ELABORATO DELL'ESAME DI STATO

"Web Community"

Leonardo Marchesini, classe 5IA

ITIS Carlo Zuccante
AS 2020/2021

SmartSocial

ABSTRACT

Italiano:

L'elaborato tratta la creazione di una *web community* per condividere dati e commenti relativi a eventi dal vivo di generi diversi (sportivo, artistico, ecc.). Ci sono due tipi di utenti che possono accedere al *social network*. Il primo è quello degli utenti anonimi, che possono solo consultare il sito, o l'applicazione, visualizzando tutti gli eventi, sia quelli in programma che quelli già svolti. Il secondo è quello di utenti registrati che, dopo aver eseguito il login con un account Google avranno un profilo completo all'interno del social. A differenza degli utenti anonimi, quelli registrati hanno la possibilità di interagire con gli eventi pubblicati, vederne solo alcuni in base a determinati parametri o addirittura pubblicarne altri. Inoltre possono rimanere sempre aggiornati ricevendo periodicamente delle *newsletter*, avendo sempre la possibilità di disabilitare questa funzione. I dati degli utenti e degli eventi sono contenuti all'interno di un database relazionale. Per rendere disponibile il social su diverse piattaforme, oltre ad utilizzare un *web server*, è stato utilizzato *Flutter* e *Flutter Web*, un *SDK* di *Dart* che consente di convertire siti web in applicazioni e viceversa.

Inglese:

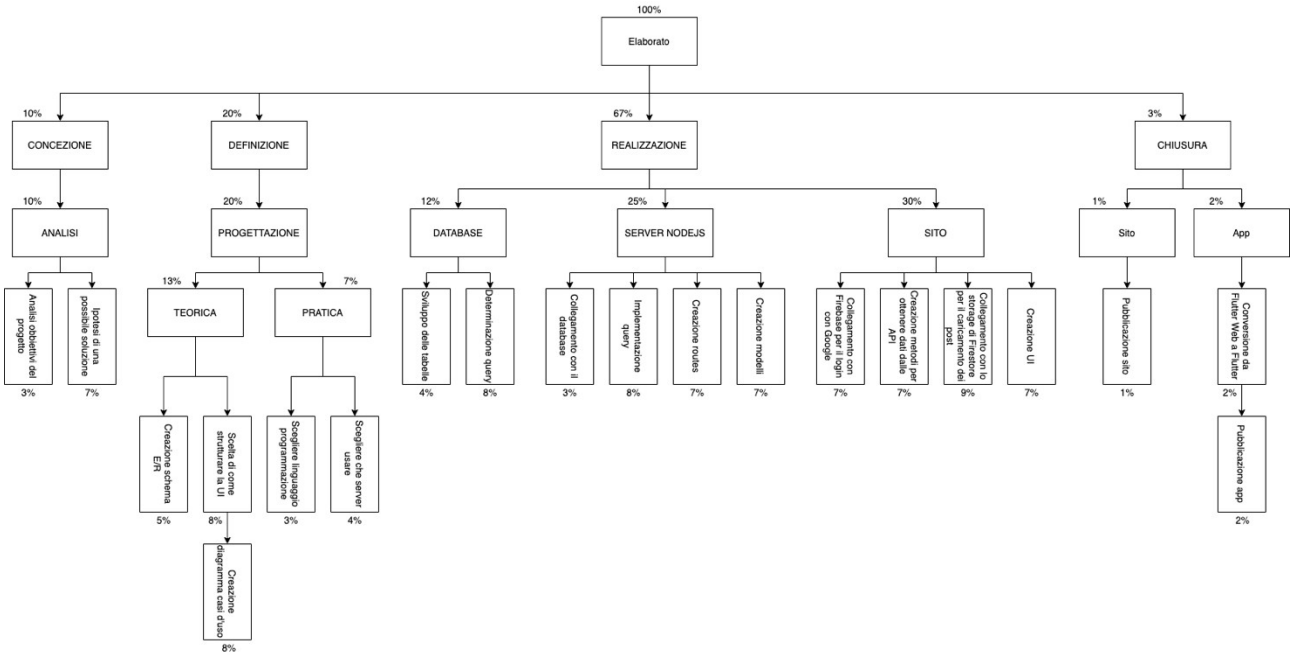
The paper aims to deal with the creation of a *web community* to share data and comments relating to living events of different genres (sports, art, etc.). Two types of users can access the *social network*. The first is that of anonymous users, who can only consult the site or application, viewing all the events, both those scheduled and those already carried out. The second is that of registered users who, after logging in with a Google account, will have a complete profile within the *social network*. Unlike anonymous users, registered users have the ability to interact with published events, see only some of them based on certain parameters, or even publish others. They can also stay up to date by receiving *newsletters* periodically, while always maintaining the ability to disable this function. User and event data are contained within a relational database. To make social media available on different platforms, in addition to using a *web server*, *Flutter* and *Flutter Web*, a *Dart SDK* that allows you to convert websites into applications and vice versa, were used.

Indice

1. Analisi obiettivi e soluzioni adottate

Il servizio mette a disposizione all'utente finale la possibilità di autentificarsi. Questa funzionalità è stata implementata tramite i servizi Google offerti da **Firebase** e un server in **NodeJS**, che fa riferimento a un database **MySQL** di tipo relazionale. Inoltre, deve consentire agli utenti registrati la possibilità di pubblicare un evento o interagire con uno già pubblicato tramite un commento, un like o una valutazione. La pubblicazione avviene tramite una funzione che, dopo il caricamento dell'immagine all'interno dello Storage offerto dai servizi Google, va a copiare il link di riferimento della stessa e va a inserirlo all'interno del database **MySQL**. Le interazioni con gli eventi, invece, avvengono tramite richieste http al server, il quale successivamente va a cambiare o caricare dei parametri all'interno del database. Un'altra richiesta è quella di inviare in maniera automatica delle newsletter agli utenti registrati, evasa tramite un plug-in del server associato ad una funzione periodica. Infine per rendere disponibile l'accesso ad ogni dispositivo si è andati ad *hostare* il tutto all'interno di un web server realizzato con **Ubuntu**.

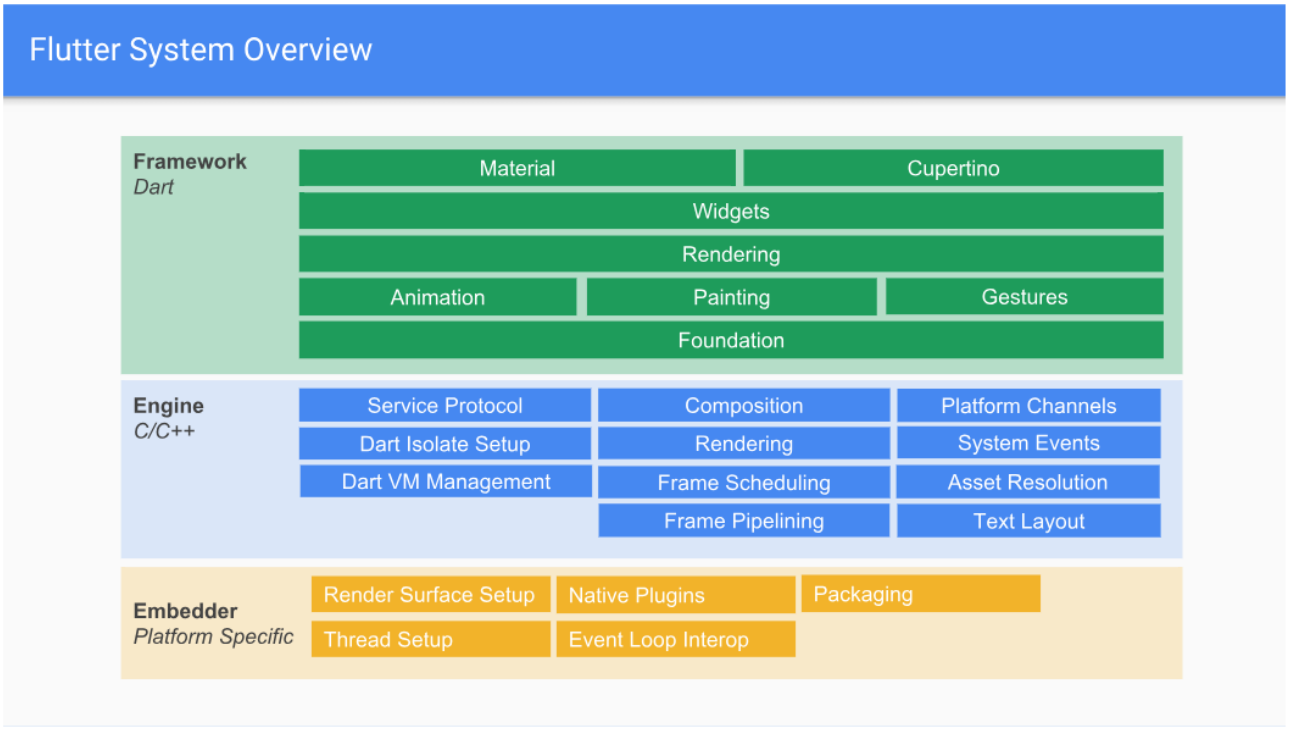
2. WBS



Click qui per visualizzare l'immagine in grande. (Per avere l'accesso è necessario accedere al file con l'account scolastico, esempio: nome.cognome@itiszuccante.edu.it)

3. Linguaggio di programmazione

Il progetto è stato scritto utilizzando il linguaggio di programmazione **Dart**, sviluppato da Google, tramite l'SDK **Flutter Web**, anch'esso sviluppato da Google, in modo da poter offrire all'utente finale la possibilità di accedervi sia da un dispositivo desktop, tramite sito web, che da un dispositivo mobile, tramite un'applicazione. L'obiettivo di Dart e, nello specifico, di Flutter Web è quello di andare a soppiantare l'ormai datato **JavaScript.** I programmi Dart possono essere eseguiti direttamente, mentre sul browser vengono convertiti in JavaScript mediante il **transcompiler Dart2js**. Nella figura che segue si può vedere nel dettaglio la struttura dell'architettura definita per Flutter.



L'architettura di Flutter si compone di tre macro blocchi principali composti a loro volta da **API** e librerie che caratterizzano ogni strato. Nel dettaglio la funzione dei tre macro blocchi si può schematizzare in:

Embedder – Platform Specific	È il livello più basso dell'architettura di Flutter ed è il cuore dell'Engine di Flutter. In questo strato vengono definiti gli embedder specifici per le piattaforme, che hanno lo scopo di legare tra loro il rendering alla gestione degli eventi di input. Per fare ciò, gli embedder interagiscono con il layer di Engine tramite delle API C/C++ di basso livello.
Engine	Questo strato intermedio è il lato C/C++ di Flutter ed è definito nel repository engine. In particolare, l'Engine include molteplici componenti di basso livello, fondamentali per il funzionamento del framework e delle sue operazioni di base.
Framework	È lo strato più importante per gli sviluppatori e offre tutte le librerie e i pacchetti necessari per lo sviluppo di un'applicazione o di un sito. Tra questi vi sono i layer relativi alle animazioni, alla definizione delle gesture, e alla creazione dei widget. Infatti, il layer Widget permette la definizione dei layer Material e Cupertino per la realizzazione dei componenti grafici secondo lo stile Android e iOS, rispettivamente, e di definire dei Widget personalizzati.

La strategia di Flutter, "tutto è un widget", applica la programmazione orientata agli oggetti a tutto, includendo l'interfaccia utente: l'interfaccia di un programma è così composta da vari **widget**, che possono essere nidificati gli uni negli altri. Ogni pulsante e testo visualizzato è un widget contenente diverse caratteristiche che possono essere modificate. I widget possono influenzarsi a vicenda e reagire, tramite funzioni integrate, a cambiamenti di stato dall'esterno.

É possibile creare widget personalizzati che possono essere combinati perfettamente con quelli esistenti. Rispetto agli strumenti di altri **SDK**, i widget offrono molta più flessibilità, ma hanno lo svantaggio di essere tutti situati nel codice sorgente del programma che risulta, pertanto, fortemente nidificato e intricato.
E' stato scelto Flutter, e nello specifico su Flutter Web, perché consente di creare un interfaccia web e un'interfaccia mobile mantenendo la stessa **UI** e le stesse funzioni, in modo da avere su entrambe le piattaforme un sistema che funziona in maniera analoga. Inoltre, tutte le funzioni di autentificazione e **storage Google** sono completamente implementate e funzionano egregiamente senza riscontrare problemi di compatibilità. In più, la programmazione basata su Widget (citata precedentemente) consente una realizzazione grafica semplice, ma allo stesso tempo completa e piacevole.

Tra i possibili linguaggi di programmazione per ottenere un risultato analogo si sarebbe potuto utilizzare **PHP** con framework **Lavarel** e **bootstrap**, ma si è scelto Flutter Web perché consente un'ottima implementazione con i servizi Google.

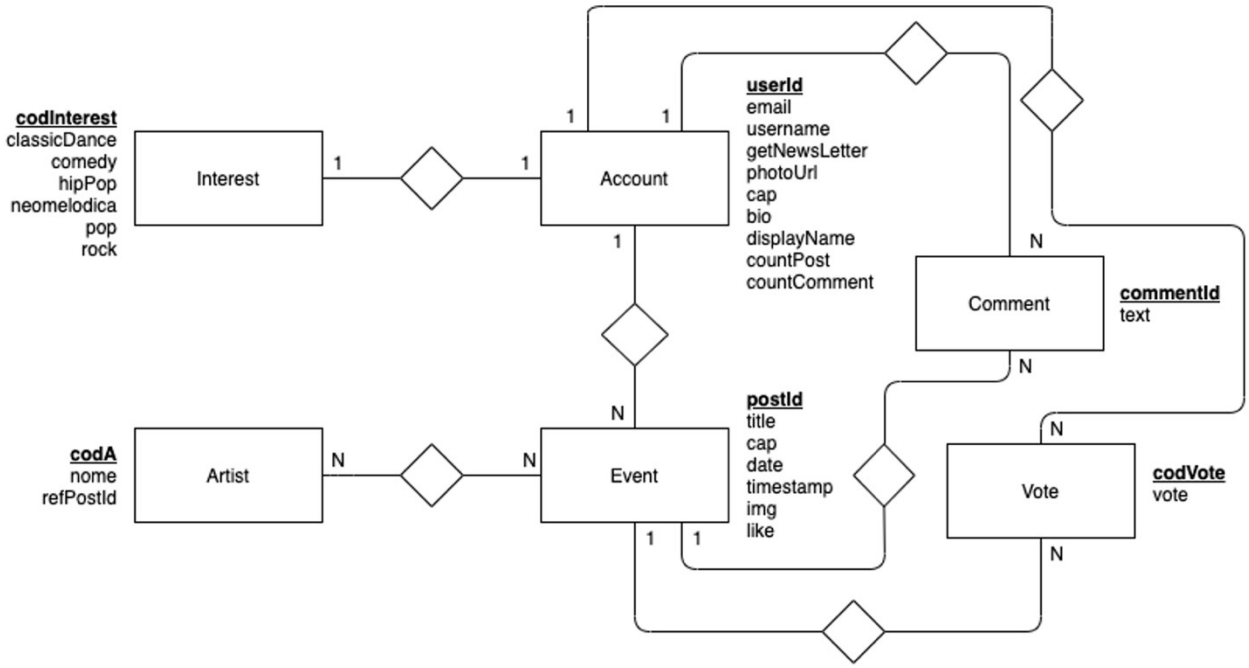
4. Database e server

4.1 Tecnologie usate

Come database si è utilizzato un database relazionale **MySQL**, *hostato* su **server Ubuntu**, che si appoggia su un server **NodeJS express**. La creazione del database è stata affiancata da un DBMS (Database Managment System), nello specifico di **phpMyAdmin**, che consente la creazione e gestione del database sia con un'interfaccia grafica che tramite apposite query in linguaggio SQL.
NodeJS è una piattaforma realizzata su **V8**, il motore **Javascript** di Chrome, che permette di realizzare applicazioni web veloci e scalabili. Node usa un modello ad eventi e un **sistema di I/O non bloccante** che lo rende leggero ed efficiente, perfetto per **applicazioni real-time** che elaborano dati in modo intensivo e che può essere distribuito su più sistemi. In generale, tutto il modello di programmazione di Node è pervaso da **callback**: si parla quindi di **programmazione asincrona**, governata da eventi. La scelta è ricaduta in questo server grazie alla sua ottima versatilità e leggerezza, come detto precedentemente. Inoltre, il **micro-framework ExpressJS** consente la creazione di **API di tipo rest** tramite JavaScript in maniera semplice e veloce, ottenendo come risultato, dopo una richiesta **HTTP**, delle informazioni sotto forma di **JSON** (Facilmente convertibile in valori utilizzabili tramite Flutter), ottime per l'interazione desiderata.

4.2 Schema E/R

Il database è stato fatto tenendo in considerazione alcuni parametri che vengono mostrati in alcune pagine. Per esempio, nella tabella "Account" sono stati aggiunti i parametri "countPost" e "countComment" per semplificare la costruzione della griglia dei *post*. La tabella "vote" è stata pensata come una "mappa" in modo da poter ottenere in maniera semplice l'esatto numero di voti che ha ottenuto un determinato Evento. La tabella "Event" ha un timestamp. Il suo funzionamento è quello di salvare la data e l'ora della pubblicazione, in modo da poterli riordinare in ordine di pubblicazione nel momento in cui si va a rappresentarli graficamente.



4.3 Schema logico

Grossetto e sottolineato = Chiave Primaria

Corsivo e grigio = Chiave esterna

Account (**userId**, email, username, getNewsLetter, photoUrl, cap, bio, displayName, countPost, countComment, *codInterest*)
Event (**postId**, title, cap, date, timestamp, img, like, *userId*)

Artist (**codA**, name, *postId*, *event*)

Comment (**commentId**, text, *postId*, *userId*)

Vote (**codVote**, vote, *postId*, *userId*)

Interest (**codInterest**, classicDance, comedy, hipPop, neomelodica, pop, rock, *userId*)

5. Sicurezza

Per implementare la sicurezza all'interno del sistema si è deciso di utilizzare vari metodi differenti. Il primo consiste nel login tramite account Google, in modo da non tenere la password dell'utente all'interno del database **MySQL** ed evitare che vengano rubate in seguito ad un attacco hacker, risolvendo così anche il problema degli attacchi **bruteforce**. Quando si esegue il login o la registrazione a un sito web dopo aver eseguito l'accesso a Chrome (O qualsiasi browser abilitato per il login con Google), quest'ultimo cripta il nome utente e la password con una chiave segreta nota soltanto al proprio dispositivo. Poi invia a Google una copia criptata dei dati. Poiché la crittografia avviene prima che i server di Google ricevano le informazioni, nessuno, inclusa Google, sa quali siano il nome utente o la password. Questo consente, oltre ad avere una sicurezza dei dati sensibili ineguagliabile, anche un'ottima protezione della privacy. Nel caso in cui ci sia stata una violazione dei dati, il browser compatibile con i servizi Google invia le credenziali criptate a Google per un confronto con un elenco criptato dei dati violati. Se il browser rileva una corrispondenza tra un set di dati criptato, viene visualizzato un avviso che chiede di cambiare la password. Google non memorizza mai i nomi utente o le password durante questa procedura.

In aggiunta, un'altra accortezza che è stata fatta è quella di non far accedere il server al database con l'account **root**, ma utilizzare un account appositamente creato utilizzando una password a 128 bit, in modo da non mettere a rischio il totale database in caso di un eventuale attacco hacker. Un'altra accortezza è stata quella di hostare le foto all'interno di un drive, in questo caso *Storage di Google*, che consente l'accesso alla foto da un account non proprietario del drive solamente tramite **token**, un "link" creato in maniera casuale impossibile da ottenere tramite un attacco bruteforce essendo un "link" di 142 caratteri, e comunque resta revocabile e ricreabile in maniera casuale.

Per incrementare la sicurezza a livello server invece si è utilizzato un *plug-in* per NodeJS, **Helmet**, che consente, tramite l'utilizzo di nove funzioni middleware, di configurare le intestazioni HTTP in modo appropriato. Nello specifico le principali funzioni sono:

- **csp**: imposta l'intestazione Content-Security-Policy per impedire attacchi **XSS** (cross-site scripting). Il Cross-site scripting è uno sfruttamento di vulnerabilità nelle applicazioni web. Gli script dannosi vengono inseriti all'interno del sito, così facendo gli hacker riescono ad accedere ad informazioni riservate o al computer dell'utente. Per fare sì che le pagine Internet possano essere utilizzate in modo esaustivo senza correre rischi, è stata introdotta la Content Security Policy (CSP). Questo standard di sicurezza ha l'obiettivo di difendere i siti web dagli attacchi cibernetici e, contemporaneamente, di proteggere gli utenti. Il compito della CSP è quello di bloccare tutti gli script scritti *inline* all'interno del codice, o per lo meno tutti quegli script che non sono stato inseriti all'interno della *whitelist* del protocollo durante la programmazione.
- **hidePoweredBy**: rimuove l'intestazione X-Powered-By, che è di default abilitata e va a mostrare informazioni "sensibili", come chi ha creato il sito o che tecnologia ha usato, che potrebbero essere usate per effettuare un attacco hacker mirato.
- **hsts**: imposta l'intestazione Strict-Transport-Security che rafforza connessioni (HTTP su SSL/TLS) sicure per il server. HSTS (HTTP Strict Transport Security) è un meccanismo di sicurezza che è stato sviluppato per proteggere le connessioni HTTPS contro gli attacchi *man in the middle*, questo meccanismo consente di comunicare ai browser che un sito può essere richiamato per un arco di tempo definito esclusivamente per mezzo della crittografia SSL/TLS. Per questo viene utilizzato il campo di intestazione Strict-Transport-Security lato server, che comprende la direttiva obbligatoria *max-age*, indica per quanto tempo un sito deve rimanere a disposizione crittografato. Nello specifico succede questo: se un utente visita un sito protetto da HSTS per la prima volta, il browser crittografa tutti i link che non sono crittografati facendoli diventare da *http* a *https*, mentre nel caso in cui la sicurezza di una connessione non può essere garantita, l'utente visualizzerà un messaggio di errore.
- **noSniff**: imposta X-Content-Type-Option per impedire l'utilizzo e il funzionamento di uno *sniffer*. Si tratta infatti di un software comunemente utilizzato per monitorare e analizzare il traffico di rete, al fine di rilevare problemi e mantenere il sistema efficiente. Tuttavia, gli *sniffer* possono essere utilizzati anche per scopi illegali. Registrano, infatti, tutto ciò che incontrano, inclusi i nomi utente e le password non criptate, pertanto possono essere sfruttati dagli hacker per accedere a qualsiasi account.
- **frameguard**: imposta l'intestazione X-Frame-Options per fornire la protezione al *clickjacking*. ll *clickjacking* è una tecnica utilizzata dagli hacker per spingere gli utenti in siti pericolosi tramite un click su un link a loro insaputa. Oltre al reinderizzamento a un sito esterno, possono essere intercettati i tasti premuti, in modo da poter scoprire i tasti cliccati per riuscire ad arrivare a una ipotetica password. X-Frame-Options disabilita i tag html *<frame>* e *<iframe>*, evitando così il caricamento di una pagina esterna all'interno di un frame.

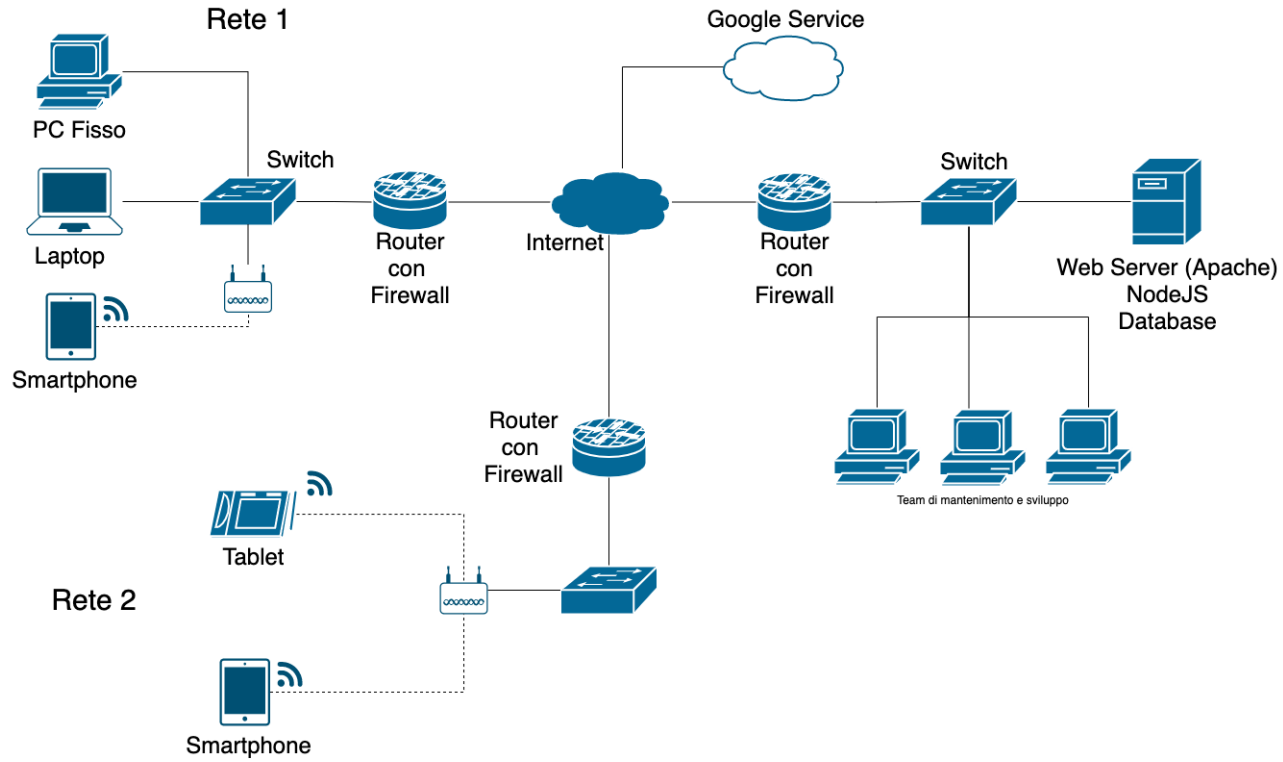
Va tenuta in considerazione anche la sicurezza applicata al server Ubuntu utilizzato per l'hosting del sito e delle sue funzionalità. Questo argomento viene approfondito nella prossima sezione.

6. Hosting e Server Ubuntu

6.1 Tecnologie usate

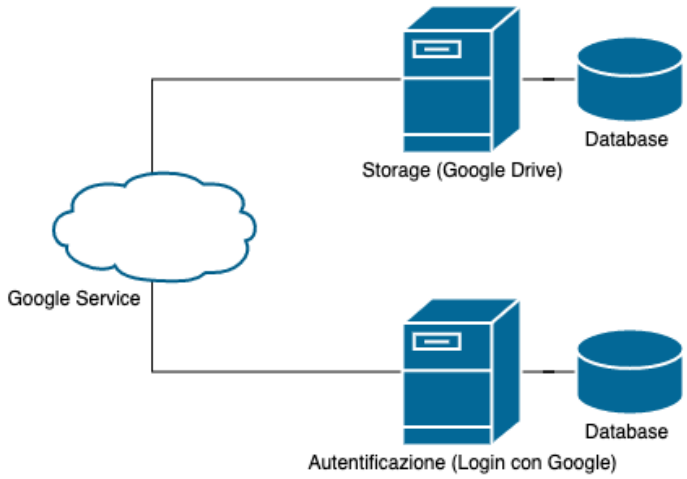
Per hostare il database MySQL e i due siti web si è utilizzato un Server Ubuntu creato tramite macchina virtuale. Nello specifico, si è andati ad utilizzare un Web Server che ha il compito di gestire le richieste di un client relative al trasferimento di pagine web. Il protocollo che si è andati ad utilizzare è il protocollo **HTTPS** (Hypertext Transfer Protocol Secure). La scelta è ricaduta su questo protocollo al posto di un semplice **HTTP** perché, essendo la versione successiva, garantisce una maggiore sicurezza. Infatti, lo stesso garantisce che i dati inviati vengano protetti tramite il protocollo Transport Layer Security (**TLS**), che fornisce tre livelli di protezione fondamentali: **Crittografia**, **Integrità dei dati** e **Autentificazione**.
Il TLS, che è il successore di SSL, crittografa il flusso di dati sulla rete, affinché questi siano letti soltanto dai legittimi destinatari.
In più si è andati a creare una **CA locale** per fare in modo che, previo inserimento manuale, il sito HTTPS risulti certificato e che TLS si attivi.

6.2 Infrastruttura di rete



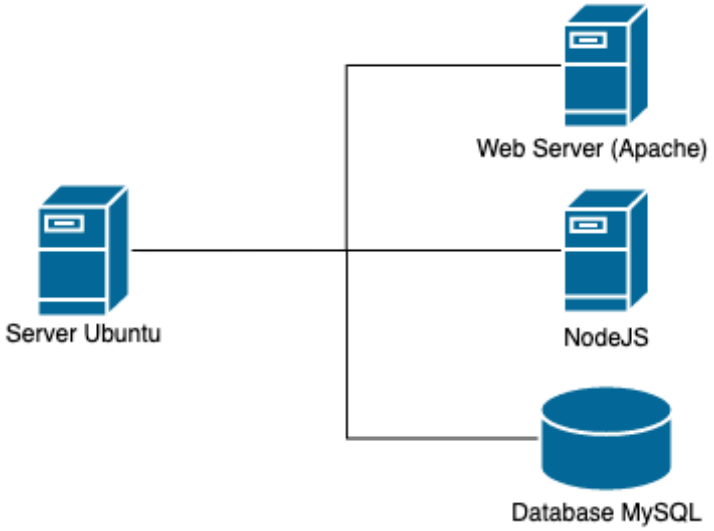
Click qui per visualizzare l'immagine in grande. (Per avere l'accesso è necessario accedere al file con l'account scolastico, esempio: nome.cognome@itiszuccante.edu.it)

2.6 Struttura Google Service



Click qui per visualizzare l'immagine in grande. (Per avere l'accesso è necessario accedere al file con l'account scolastico, esempio: nome.cognome@itiszuccante.edu.it)

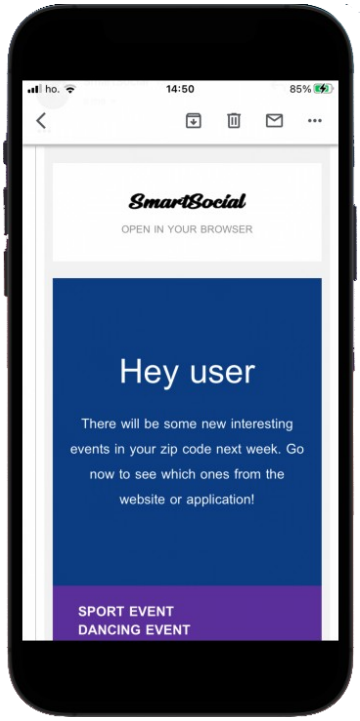
2.7 Struttura Web Server

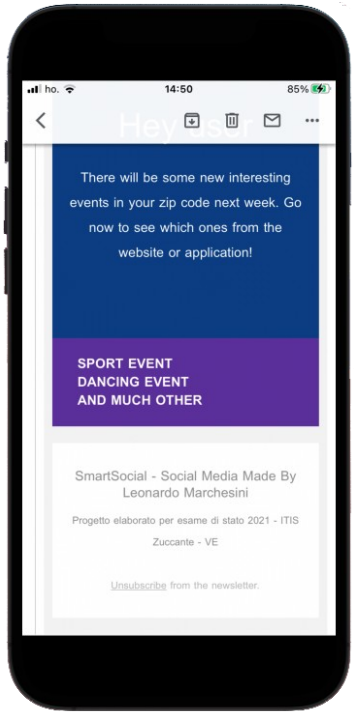


Click qui per visualizzare l'immagine in grande. (Per avere l'accesso è necessario accedere al file con l'account scolastico, esempio: nome.cognome@itiszuccante.edu.it)

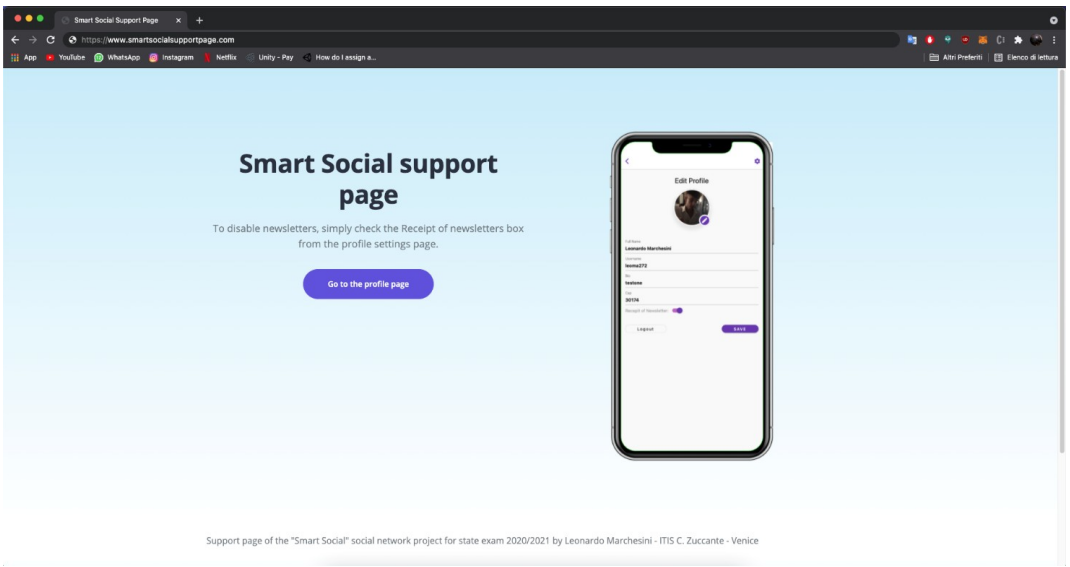
7. Invio delle email

Il sistema mette a disposizione l'invio delle email automatico tramite il plug-in nodemailer. Per evitare che le newsletter finiscano su spam si è andati a richiedere il certificato di autenticità tramite API Google, che, dopo aver registrato il sito web, si ottiene un codice OAuth2 per andare a richiedere l'autorizzazione da parte di Gmail per l'invio automatico delle email. Tutte le email inviate saranno a nome *leonardo.marchesini@itiszuccante.edu.it*, con l'acronimo "SmartSocial", e avranno un form HTML all'interno modellato appositamente per essere visto sia da smartphone che da PC.





Si può notare che in basso all'email è presente la scritta "Unsuscribe from the newsletter" in un font color grigio, cliccando su quella scritta si verrà reinderizzati a una pagina dove saranno presenti le istruzioni su come andare a disattivare le newsletter.

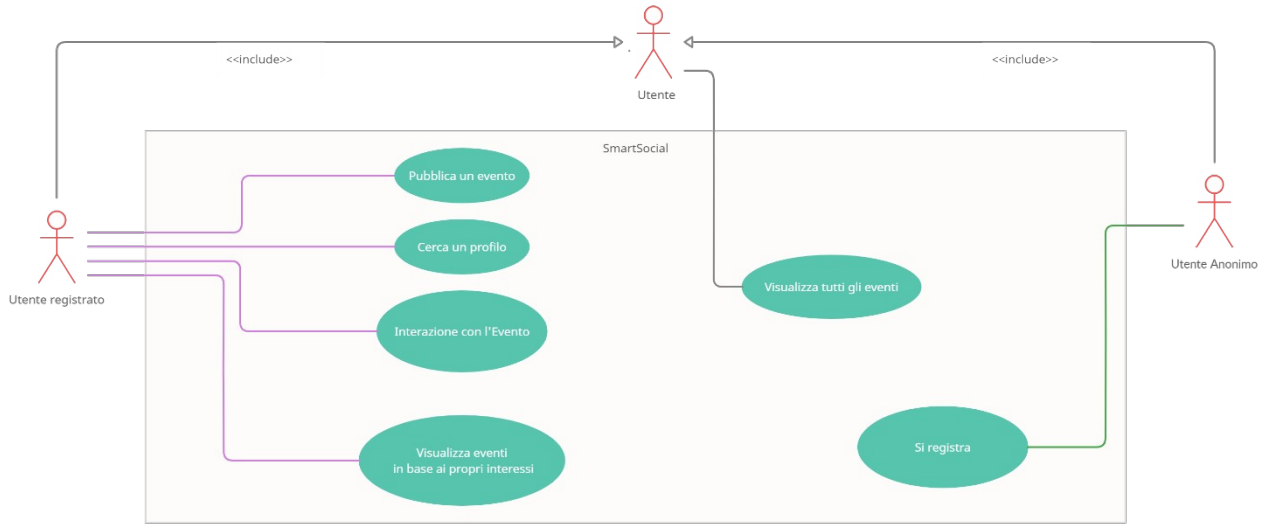


Click qui per visualizzare l'immagine in grande. (Per avere l'accesso è necessario accedere al file con l'account scolastico, esempio: nome.cognome@itiszuccante.edu.it)

Il metodo che viene usato per evitare che le email finiscano per essere catalogate come spam consiste nella creazione di un *token* di riferimento temporaneo, questo *token* viene utilizzato da Gmail per l'invio dell'email e ha valenza di un'ora. Il plug-in *nodemailer* consente la creazione automatica di questo *token*, per evitare che venga creato ogni volta che scade manualmente, e consente l'invio automatizzato dell'email (per esempio, ogni volta che si pubblica un post o ogni settimana ad una determinata ora).

8. Metodi e Codice

8.1 Diagramma dei casi d'uso



Click qui per visualizzare l'immagine in grande. (Per avere l'accesso è necessario accedere al file con l'account scolastico, esempio: nome.cognome@itiszuccante.edu.it)

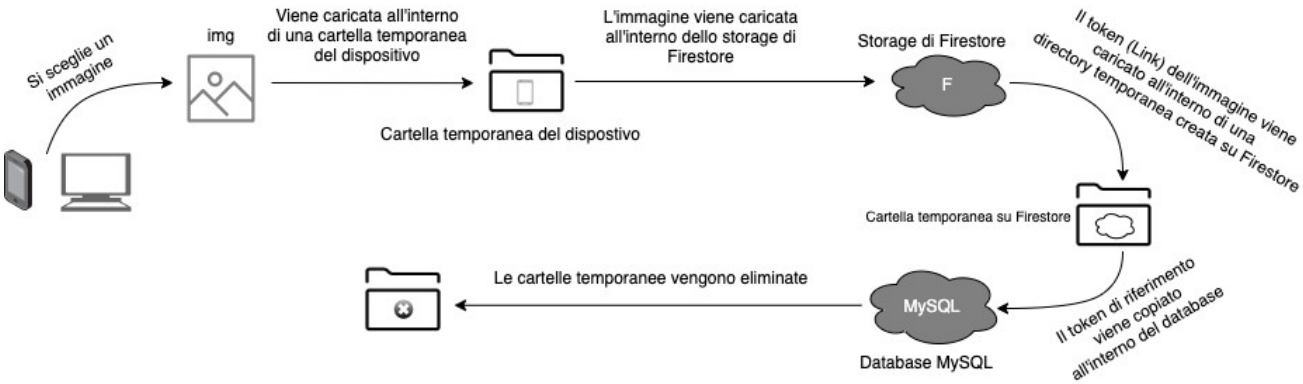
8.2 Schermata inziale, login e permessi

Una volta entrati all'interno del sito o dell'applicazione mobile verranno date due possibilità di accesso. La prima è quella di accedere anonimamente, quindi senza dover effettuare nessuna registrazione o inserire credenziali. L'accesso anonimo consente una possibilità limitata di navigazione all'interno del social network, infatti sarà possibile solamente visualizzare gli eventi pubblicati dagli altri utenti, senza la possibilità di poterci interagire. La seconda opzione è quella del login con un account Google, questo servizio è stato implementato grazie ad una funzione presente all'interno di Firebase, che verrà utilizzato solo per il riconoscimento e per lo storage dell'immagine dell'evento, che nel caso in cui venisse fatto per la prima volta sarà necessario inserire l'username che si vuole utilizzare, il CAP in cui si abita o di cui si vuole avere notizie relative agli eventi e le categorie a cui si è interessati (Le categorie sono a numero finito e comprendono gli ambiti di gusti musicali, danza, scene teatrali e sport). Dopo che si avranno completato tutti i campi richiesti viene inviata una richiesta *post* al server che andrà a comunicare al database che si vuole aggiungere un account, contente i dati dell'account Google forniti dallo stesso tramite l'autentificazione e le scelte effettuate nella pagina di registrazione. Tutta la parte del login viene gestita dai metodi *handeSignIn()*[1], *createUserInFirestore()*[2], *makePostRequest()*[3] e *makePostRequestInterest()*[4]. Nel caso in cui invece l'utente sia già stato registrato verrà semplicemente reindirizzato all'interno del social network senza il bisogno di effettuare il login con Google, a patto che il browser abbia impostato come profilo predefinito quello con cui si è l'account.

I vantaggi che ha un utente registrato rispetto a uno anonimo sono molteplici, tra cui: Interagire con le pubblicazioni altrui inserendo un commento, un "like" o un voto che va da 1 a 5, ricercare gli utenti inserendo il loro username, pubblicare un evento ed avere una visione completa del proprio profilo tramite la pagina apposita.

8.3 Creazione di un evento e pubblicazione di una foto

L'utente registrato, tra le molte funzionalità in più rispetto ad un anonimo, ha la possibilità di creare e pubblicare un evento. Nel momento in cui si vuole fare una pubblicazione viene data la possibilità all'utente di scegliere se scattare una foto in quel momento, nel caso in cui il computer che si sta utilizzando ha una webcam collegata o se viene utilizzata l'applicazione mobile, o di caricarne una già presente all'interno del computer o dalla galleria dello smartphone, entrambe queste funzionalità sono possibili grazie al pacchetto *Image Picker*. Dopo aver scattato o selezionato l'immagine desiderata verrà creata una directory temporanea, grazie al pacchetto *path_provider* in cui ci sarà il collegamento temporaneo all'immagine, che consentirà il caricamento all'interno dello *storage* di *Firestore*. La gestione della selezione dell'immagine viene fatta dalle funzioni *handleTakePhoto()* [5] e *handleChooseFromGallery()*[6]. Successivamente sarà richiesto di inserire le informazioni riguardati l'evento (Titolo dell'evento, categoria, CAP di dove si svolgerà, data di svolgimento e artisti coinvolti), una volta inserite tutte le informazioni l'immagine sarà caricata all'interno dello *storage* di *Firestore* e il suo *token* (Link) di riferimento verrà temporaneamente salvato all'interno di una raccolta temporanea, tutto questo procedimento viene fatto dalla funzione *handleSubmit()*[7] che contiene al suo interno le due funzioni *createPostInFirestore()*[8] e *compressImage()*[9], questo procedimento viene fatto per permette poi, inviando una richiesta *get* a *Firestore*, di ottenere il *token* dell'immagine e tramite una richiesta *post* al server, fatta grazie al metodo *uploadEvent()*[10], verrà aggiunto al database un elemento all'interno della tabella *Eventi* contenente oltre alle informazioni inserite dall'utente il *token* di riferimento dell'immagine, ottenuto precedentemente, il contatore (Inizializzato a zero) dei *like* dell'evento e il *timestamp* (Data e ora di pubblicazione). Una volta che il processo è stato completato la raccolta temporanea all'interno del database *Firestore* verrà eliminata, mantenendo però l'immagine all'interno dello *storage*, per consentire così l'accesso tramite token.
Nell'immagine che segue si può vedere graficamente il procedimento di caricamento dell'immagine:



Click qui per visualizzare l'immagine in grande. (Per avere l'accesso è necessario accedere al file con l'account scolastico, esempio: nome.cognome@itiszuccante.edu.it)

8.4 Home Page, Visualizzazione e Interazione degli Eventi

Sia l'utente registrato che l'utente anonimo hanno la possibilità di visualizzare tutti gli eventi pubblicati. Funzionalità resa possibile dalla funzione *getAllPost()*[11], il quale compito è quello di ottenere dalle API, messe a disposizione dal server NodeJS express, i riferimenti di tutti gli eventi postati dagli utenti, in ordine cronologico di pubblicazione, e i dati dell'account che ha effettuato la pubblicazione. Una volta che gli eventi sono stati caricati è consentito all'utente registrato interagirvi, lasciando per esempio un *like*, *commento* o una *valutazione* che va da 1 a 5. Tutte queste funzioni sono possibili grazie ai relativi metodi *post* (*updateLikeByPostId()*[12], *postComment()*[13] e *postStar()*[14]), che funzionano inviando una richiesta *post* al server contenente il valore che si vuole aggiungere o modificare della relativa tabella del database. In particolare il metodo *postComment()* va a creare un entità commento all'interno della tabella *comment*, questa entità avrà come parametri i riferimenti di chi lo vuole lasciare e i riferimenti del post su cui si vuole lasciare il commento. Oltre a pubblicarli è possibile anche vedere i commenti fatti dalle altre persone grazie al metodo *getPostComment()*[15] che consente di avere una lista contenente tutti i commenti, con i relativi dati dell'utente che l'ha scritto, lasciati a quel post. La funzione *postStar()*invece va ad aggiungere all'interno della tabella *vote* il codice utente, un intero da 1 a 5 e il riferimento del post su cui è stato lasciato il voto, facendo così è possibile trovare il voto medio, che sarà rappresentato sotto forma di stelline, semplicemente andando a trovare la somma di tutte le valutazioni rilasciate a quel post e dividerla per il numero di voti presenti relativi a quel post.

8.5 Visualizzazione mirata

Nella pagina *rule* è possibile visualizzare alcuni post o alcuni account in base a determinati parametri. Nello specifico è possibile visualizzare:

- Elenco dei membri che non hanno mai inserito un commento. Consentito grazie alla funzione *getUserByNoComment()*.[16] Funzione che va a richiamare la seguente query:

```
SELECT * FROM account LEFT JOIN comment ON account.id =
comment.userId WHERE text IS NULL
```

- I dati dell'utente che ha registrato il maggior numero di eventi. Consentito grazie alla funzione *getUserMaxPost()*.[17] Funzione che va a richiamare la seguente query:

```
SELECT * FROM account, post WHERE account.id = post.ownerId
GROUP BY post.ownerId HAVING MAX(post.ownerId)
```

- Elenco degli eventi, sia già svolti che in programmazione, di un determinato cap. Consentito grazie alla funzione *getPostByCap()*.[18] Funzione che va a richiamare la seguente query:

```
SELECT * FROM post WHERE place = (SELECT cap FROM account
WHERE id = currentUser.id)
```

Tutte queste funzionalità vengono rese possibili grazie a delle query che vanno a interrogare il database richiedendo un post o un account che soddisfi determinate caratteristiche.

8.6 Profilo, resoconto delle pubblicazioni e modifiche

L'utente registrato ha la possibilità di visualizzare il proprio profilo con tutte le informazioni, interessi e tutti gli eventi postati, questo è possibile grazie alle tre funzioni, *getUser()*[19], *getInterest()*[20] e *getPostById()*[21], che vanno a fare delle richieste get al server per ottenere dalle API i dati relativi ad un determinato utente. Inoltre dalla pagina profilo è possibile accedere ad una pagina di impostazioni, dove si possono modificare tutte le voci che saranno visibili agli altri utenti (Nome e cognome, username, bio, interessi) tutte queste modifiche sono possibili grazie alla funzione *changeInfo()*[22]. Inoltre c'è la possibilità di disattivare la ricezione delle newsletter, inviate automaticamente, grazie al metodo *removeNewsLetter()*, che si appoggia a *changeInfo()*.

9. Commento personale

9.1 Eventuali Migliorie:

Nonostante la presenza delle funzioni di base, il social network potrebbe essere migliorato con l'aggiunta di funzioni esterne. Ad esempio, si potrebbe implementare una "chat" in tempo reale tra gli utenti registrati, con magari la possibilità di citare un determinato evento.

9.2 Difficoltà riscontrate:

Durante la realizzazione dell'elaborato sono state riscontate difficoltà relative alla pubblicazione degli eventi e le relative immagini, sopratutto nel trasferirle all'interno del database relazionale. Nonostante questo il resto del lavoro è riuscito senza troppi problemi, anche grazie alle competenze extrascolastiche acquisite durante l'anno.

9.3 Competenze acquisite e autoapprendimento:

Grazie a questo progetto è stato realizzato un sistema completo che rende possibile la comunicazione tra varie tecnologie e piattaforme (Smartphone, PC, Tablet, ecc.). Per portarlo a compimento è stato utilizzato in maniera sostanziale l'autoapprendimento affiancato alle competenze acquisite durante l'anno scolastico. L'elaborato si è rilevato utile nell'imparare come unire tutte le varie tecnologie e competenze, acquisite fino ad ora in maniera separata, cosa di sicuro fondamentale per un eventuale futuro lavorativo.

10. Progetto GitHub

Link alla repository contenente:
https://github.com/leonardoMarchesini/elaboratoLeonardoMarchesini.git

- Sito Web
- Applicazione
- Server

11. Legenda

Tutti i metodi citati con le due parentesi quadre blu, [n], indicano il riferimento della riga del file *method.dart*, che si trova all'interno della cartella *lib->database->method.dart*

[1] = Riga 55	[6] = Riga 148	[11] = Riga 231	[16] = Riga 288
[2] = Riga 69	[7] = Riga 158	[12] = Riga 245	[17] = Riga 302
[3] = Riga 96	[8] = Riga 178	[13] = Riga 255	[18] = Riga 312
[4] = Riga 117	[9] = Riga 192	[14] = Riga 266	[19] = Riga 327
[5] = Riga 136	[10] = Riga 211	[15] = Riga 277	[20] = Riga 338
[21] = Riga 349	[22] = Riga 365		

12. Bibliografia e Riferimenti

Stesura della teoria:

- Database:
  - Appunti presi in classe
- Linguaggio di programmazione:
  - https://flutter.dev
- Sicurezza:
  - https://ionos.com
  - https://webtechsurvey.com
  - https://html.com
  - https://codingjam.it
- Server:
  - https://redhat.com