
6. }
7.
```

Sono presenti altre due chiamate ad endpoint, una per la funzionalità della condivisione tramite e-mail e l'altra per la condivisione tramite Facebook. La prima crea anch'essa un file JSON tramite le informazioni inserite dall'utente e lo invia alla funzione lambda che si occupa a sua volta di inoltralo all'endpoint di SendGrid per l'effettivo invio della e-mail.

```
1. sendEmail(content){
2.     if(this.recieverEmail === 'null' || this.subjectEmail ==
   = 'null' || this.textEmail === '')
3.       this.modalService.open(content, {centered: true});
4.     else {
5.       this.emailForm = {
6.
7.           "personalizations": [
8.             {
9.               "to": [
10.                {
11.                  "email": this.recieverEmail
12.                }
13.              ],
14.              "subject": this.subjectEmail
```

```
15.              }
16.            ],
17.            "from": {
18.              "email": this.userEmail,
19.            },
20.            "content": [
21.              {
22.                "type": "text/plain",
23.                "value": this.textEmail
24.              }
25.            ]
26.          }
27.          this.sendPostEmail(this.emailForm).subscribe(
28.            res => {
29.              console.log(res.toString());
30.            }
31.          );
32.      }
33.
34.    }
35.
36.    sendPostEmail(data: any): Observable<String> {
37.      const headers = {'content-tupe': 'application/json'}
38.      const body = JSON.stringify(data);
39.      console.log(body);
40.      return this.httpClient.post<String>(
   this.urlApiemail, data);
41.    }
42.
```

La seconda chiamata, invece, si occupa di effettuare una richiesta HTTP GET, per il Pvideo appena creato, ad una funzione lambda che funge da proxy ed inserisce i meta tag all'HTML del video che Facebook utilizzerà come informazioni per creare un post sul proprio sito.

```
1. share() {
2.     console.log(this.guid);
3.     FB.ui({
4.       method: 'share',
5.       href: this.urlApiShare+this.guid,
6.       display: 'popup',
7.     }, function(response){});
8.   }
9.
```

4.4 Back-end

Per quanto riguarda il back-end, come è stato anticipato nel capitolo 3.2, sono state realizzate tre funzioni lambda mediante il servizio AWS omonimo.

4.4.1 Lambda-create

La prima, *lambda-create*, è scritta in Java e permette di trasformare il file JSON, che arriva in input tramite l'API Gateway dal front-end, in un file XML compatibile con quello prestabilito dalla Doxee Platform, per inviarlo successivamente ad essa che si occuperà della creazione del video.
Come prima cosa si occupa di leggere il JSON, presente nel campo body del pacchetto ricevuto dall'API Gateway, e salvare tutti i dati necessari in stringhe.

```
1. JsonNode eventApi = OBJECT_MAPPER.readTree(input); // root n
   ode
2. JsonNode eventNode;
3. String body = eventApi.path("body").asText();
4. eventNode = OBJECT_MAPPER.readTree(body);
5. //Retrieve of recipient's attributes
6. String name = eventNode.path("name").asText();
7. String userID = eventNode.path("userID").asText();
8.
9. //Retrieve of scene1's elements
10.  String title = eventNode.path("title").asText();
11.  String enable_1 = eventNode.path("enable_1").asText();
12.
13.  //Retrieve of scene2's elements
14.  String enable_2 = eventNode.path("enable_2").asText();
15.  String photo_1 = eventNode.path("photo_1").asText();
16.  String title_1 = eventNode.path("title_1").asText();
17.  String subtitle_1 = eventNode.path("subtitle_1").asText();
18.
19.  //Retrieve of scene3's elements
20.  String enable_3 = eventNode.path("enable_3").asText();
21.  String description_1 = eventNode.path("description_1").asT
   ext();
22.
23.  //Retrieve of scene4's elements
24.  String enable_4 = eventNode.path("enable_4").asText();
25.  String photo_2 = eventNode.path("photo_2").asText();
26.  String title_2 = eventNode.path("title_2").asText();
27.  String subtitle_2 = eventNode.path("subtitle_2").asText();
28.
29.  //Retrieve of scene5's elements
30.  String enable_5 = eventNode.path("enable_5").asText();
31.  String description_2 = eventNode.path("description_2").asT
   ext();
32.
33.  //Retrieve of scene6's elements
34.  String enable_6 = eventNode.path("enable_6").asText();
35.  String photo_3 = eventNode.path("photo_3").asText();
36.  String title_3 = eventNode.path("title_3").asText();
```

```
37.  String subtitle_3 = eventNode.path("subtitle_3").asText();
38.
39.  //Retrieve of scene7's elements
40.  String enable_7 = eventNode.path("enable_7").asText();
41.  String director = eventNode.path("director").asText();
42.  String propic = eventNode.path("propic").asText();
43.  String actor_1 = eventNode.path("actor_1").asText();
44.  String actor_2 = eventNode.path("actor_2").asText();
45.  String actor_3 = eventNode.path("actor_3").asText();
```

Successivamente inserisce i dati negli oggetti classe delle scene, i quali, grazie alla libreria Java javax.xml.bind.annotation, verranno mappati all'interno di uno schema XML.

```
1. //Setting scene1's attributes
2. Scene1 scene1 = new Scene1();
3. scene1.setTitle(title);
4. scene1.setEnable(enable_1);
5.
6. //Setting scene2's attributes
7. Scene2 scene2 = new Scene2();
8. scene2.setEnable(enable_2);
9. scene2.setPhoto(photo_1);
10.  scene2.setTitle(title_1);
11.  scene2.setSubtitle(subtitle_1);
12.
13.  //Setting scene3's attributes
14.  Scene3 scene3 = new Scene3();
15.  scene3.setEnable(enable_3);
16.  scene3.setDescription(description_1);
17.
18.  //Setting scene4's attributes
19.  Scene4 scene4 = new Scene4();
20.  scene4.setEnable(enable_4);
21.  scene4.setPhoto(photo_2);
22.  scene4.setTitle(title_2);
23.  scene4.setSubtitle(subtitle_2);
24.
25.  //Setting scene5's attributes
26.  Scene5 scene5 = new Scene5();
27.  scene5.setEnable(enable_5);
28.  scene5.setDescription(description_2);
29.
30.  //Setting scene6's attributes
31.  Scene6 scene6 = new Scene6();
32.  scene6.setEnable(enable_6);
33.  scene6.setPhoto(photo_3);
34.  scene6.setTitle(title_3);
35.  scene6.setSubtitle(subtitle_3);
36.
37.  //Setting scene7's attributes
38.  Scene7 scene7 = new Scene7();
39.  scene7.setEnable(enable_7);
40.  scene7.setDirector(director);
```

```
41.  scene7.setPropic(propic);
42.  scene7.setActor_1(actor_1);
43.  scene7.setActor_2(actor_2);
44.  scene7.setActor_3(actor_3);
45.
46.  //Setting video's elements
47.  Video video = new Video();
48.  video.setScene1(scene1);
49.  video.setScene2(scene2);
50.  video.setScene3(scene3);
51.  video.setScene4(scene4);
52.  video.setScene4(scene4);
53.  video.setScene5(scene5);
54.  video.setScene6(scene6);
55.  video.setScene7(scene7);
56.
57.  //Setting template's element
58.  Template template = new Template();
59.  template.setVideo(video);
60.
61.  //Setting recipient's attributes
62.  Recipient recipient = new Recipient(name, userID);
63.
64.  //Setting recipients' element
65.  Recipients recipients = new Recipients();
66.  recipients.setRecipient(recipient);
67.
68.  //Setting file's element
69.  File file = new File();
70.  file.setRecipients(recipients);
71.  file.setTemplate(template);
72.
```

Una volta ottenuto l'oggetto file, contenente lo schema XML, vengono creati due pacchetti: il dataXML e il bomXML. Il primo, come suggerisce il nome, è quello contenente i dati, il secondo, il quale acronimo sta per Byte Order Mark, consente di specificare il tipo di codifica del primo e fornire altre informazioni all'endpoint. Entrambi, prima di essere inviati, vengono trasformati in array di byte e codificati in base64. Per potersi interfacciare con la Doxee Platform, l'header del pacchetto viene, inoltre, fornito di una stringa in base64 contenente le credenziali per la basic access authentication.

```
1. Form form = new Form();
2. JAXBContext contextObj = JAXBContext.newInstance(File.class)
   ;
3. Marshaller marshallerObj = contextObj.createMarshaller();
4. marshallerObj.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT,
   true);
5.
6. //dataXml
```

```
7. ByteArrayOutputStream bos = new ByteArrayOutputStream();
8. marshallerObj.marshal(file, bos);
9.
10.  String encodedData = new String(b64enc.encode(bos.toString
   ()));
11.  form.add("dataXml", encodedData);
12.  context.getLogger().log("dataXml:" + bos.toString());
13.
14.  //bomXml
15.  byte[] bomByteArray = createBom(title, bos.size()).getByte
   s();
16.  String encodedBom = new String(b64enc.encode(bomByteArray)
   );
17.  form.add("bomXml", encodedBom);
18.  context.getLogger().log("bomXml:" + new String(bomByteArra
   y));
19.
20.  Client client = null;
21.
22.  ClientConfig config = new DefaultClientConfig();
23.  client = Client.create(config);
24.
25.  String auth = odp_user + ":" + odp_pwd;
26.  context.getLogger().log(" auth : " + auth);
27.  byte[] encodedAuth = org.apache.commons.codec.binary.Base6
   4.encodeBase64(auth.getBytes(StandardCharsets.ISO_8859_1));
28.  String authHeader = "Basic " + new String(encodedAuth);
29.
30.  context.getLogger().log(" odp_url : " + odp_url);
31.
32.  WebResource service = client.resource(odp_url);
33.
34.  ClientResponse response = service.header(HttpHeaders.AUTHO
   RIZATION, authHeader).type(MediaType.APPLICATION_FORM_URLENC
   ODED).post(ClientResponse.class, form);
35.
36.  if (response.getStatus() == 200) {
37.        String pUrl = getpUrl(response);
38.        context.getLogger().log("pUrl: " + pUrl);
39.        OBJECT_MAPPER.writeValue(output, pUrl);
40.  }
41.
```

4.4.2 Lambda-proxy

Questa funzione lambda si è resa necessaria a causa dell'impossibilità di modificare l'HTML della pagina del Pvideo in base alle proprie preferenze.

Per poter essere condivisa su Facebook, una pagina web ha bisogno di alcuni tag particolari che indicano al social network con quali informazioni presentare la pagina agli altri utenti.

Questa funzione, scritta in JavaScript, funge da proxy tra il front-end e il server di archiviazione dei Pvideo. Interponendosi tra i due modifica l'HTML del video personalizzato aggiungendo i meta tag per il social share tramite Facebook, definendo un titolo, il contenuto, un'immagine di presentazione e una descrizione.

```
1. exports.handler = function(event, context, callback) {
2.   console.log("EVENT: \n" + JSON.stringify(event, null, 2))
3.
4.   let guid = event.pathParameters.guid;
5.
6.
7.   const https = require('https')
8.
9.   https.get('https://test2.dev.doxee.com/da-
   purl/document/'+guid, (res) => {
10.        console.log("Got response: " + res.statusCode);
11.        let body = '';
12.
13.        res.on("data", (chunk) => { body += chunk});
14.        res.on("end", () => {
15.
16.          body = body.replace(
17.                /<meta name="viewport" content="width=device
   -width, initial-scale=1, maximum-scale=1">/,
18.                `<meta name="viewport" content="width=device
   -width, initial-scale=1, maximum-
   scale=1">\n\t<meta property="og:type" content="website"/>` +
19.                `\n\t<meta property="og:title" content="My P
   video Trailer"/>\n\t<meta property="og:description" content=
   "Check out my new adventure in this Pvideo Trailer that i ju
   st crea"/>` +
20.                `\n\t<meta property="og:image" content="http
   s://media-
   test2.dev.doxee.com/trailer/static/PvideoTrailer.png"/>` +
21.                `\n\t<meta property="og:image:width" content
   ="1200"/>\n\t<meta property="og:image:height" content="675"/
   >\n\t<meta property="og:image:type" content="image/png"/>` +
22.                `\n\t<meta property="og:site_name" content="
   PvideoTrailer"/>`
23.          );
24.          console.log("BodyModified: " + body);
25.
26.          let response = {
27.            "isBase64Encoded": false,
28.            "statusCode": 200,
29.            "headers": { "Content-Type": "text/html" },
30.            "multiValueHeaders": {},
31.            "body": body
32.          }
33.          callback(null, response);
34.        });
35.    }).on('error', function(e) {
36.        console.log("Got error: " + e.message);
37.    });
```

```
38.
39.     console.log("\nGuid: " + guid)
40.
41.
42.  }
43.
```

4.4.3 Lambda-email

La terza ed ultima lambda, la *lambda-email*, anch'essa scritta in JavaScript, viene usata per nascondere all'utente le credenziali di accesso all'API di SendGrid. Si occupa, infatti, di leggere il file JSON passato dall'API Gateway contenente tutte le informazioni per l'invio di una e-mail. Una volta ottenute, il suo compito consiste nell'effettuare una richiesta HTTP POST all'endpoint di SendGrid, inserendo le credenziali di accesso.

```
1. exports.handler = function(event, context, callback) {
2.   console.log("EVENT: \n" + JSON.stringify(event, null, 2))
3.
4.   let isBase64Encoded = event.isBase64Encoded;
5.   let emailbody = JSON.parse(event.body);
6.
7.   if(isBase64Encoded)
8.       emailbody = Buffer.from(emailbody, 'base64');
9.
10.
11.       var http = require("https");
12.
13.       var options = {
14.         "method": "POST",
15.         "hostname": "api.sendgrid.com",
16.         "port": null,
17.         "path": "/v3/mail/send",
18.         "headers": {
19.           "authorization": "Credentials",
20.           "content-type": "application/json"
21.         }
22.       };
23.
24.       var req = http.request(options, function (res) {
25.         var chunks = [];
26.
27.         res.on("data", function (chunk) {
28.           chunks.push(chunk);
29.         });
30.
31.         res.on("end", function () {
32.           var body = Buffer.concat(chunks);
```

```
            console.log(body.toString());
          });
        }).on('error', function(e) {
            console.log("Got error: " + e.message);
        });

        req.write(JSON.stringify(emailbody));
        req.end();
  }

```

4.5 Template del Pvideo

I template dei Pvideo vengono realizzati tramite Adobe Animate e un plugin, creato da Doxee, che ne permette la creazione.

Tramite questi due strumenti occorre definire una struttura del JSON da cui il video andrà a reperire i dati personalizzati e una serie di scene che andranno a costruire il video. All'interno di queste scene sarà possibile inserire vari elementi di personalizzazione, come testi, immagini, video e audio, tutti con la possibilità di essere statici o dinamici con contenuto reperito dal JSON. Inoltre, è possibile dare a tutti questi contenuti animazioni ed effetti per rendere più piacevole la visione del video.

Trattandosi di Facebook, il social network in esame, abbiamo pensato ad un video che potesse attirare l'attenzione dei ragazzi come prototipo. Il Pvideo in questione consiste in un trailer di una possibile vacanza o esperienza di un utente, in cui la struttura delle scene principali è la seguente:

- *Introduzione:* In questa scena è possibile decidere il titolo del proprio trailer.
- *Scena con foto:* Qui l'utente può inserire una propria foto, presente sul suo profilo Facebook, e dargli un titolo principale e un sottotitolo.
- *Descrizione:* Una scena in cui è presente solo del testo per poter dare un po' di contorno al video.
- *Crediti:* La scena finale, dove sono presenti i riconoscimenti delle persone che hanno partecipato al video.

A parte la scena dell'introduzione e dei crediti, quella con le foto e con le descrizioni si ripetono rispettivamente tre e due volte, arrivando ad un totale di sette scene.

Conclusione e sviluppi futuri

In questa tesi si è cercato di introdurre i lettori al mondo della Customer Experience, dei suoi strumenti: in particolare l'utilizzo dei video per poter interagire coi propri clienti e di fornire una spiegazione alle motivazioni che hanno portato alla realizzazione di questo progetto.

L'obiettivo principale raggiunto da questo elaborato è la prova della possibilità di creazione di un video personalizzato tramite dati provenienti da social network, in questo caso specifico, dell'integrazione tra il prodotto aziendale Doxee Pvideo con Facebook.

Trattandosi di un proof of concept, questo progetto è ancora lontano dal suo utilizzo su vasta scala, dimostrando però il suo potenziale, che, sicuramente in futuro, sarà applicato e magari ampliato ad altri social network. Potrebbe includere un sistema di gestione dei precedenti video creati dall'utente di cui attualmente non dispone, o offrire altre opzioni di personalizzazione. Non è escluso neanche il cambio di direzione nella trasformazione del video in un contenuto più professionale. Una delle idee nate verso la fine è stata la trasformazione del progetto in un curriculum vitae multimediale, il quale, prendendo le informazioni degli utenti da LinkedIn, possa costruire un video di presentazione da sostituire o accompagnare al proprio CV.

Bibliografia

[1] Customer experience: https://www.digital4.biz/marketing/big-data-e-analytics/customer-experience-cos-e-perche-e-strategica-per-le-aziende-e-quali-sono-le-tecnologie-per/

[2] Cisco Annual Report: https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html

[3] Marketing video: https://www.obliquodesign.com/video/marketing-video-come-integrarlo-nella-comunicazione

[4] Importanza dei social network: https://sociologicamente.it/limportanza-dei-social-network-per-le-aziende-quando-esserci-e-il-miglior-modo-di-comunicare/

[5] Social sharing: https://socialandtech.net/social-sharing-cosa-ci-spinge-alla-condivisione/

[6] Angular Wikipedia: https://it.wikipedia.org/wiki/Angular

[7] Angular: https://angular.io/features

[8] TypeScript: https://www.typescriptlang.org/

[9] TypeScript Wikipedia: https://it.wikipedia.org/wiki/TypeScript

[10] Amazon Web Services: https://aws.amazon.com/it/what-is-aws/?nc1=f_cc

[11] Amazon Cloudfront: https://aws.amazon.com/it/cloudfront/?nc2=type_a

[12] Amazon S3: https://aws.amazon.com/it/s3/?nc2=type_a

[13] Amazon Lambda: https://aws.amazon.com/it/lambda/

[14] Doxee Pvideo: https://www.doxee.com/it/prodotti/interactive-experience/pvideo-video-personalizzati/

[15] SendGrid Wikipedia: https://en.wikipedia.org/wiki/SendGrid

[16] Facebook API Graph: https://developers.facebook.com/docs/graph-api/overview/

[17] User Story: https://www.atlassian.com/agile/project-management/user-stories