

UNIVERSITÀ DELLA CALABRIA

DIPARTIMENTO DI MATEMATICA E INFORMATICA



Corso di Laurea in Informatica

Link 4 Students - Portale di incontro per studenti realizzato con Svelte e Firebase

Relatore:

Prof. Mario Alviano

Candidato:

Daniele Avolio

Mat. 201018

ANNO ACCADEMICO 2020/2021

Our journey may have been meaningless.

Our past may have been a mistake.

But we're not going back.

Even if this world comes to an end.

Because this...

This is the world with the people we cherish.

(Yoko Taro)

Sommario

In questo lavoro di tesi è stato progettato e implementato un sito web in cui gli studenti hanno la possibilità di interagire con i contenuti dei Corsi di Laurea. Le feature principali permettono di gestire gli esami sostenuti e di poter chiedere aiuto ad altri studenti che hanno sostenuto o devono sostenere l'esame. Inoltre, il sistema consente il caricamento di appunti e altro materiale didattico da associare ai diversi esami, in modo da rendere tali risorse disponibili a tutti gli utenti. Il profilo di ogni studente è personalizzabile ed è possibile decidere se mostrare o meno i contenuti agli altri studenti. I contenuti del sito vengono revisionati dagli amministratori con strumenti appositi all'interno di un'apposita dashboard.

Indice

Indice	3
Elenco delle figure	5
1 Introduzione	7
1.1 Contesto e Motivazione	7
1.2 Obiettivi	8
1.3 Contributo principale	8
1.4 Problemi tecnici riscontrati	9
2 Tecnologie Utilizzate	11
2.1 Firebase	11
2.1.1 Database orientato ai documenti	11
2.2 Svelte	14
2.3 Sveltekit	17
2.4 HTML E CSS	17
2.5 Javascript	18
2.6 Node e NPM	19
3 Contributo principale	21
3.1 Panoramica e struttura del sito	21
3.1.1 Pagina del profilo	23
3.1.2 Sezione dei Corsi di Laurea	24
3.1.3 Sezione degli insegnamenti	25
3.1.4 Sezione degli annunci	26

<i>INDICE</i>	4
3.2 Configurazione iniziale di Firebase	26
3.3 Lettura dei dati da Firebase	26
3.4 Funzionalità principali	29
3.4.1 Gestione della media degli esami	29
3.4.2 Gestione della difficoltà e utilità degli esami	36
3.4.3 Gestione dei contenuti tramite dashboard	40
4 Discussione	50
4.1 Elevato numero di letture	50
4.2 Backend e sicurezza	55
4.2.1 Express.js e Firebase	55
4.3 Ulteriore soluzione per integrità dei dati	56
5 Conclusioni e sviluppi futuri	61
5.1 Integrazione con Esse3	62
5.2 Espansione ad ulteriori Università	63
Bibliografia	65

Elenco delle figure

2.1	Esempio di Documento in Firebase	13
2.2	Esempio di cambio valore variabile in React	14
2.3	Esempio di cambio valore variabile in Svelte	15
2.4	Costruzione dei Component BoxLink	17
2.5	Esempio di package.json	20
3.1	Homepage di Link4Students	22
3.2	Struttura del Component BoxLink	23
3.3	Pagina del profilo utente	24
3.4	Pagina dei Corsi di Laurea	25
3.5	Pagina degli Insegnamenti	25
3.6	Esempio di richiesta Firebase	27
3.7	Esempio di getDocs() per le informazioni utente	28
3.8	Esempio pagina Insegnamento	30
3.9	Grafico chiamate	32
3.10	Grafico del problema	32
3.11	Funzione di aggiunta esame al libretto	33
3.12	Funzione di calcolo media dell'esame	35
3.13	Form per la recensione	36
3.14	Struttura Documento e Component	37
3.15	Esempio di libertà di accesso	37
3.16	Regole per gli accessi al documento recensione	38
3.17	Calcola media difficoltà e utilità	39
3.18	Controllo utente in Dashboard	40

3.19 Dashboard - Sezione segnalazioni	41
3.20 Costruzione di UtenteDash	42
3.21 Implementazione di setBanTime	43
3.22 Form di Revisione Appunto	45
3.23 Implementazione di aggiungiCorso	46
3.24 Implementazione di eliminaCorso	47
3.25 Sezione di Controllo	48
3.26 Implementazione di checkMediaEsame	49
4.1 Grafico di utilizzo di Firebase per 30 Giorni	51
4.2 Utilizzo del 24 e 25 Gennaio	51
4.3 Esempio di caricamento pagina profilo utente	52
4.4 Grafico di confronto tra i 2 metodi	54
4.5 Regole per la sub-collection dativalidati	57
4.6 Flow chart della gestione dei dati validati	58
4.7 Esempio di struttura del documento dell'insegnamento aggiornato	59
5.1 Diagramma integrazione API Esse3	63

Capitolo 1

Introduzione

1.1 Contesto e Motivazione

Tutte le Università hanno un sito che permette la gestione del proprio libretto universitario e di consultare gli esami nel proprio piano di studi, ma la maggior parte non permette altro oltre a questo.

La possibilità di consultare contenuti riguardanti un determinato esame, ad esempio, non è presente in Esse3¹, il sistema informativo adottato dall'Università della Calabria. Infatti, Esse3, non permette di interagire con i contenuti riguardanti gli esami, poiché inesistenti nella piattaforma. Uniwhere², un'applicazione per iOS e Android, permette di interagire con i contenuti dei corsi della propria università e di avere un libretto dove tracciare la propria carriera. La possibilità, però, di poter dare un contributo vero e proprio e dare *una mano* ad altri studenti che frequentano lo stesso Corso di Laurea non è presente in nessuno dei due esempi sopraccitati.

Altri esempi di *portali* che permettono la cooperazione tra studenti sono DocSity³ o AppuntiUniversitari⁴ in cui è data la possibilità agli studenti di caricare i propri appunti per poterli rivendere e aiutare altri studenti.

¹<https://unical.esse3.cineca.it>

²<https://uniwhere.com/>

³<https://www.docsity.com/it>

⁴<https://appuntiuniversitari.online/>

1.2 Obiettivi

Gli obiettivi di questo lavoro di tesi sono lo sviluppo, la progettazione e l'implementazione di un sito web che permette agli studenti di potersi registrare per poter interagire con i contenuti del proprio Corso di Laurea. In particolare, attraverso il sito lo studente potrà fruire dei seguenti servizi:

- Accesso ad un libretto personalizzato dove inserire i voti del proprio percorso di Laurea
- La media degli esami inseriti
- La possibilità di scrivere per ogni esame superato una recensione dell'esame, dando una valutazione testuale e numerica della difficoltà e dell'utilità dell'esame
- La possibilità di fare domande riguardanti l'esame e ricevere risposte dagli altri utenti
- La possibilità di caricare i propri appunti per un determinato esame
- Poter collegarsi con altri studenti tramite una lista di social network presenti nel profilo utente

1.3 Contributo principale

Entrando nel dettaglio, il sito permette di avere un'interfaccia semplice che racchiude le informazioni più importanti di un esame, ovvero CFU, professore, codice corso, anno di corso e voto medio del superamento dell'esame. Per ogni esame, se superato, è possibile lasciare una recensione che andrà a dare una valutazione sulla difficoltà e sull'utilità dell'esame. Inoltre, è possibile fare delle domande per ogni esame, anche in modo anonimo, alle quali solamente gli studenti registrati potranno rispondere. In più, è disponibile una sezione appunti in cui è possibile caricare degli appunti per il singolo esame, dando un titolo ed una descrizione di quest'ultimo; prima di essere caricato e disponibile

a tutti, verrà visionato da un amministratore, tramite un'apposita dashboard. Se una recensione, una domanda o un'immagine del profilo dovesse essere di cattivo gusto o disturbante per qualche utente del sito, sarà possibile mandare una segnalazione al sistema, la quale verrà revisionata da un amministratore, il quale, eventualmente, prenderà provvedimenti in base alla gravità della segnalazione.

Struttura del lavoro di tesi

La presentazione di questo lavoro di tesi è strutturata nel seguente modo: Il Capitolo 2 mostra le principali tecnologie utilizzate con un esempio generico di utilizzo, introducendo i principali concetti di programmazione legati ad esse. Il Capitolo 3 è il capitolo fondamentale, poiché contiene una panoramica generale del sito seguita poi dalle principali implementazioni di funzioni e algoritmi, esplicitando i problemi all'interno del progetto. Il Capitolo 4 contiene alcune possibili soluzioni a dei problemi che vengono esposti durante le varie implementazioni. Il Capitolo 5 conclude il lavoro di tesi introducendo alcuni che possono essere dei risvolti futuri che il progetto potrebbe avere.

1.4 Problemi tecnici riscontrati

Il progetto utilizza per la gestione dei dati Firebase⁵, una piattaforma creata da Google per la creazione di applicazioni mobile e web application. La mancanza di un servizio di back-end vero e proprio ha portato a delle scelte di modellazione dei dati e gestione dei permessi utente diverso dal solito. Il problema principale è quello di non avere la possibilità di eseguire del codice lato server poiché, come verrà spiegato nel Capitolo 2, Firebase ha delle limitazioni riguardo al suo utilizzo poiché offre diversi piani tariffari. Il progetto è stato sviluppato utilizzando il *Piano Free* di Firebase, perciò non è stato

⁵<https://firebase.google.com>

possibile utilizzare il servizio di Cloud Functions ⁶. I dettagli del problema e dell'implementazione verranno discussi nel Capitolo 3.

⁶<https://firebase.google.com/docs/functions>

Capitolo 2

Tecnologie Utilizzate

In questo capitolo verranno illustrare le tecnologie utilizzate per lo sviluppo del sito web, aggiungendo qualche esempio di utilizzo di quelle più importanti.

2.1 Firebase

Firebase è un servizio offerto da Google che permette di creare applicazioni mobile e web in modo semplice. Il servizio offerto fornisce un pacchetto di funzionalità elevato, partendo da un servizio di *autenticazione* fino a funzionalità utili per il *Machine Learning*.

L'utilizzo di Firebase all'interno del progetto è stato principalmente quello di sfruttare il Firestore Database, un servizio Database di tipo **non relazionale** anche detto **NO-SQL** che facilita di molto al programmatore la gestione dei dati all'interno del proprio sito, tramite l'enorme mole di funzioni offerte da Firebase stesso per inserire e ottenere i dati dal database.

2.1.1 Database orientato ai documenti

La scelta di un database NOSQL porta ad avere alcune libertà in più rispetto ad un database relazionale. Quella principale è che i dati sono memorizzati non in tabelle, bensì in veri e propri documenti identificati da un

codice univoco e situati in un percorso ben definito. La struttura di database orientato ai documenti è la seguente:

Collection

Le collection funzionano similmente alle tabelle di un database relazione; è il 'luogo' dove i documenti sono immagazzinati e nel quale solitamente vengono cercati attraverso le query. La differenza sostanziale dalle tabelle è che al proprio interno la struttura dei documenti non è statica, bensì dinamica. Ciò significa che le proprietà dei documenti possono essere diverse tra i documenti appartenenti alla stessa collection, cosa che in un database relazione non sarebbe possibile.

Document

I documenti sono il vero e proprio nucleo del database NO-SQL. All'interno sono contenuti i dati suddivisi in proprietà secondo una struttura standard identificata dal programmatore, ma che non è vincolata ad essere la stessa per ogni documento. A differenza delle righe delle tabelle, il contenuto di una proprietà può includere un particolare tipo di entità chiamata **SubCollection**, che è un vero e proprio documento innestato all'interno di un altro documento. Ogni documento viene identificato solitamente da una stringa o un codice identificativo univoco per ogni documento. Le codifiche più comuni per i documenti sono XML, YAML, JSON, e BSON.

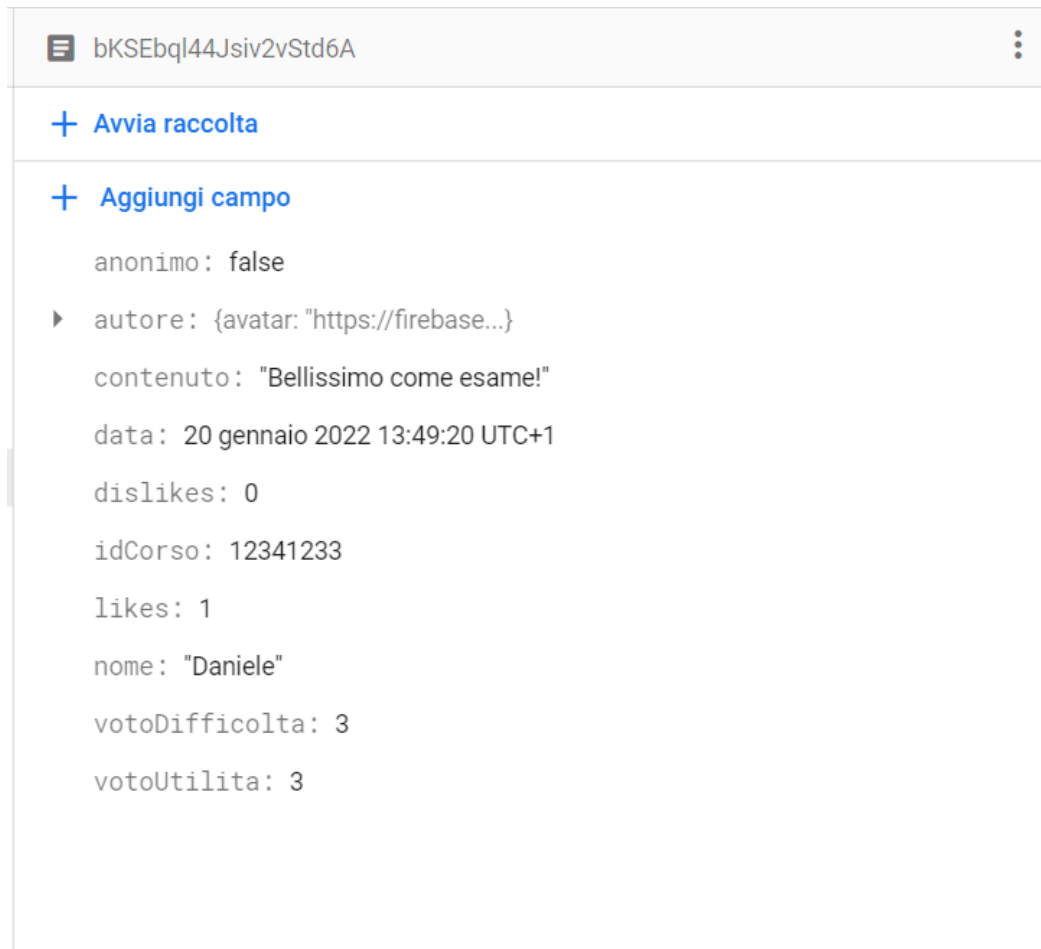


Figura 2.1: Esempio di Documento in Firebase

Ciò che Firebase offre è inizialmente un piano gratuito con delle limitazioni dipendenti dalla grandezza dell'applicazione e dal numero di utenti che la utilizzeranno. Precisamente, le limitazioni si basano in base ai servizi utilizzati. Per esempio:

- Il limite di letture di documenti giornalieri sul database è di 50.000
- Il limite di scritture di documenti giornalieri sul database è di 20.000

Oltre al piano gratuito, Firebase offre anche dei piani a pagamento¹ che seguono il concetto di *Pay as you go*, cioè quello di pagare in base all'utilizzo che viene fatto dell'applicazione.

¹<https://firebase.google.com/pricing>

Ad esempio: Dopo aver superato la soglia delle 50.000 letture al giorno viene applicata una tariffa calcolata in base ai criteri scelti dal provider del servizio.

2.2 Svelte

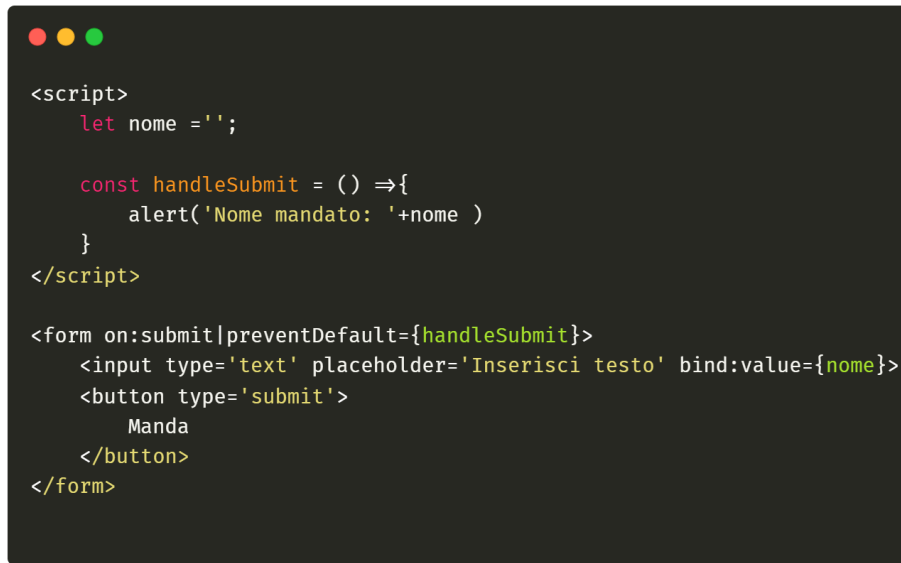
Svelte è un compilatore per Javascript sviluppato nel 2016 da Rich Harris che nell'ultimo periodo ha cominciato a prendere sempre più piede nello sviluppo web front-end. Svelte permette di creare applicazioni web complesse in modo semplice e con una sintassi facile da apprendere e poco articolata da scrivere. Il punto di forza di Svelte quello di compilare il codice e renderlo disponibile dopo il processo di build, avendo performance notevoli. Per capire quanto Svelte semplifica lo sviluppo ai programmatori, vediamo un esempio di codice a confronto.

```
class Form extends React.Component {
  constructor(props) {
    super(props);
    this.state = {value: ''};
    this.handleChange = this.handleChange.bind(this);
    this.handleSubmit = this.handleSubmit.bind(this);
  }

  handleChange(event) {    this.setState({value: event.target.value}); }
  handleSubmit(event) {
    alert('Evento del nome: ' + this.state.value);
    event.preventDefault();
  }

  render() {
    return (
      <form onSubmit={this.handleSubmit}>
        <label>
          Name:
          <input type="text" value={this.state.value} onChange={this.handleChange} />
        </label>
        <input type="submit" value="Submit" />
      </form>
    );
  }
}
```

Figura 2.2: Esempio di cambio valore variabile in React



```
<script>
  let nome = '';

  const handleSubmit = () =>{
    alert('Nome mandato: '+nome )
  }
</script>

<form on:submit|preventDefault={handleSubmit}>
  <input type='text' placeholder='Inserisci testo' bind:value={nome}>
  <button type='submit'>
    Manda
  </button>
</form>
```

Figura 2.3: Esempio di cambio valore variabile in Svelte

Svelte si basa su un paradigma di programmazione basato sui *componenti*. Un Componente è un'entità di codice separata dal codice principale, come nella *programmazione orientata agli oggetti*; questo modo di suddividere il codice permette di separare le unità che compongono l'interfaccia grafica, rendendo di facile lettura il codice del progetto e riducendo il numero di righe di codice, siccome i Componenti sono spesso *riutilizzabili*.

Struttura di un Component

Un Component in Svelte [3] viene creato quando ad un file viene data l'estensione `.svelte`. La struttura di un Component è divisa in 3 parti. La prima è lo *script* che contiene tutta la logica del component, come lo sono ad esempio le variabili che verranno utilizzate nell'HTML, le funzioni che vengono chiamate quando un determinato evento accade, ecc... La seconda parte è quella dell'*HTML*, che contiene tutta la struttura del Component. La terza parte è quella dello *styling*, identificato dal tag *style* che corrisponde al file css

del component, contenente tutti i selettori di stile che personalizzano l'HTML come lo sviluppatore decide.

Parametri dinamici nei Component

Quando si parla di parametri passati dall'esterno ad un Component, si fa riferimento a quelle che vengono definite *proprietà*, spesso abbreviate in *props*. Queste proprietà sono passate direttamente quando il Component viene creato all'interno del codice.

In Svelte per passare una variabile ad un Component bisogna specificare che quella variabile viene fornita dall'*esterno*, aggiungendo prima della dichiarazione la keyword *export*.

Ci sono due modi per passare una variabile ad un Component:

- Quando la variabile da passare ha lo stesso nome sia all'interno del Component sia all'interno del Parent che passa la variabile, allora semplicemente si scrive il nome *della variabile* tra parentesi graffe:

$$\{\text{nomeVariabile}\}$$

- Se la variabile all'interno del Component ha un nome diverso dalla variabile che viene passata dal Parent, bisogna assegnare alla variabile all'interno del Component il valore della variabile nel Parent, tramite l'operatore di assegnamento e le parentesi graffe:

$$\text{variabileComponent}=\{\text{variabileParent}\}$$

N.B: Le variabili che vengono passate ad un Component sono una copia di quelle passate dal Parent e in caso di cambiamento non andranno ad alterare il contenuto della variabile nel Parent. Se ci fosse bisogno di avere questo collegamento tra le due variabili, basta aggiungere la keyword *bind*: quando la variabile viene passata al Component. In questo modo ogni cambiamento verrà sincronizzato. Un esempio di applicazione può essere un *Component bottone 'Aumenta variabile'* che quando viene cliccato aumenta il valore di una variabile nel Parent.

```
<div class="main-links">
  {#if !$authStore.isLoggedIn}
    <BoxLink nome="Unisciti a noi!" emoji="login" linkto="/reg/joinus" />
  {:else}
    <BoxLink nome="Profilo" emoji="account_circle" linkto="/profilo/{$authStore.user.uid}" />
  {/if}
  <BoxLink nome="Corsi" emoji="school" linkto="/corsi" />
  <BoxLink nome="Ricerca" emoji="search" linkto="/ricerca"/>
</div>
```

Figura 2.4: Costruzione dei Component BoxLink

2.3 Sveltekit

Sveltekit è un framework basato su Svelte che permette di creare applicazioni web fornendo un'esperienza di sviluppo più comoda e rapida. Diversamente dalle altre applicazioni basate su server side rendering, SvelteKit rende l'esperienza più rapida e fluida. É basato su Snowpack, un tool che velocizza il processo di sviluppo, compilando i file una singola volta e, successivamente, ricompilandoli solamente in caso di cambiamenti all'interno del codice. Il punto di forza di Sveltekit è la possibilità di avere un routing delle pagine basato sul file-system; ciò significa che la struttura dell'applicazione è basata sulla struttura del codice. Quindi, come per React, non c'è bisogno di utilizzare un router esterno per la propria applicazione web, come ad esempio Redux. Le funzionalità di Sveltekit sono numerose e le principali utilizzate in questo progetto sono state il *server-side rendering* e il *routing dinamico*.

2.4 HTML E CSS

HTML

HTML non è un linguaggio di programmazione, bensì un linguaggio di markup, ovvero un insieme di regole che, scritte in un certo modo, portano ad una rappresentazione grafica del contenuto. E' il linguaggio più famoso per la realizzazione di pagine web, e la sua struttura è formata da `< tag >`. Un contenuto è solitamente preceduto da un tag di apertura `< tag >` e da un tag

di chiusura `< /tag >`. Tramite i tag è possibile suddividere il contenuto della pagina web in diversi tipi: Esempio:

- Paragrafo `< p >`
- Lista `< ul >`
- Immagine `< img >`
- Collegamento `< a >`
- Ecc...

CSS

HTML è solamente lo scheletro di una pagina web. Per renderla diversa da tutte le altre si utilizzano i Cascadian Style Sheets (CSS) letteralmente fogli di stile a cascata. I CSS sono usati per formattare il contenuto della pagina, utilizzando delle “regole” chiamate recommendations, e per rendere più semplice la stilizzazione della pagina web. Tramite il CSS è possibile andare a modificare come il contenuto della pagina viene mostrato.

2.5 Javascript

Javascript è un linguaggio di programmazione basato sugli oggetti e sugli eventi. Viene utilizzato per la maggior parte per la programmazione lato client, e negli ultimi anni è stato implementato anche nella programmazione lato server. Viene utilizzato per l’implementazione degli eventi che avvengono all’interno di una pagina web, ad esempio il click del mouse su un bottone. Nel tempo sono stati sviluppati nuovi metodi e funzioni che aiutano il programmatore ad accedere e gestire le strutture dati in modo più semplice, rendendolo uno dei linguaggi di programmazione più utilizzati al mondo.

Typescript

Typescript, come fa intuire il nome, è un'estensione del linguaggio Javascript, che aggiunge funzionalità in più come la possibilità di dare un tipo ai dati, creare classi, interfacce, moduli ecc. Per essere interpretato dal browser, il file con estensione .ts dev'essere prima compilato in Javascript, e non può essere letto direttamente.

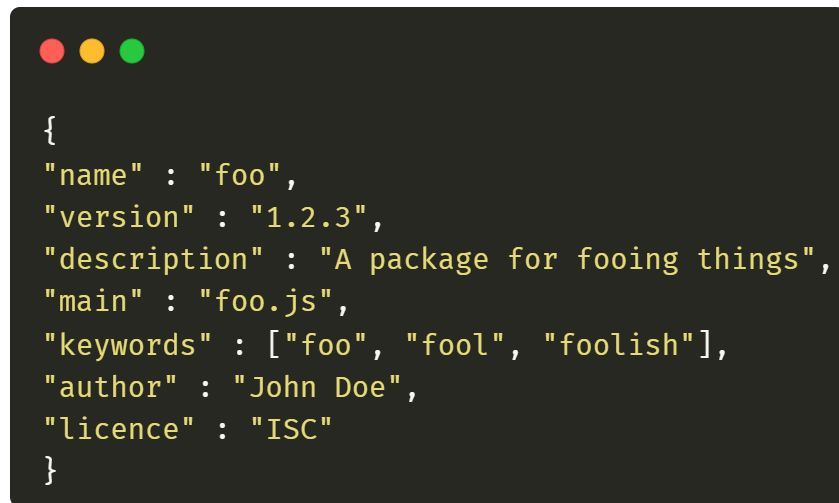
2.6 Node e NPM

Node

Node è una piattaforma di sviluppo a server-side per creare applicazioni web-based in semplice e scalabile basata sul V8 Engine di Chrome. Le funzionalità principali sono un modello asincrono e basato sugli eventi, ha una velocità in fase di esecuzione molto alta ed è facilmente scalabile. Negli ultimi anni è diventato sempre più popolare e viene usato per molteplici tipi di progetti, dalle applicazioni real-time alle single-page applications.

NPM

NPM è l'acronimo di Node Package Manager, un gestore di pacchetti, chiamati moduli, che contengono codice Javascript utilizzabile all'interno del proprio progetto. Tutti i pacchetti sono definiti in un file chiamato package.json, la cui struttura è la seguente:



```
{
  "name" : "foo",
  "version" : "1.2.3",
  "description" : "A package for fooing things",
  "main" : "foo.js",
  "keywords" : ["foo", "fool", "foolish"],
  "author" : "John Doe",
  "licence" : "ISC"
}
```

Figura 2.5: Esempio di package.json

Capitolo 3

Contributo principale

Obiettivo principale di questo lavoro di tesi è favorire l'interazione fra gli studenti di una stessa università attraverso la condivisione di contenuti inerenti gli esami dei vari corsi di laurea. In prima istanza, la fruizione dei servizi offerti dal sito realizzato è ristretta agli studenti dell'Università della Calabria.

Lo studente ha la possibilità di scrivere delle *recensioni* per gli esami superati, dando una valutazione più approfondita sulla *difficoltà* e *l'utilità*, di porre delle *domande* riguardo un determinato esame, di *rispondere* alle domande degli altri studenti, di caricare i propri *appunti* per un determinato esame e di aggiungere gli esami superati al proprio libretto.

Se lo studente dovesse trovare dei contenuti non idonei al sito o che vanno contro le regole, avrà la possibilità di creare delle *segnalazioni* che gli amministratori riceveranno e, tramite l'apposita dashboard, potranno gestire.

In questa sezione vengono mostrate le funzioni principali e la loro implementazione, tramite spezzoni di codice e grafici.

3.1 Panoramica e struttura del sito

Il sito è strutturato in macro-sezioni contenenti al loro interno i collegamenti con i contenuti principali.

Le sezioni accessibili sono:

- Sezione del profilo, personale e di altri utenti

- Sezione dei Corsi di Laurea
- Sezione degli insegnamenti
- Sezione degli annunci

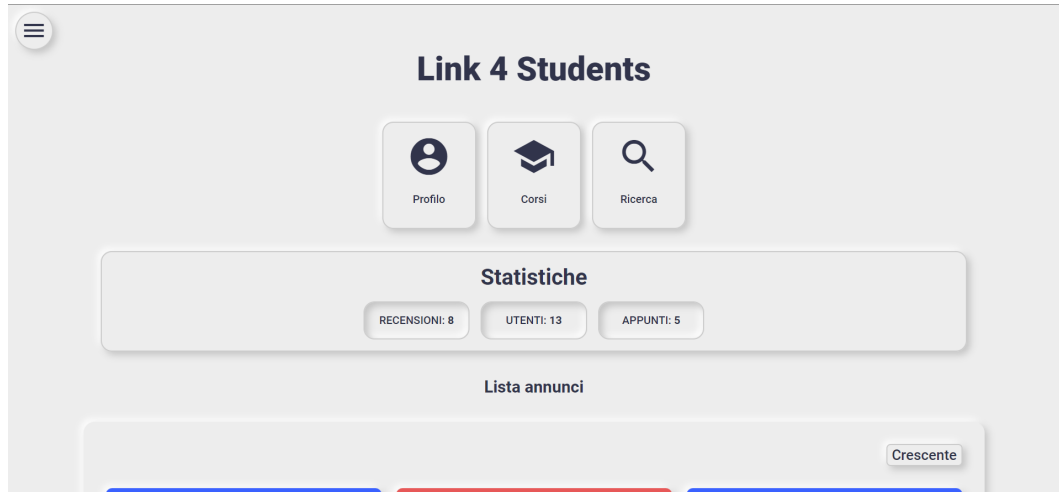


Figura 3.1: Homepage di Link4Students

Come citato nel Capitolo 2, all'interno del progetto viene utilizzato un paradigma di programmazione basato sui *componenti*. Proprio per questo motivo, la quasi totalità degli elementi mostrati in Figura 3.1 è un *component*.

Come menzionato precedentemente, Svelte permette di creare dei componenti riutilizzabili all'interno del progetto. Prendendo come esempio la figura 3.1, all'interno vengono mostrate delle *card*, una delle quali ha su scritto *profilo*. La card è formata da un'*icona* e un *testo*, e questo si ripete per le altre due card al centro della Homepage. La struttura è molto semplice, ma questa permette una scrittura del codice più leggibile e facile da evolvere in futuro. Il nome del component, in questo caso, è `BoxLink`, con estensione `.svelte` che identifica i file che assumono il ruolo di component. Per essere utilizzati all'interno del progetto devono essere importati tramite la keyword *import* seguita dal nome del file senza l'estensione, seguito successivamente dalla keyword *from* ed il percorso relativo del file.



Figura 3.2: Struttura del Component BoxLink

3.1.1 Pagina del profilo

La pagina dell'utente contiene le informazioni che caratterizzano lo studente come: *nome*, *cognome*, *Corso di Laurea*, *anno di corso*, ecc... Inoltre, è presente una *bio* che ogni studente può personalizzare a suo piacimento tramite una textarea presente nella sezione di modifica del profilo. Più in fondo sono presenti i collegamenti che lo studente effettua con gli altri studenti del sito, insieme alla lista degli esami presenti nel suo libretto e gli appunti da lui caricati all'interno di Link4Students.

È fornita la possibilità di nascondere alcune delle informazioni come il libretto e la lista dei collegamenti in base alle proprie preferenze.

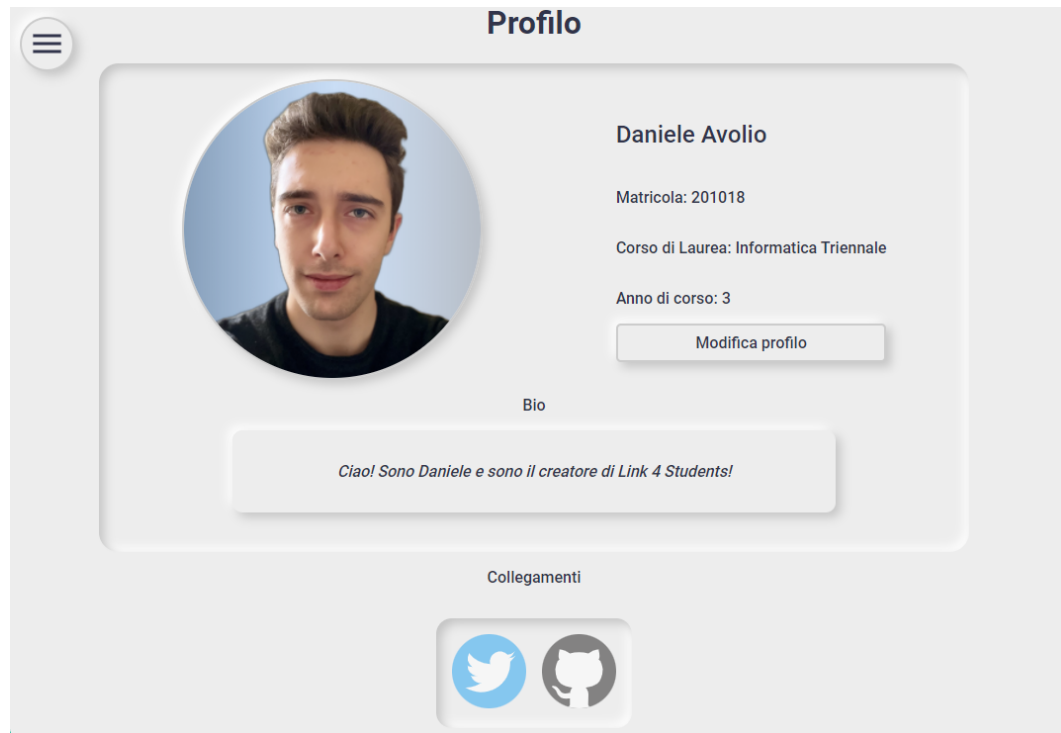


Figura 3.3: Pagina del profilo utente

3.1.2 Sezione dei Corsi di Laurea

La sezione dei Corsi di Laurea permette allo studente di navigare tra tutti i Corsi di Laurea che sono presenti nel Database di Firebase; ogni Corso di Laurea viene aggiunto manualmente dagli amministratori. Da lì, è possibile accedere alla pagina del singolo Corso di Laurea e visualizzare la lista di tutti gli insegnamenti presenti.

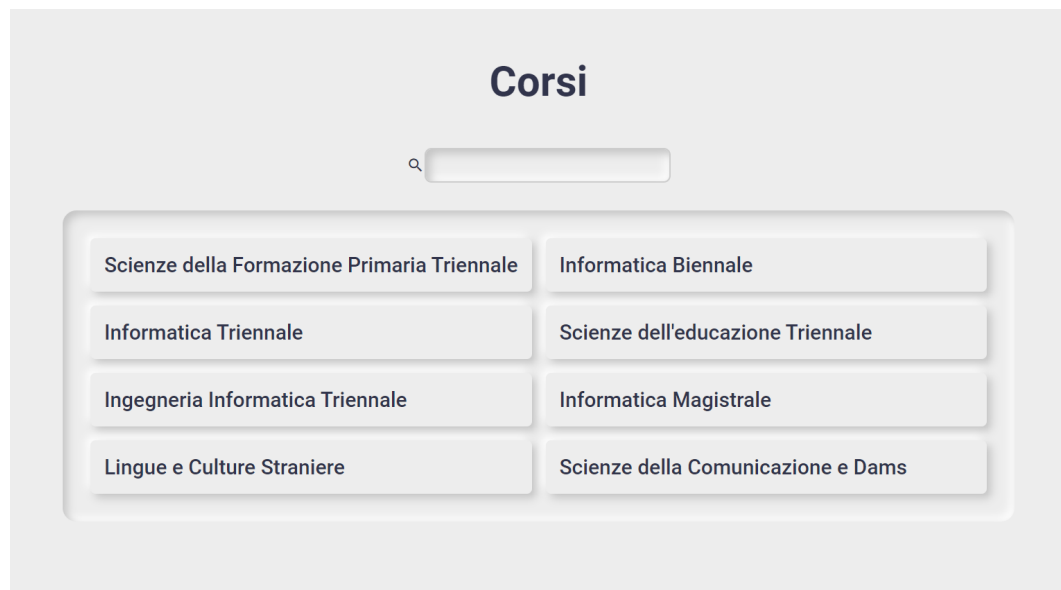


Figura 3.4: Pagina dei Corsi di Laurea

3.1.3 Sezione degli insegnamenti

La sezione degli Insegnamenti mostra la lista di tutti gli esami di un determinato Corso di Laurea. Oltre alla lista è fornito un dropdown menu che ordina in modo crescente in base a *CFU*, *Anno* e *Nome del corso*, scelti dall'utente. Oltre a questo è presente un campo di testo che filtra la lista in base a ciò che viene scritto al proprio interno.



Figura 3.5: Pagina degli Insegnamenti

3.1.4 Sezione degli annunci

La sezione degli annunci è una pagina che mostra tutti gli annunci che vengono fatti dagli amministratori, tramite l'apposita dashboard. Un annuncio può essere un' *avvertenza* o un *messaggio*, mostrando un colore diverso in base alla tipologia.

3.2 Configurazione iniziale di Firebase

Prima di utilizzare Firebase, bisogna configurarlo per renderlo disponibile all'interno del progetto.

In principio, bisogna creare un progetto sul portale Firebase, passaggio che non viene citato in questa tesi. Dopodiché bisogna installare la SDK di Firebase tramite npm utilizzando il comando:

```
npm install firebase
```

Successivamente, Firebase rende disponibile uno spezzone di codice da inserire all'interno del progetto per poter utilizzare le funzionalità da loro offerte. Questo spezzone di codice è inserito all'interno del progetto creando un file '*firebase.js*' e importandolo all'interno di tutti i file in cui c'è bisogno di utilizzare le funzioni. Senza questa configurazione avere l'accesso alle funzioni di Firebase sarebbe impossibile.

3.3 Lettura dei dati da Firebase

Per capire come funzionerà l'intero progetto, in questa Sezione 3.3 viene mostrato un esempio generico di come i dati vengono ottenuti tramite chiamate all'endpoint di Firebase tramite le *API* forniteci.

Esempio generico di richiesta

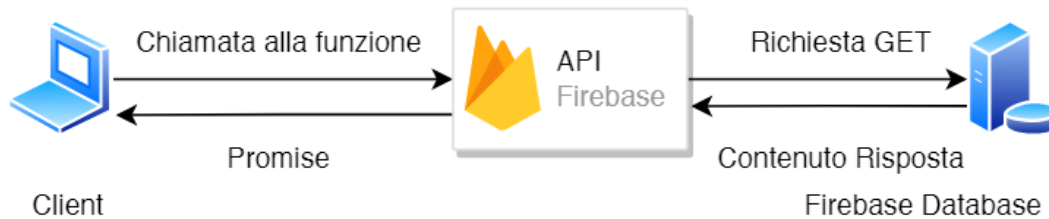


Figura 3.6: Esempio di richiesta Firebase

Come mostrato in Figura 3.6, il client effettua delle chiamate tramite le funzioni di Firebase per ottenere una Promise che, in caso di risoluzione, contiene all'interno ciò che dev'essere mostrato all'utente. Un esempio semplice di utilizzo è quando bisogna mostrare le informazioni dell'utente su schermo.

Nel database ogni utente viene salvato tramite un identificativo in una *collection* tramite un apposito documento. All'interno del documento si trovano i dati che costituiscono lo studente come il nome, il cognome, l'anno di corso, ecc.. Questi dati vengono richiesti a Firebase tramite la funzione *getDoc()* [1] in caso di singolo documento e *getDocs()* [1] in caso di documenti multipli, come una lista di *esami* superati dall'utente; potrebbe capitare di dover utilizzare *getDocs()* anche quando serve ottenere un singolo documento, poiché la funzione permette di filtrare i contenuti della query, mentre *getDoc()* non lo permette.

```
const uid = page.params.uid;
const collectionRef = collection(db, 'users');
const queryToDo = query(collectionRef, where('uid', '==', uid));
await getDocs(queryToDo).then(async (document) => {
  let docs = document.docs;
  profilo = docs[0].data();
});
```

Figura 3.7: Esempio di `getDocs()` per le informazioni utente

Il codice mostrato in Figura 3.7 esegue le seguenti operazioni in ordine:

1. Ottiene il valore dell'uid utente dai parametri della pagina.
2. Assegna il valore di `CollectionReference` [1] alla variabile `collectionRef` tramite la funzione `collection()` di Firebase.
3. Viene creata una *query* tramite la funzione di Firebase `query()` [1] che prende come parametro la `CollectionRef`. E' possibile filtrare i risultati tramite la funzione *where* [1].
4. Viene chiamata la funzione `getDocs()` passando come parametro la query creata precedentemente.
5. Come accennato poche righe fa, quando bisogna ottenere informazioni filtrando i documenti bisogna utilizzare la funzione `getDocs()` e, siccome la funzione in questione ritorna una `Promise` contenente il risultato della query come `QuerySnapshot` [1], sarà possibile accedere alla lista di documenti ottenuti tramite la proprietà `docs`. La proprietà `docs` ritorna una lista di documenti all'interno della `QuerySnapshot` e quindi bisogna accedere al contenuto come se fosse un array. Alternativamente, si sarebbe potuta utilizzare la funzione `pop()` di Javascript [2].

6. La variabile `profilo` ottiene le proprietà del documento che rappresenta l'utente tramite la funzione `data()` [1] e rende i dati accessibili, su tutta la pagina.

3.4 Funzionalità principali

All'interno del sito sono presenti numerose funzionalità, ma le principali si possono racchiudere in un gruppo di 3:

1. Gestione della media degli esami
2. Gestione della difficoltà e utilità degli esami
3. Gestione dei contenuti tramite dashboard amministratore

3.4.1 Gestione della media degli esami

Come scritto in Sottosezione 3.1.3, ogni corso ha una pagina apposita che contiene al proprio interno le informazioni che compongono un insegnamento quali *Nome*, *CFU*, *Codice Corso*, *Anno*, *Scheda Insegnamento (se disponibile)* e *Ore Insegnamento (se disponibili)*. Oltre a queste informazioni sono presenti altri 3 dati importanti:

1. Media dell'esame
2. Media difficoltà esame
3. Media utilità esame

Le ultime due verranno trattate in 3.4.2. La prima verrà trattata qui.



Figura 3.8: Esempio pagina Insegnamento

Come visto in Figura 3.8, la valutazione degli studenti non è sempre presente, e questo dipende dal fatto che ci siano recensioni o meno, come si vedrà nel punto 3.4.2. Si vede, però, un numero che spicca in confronto al resto, ovvero *22*. Quel numero è la media dei voti di superamento dell'esame, calcolata come verrà spiegato in seguito.

Per poter agire su questa media, lo studente dovrà caricare il voto di superamento dell'esame nel proprio libretto, tramite l'apposita funzione presente nella pagina del profilo. Tramite un bottone *Aggiungi Esame al Libretto* è possibile aprire una *form* che mostra un *dropdown menu* con una lista contenente gli esami presenti nel Corso di Laurea dello studente che ancora non stati aggiunti al libretto, un campo di inserimento numerico e una checkbox in caso di superamento dell'esame con *lode*.

Struttura del documento esameLibretto

Quando viene aggiunto un esame nel libretto dello studente viene creato un documento all'interno della collection *esameLibretto*; questo documento contiene delle informazioni che identificano per ogni studente gli esami tramite delle proprietà univoche, tra cui l'UID utente e l'ID dell'esame. Per la struttura del documento è stata fatta una scelta di modellazione specifica, questo per evitare di dover fare troppe letture per mostrare un singolo esame nel libretto studente. La struttura del documento è:

- codiceCorso
- lode
- **nomeCorso**
- uidCorso
- uidUtente
- voto

In particolare, la scelta di inserire il *nome del corso* nel documento è stata fatta per evitare di dover effettuare un'ulteriore query per ottenere il nome dell'esame dal documento con *ID Documento uguale a uidCorso*;

Esempio: *Se per ogni esame da mostrare nel libretto si dovesse fare una query per ottenere il nome del singolo esame, il numero di chiamate da fare sarebbe pari a:*

$$N = k * 2$$

con $N = \text{Numero Chiamate}$ e $k = \text{numero di esami aggiunti al libretto dallo studente}$.

Se invece il documento al proprio interno ha già il nome dell'esame al momento della creazione, si evita la seconda chiamata, avendo come risultato un numero di chiamate:

$$N = k$$

con $N = \text{Numero Chiamate}$ e $k = \text{numero di esami aggiunti al libretto dallo studente}$.

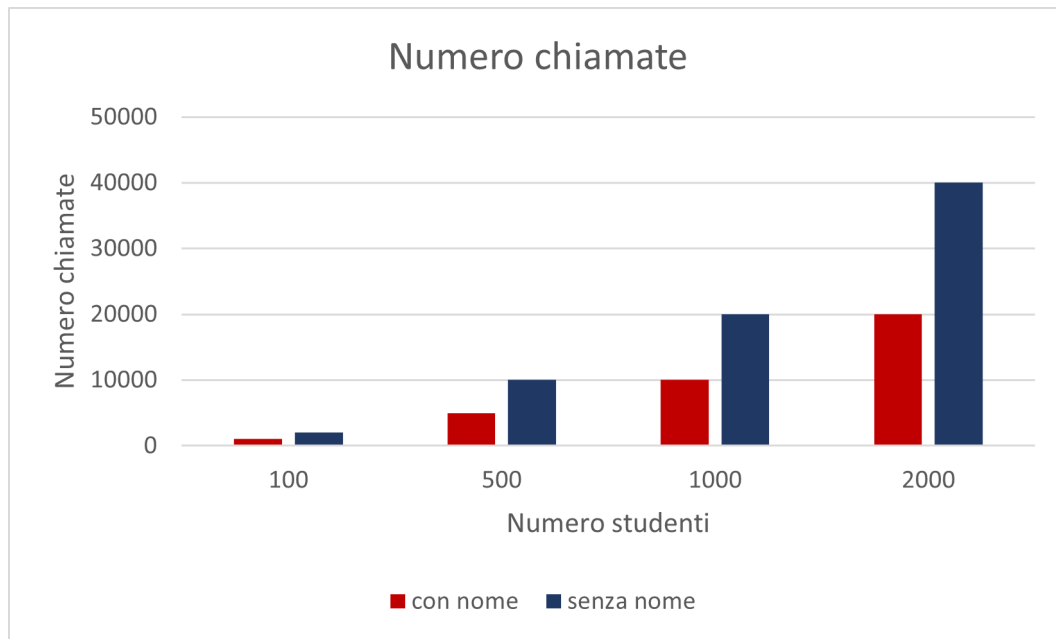


Figura 3.9: Grafico chiamate

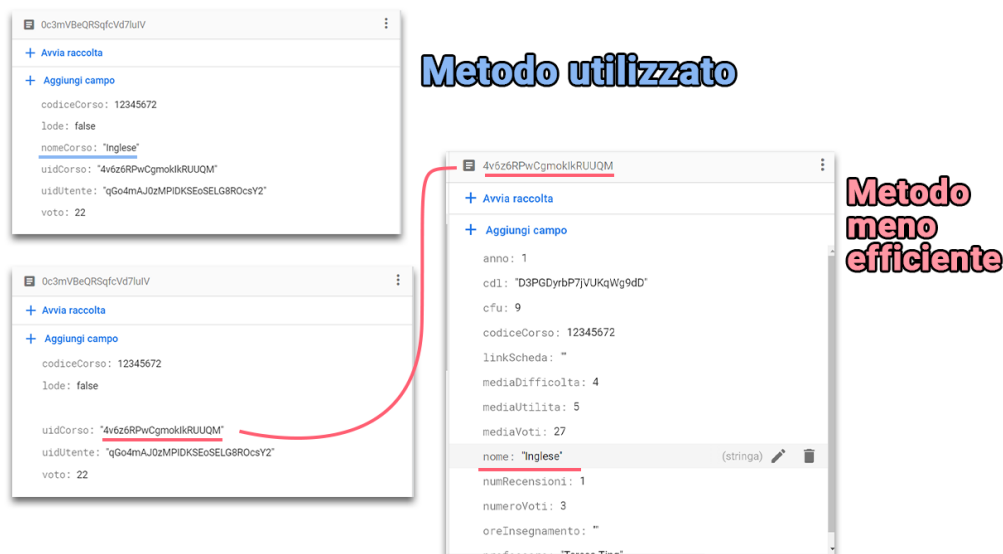


Figura 3.10: Grafico del problema

Per aggiungere l'esame al libretto viene utilizzata una funzione chiamata *aggiungiAlLibretto* che viene chiamata quando l'evento submit viene mandato al form. La funzione costruisce al proprio interno un oggetto *dati* che ha come valori tutte le proprietà necessarie per costruire un documento di tipo *esamiLibretto*. La funzione è asincrona, poiché la funzione che si occupa di mo-

dificare la media dell'esame dev'essere eseguita successivamente al caricamento del documento su Firebase.

Il metodo `addDoc()` [1] necessita di una `CollectionReference` come primo parametro, che viene costruita tramite l'istanza dell'oggetto `Firestore` chiamata `db`, precedentemente istanziato nel file di configurazione `firebase.js` (vedi Sezione 3.2), e una stringa rappresentante il *path* della `Collection`, in questo caso `'esamiLibretto'`, mentre come secondo parametro `addDoc` prende l'oggetto dati.

Da notare che non viene passato nessun ID al documento, questo perché la funzione `addDoc` assegna automaticamente un ID al documento creato.

Il metodo ritorna una `promise` che, se risolta, aggiorna il contenuto di una variabile messaggio presente nella pagina, informando l'utente che l'esame è stato caricato con successo nel libretto.

```
const aggiungiAllibretto = async () => {  
  let dati = {  
    uidUtente: $authStore.user.uid,  
    codiceCorso: corsoScelto.data().codiceCorso,  
    nomeCorso: corsoScelto.data().nome,  
    voto: voto,  
    lode: lode,  
    uidCorso: corsoScelto.id  
  };  
  
  // Aggiungo l'esame al libretto  
  await addDoc(collection(db, 'esamiLibretto'), dati).then((docum) => {  
    messaggio = 'Esame aggiunto al libretto!';  
  });  
  
  // Gestione media voto  
  cambiaMediaVotoEsame();  
};
```

Figura 3.11: Funzione di aggiunta esame al libretto

Successivamente all'aggiunta dell'esame nel libretto studente, viene eseguita la funzione `cambiaMediaVotoEsame()`. La funzione `getDoc`[1] ottiene il documento dell'esame tramite la variabile `corsoScelto`, avendo fatto un binding tra la variabile ed il valore del dropdown menu del form. Una volta ottenuto il documento viene fatto un controllo sulla proprietà *mediaVoti*.

I casi possono essere 2:

1. Non esiste nessuna media dei voti poiché nessuno ha mai aggiunto l'esame al proprio libretto.
2. Esiste già una media dei voti siccome qualcuno ha aggiunto l'esame al proprio libretto.

Caso 1: É il caso più semplice, poiché le uniche cose da fare sono creare una variabile *mediaVoti* e assegnarle il valore del voto appena inserito e creare un'altra variabile *numeroVoti* dando come valore 1. Creato un oggetto *data* contenente le due variabili, viene chiamata la funzione *setDoc()* per assegnare al documento del corso le nuove proprietà.

N.B: Viene passato un secondo oggetto alla funzione *setDoc()*, ovvero *{merge: true}*. Questo oggetto permette al documento di non essere sovrascritto completamente con il nuovo oggetto *data*, ma di *modificare* esclusivamente le proprietà differenti dalla versione precedente alla modifica.

Caso 2: É il caso più complesso. Le operazioni di scrittura dei dati sono le stesse del punto 1, ma cambia il calcolo della media dei voti.

$$vecchioTotale = mediaVoti * numeroVoti; \quad (3.1)$$

Trovato il totale dei voti presenti, viene aumentato il numero dei voti inseriti di 1. Avendo ora il *totale* ed avendo il *numero* di voti, per calcolare la media bisogna applicare la formula:

$$mediaVoti = (vecchioTotale + nuovoVoto) / numeroVoti; \quad (3.2)$$

L'unica differenza in scrittura rispetto al punto 1 è che per aumentare il *numeroVoti* dell'esame viene chiamata la funzione *increment[1]*, che ottiene automaticamente il valore della proprietà da Firebase e lo aumenta del valore in parentesi.

```

const cambiaMediaVotoEsame = () => {
  getDoc(doc(db, 'corsidelcdl', corsoScelto.id)).then((corso) => {
    // Ho il corso.. devo prendermi i dati

    if (corso.data().mediaVoti == null) {
      // Non è mai esistito un voto, allora.
      let mediaVoti = voto;
      let numeroVoti = 1;

      let data = {
        mediaVoti,
        numeroVoti
      };

      setDoc(doc(db, 'corsidelcdl', corsoScelto.id), data, { merge: true });
    }
    // Se invece esistono già voti registrati nei libretti degli altri
    else {
      let vecchioTotale = corso.data().mediaVoti * corso.data().numeroVoti;
      let nuovoNumVoti = corso.data().numeroVoti + 1;
      let nuovaMedia = Math.round((vecchioTotale + voto) / nuovoNumVoti);
      // ho la nuova media
      // ho il nuovo numero di recensioni
      let data = {
        mediaVoti: nuovaMedia,
        numeroVoti: increment(1)
      };
      // posso settare il doc
      setDoc(doc(db, 'corsidelcdl', corsoScelto.id), data, { merge: true });
    }
  });
};

```

Figura 3.12: Funzione di calcolo media dell'esame

Problema tecnico

Come già menzionato in Sezione 1.4 nel progetto sono presenti dei problemi tecnici dovuti all'inesistenza di un server per il *backend*. L'operazione di modifica della media del voto è eseguita direttamente dal Client, senza passare da un server. Per questo motivo sono state scritte delle regole che impediscono all'utente di compiere alcune azioni, come modificare direttamente i contenuti altrui o alcuni contenuti del sito. Però, poiché l'utente modifica direttamente la media dell'esame, ci potrebbero essere dei comportamenti malevoli volti a modificare e rendere inconsistente i dati tra loro. Questo problema sarebbe inesistente se le operazioni di scrittura di alcuni contenuti fossero eseguite da un backend puro. Firebase offre queste possibilità tramite le *Cloud Functions* come menzionato in 1.4, sottoponendosi ad un piano a pagamento.

La risoluzione del problema verrà trattata successivamente nella Sezione 3.4.3 in cui si adopererà un meccanismo di controllo utilizzabile dagli amministratori del sito.

3.4.2 Gestione della difficoltà e utilità degli esami

Quando uno studente aggiunge al proprio libretto un esame, gli viene data la possibilità di scrivere una *recensione*. Questa possibilità viene mostrata direttamente dalla *pagina dell'insegnamento* tramite un apposito bottone.

Il form per la scrittura della recensione fornisce una scala variabile da 1 a 5 per dare una valutazione sulla difficoltà e l'utilità dell'esame secondo lo studente, seguita da una textarea per dare un parere personale di massimo 200 caratteri sull'esame e una checkbox per scegliere se caricare la recensione in anonimo.



The screenshot shows a web form titled "Recensione per: Algoritmi paralleli e sistemi distribuiti". It contains two rating sections. The first section, "Quanto reputi difficile l'esame?", has five buttons labeled 1, 2, 3, 4, and 5, with the button '3' selected and highlighted in blue. The second section, "Quanto reputi utile l'esame?", also has five buttons labeled 1, 2, 3, 4, and 5, with the button '3' selected and highlighted in blue. Below these sections is a text area labeled "Lascia la tua recensione." with the placeholder text "Dai un tuo parere, fai un commento o suggerimento per l'esame! (Massimo 200 caratteri)". At the bottom of the form is a blue button labeled "Pubblica" and a checkbox labeled "Anonimo?" which is currently unchecked.

Figura 3.13: Form per la recensione

Quando il form cattura l'evento submit, viene chiamata la funzione *mandaRecensione*. La logica della funzione è molto simile alla funzione implementata in 3.4.1

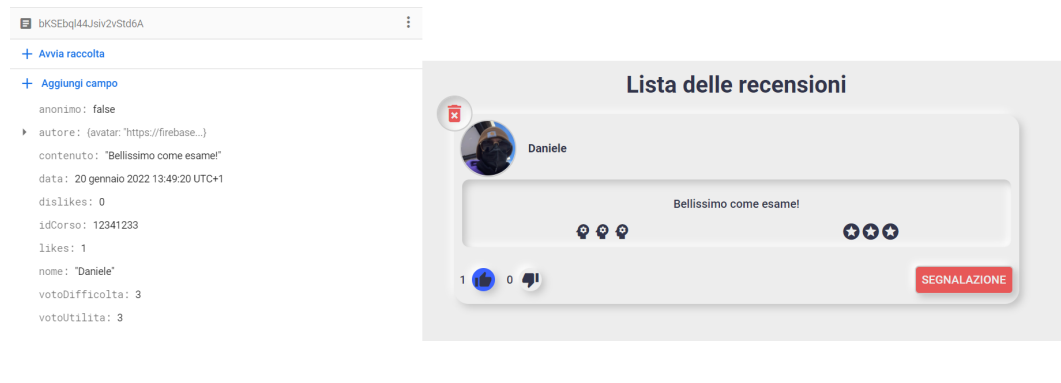


Figura 3.14: Struttura Documento e Component

Nella struttura del Documento mostrato in Figura 3.14 è presente una proprietà chiamata *autore* la quale contiene all'interno le informazioni dell'autore quali *nome*, *idAutore* e *avatar*. Il motivo di questa divisione del documento è dovuta al fatto che Firebase permette di aggiungere delle regole per gestire gli accessi e i permessi di scrittura e lettura dei dati dal database all'intero documento. Questo implica se un utente avesse la possibilità di leggere, ad esempio, il contenuto della recensione, questo significa che tutte le proprietà a livello del contenuto sono accessibili da chiunque rispetti le regole impostate allo sviluppatore.

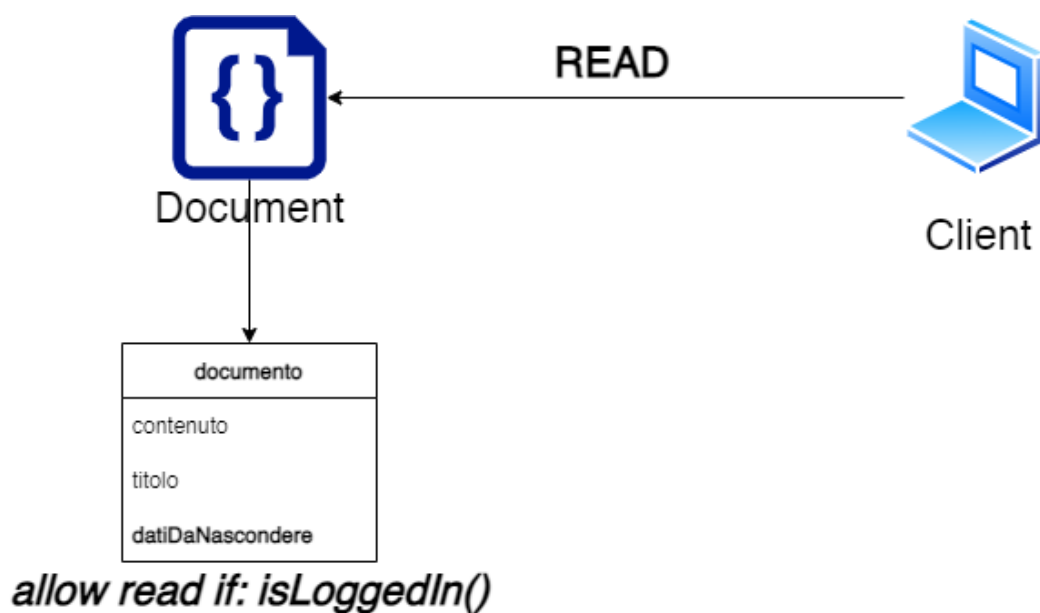


Figura 3.15: Esempio di libertà di accesso

In Figura 3.15 il Client richiede il documento e ha la possibilità di accedere a tutte le proprietà del documento, se le regole in lettura fossero

allow read if: isLoggedIn()

dove `isLoggedIn` è una funzione ipotetica per controllare che l'utente sia loggato. Se questa fosse la struttura, chiunque avesse accesso al documento e fosse loggato potrebbe leggere tutte le proprietà.

Per questo motivo, la proprietà *autore* del documento *recensione* è un sotto-documento che all'interno contiene le proprietà citate precedentemente. Avendo un sotto-documento è possibile scrivere delle regole che lo avranno come target, modificando i suoi permessi per la lettura e scrittura. In questo caso, poiché la recensione potrebbe essere anonima, la possibilità di lettura di quei dati dev'essere data esclusivamente agli *amministratori* oppure al *proprietario* della recensione. Se la recensione invece non fosse anonima, chiunque potrebbe leggere il contenuto della proprietà *autore*.

Le regole che gestiscono questi accessi sono le seguenti:

```
//Se sei loggato, puoi scrivere recensioni. Altrimenti non puoi.
//Se sei l'autore, puoi eliminare la recensione.
match/recensioni/{rec}{
  allow create, update: if request.auth != null
  allow delete: if request.auth.uid == resource.data.autore.idAutore ||
    exists(/databases/${database}/documents/amministratori/${request.auth.uid})
  allow read: if true;

  //Per vedere le informazioni nascoste della recensione devi essere il proprietario
  match/autore/{autore}{
    allow create: if request.auth != null;
    allow read: if get(/databases/${database}/documents/recensioni/${rec}).data.anonimo == false ||
      get(/databases/${database}/documents/recensioni/${rec}).data.autore.idAutore == request.auth.uid
  }
}
```

Figura 3.16: Regole per gli accessi al documento recensione

Nel momento di caricamento di una recensione, vengono richiesti i dati dell'autore tramite la funzione `getDoc` di Firebase e vengono salvati in un oggetto *data*. Successivamente, trovandosi nella pagina del corso quando viene fatta la richiesta, viene richiesto il documento del corso per il quale viene scritta la recensione. Per il calcolo della media viene chiamata la funzione *calcolaMedia* che come parametri riceve il documento contenente i dati del

corso ricavato precedentemente e il valore di *difficoltà* e *utilità* che viene inserito nel form.

La funzione `calcolaMedia` è molto simile alla funzione utilizzata per calcolare la media del voto di superamento di un dato esame. L'unica cosa che cambia è che le medie da calcolare sono due a differenza di una. Le formule che vengono utilizzate sono pressoché le stesse:

$$mediaDifficolta' = (vecchiaDifficolta' + difficolta') / numeroRecensioni \quad (3.3)$$

$$mediaUtilita' = (vecchiaUtilita' + utilita') / numeroRecensioni \quad (3.4)$$

Oltre a questo, viene aumentato il numero di recensioni di 1 tramite la funzione `increment` [1] fornita da Firebase.

```
const calcolaMedia = (esame, difficolta, utilita) => {
  let numRecensioni;
  let mediaDifficolta;
  let mediaUtilita;
  if (esame.numRecensioni) {
    numRecensioni = esame.numRecensioni + 1;
  } else numRecensioni = 1;

  // Se esiste la media, allora salvo la vecchia e poi calcolo la nuova
  if (esame.mediaDifficolta) {
    let vecchiaMediaTot = esame.mediaDifficolta * esame.numRecensioni;
    mediaDifficolta = Math.floor((vecchiaMediaTot + difficolta) / numRecensioni);
  } else mediaDifficolta = difficolta;
  // Se esiste la media, allora salvo la vecchia e poi calcolo la nuova
  if (esame.mediaUtilita) {
    let vecchiaMediaTot = esame.mediaUtilita * esame.numRecensioni;
    mediaUtilita = Math.floor((vecchiaMediaTot + utilita) / numRecensioni);
  } else mediaUtilita = utilita;

  let datiMedia = {
    numRecensioni,
    mediaDifficolta,
    mediaUtilita
  };
  return datiMedia;
};
```

Figura 3.17: Calcola media difficoltà e utilità

Anche questa funzionalità soffre dello stesso problema menzionato nella sezione precedente, quello della mancanza del server reale per la gestione del-

l'integrità dei dati. La risoluzione del problema verrà discussa nella sezione successiva, attraverso la *Gestione dei contenuti tramite dashboard*

3.4.3 Gestione dei contenuti tramite dashboard

La gestione dei contenuti all'interno del sito, come i Corsi di Laurea, gli insegnamenti, gli annunci e le segnalazioni vengono gestite tramite dashboard, creata per essere utilizzata solamente dagli amministratori.

Un utente viene definito amministratore se esiste un documento all'interno di una Collection chiamata *amministratori* avente come ID del Documento l'UID di un utente. Quando un utente accede alla dashboard, dev'essere effettuato un controllo per verificare che colui che sta accedendo alla pagina è un amministratore. Per fare questo controllo, quando la pagina viene caricata viene eseguita una funzione che controlla inizialmente *se l'utente ha effettuato l'accesso* e successivamente viene controllata se una variabile all'interno di uno *store* [3] utilizzato per le autorizzazioni e l'accesso fosse *true*.

Se questo controllo va a buon fine, l'utente rimane all'interno della pagina e vengono ricavate le sue informazioni. Altrimenti, in caso di fallimento, viene rimandato alla *Homepage* del sito.

```
onMount(async () => {  
  if ($authStore.isLoggedIn) {  
    await getDoc(doc(db, 'users', $authStore.user.uid)).then((doc) => {  
      user = doc;  
    });  
  } else {  
    goto('/');  
  }  
}
```

Figura 3.18: Controllo utente in Dashboard

La Dashboard è divisa in 3 Macro-sezioni che al loro interno incorporano altre sezioni per gestire i contenuti del sito. Le sezioni sono:

- Sezione di gestione delle segnalazioni
- Sezione di gestione dei contenuti
- Sezione di controllo

Sezione di gestione delle segnalazioni

In questa sezione vengono gestite tutte le segnalazioni create dagli utenti quando rilevano che un contenuto all'interno del sito non è opportuno o che non rispetta le regole. Più precisamente, le segnalazioni vengono effettuate su: *Utenti*, *Recensioni*, *Domande* e, in aggiunta, in questa sezione vengono gestiti anche gli *Appunti* caricati dagli utenti, poiché prima di essere disponibili a tutti per essere scaricati, l'amministratore deve accertarsi che il contenuto caricato non sia illegale o non conforme alle regole del sito (ad esempio Tracce d'Esame).



Figura 3.19: Dashboard - Sezione segnalazioni

Gestione e struttura di una segnalazione

L'amministratore tramite la dashboard può consultare la lista di segnalazioni riguardanti i vari tipi di contenuti tramite un menù verticale che modifica ciò che viene mostrato a lato della schermata. In Figura 3.19 viene mostrata la schermata di Segnalazione per gli Utenti, che riporta la lista di utenti che hanno ricevuto una segnalazione, avendo la possibilità di impartire una punizione espressa in giorni tramite un campo di testo al cui interno andrà inserito il numero di giorni desiderato. Per capire il motivo della segnalazione si utilizza il pulsante *Mostra Report* che espande la segnalazione mostrandone l'*autore* e la *motivazione* che lo ha portato a segnalare il contenuto. Se l'amministratore

dovesse ritenere la segnalazione senza senso, potrebbe decidere di *Risolvere la segnalazione* eliminandola senza arrecare all'autore del contenuto nessuna punizione.

Nel caso di un utente, è stato creato un Component chiamato *UtenteDash*, che permette la gestione delle segnalazioni all'interno di esso. Quando viene creato riceve come parametri due funzioni per la gestione della lista all'interno della dashboard e un oggetto contenente le informazioni utili a gestire la segnalazione. Se non ci fossero utenti segnalati, ciò che verrebbe mostrato è un Component chiamato *VuotoDash* avente come unico scopo quello di mostrare un messaggio a schermo con su scritto 'Nessuna segnalazione'. Per ricavare la *lista di utenti segnalati* viene fatta una semplice query sulla Collection di *segnalazioniUtenti* e viene riportato il contenuto della Response in una variabile chiamata *listaUtenti*. Lo stesso discorso avviene per la *recensioni*, per le *domande* e per gli *appunti*.

```
{#if listaUtenti.length == 0}
  <VuotoDash oggetti="utenti" />
{:else}
  <div class="lista-generica">
    {#each listaUtenti as oggettoSegnalazione (oggettoSegnalazione.segnalazione.id)}
      <UtenteDash
        {oggettoSegnalazione}
        {cambiaUtentiSegnalati}
        {risolviSegnalazioneUtente}
      />
    {/each}
  </div>
{/if}
```

Figura 3.20: Costruzione di *UtenteDash*

Al contrario di quando un amministratore decide di non arrecare punizioni ad un utente, se l'amministratore decidesse di arrecare una sospensione, quello che succede è che verrebbe aggiunto un Documento ad una Collection chiamata *banTimes*. All'interno di questa Collection vengono caricati dei documenti aventi come ID proprio l'UID che distingue un utente all'interno del sito. Come unica proprietà del Documento è stato scelto di inserire *time*, una proprietà che corrisponde al tempo di scadenza della punizione arrecata. Il modo in cui questo calcolo viene fatto è il seguente:

- Viene preso il valore all'interno del campo di inserimento nel Component UtenteDash.
- Viene invocata la funzione `now()` di `Timestamp` [1].
- Il valore di ritorno di `now()` espresso in millisecondi viene trasformato in secondi.
- A questo valore di secondi viene sommato un nuovo valore, calcolato come segue:

$$\text{Valore} = 86000 * K$$

dove K = giorni di attesa o valore ottenuto dal campo di inserimento e 86000 sono i *secondi in un giorno*.

Una volta ottenuto questo valore, viene creato il documento tramite la proprietà `setDoc()` [1] che richiede direttamente l'inserimento di un ID del documento che, come detto poche righe prima, dovrà essere l'UID dell'utente da sospendere.

```
const setBanTime = (uid) => {  
  // 86400 è il numero di secondi in un giorno  
  let giorniInSecondi = giorniSospensione * 86400;  
  let tempo = Timestamp.now().seconds + giorniInSecondi;  
  
  setDoc(  
    doc(db, 'banTimes', uid),  
    {  
      time: tempo  
    },  
    { merge: true }  
  ).then(() => {  
    // Si elimina la segnalazione  
    deleteDoc(doc(db, 'segnalazioniUtenti', oggettoSegnalazione.segnalazione.id));  
    // Rimuovere localmente la recensione  
    cambiaUtentiSegnalati(oggettoSegnalazione.segnalazione.id);  
  });  
};
```

Figura 3.21: Implementazione di `setBanTime`

Il controllo sulla sospensione dell'utente viene effettuato al momento del *login* attraverso una funzione che, tramite l'UID dell'utente appena loggato, controlla se esiste un Documento con l'ID uguale al proprio UID. Se questo

controllo dovesse andare a buon fine verrà verificato se il valore *time* dovesse essere *maggiore* rispetto al valore attuale di `Timestamp.now()`. In caso successo, l'utente verrebbe notificato del fatto che la sospensione è ancora attiva e verrebbe automaticamente disconnesso.

Revisione degli appunti

Come detto precedentemente, gli appunti caricati dagli utenti prima di essere disponibili devono essere controllati manualmente da un amministratore. Questo avviene tramite una lista di appunti caricati nella stessa sezione delle segnalazioni, in cui l'amministratore può scaricare *uno ad uno* tutti gli appunti che decide approvare. Per questo passaggio non esiste nessuno storico e una volta che l'appunto è stato revisionato non viene più mostrato in quella lista. Per scaricare l'appunto è presente un apposito pulsante che permette l'operazione. Se questo contenuto dovesse essere consono al sito secondo l'amministratore, potrà revisionarlo cliccando sul pulsante *REVISIONA*, che mostrerà un form in cui all'interno ci sarà la possibilità di inserire un link. Il link da inserire sarà ricavabile caricando il file in Google Drive ¹ e rendendo accessibile il contenuto a chiunque avesse accesso ad esso. Altrimenti, se il contenuto non fosse legale o andasse contro le regole del sito, l'amministratore può semplicemente eliminarlo tramite l'apposito bottone.

¹https://www.google.com/intl/it_it/drive/

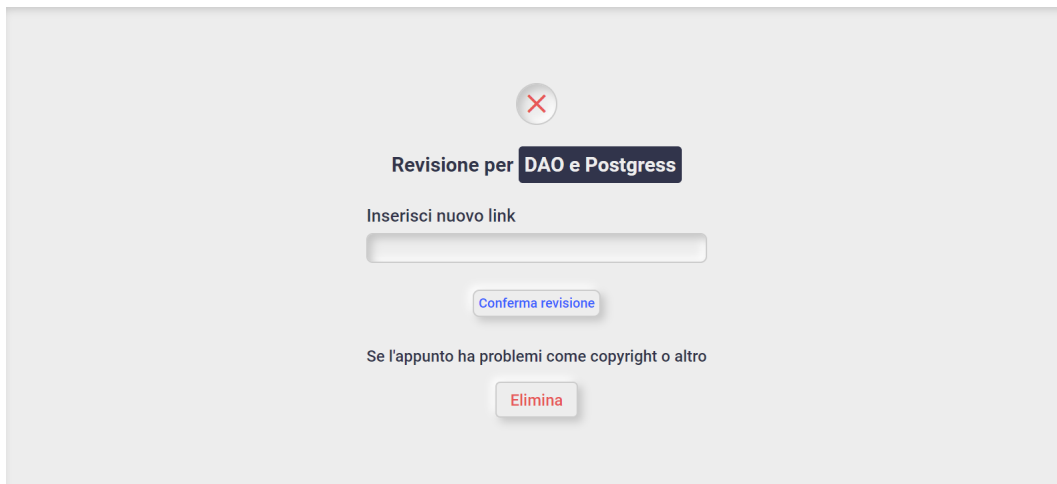
The image shows a web form for revising a note. At the top, there is a red 'X' icon in a circle. Below it, the text 'Revisione per' is followed by a dark blue button labeled 'DAO e Postgress'. Underneath, there is a text input field with the placeholder 'Inserisci nuovo link'. Below the input field is a blue button labeled 'Conferma revisione'. At the bottom, there is a line of text 'Se l'appunto ha problemi come copyright o altro' followed by a red button labeled 'Elimina'.

Figura 3.22: Form di Revisione Appunto

L'implementazione di questa funzione è estremamente semplice. Poiché l'appunto è mostrato direttamente nella lista si conosce l'ID del suo Documento nel Database. Per questo motivo è facile accedere direttamente al documento e modificare la proprietà *revisionato*, una booleana che permette al contenuto di essere scaricato agli utenti in caso di valore *true*, rendendola da *false* a *true*. Per fare in modo che quest'azione potesse essere effettuata esclusivamente da un *amministratore* sono state scritte delle regole apposite per la limitazione di scrittura sui Documenti in quella Collection.

Sezione di gestione dei contenuti

La sezione di gestione dei contenuti riguarda tutti i contenuti che sono presenti all'interno del sito che non sono stati aggiunti dagli utenti, ad esempio i *Corsi di Laurea*, gli *Insegnamenti dei Corsi di Laurea* e gli *Annunci*. La funzione più grande all'interno della Sezione è sicuramente quella di *Aggiunta ed Eliminazione* di un Insegnamento all'interno di un Corso di Laurea. Per fare ciò gli amministratori utilizzando un form presente nella sezione *Insegnamenti* accessibile dal collegamento presente in Figura 3.20

L'aggiunta di un Insegnamento è piuttosto semplice, poiché bisogna esclusivamente aggiungere i dati del form in un *oggetto* e utilizzare quell'oggetto per assegnare le proprietà al Documento all'interno della Collection *corsidelcdl*.

Il codice in Figura 3.23 è esplicativo per capire il funzionamento dell'implementazione. Unico controllo che viene effettuato è quello sul *codiceCorso* che dev'essere univoco per ogni Insegnamento aggiunto all'interno del Database.

```
const aggiungiCorso = () => {
  if (listaCorsi.find((elem) => elem.data().codiceCorso == codiceCorso)) {
    messaggio = 'Corso già esistente! 🚫';
  } else {
    let dati = {
      anno: annoCorso,
      cd1: cd1.id,
      cfu: cfu,
      codiceCorso: codiceCorso,
      nome: nomeCorso,
      professore: professore,
      oreInsegnamento: oreInsegnamento,
      linkScheda: linkScheda
    };

    const queryCheck = query(
      collection(db, 'corsidelcd1'),
      where('codiceCorso', '==', codiceCorso)
    );
    getDocs(queryCheck).then((res) => {
      if (res.empty) {
        addDoc(collection(db, 'corsidelcd1'), dati)
          .then(() => {
            messaggio = `Corso aggiunto con successo a ${cd1.data().nome}`;
          })
          .catch((error) => {
            messaggio = error.message;
          });
      } else {
        alert('Corso con lo stesso codice già esistente!');
      }
    });
  }
};
```

Figura 3.23: Implementazione di aggiungiCorso

L'eliminazione di un Insegnamento è totalmente diversa, poiché dopo la sua creazione un Insegnamento potrebbe avere delle *recensioni*, delle *domande* o degli *appunti*. Alla sua eliminazione, ognuno di questi elementi dev'essere eliminato dal Database per non lasciare Documenti vaganti senza nessun collegamento coi contenuti del sito. Per questo motivo, se l'amministratore decidesse di eliminare un Insegnamento dal Corso di Laurea, dovrebbe seguire i seguenti passaggi:

- Eliminare tutte le recensioni
- Eliminare tutte le domande
- Eliminare tutti gli appunti

- Eliminare l’Insegnamento

Poiché queste operazioni sono lunghe da effettuare manualmente, è presente la funzione *eliminaCorso* che si occupa di effettuare tutti i dovuti passaggi.

```
const eliminaCorso = async () => {  
  caricamento = true;  
  await eliminaRecensioni(corso).then(() => {  
    eliminaDomande(corso).then(() => {  
      eliminaAppunti(corso).then(() => {  
        deleteDoc(corso.ref)  
          .then(() => {  
            })  
          .catch((err) => {});  
      });  
    });  
  });  
};
```

Figura 3.24: Implementazione di eliminaCorso

Sezione di controllo

Nella sezione di controllo vengono mostrati 3 micro-elementi. In particolare, viene mostrata una lista degli utenti del sito, con la possibilità di filtrare i nomi tramite un campo di testo. Oltre a ciò è presente un *registro delle attività* che mostra le *domande*, *risposte*, *recensioni* e *appunti* pubblicati dagli utenti in ordine decrescente di pubblicazione, avendo un limite di 5 per ognuno.

L'elemento più importante è ciò che si trova in fondo alla pagina. Un pulsante dedicato agli amministratori con il quale possono effettuare un controllo sull'*integrità dei dati* del Database. Con esso si risolve il problema citato nelle precedenti sezioni della tesi.

N.B: Questa funzione effettua un elevatissimo numero di letture, con altrettante scritture. Perciò il suo utilizzo è consigliato con un ritmo di massimo 1 utilizzo al giorno.

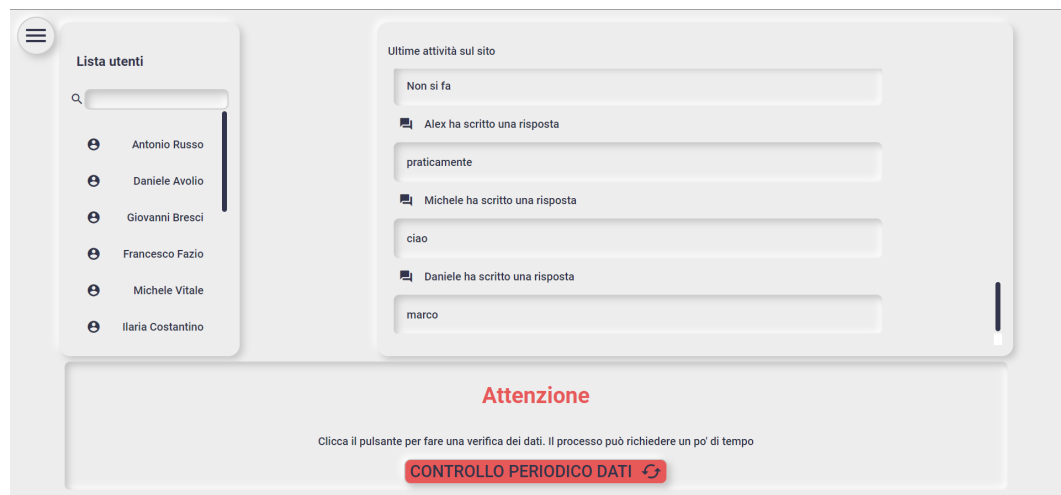


Figura 3.25: Sezione di Controllo

Ciò che questa funzione permette di fare è *controllare l'integrità dei dati* in modo manuale e, in caso di manomissione, cambiare direttamente il contenuto manomesso. Il motivo per il quale il numero di letture è così alto è che viene controllato *ogni singolo Insegnamento del Database*.

Quando viene chiamata la funzione di controllo, per ogni Insegnamento di ogni Corso di Laurea viene calcolata manualmente la *media*. Successivamente, per ogni esame vengono calcolate tutte le medie di *difficoltà e utilità*, controllando ogni *recensione* appartenente a quell'esame. Arrivati a questo punto dovrebbe essere chiaro il motivo per il quale la funzione è destinata solo agli amministratori ed è consigliato l'utilizzo di massimo una volta al giorno. Se le medie calcolate dovessero essere *diverse* dal valore di media all'interno del documento dell'Insegnamento, quest'ultimo verrà *sovrascritto* per avere il nuovo valore integro al 100%, calcolato su tutti i dati disponibili del Database.

```

const checkMediaEsame = async () => {
  const queryEsami = query(collection(db, 'corsidelcdl'));
  let esami = await getDocs(queryEsami);

  // Per ogni esame, controllo la media
  for (const esame of esami.docs) {
    let mediaCorrente = esame.data().mediaVoti;
    let mediaCalcolata = 0;
    let sommaCalcolata = 0;
    // Controllo, per ogni esame, che ci siano il giusto numero di voti e la somma corretta
    const queryLibretto = query(
      collection(db, 'esamilibretto'),
      where('uidCorso', '==', esame.id)
    );
    let esamiRegistrati = await getDocs(queryLibretto);
    for (const esameRegistrato of esamiRegistrati.docs)
      sommaCalcolata += esameRegistrato.data().voto;

    // Calcolo la media
    mediaCalcolata = Math.floor(sommaCalcolata / esamiRegistrati.docs.length);

    if (mediaCorrente == null || mediaCalcolata == NaN) {
      // fai niente
    } else {
      // Se la media è errata, sostituiscila con quella corretta
      await setDoc(
        doc(db, 'corsidelcdl', esame.id),
        {
          mediaVoti: mediaCalcolata
        },
        { merge: true }
      );
    }
    progress.value += 50 / esami.docs.length;
  }
}

```

Figura 3.26: Implementazione di checkMediaEsame

L'implementazione del calcolo della media di difficoltà e utilità è praticamente identica, perciò è omessa dall'esempio.

Per capire il livello di progresso del processo di controllo è stato implementato una formula che mostra graficamente con una *progress bar* l'attuale percentuale di completamento del processo. L'aumento della barra è stato calcolato in base al numero di insegnamenti che sono presenti all'interno del sito, aggiungendo un valore pari a $50/\text{numeroInsegnamenti}$ che permette di arrivare al completamento della prima parte della progress bar. L'ulteriore 50% viene raggiunto tramite la funzione di calcolo media difficoltà e utilità.

Capitolo 4

Discussione

Lo sviluppo del sito oggetto di questo lavoro di tesi ha richiesto la definizione di diversi algoritmi e l'implementazione di svariate funzionalità, le tre di maggior rilievo discusse nella sezione precedente. Questa sezione riprende le funzionalità già descritte e altre di minor rilievo con l'obiettivo di illustrare i problemi che potrebbero esistere in caso di evoluzione del progetto. In particolare, la discussione è focalizzata alle conseguenze di un aumento del numero di utenti e le possibili risoluzioni applicabili.

4.1 Elevato numero di letture

Durante lo sviluppo e durante le fasi di debugging il *numero* di letture e scrittura sul database è stato molto elevato, dovendo controllare che tutti i documenti fossero correttamente compilati e caricati o cancellati. Per questo motivo, uno dei miglioramenti possibili al progetto potrebbe essere quello di cambiare radicalmente la struttura di alcuni documenti all'interno di alcune Collection.

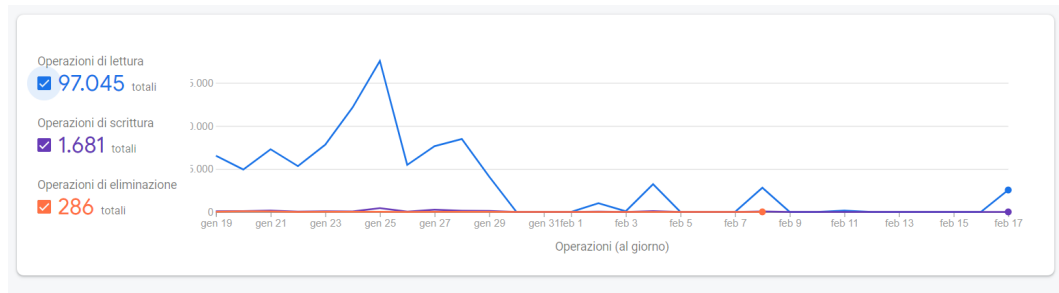


Figura 4.1: Grafico di utilizzo di Firebase per 30 Giorni

In Figura 4.1 viene illustrato l'andamento dell'utilizzo delle funzioni per leggere e scrivere i dati sul database. In particolare si nota un utilizzo *intenso* nelle giornate 24 e 25 Gennaio, poiché sono state le ultime riguardanti le implementazioni e controllo delle funzionalità. Andando avanti nei giorni si nota un andamento molto più *leggero*, poiché i controlli da effettuare sul codice e sugli algoritmi implementati erano ormai quasi nulli. In dettaglio, l'utilizzo del 24 e 25 Gennaio è stato il seguente:



Figura 4.2: Utilizzo del 24 e 25 Gennaio

Le scritture nel sito non sono quasi mai arrivate ad un numero critico, ma questo era abbastanza scontato poiché il numero di documenti da creare è notevolmente inferiore rispetto al numero di letture. Le letture, che nel caso di Figura 4.2 sono 17.541 effettuate esclusivamente da una singola persona, sono il problema principale di questo progetto. Questo problema è derivato dal fatto che i documenti all'interno del sito vengono letti ogni volta che una pagina viene caricata nella maggior parte dei casi. Un esempio potrebbe essere il *caricamento del profilo utente*.

Caricamento del profilo utente

Quando viene visitata la pagina del profilo di un utente, vengono effettuate numerose chiamate per ottenere alcune delle informazioni più importanti da mostrare nella pagina. Poiché molte delle funzionalità sono vincolate ad utilizzare dei dati da ottenere dal database, come *la lista dei insegnamenti* di un determinato Corso di Laurea che viene usata per costruire il form di aggiunta degli insegnamenti del libretto, questi dati vengono richiesti quando la pagina viene caricata.

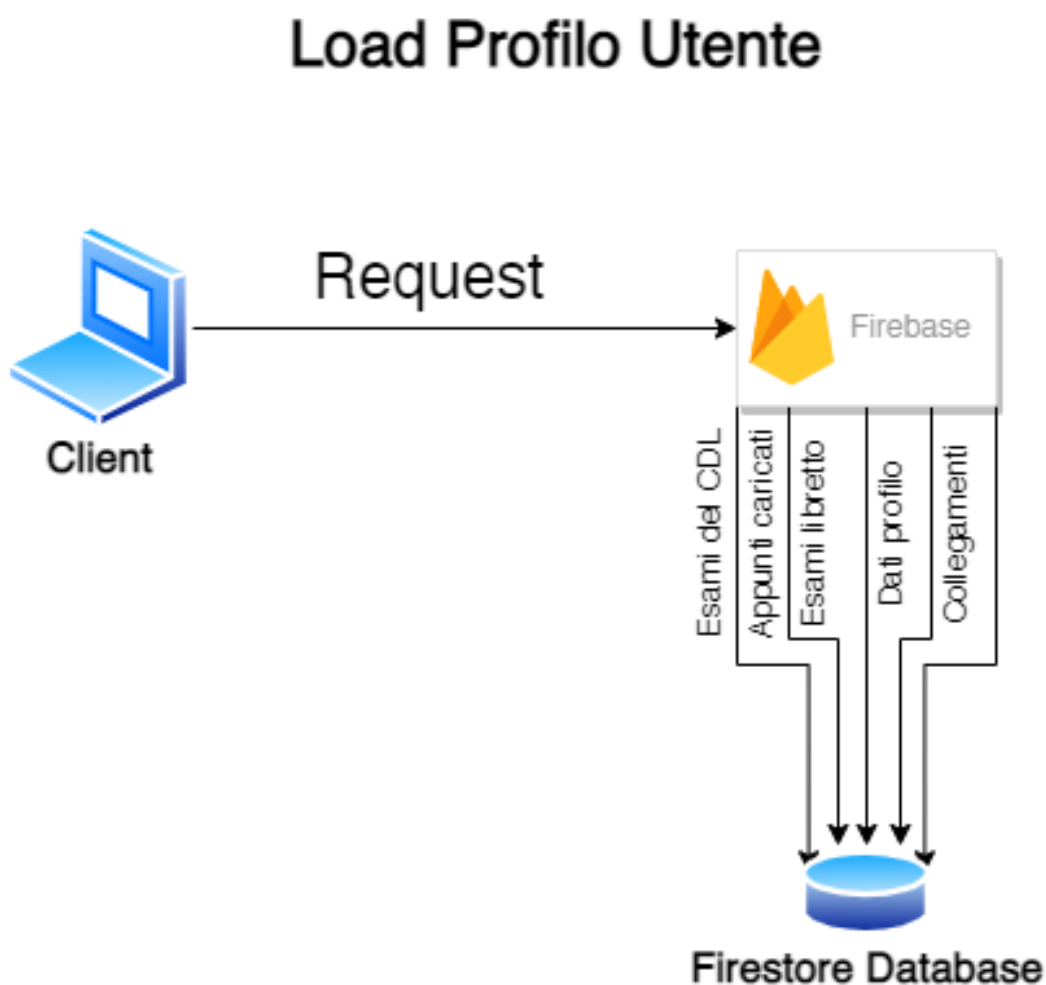


Figura 4.3: Esempio di caricamento pagina profilo utente

Come mostrato in Figura 4.3, ogni volta che viene caricato il profilo di un utente, vengono effettuate almeno 5 chiamate alle funzioni di Firebase per

ottenere i documenti necessari a costruire la pagina. Il numero di letture, però, viene calcolato in base ai documenti che vengono restituiti dalla funzione.

Ad esempio: Se il numero di *esami superati* di uno studente fosse 16, la chiamata alla funzione sarebbe unica, ma il numero di letture sarebbe pari al numero di documenti letti dal database. In questo caso, il numero di letture sarebbe pari a 16.

Avendo una visione più estesa del problema, pensando a più utenti che caricano più pagine nell'arco della giornata, il numero di letture sul database arriva ad essere molto elevato. La soluzione di questo problema comporterebbe dei cambiamenti radicali nell'implementazione della quasi totalità delle pagine. Precisamente, i cambiamenti dovrebbero essere 2:

1. Strutturare i documenti con più proprietà per evitare di dover ricavare alcuni dati con altre letture.
2. Cambiare la logica di richiesta di alcuni documenti, dovendoli leggere esclusivamente on demand.

Per il punto 1 bisognerebbe analizzare per intero il database, capendo cosa potrebbe evolvere da documento a sé stante a diventare una proprietà di un qualche altro documento. Ad esempio, per evitare di dover leggere gli esami di uno studente uno ad uno avendo 1 lettura per documento, si potrebbero trasformare quei documenti in un *array* da inserire all'interno del documento dell'utente, rendendo tutti gli esami accessibili con una singola lettura.

Per verificare se questa ristrutturazione possa essere una soluzione al problema bisogna ragionare sulla logica di inserimento degli esami nel libretto e nell'utilizzo che si fa di queste informazioni. Se l'utilizzo di questi dati fosse possibile nonostante questo cambiamento, allora questa evoluzione potrebbe avere esito positivo e porterebbe ad una diminuzione del numero di letture da fare per arrivare al medesimo risultato. Questi cambiamenti, però, hanno bisogno di uno studio approfondito sulla gestione futura per quanto riguarda l'elaborazione di questi dati, poiché un documento potrebbe essere richiesto in più parti del sito. Ciò significa che andrebbe rimodellata la strut-

tura rispettando tutti i vincoli che il sito ha in confronto con quei determinati dati.

Il risparmio riguardo il numero di letture che si otterrebbe spostando gli esami dell'utente all'interno del documento dell'utente stesso è rappresentato da questo grafico:

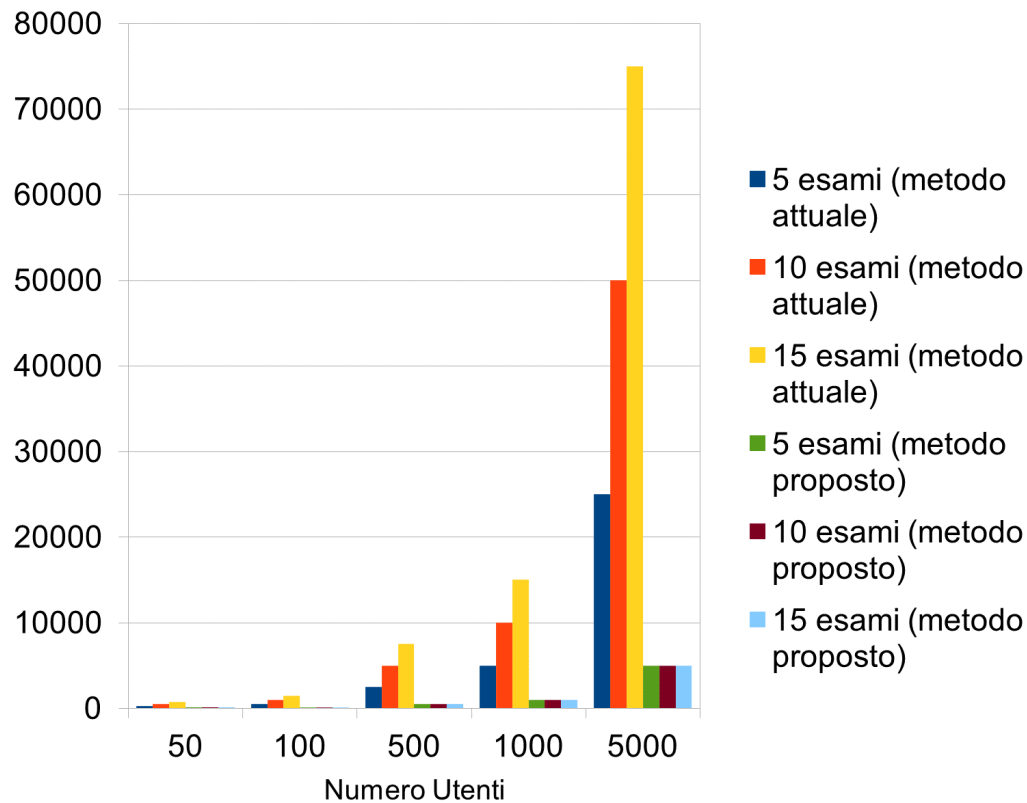


Figura 4.4: Grafico di confronto tra i 2 metodi

In Figura 4.4 viene mostrato come diminuirebbe il numero di letture se gli esami dell'utente fossero caricati all'interno del proprio documento, al contrario di avere un documento per ogni esame caricato nel libretto. È facile notare la differenza notevole che c'è tra i due metodi, poiché il numero di letture che vengono fatte utilizzando il *metodo proposto* è notevolmente inferiore rispetto al *metodo attuale*. Ovviamente, questo cambiamento non potrebbe essere apportato senza effettuare i dovuti accertamenti, controllando tutte le funzionalità del sito che comprendono l'utilizzo degli esami nel libretto utente.

Per il punto 2, invece, il quantitativo di modifiche da effettuare e pianificare è nettamente inferiore. Un esempio pratico è il passaggio di una *lista di documenti* che viene passata ad un Component generico. Per passare la lista, solitamente, si effettua una richiesta al caricamento della pagina, per poi costruire il Component utilizzando la lista passata. Se la lista dovesse contenere un quantitativo elevato di documenti, potrebbe essere dispendioso a lungo andare se effettivamente il Component non dovesse essere utilizzato, avendo così delle letture inutili.

Se, invece, la richiesta per ottenere la lista dovesse essere effettuata *esclusivamente* alla costruzione del Component, allora il quantitativo di chiamate *inutili* sarebbe ridotto. Sicuramente il risparmio sulle letture non sarebbe alto quanto una modellazione diversa dei dati come citato nel Punto 1, ma comunque porterebbe ad un miglioramento delle prestazioni da non sottovalutare.

4.2 Backend e sicurezza

Come ripetuto più volte, all'interno del progetto è presente un problema legato alla *sicurezza*, aggirato utilizzando un utility creata all'interno della dashboard dell'amministratore. Il progetto, essendo sviluppato utilizzando un piano gratuito, non permette l'utilizzo delle Cloud Functions da eseguire sui server di Google, perciò una soluzione da attuare sarebbe quella di utilizzare un backend esterno a Firebase.

4.2.1 Express.js e Firebase

Poiché l'utilizzo delle Cloud Functions è negato, una delle alternative potrebbe essere quella di creare una propria API per gestire il caricamento dei contenuti e la gestione dell'elaborazione dei dati, collegandosi in modo indiretto a Firebase. L'utilizzo di un framework come *Express.js* potrebbe essere una soluzione.

Cos'è Express.js

Express.js è un framework open-source basato su Node.js che permette la creazione di API di routing e permette di rispondere a delle richieste HTTP. La semplicità di Express permette di sviluppare applicazioni web in modo rapido e dando molta flessibilità ed efficienza al programmatore.

Una soluzione per il problema della sicurezza potrebbe essere quello di spostare tutte le operazioni di scrittura critiche, come lo sono ad esempio l'aggiunta degli esami nel libretto degli studenti o l'aggiunta delle recensioni nelle pagine dei corsi dal Client al Server, utilizzando Express per sviluppare un API per gestire il tutto.

Scelte a confronto

Mettendo in conto che i *costi* che potrebbe avere il mantenimento di un Server e il *tempo* per implementare le funzionalità delle API per gestire il problema, converrebbe fare un confronto tra i costi del piano dedicato alle Cloud Functions di Firebase e il lavoro da svolgere per mantenere e sviluppare un sistema di backend da zero. Dipendentemente dai limiti imposti dal progetto, Firebase fornisce supporto al programmatore e funzioni decisamente più semplici da utilizzare rispetto ad un backend da implementare totalmente da zero.

4.3 Ulteriore soluzione per integrità dei dati

Un'ulteriore soluzione per fare in modo che i dati mostrati all'interno del sito siano effettivamente quelli corretti potrebbe essere *dividere* in due parti le informazioni. In particolare, per i dati che ne avessero bisogno si potrebbero creare due raggruppamenti diversi di dati: i dati *validati dagli amministratori* e i dati *caricati dagli utenti*.

I *dati validati dagli amministratori* sono quei dati che hanno subito il trattamento di verifica tramite dashboard amministratore (vedi Sezione 3.4.3) e che sono corretti al 100%.

I *dati caricati dagli utenti* sono i dati che subiscono una variazione ogni volta che un utente interagisce con loro, ad esempio la *media di superamento* di un esame.

La nuova soluzione comporterebbe la creazione di una *sub-collection* all'interno del documento che rappresenta l'insegnamento. Il motivo per cui c'è bisogno di questo ulteriore passaggio è quello di regolamentare i diritti di accesso alla *scrittura* del documento. Questa struttura per la creazione di una gerarchia di regole è stata trattata in 2.1.1. Poiché i dati verificati dagli amministratori dovranno essere visibili a tutti ma modificabili esclusivamente dagli amministratori, le regole della sub-collection saranno le seguenti:

```
// Per modificare i corsi del cdl devi essere loggato
match/corsidelcdl/{corso}{
  allow write: if request.auth.uid != null
  allow read: if true;

  // Per leggere scrivere in dati validati bisogna essere admin
  match/dativalidati/{dati}{
    allow write: if isAdmin();
    allow read:if true;
  }
}
```

Figura 4.5: Regole per la sub-collection dativalidati

In questo modo se qualcuno dovesse effettuare una richiesta per scrivere sul documento e non fosse un amministratore, allora la richiesta sarebbe negata. Avendo l'accesso regolamentato non sarà possibile avere dati falsi all'interno di questa sub-collection.

Gestione della verifica

Questi nuovi dati da verificare avrebbero una piccola problematica, ovvero quella di non essere aggiornati in tempo reale con i dati del database. Questo perché la verifica avverrebbe manualmente quando un amministratore decidesse di utilizzare lo strumento di controllo all'interno della dashboard; fino ad allora i dati caricati dagli utenti e i dati verificati non sarebbero sin-

cronizzati. Il processo di verifica dei dati viene illustrato tramite il seguente flow chart:

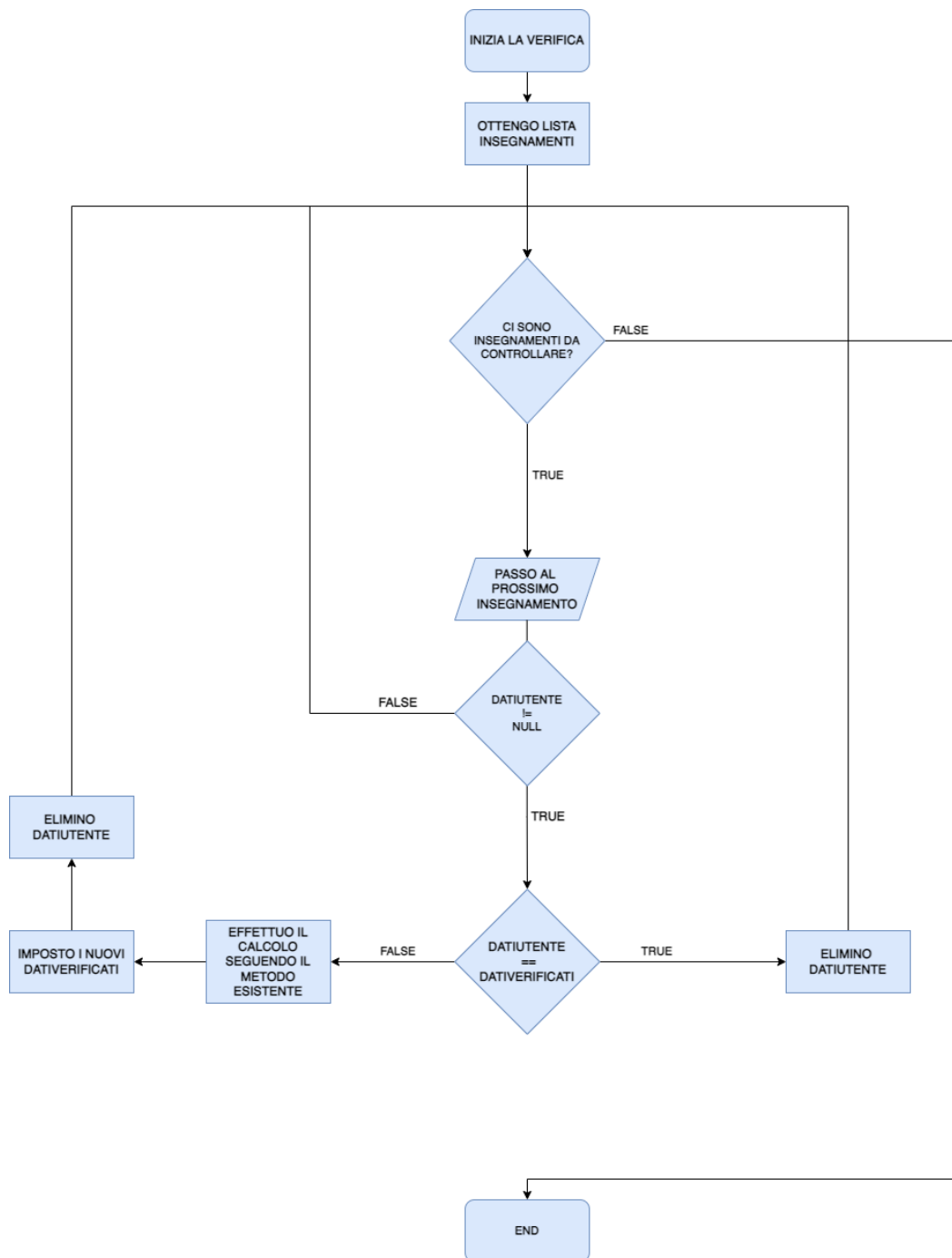


Figura 4.6: Flow chart della gestione dei dati validati

Per riepilogare ciò che viene mostrato in Figura 4.6, si parte con l'esecuzione dell'algoritmo che *ottiene* la lista di insegnamenti e tramite un ciclo che parte dal primo elemento della lista fino all'ultimo controlla se la proprietà

datiutente all'interno del documento esiste o meno. Se il controllo avesse esito *negativo* allora vorrebbe dire che i dati sono sincronizzati e si passerebbe al prossimo insegnamento, se esistente. Se, invece, il controllo risultasse negativo allora bisognerebbe controllare se i *dativalidati* e i *datiutente* siano uguali. In caso di esito positivo, i *datiutente* sarebbero inutili e basterebbe eliminarli per evitare di dover effettuare i calcoli per impostare la proprietà *dativalidati*. In caso di esito negativo, invece, vorrebbe dire che i dati non sono sincronizzati, perciò ci sarebbe bisogno di eseguire le funzioni necessarie per i calcoli per impostare i valori corretti alla proprietà *dativalidati* citate in Sezione 3.4.3. Successivamente a ciò il contenuto di *datiutente* verrebbe eliminato per passare all'insegnamento successivo, se esistente.

Un esempio di documento aggiornato potrebbe essere il seguente:

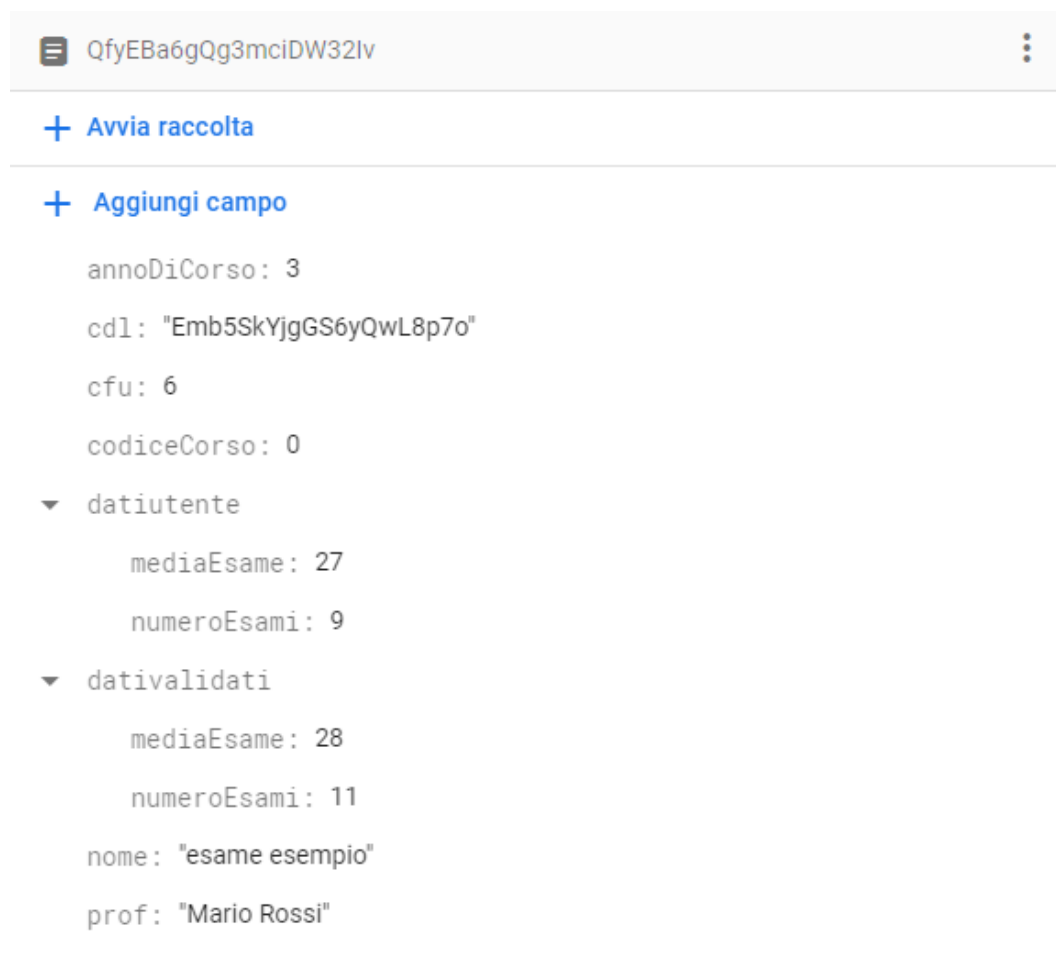


Figura 4.7: Esempio di struttura del documento dell'insegnamento aggiornato

Dipendentemente dalla scelta dello sviluppatore, all'interno della pagina dell'insegnamento dove viene mostrata la media dell'esame si potrebbe optare per due soluzioni:

- Il voto medio mostrato è sempre quello dei *dativalidati*, nonostante il problema della sincronizzazione non immediata.
- Il voto medio mostrato viene scelto in base al seguente criterio:
 - Se *datiutente* == *dativalidati*, vengono mostrati i dati validati.
 - Se *datiutente* != *dativalidati*, vengono mostrati i dati utente con l'aggiunta di un simbolo / tooltip che segnala la possibilità che i dati non siano effettivamente corretti e che saranno aggiornati al più presto.

Metodo alternativo

Alternativamente a ciò che è stato detto in questa questa sezione, ci potrebbe essere un metodo più drastico per avere i dati mostrati costantemente corretti, ma che comporterebbe la totale eliminazione della logica di aggiornamento della media lato client. Questa alternativa imporrebbe come unico modo per calcolare la media degli esami l'utilizzo dello strumento all'interno della dashboard dell'amministratore, eliminando ogni sorta di logica di calcolo e controllo al di fuori di essa. Questo, come detto precedentemente, dipende dallo sviluppatore, poiché non verrebbe mostrato alcun dato all'interno della pagina dell'insegnamento fino a quando l'amministratore non decidesse di verificare i dati.

Senza dubbio i miglioramenti possibili non sono finiti qui, poiché non è stata trattata la gestione di *autenticazione* e la gestione dello *storage* entrambi forniti da Firebase, ma per questa Tesi di Laurea è stato scelto di limitare la lista di miglioramenti a questi problemi.

Capitolo 5

Conclusioni e sviluppi futuri

In questo lavoro di tesi è stato progettato e implementato un sito web per permettere agli studenti di interagire tra loro e caricare contenuti inerenti agli esami appartenenti al proprio Corso di Laurea e potersi supportare a vicenda.

Lo sviluppo e l'implementazione sono basati per la maggior parte su *Javascript*, utilizzando SvelteKit come *Framework* e *Firebase* come "backend as a service". L'intera gestione delle interazioni tra Client e Server avviene tramite *Firebase*, il che ha portato a delle scelte di modellazione di dati e implementazioni di algoritmi vincolati a rispettare dei limiti dovuti alla sicurezza. Per questo motivo sono state implementate delle regole per limitare ciò che gli utenti possono e non possono fare all'interno del sito. Gli studenti, dopo la registrazione, hanno la possibilità di scegliere se rendere il contenuto del loro profilo pubblico, limitarlo solamente alle persone con cui sono *collegate* oppure renderlo privato a chiunque.

I contenuti caricati dagli utenti vengono supervisionati dagli amministratori che, tramite una dashboard apposita, applicano delle *punizioni* agli utenti che non rispettano le regole sospendendoli per un determinato lasso di tempo scelto da loro.

5.1 Integrazione con Esse3

Una futura espansione del progetto è senza ombra di dubbio un'integrazione con Esse3, il sistema informativo per la gestione della didattica e della segreteria utilizzato dall'Università della Calabria e da altri atenei italiani. Il poter utilizzare le *API* di Esse3 faciliterebbe l'aggiunta dei Corsi di Laurea e dei singoli insegnamenti all'interno del database, ma non solo. Oltre a queste semplificazioni, lo studente avrebbe la possibilità di accedere al proprio libretto direttamente dal sito, avendolo sincronizzato automaticamente. Inoltre, il problema della sicurezza citato nelle sezioni precedenti non esisterebbe più, poiché i dati non verrebbero caricati *dagli utenti nel database*, ma verrebbero passati direttamente dalle *API di Esse3*.

Questa funzionalità diminuirebbe di molto il carico di lavoro agli amministratori che altrimenti dovrebbero aggiungere all'interno del database ogni Corso di Laurea e ogni insegnamento a mano. Inoltre sarebbe possibile evitare di dover creare un account manualmente, poiché alcuni sistemi dell'Università della Calabria sfruttano un servizio unificato di autenticazione, come ad esempio il "*Servizio online Centro Residenziale*". Questo renderebbe ancora più sicuro il metodo di verifica dell'account, che attualmente è implementato tramite una funzione di *verifica via email* che viene fornita da Firebase.

Un'integrazione del genere comporterebbe dei cambiamenti a livello di implementazione e gestione dei dati, poiché la gestione avviene tramite una comunicazione Client - Firebase, aggiungendo e rimuovendo contenuti all'interno del sito. Se invece alcuni dati come:

- Voto di superamento dell'esame
- Lista di Corsi di Laurea
- Lista degli Insegnamenti
- Tasse da pagare
- Appelli disponibili

- Ecc..

dovessero essere disponibili direttamente utilizzando l'account di un utente utilizzando queste API, il progetto evolverebbe ad un livello superiore di complessità, ma allo stesso tempo lo trasformerebbero dall'essere un semplice portale per studenti ad un vero e proprio sito di *gestione della carriera universitaria*.

Integrazione API Esse3

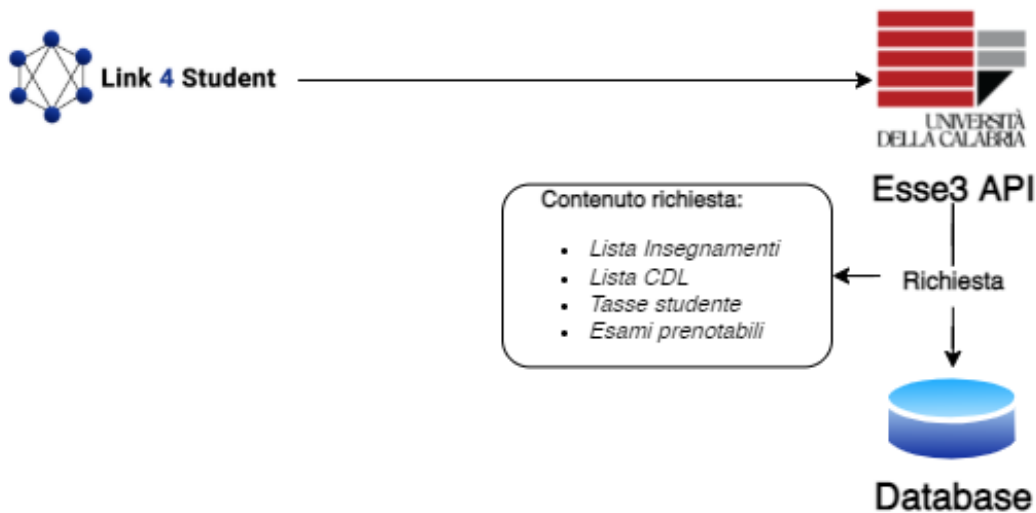


Figura 5.1: Diagramma integrazione API Esse3

5.2 Espansione ad ulteriori Università

Il progetto attualmente è indirizzato per essere utilizzato esclusivamente dagli studenti dell'Università della Calabria. Essendo molto generico nelle implementazioni e non avendo un supporto di un database esterno, non viene difficile pensare che una delle future implementazioni potrebbe essere quella di espandere il progetto al di fuori del territorio della Calabria.

Poiché, come precedentemente spiegato in Sezione 5.1, l'integrazione con un sistema di API Universitarie permetterebbe l'accesso ai dati e gestione efficace dei contenuti del sito, non sarebbe complesso riuscire a soddisfare le

esigenze di altri atenei italiani. Le complicazioni potrebbero esserci se le API di ogni ateneo fossero completamente diverse l'une dalle altre. Se, invece, Esse3 avesse uno schema di API costante tra tutte le Università che ne fanno uso, sarebbe più semplice generalizzare il codice e diminuire il carico di lavoro.

Questa complicità non implica che le API diverse tra loro siano inutilizzabili all'interno del sito, anzi. L'unica "problematica" sarebbe quella di avere un carico di lavoro maggiore rispetto alla precedente opzione. Oltre a questo, ogni ateneo potrebbe avere delle funzionalità personalizzabili all'interno del sito. Ad esempio:

Un ateneo vorrebbe implementare un sistema di sconti universitari che permetterebbero alle aziende di fornire degli sconti agli studenti che utilizzano il servizio.

Questa è solamente una delle opzioni che potrebbero essere aggiunte all'interno di un sistema universitario. Oltre a ciò, le associazioni studentesche potrebbero avere uno spazio apposito all'interno del sito di ogni Università, dove potrebbero caricare i propri contenuti e avere una "bacheca" per avere un canale di comunicazione con gli studenti.

Ulteriori implementazioni potrebbero essere effettuate in futuro dipendentemente dalle esigenze che il richiedente del prodotto dovesse avere.

Bibliografia

- [1] Firebase. Firebase documentation. <https://firebase.google.com/docs/reference/js>.
- [2] Mozilla and individual contributors. Javascript reference. <https://developer.mozilla.org/en-US/docs/Web/JavaScript>.
- [3] Svelte Team. Svelte documentation. <https://svelte.dev/docs>.
- [4] Svelte Team. Sveltekit documentation. <https://kit.svelte.dev/docs/introduction>.

Ringraziamenti

A conclusione di questo lavoro di tesi voglio ritagliare un piccolo spazio per inserire i dovuti ringraziamenti per le persone che mi sono sempre state vicine e per le quelle che sono entrate nella mia vita grazie a questo percorso.

In primis voglio ringraziare il mio relatore, il professore Alviano Mario per avermi guidato e dato dei preziosi consigli per arrivare alla conclusione di questo elaborato.

Ringrazio la mia famiglia, in particolare modo mia Madre e mio Padre per i sacrifici ai quali sono andati in contro per fare in modo che io potessi arrivare fino a questo punto. Ringrazio mio fratello Francesco per avermi indirizzato verso questo percorso, dandomi sempre il supporto di cui avevo bisogno. Un grazie a Nonna e Nonno per esserci sempre stati, dandomi supporto morale e non durante qualsiasi occasione.

Un ringraziamento va alle persone che ho conosciuto grazie all'Università, con cui ho passato momenti belli e momenti meno belli, ma che rimarranno sempre scolpiti nel mio cuore.

In particolare, un ringraziamento ad Alex, persona stupenda con cui non mi aspettavo di legare così tanto e a cui voglio più bene di quanto pensi. A Domenico, che c'è sempre stato dall'inizio e che è rimasto fino alla fine, nonostante non avessimo avuto molto tempo tempo per stare insieme. A Simone, la persona con la quale ho trascorso forse più tempo da quando sono arrivato all'Università, con il quale ho sempre potuto parlare e soprattutto discutere di tutto senza nessun problema. Un grazie anche a Giuseppe, Stefano, Emanuele, Lorenzo, Carlo, a Peppe e Alessandro con i quali ho costruito dei bellissimi ricordi.

Un ringraziamento alle persone che ci sono state fin dall'inizio con le quali ho passato più tempo. A Roberto, ormai presente da quando ho memoria, a Giovanni con il quale il rapporto va oltre quello familiare, a Marco che anche se lontano è sempre vicino, a Laura che da poco è entrata a far parte di questa famiglia e ad Antonio sempre più presente e disponibile.

Un ringraziamento speciale va a Pasquale, che nonostante la lontananza, il tempo passato e le strade diverse che abbiamo intrapreso è sempre presente e mai troppo lontano per avere spazio nella mia vita.

Infine, un ringraziamento speciale va a Ilaria, la mia compagna di vita che ormai da anni è al mio fianco e continua a darmi la forza per andare avanti e raggiungere i miei obiettivi, tirandomi su il morale, incitandomi a non mollare mai e dandomi la grinta necessaria a superare qualsiasi ostacolo durante questo cammino.

Grazie.