



UNIVERSITÀ
DI TRENTO

DIPARTIMENTO DI INGEGNERIA
E SCIENZA DELL'INFORMAZIONE

Sleep Code

PROGETTO PER IL CORSO DI INGEGNERIA DEL SOFTWARE
ANNO ACCADEMICO 2023-2024

Documento di Sviluppo

Descrizione:

Numero documento: D4

Versione documento: 1.0

Membri del gruppo:

Raffaele CASTAGNA

Alberto ROVESTI

Zeno SALETTI

Numero gruppo: G17

Ultima revisione: 3 febbraio 2024

Indice

1	Scopo del documento	3
2	User Flows	4
2.1	Azioni riguardanti l'autenticazione	5
2.2	Azioni riguardanti l'utilizzo del sito	6
3	Documentazione e implementazione dell'applicazione	7
3.1	Struttura del Progetto	8
3.1.1	Directory: __mock__	9
3.1.2	Directory: __test__	9
3.1.3	Directory: public	9
3.1.4	Directory: src	9
3.1.5	.env.local	10
3.1.6	.gitignore	10
3.1.7	jest.config.js	10
3.1.8	package.json	10
3.1.9	README.md	10
3.1.10	postcss.config.js	10
3.1.11	tailwind.config.js	10
3.1.12	tsconfig.json	10
3.2	Dependencies del progetto	11
3.3	Dati del Progetti e Database	11
3.3.1	Database	11
3.4	Project API	16
3.4.1	Estrazione delle risorse dal class diagram	16
3.4.2	API riguardanti l'utente	16
3.4.3	API riguardanti i problemi	16
3.4.4	Resource Models	17
3.5	Sviluppo API	19
3.5.1	Signup	20
3.5.2	deleteAccount	21
3.5.3	changePassword	22
3.5.4	getProblems	23
3.6	Documentazione API	24
4	Frontend	25
4.1	Home	25
4.2	Login	27
4.3	Signup	28
4.4	Recupero Password	29
4.5	Pannello Admin	30
4.6	Pagina Profilo	31

4.7	Catalogo	32
4.8	Problema	33
5	Github Repository e Deployment Info	34

Consigli utili per la consultazione del testo: Se il lettore per file `.pdf` attualmente in uso lo consente, è possibile navigare con più semplicità e velocità all'interno di questo documento cliccando sugli elementi dell'indice.

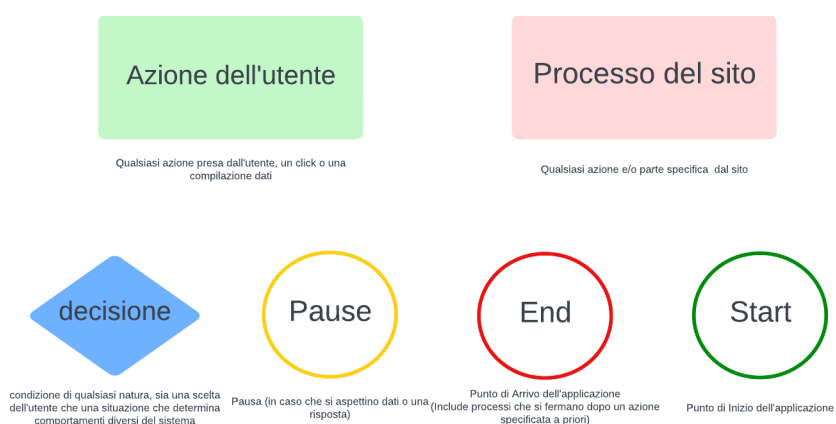
1 Scopo del documento

Il presente documento riporta tutte le informazioni richieste e necessarie per lo Sviluppo di una parte dell'applicazione Sleepcode. In particolare, presenta:

- User Flow legato al ruolo dell'utente (amministratore,autenticato e non)
- User Flow legato all'uso del sito
- Documentazione delle Api attraverso API Model e Modello delle risorse
- Api Fornite per interagire con l'applicazione
- Descrizione delle api fornite
- Risultati delle test suite applicata sulle api

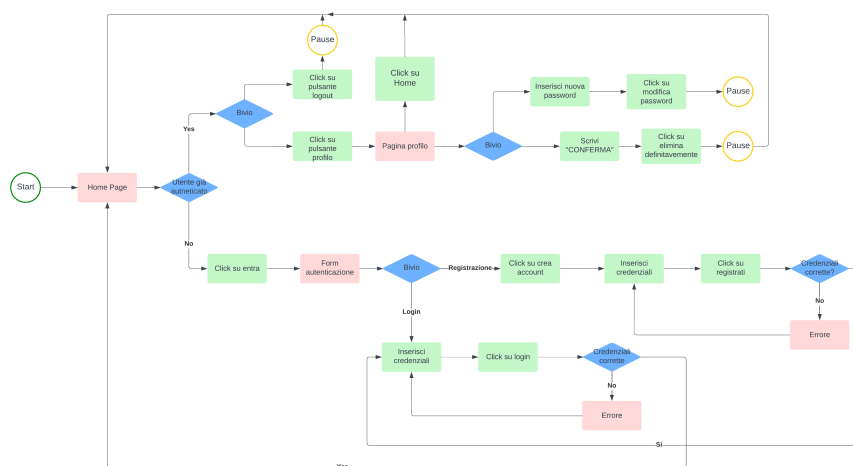
2 User Flows

In questa sezione del documento di sviluppo vengono riportati gli user Flows. Lo scopo degli User Flows è quello di poter specificare le azioni disponibili all'utente e le conseguenze di esse. Sono stati individuati 2 tipi di User flow, uno per tutto ciò che riguarda l'autenticazione e il profilo utente, e un'altro che riguarda le azioni disponibili ai diversi ruoli di utente. Teniamo a ricordare che tutte le immagini sono disponibili ad alta risoluzione nell'appropriata cartella del D4. Di seguito esponiamo la legenda per i simboli utilizzati



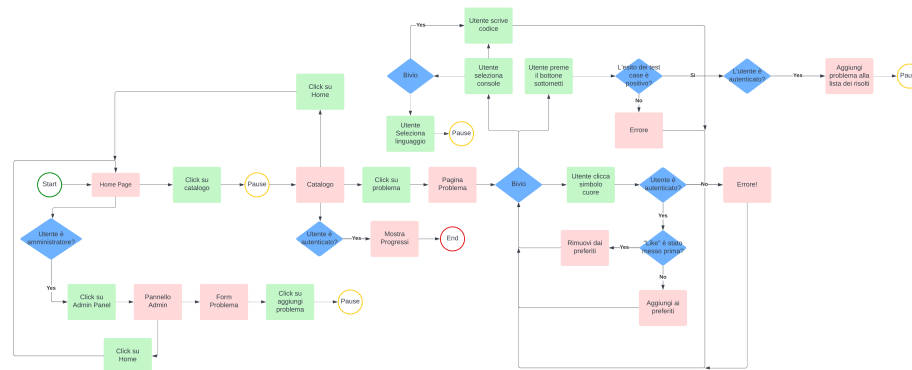
2.1 Azioni riguardanti l'autenticazione

Questo User Flow è specifico per tutte le azioni che riguardo l'autenticazione e ciò che fa parte di essa. Si ricorda che in ogni momento della navigazione l'utente è in grado di poter autenticarsi, tornare alla home e al catalogo tramite una apposita Navbar che è presente in ogni pagina del sito web. Parte di queste interazioni sono state rimosse per rendere l'User Flow più leggibile.



2.2 Azioni riguardanti l'utilizzo del sito

Questo User Flow è specifico per tutte le azioni che ogni tipo di utente può intraprendere nell'utilizzo del sito, teniamo a precisare che la funzione di aggiunta di un problema tramite DB non è stata sviluppata, al momento il form esiste ma non dialoga col database, ci scusiamo per l'inconvenienza. Ricordiamo che tramite Navbar l'utente è in grado di intraprendere tutte le azioni descritte nell'User Flow precedente.

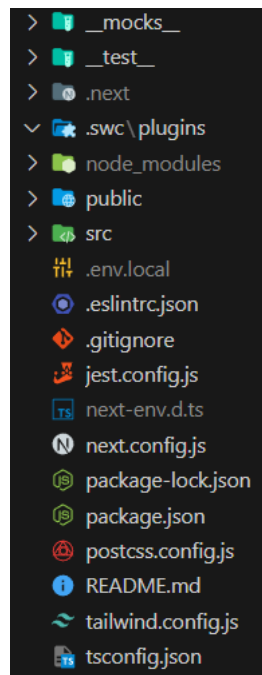


3 Documentazione e implementazione dell'applicazione

Nella precedente sezione abbiamo illustrato tutte le funzioni attualmente implementate nell'applicazione e un'idea di come l'utente può interagire con esse. L'applicazione **SleepCode** è stata sviluppata utilizzando [Next.js](#) vers. 14.0.3, un framework Javascript free-open source basato su [React](#)

3.1 Struttura del Progetto

Il software utilizzato per version control utilizzato è [Git](#), come repository abbiamo utilizzato [Github](#), il codice e la sua history è presente su una repository del membro del Team Raffaele Castagna, l'ultima versione stabile e quella utilizzata per hostare il sito è disponibile presso la repository CodeBase, all'interno di essa, troveremo le seguenti cartelle:



Ricordiamo che le cartelle `.next`, `.swc`, `.env` insieme a `package-lock.json`, `next-env.d.ts`, `eslintrc.json` sono state generate automaticamente

3.1.1 Directory: `__mock__`

Questa cartella contiene delle funzioni "mock" che permettono a Jest di poter effettuare il testing senza fare chiamate dirette al database.

3.1.2 Directory: `__test__`

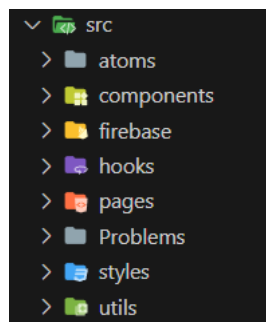
Questa cartella contiene tutti i test case e test suite dedicate al testing.

3.1.3 Directory: `public`

Questa cartella contiene tutte le immagini utilizzate all'interno del sito.

3.1.4 Directory: `src`

Questa cartella contiene tutte le parti del progetto sia front-end che back-end del progetto, procediamo con una vista più dettagliata



- **Atoms:** Abbiamo utilizzato una libreria di State management sviluppata da Google per tenere traccia dello stato dell'utente, la libreria utilizzata è: [Recoil](#), gli atoms non sono altro che gli stati di certi componenti del sito.
- **Components:** Questa cartella contiene tutti i vari componenti del sito che vengono utilizzati dalle pagine, essi possono essere semplici [Scheletri](#) o componenti di maggior importanza.
- **Firebase:** Questa cartella contiene tutti i file relativi al setup di [Firebase](#), il database utilizzato per il progetto.
- **Hooks:** Questa cartella contiene gli hooks sviluppati durante il progetto per compiere una funzione.
- **Pages:** Questa Cartella contiene le varie pagine le loro **routes** e le **API**.
- **Problems:** Questa cartella contiene il tipo di dato generico per i problemi.

- **Styles:** Questa cartella contiene diversi stili pre-impostati utilizzati assieme a [TailwindCSS](#)
- **Utils:** Questa cartella contiene i diversi testi e informazioni relative ai problemi attualmente disponibili, oltre che a form validators e funzioni comuni.

3.1.5 .env.local

Questa file contiene tutte le **variabili locali**, utilizzate per la connessione al database e necessarie per il corretto funzionamento dell'applicativo.

3.1.6 .gitignore

Questa cartella specifica quali file **git** non deve includere nelle varie pull/push requests. (.env.local è più importante in quanto contiene la **Chiave segreta**)

3.1.7 jest.config.js

Questo file contiene la configurazione di **Jest**, libreria utilizzata nel testing.

3.1.8 package.json

Questo file contiene le **dependency** del framework

3.1.9 README.md

Questo file contiene informazioni generali sul progetto.

3.1.10 postcss.config.js

Questo file è stato auto-generato da [TailwindCSS](#)

3.1.11 tailwind.config.js

Questo file contiene **pallet** di colori utilizzati da [TailwindCSS](#)

3.1.12 tsconfig.json

Questo file contiene regole utilizzate da [ESLint](#) un **patter checker** utilizzato durante lo sviluppo

3.2 Dependencies del progetto

Il progetto utilizza diverse librerie, procediamo ad elencarne le più importanti e spiegare il loro funzionamento.

- **CodeMirror**: Utilizzata nel front-end avere uno stile simile a Vs-code durante la scrittura del codice.
- **Split**: Utilizzata nel front-end per rendere la pagina dei problemi dinamica (L'utente è in grado di modificare la grandezza dei componenti).
- **Toastify**: Libreria che offre componenti UI utilizzati per dialogare con l'utente.
- **Recoil**: Libreria di State Management sviluppata da Google.
- **Librerie fornite da Firebase**: Abbiamo utilizzato diverse librerie fornite da firebase come **auth,sdk,admin sdk** e molte altre.
- **Yup**: Utilizzato per la validazione RegEx di dati inseriti
- **ESLint**: Pattern checker utilizzato durante lo sviluppo per ottenere un codice di alta qualità
- **Jest**: Libreria utilizzata per il testing
- **node-mocks-http**: Libreria utilizzata per mandare richieste API finte durante il testing.

Oltre a queste abbia altre dipendenze minore tra vari **hooks** e pacchetti necessari per altri pacchetti inutili da elencare.

3.3 Dati del Progetti e Database

Per il corretto funzionamento del sito, l'applicazione necessita della consultazione di alcuni file locali in seguito elencheremo alcune strutture dati utilizzate.

3.3.1 Database

Il database scelto per memorizzare i dati è **Firebase**, per essere più specifici **Firestore**, il sottoinsieme che si occupa della memorizzazione dei dati in cloud. Sono state individuate due collezioni da dover inserire nel database e due collezioni da tenere in locale per facilitare lo sviluppo così da poter operare in maniera 'strict'

- **Problem**

Questo modello rappresenta un Problema Generico all'interno dell'applicazione, ciò che l'applicazione sa su un problema è differente da ciò che il database tiene memorizzato, questo sia per motivi di facilità che limiti sui dati che possiamo tenere nel cloud. Ci sono alcuni campi la quale funzione non è immediatamente chiara, quindi elencheremo la funzione di ogni campo.

- **id**: l'id del problema, per semplicità l'id dei problemi è il loro titolo in minuscolo con "-" al posto degli spazi.
- **title**: Il nome del problema.
- **problemStatement**: La descrizione del problema.
- **examples**: Contiente tutti gli esempi che vogliamo mostrare all'utente.
- **constraints**: Nel caso gli input e/o output abbiano certe regole da rispettare questa campo le conterrà.
- **order**: Ad ogni problema assegneremo un numero che verrà utilizzato nell'ordinamento dei problema nella pagina principale.
- **startedCode**: Le linee di testo presenti appena si apre un problema per la prima volta
- **handlerFunction**: Funzione associata ad ogni problema che permette la sottomissione e il controllo del codice scritto dall'utente.
- **starterFunctionName**: Nome della funzione associata al problema.

```
export type Problem = {  
  id: string;  
  title: string;  
  problemStatement: string;  
  examples: Example[];  
  constraints: string;  
  order: number;  
  starterCode: string;  
  handlerFunction: ((fn: any) => boolean) | string;  
  starterFunctionName: string;  
};
```

- **Example**

Questo modello rappresenta come deve essere strutturato un esempio all'interno del problem. Include due campi opzionali (**explanation, img**) che possono non essere presenti su alcuni problemi.

```
export type Example = {  
  id: number;  
  inputText: string;  
  outputText: string;  
  explanation?: string;  
  img?: string;  
};
```

- **DBProblem**

Questo modello rappresenta i dati che vengono raccolti dal database riguardanti ogni problema, anche qui abbiamo un campo opzionale **videoId**.

```
export type DBproblem = {  
  id:string;  
  title:string;  
  category:string;  
  difficulty:string;  
  likes: number;  
  order: number;  
  videoId?: string;  
}
```

- **userData** Questo Modello rappresenta i dati che riguardano ogni utente al momento della registrazione, il ruolo di "**User**" viene assegnato inizialmente ad ogni utente e successivamente attraverso console di Firestore verrà cambiato manualmente a "**Administrator**" se si vuole promuovere l'utente.

```
const userData = {  
  uid: newUser.user.uid,  
  email: newUser.user.email,  
  displayName: username,  
  createdAt: Date.now(),  
  updatedAt: Date.now(),  
  likedP: [],  
  solvedProblems: [],  
  role: "User",  
};
```


3.4 Project API

In questa parte del documento presentiamo le API dell'applicazione Sleepcode. Le descriveremo prima e successivamente mostreremo il loro codice.

3.4.1 Estrazione delle risorse dal class diagram

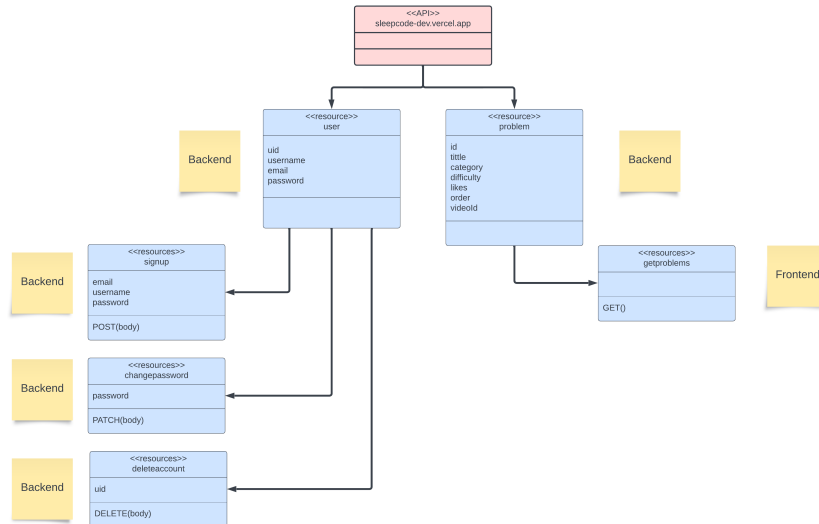
Analizzando il diagramma delle classi abbiamo individuato due risorse principali: l'utente e i problemi, di seguito riportiamo le api:

3.4.2 API riguardanti l'utente

- **signup (POST)**: Questa API permette all'utente non ancora autenticato di creare un account sulla nostra piattaforma per tracciare i progressi. Se tutte le informazioni sono valide l'account verrà creato altrimenti l'utente riceverà un errore.
- **deleteaccount (DELETE)**: Questa API permette all'utente autenticato di eliminare il proprio account e tutte le informazioni associate ad esso. (I "like" lasciati dall'utente rimarranno nel Database in modo da avere uno storico dei problemi più accurato)
- **changepassword (PATCH)**: Questa API permettet all'utente autenticato di poter cambiare la propria password dal sito stesso. Se le informazioni sono valide la password verrà cambiata altrimenti l'utente riceverà un errore.

3.4.3 API riguardanti i problemi

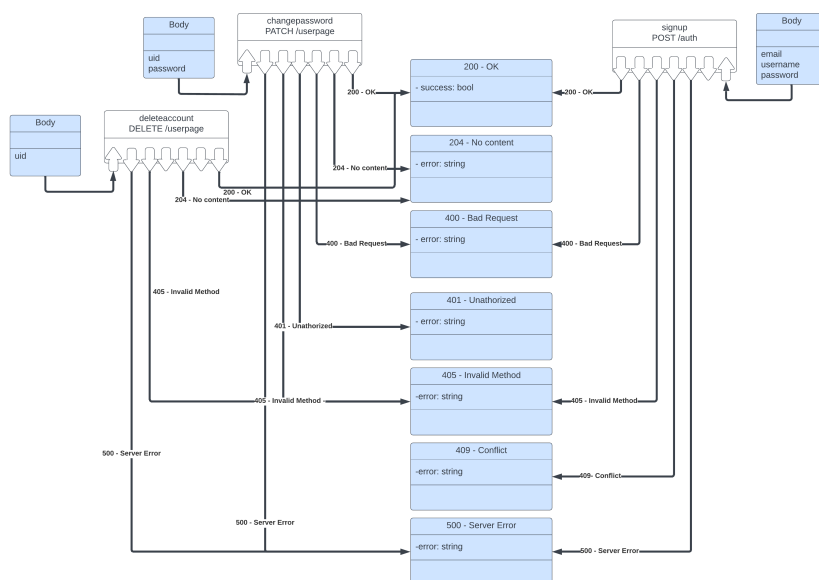
- **getProblems (GET)**: Questa API viene chiamata quando un utente (autenticato o no) si connette alla pagina del catalogo, essa ritorna la lista di problemi disponibili in quel momento.



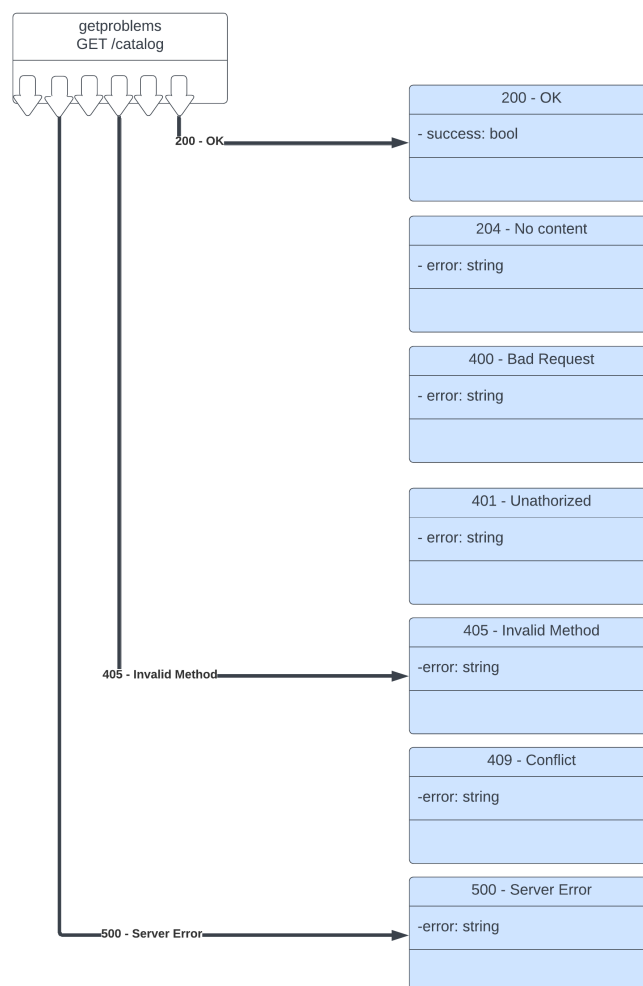
3.4.4 Resource Models

Il Resource model esprime, per ogni API, le diverse risposte, come sono strutturate le richieste e come ci si può accedere. Ogni API dovrà elaborare il body (se lo richiede), ovvero le informazioni necessarie per il corretto funzionamento, e risponderà appropriatamente con uno [Status Code](#). Il body viene rappresentato tramite una freccia che entra e le risposte sono rappresentate tramite delle frecce che escono.

• User API



- **Problem API**



3.5 Sviluppo API

In questa sezione mostreremo il codice relativo alle API.

3.5.1 Signup

```
async function signupHandler(req: NextApiRequest, res: NextApiResponse) {  
  const auth = getAuth();  
  if (req.method !== "POST") {  
    return res.status(405).send({  
      success: false,  
      error: "HTTP method not valid, POST accepted.",  
    });  
  }  
  const schema = Yup.object().shape({  
    email: regSchema.email,  
    username: regSchema.username,  
    password: regSchema.password,  
  });  
  try {  
    await schema.validate(req.body);  
  } catch (error) {  
    return res.status(400).send((error as Error).message);  
  }  
  
  try {  
    const { email, username, password } = req.body;  
    const newUser = await createUserWithEmailAndPassword(  
      auth,  
      email,  
      password  
    ).catch((error) => {  
      console.log((error as Error).message, 409)  
      return res.status(409).send("Email già in utilizzo");  
    });  
    if (!newUser)  
      return res.status(409).send({  
        success: false,  
        error: "Email già in utilizzo",  
      });  
    const userData = {  
      uid: newUser.user.uid,  
      email: newUser.user.email,  
      displayName: username,  
      createdAt: Date.now(),  
      updatedAt: Date.now(),  
      likedP: [],  
      solvedProblems: [],  
      role: "User",  
    };  
    await setDoc(doc(firestore, "users", newUser.user.uid), userData);  
    return res.status(200).send({ success: true });  
  } catch (error) {  
    console.log((error as Error).message, 500)  
    return res.status(500).send((error as Error).message);  
  }  
}
```

Questa API permette ad un utente di registrarsi all'interno dell'applicazione, richiede un'email, un username e una password, attraverso la libreria Yup

verificheremo che le informazioni inserite dall'utente siano accettabili, se accettabili, il sistema proverà a registrare l'utente. In caso abbiamo problemi di conflitto nella creazione dell'utente l'HTTP response avrà come Status Code 409, in caso di una password e/o username malformati avremo uno Status code 400, in caso di un qualsiasi errore non precedentemente previsto avremo HTTP 500, in caso la richiesta è malformata avremo HTTP 405, e in caso di successo HTTP 200.

3.5.2 deleteAccount

```
async function deletehandler(req: NextApiRequest, res: NextApiResponse) {  
  const uid = req.body  
  if (req.method !== "DELETE") {  
    return res.status(405).send({  
      success: false,  
      error: "HTTP method not valid, DELETE accepted."  
    });  
  }  
  
  try {  
    Complexity is 3 Everything is cool!  
    await adminAuth.deleteUser(uid).catch((error: any) => {  
      return res.status(204).send("Errore nell'eliminazione, riprova")  
    })  
    const userRef = doc(firestore, "users", uid)  
    await deleteDoc(userRef).catch((error) => {  
      return res.status(204).send("Errore nell'eliminazione, riprova")  
    });  
  
    return res.status(200).send({ success: true });  
  
  } catch (error) {  
    return res.status(500).send((error as Error).message);  
  }  
}
```

Questa API permette ad un utente autenticato di eliminare il proprio account e tutti i dati relativi ad esso (ricordiamo che per avere uno storico più accurato i like non verranno rimossi dai problemi), dopo che la richiesta è stata ricevuta elimineremo prima l'account dell'utente e successivamente tutti i dati contenuti nel database. In caso la richiesta sia malformata avremo HTTP 405, in caso di problemi (come utente inesistente e/o già eliminato) avremo HTTP 204, in caso di problemi col server avremo HTTP 500, e in caso di successo HTTP 200

3.5.3 changePassword

```
async function changePwhandler(req: NextApiRequest, res: NextApiResponse) {  
  if (req.method !== "PATCH") {  
    return res.status(405).send({  
      success: false,  
      error: "HTTP method not valid, PATCH accepted."  
    });  
  }  
  const schema = Yup.object().shape({  
    uid: uSchema.uid,  
    password: regSchema.password,  
  });  
  try {  
    await schema.validate(req.body);  
  } catch (error) {  
    return res.status(400).send((error as Error).message);  
  }  
  
  try {  
    const {uid, password} = req.body;  
    adminAuth.getUser(uid).catch((error) => {  
      return res.status(204).send("Utente inesistente");  
    });  
    adminAuth.updateUser(uid, {  
      password: password  
      Complexity is 3 Everything is cool!  
    }).catch((error: any) => {  
      return res.status(500).send("Errore del server, riprova");  
    });  
    return res.status(200).send({success: true});  
  } catch (error) {  
    if ((error as Error).message == "auth/id-token-expired") return res.status(401).send("Perfavore Riloggiati");  
    return res.status(500).send((error as Error).message);  
  }  
}
```

Questa API permette all'utente di modificare la propria password, sempre che rispetti i requisiti per essere password. Dopo aver Ricevuto una richiesta, controlleremo se la password rispetta i requisiti per essere tale, dopodichè controlleremo l'esistenza dell'utente e, se esso esiste cambieremo la password con quella fornita. In caso la richiesta sia malformata avremo HTTP 405, in caso la password fornita sia malformata avremo HTTP 400, in caso di utente inesistente avremo HTTP 204, in caso qualcosa vada storto con l'operazione di modifica avremo HTTP 500, e in caso di successo HTTP 200.

3.5.4 getProblems

```
async function getproblemshandler(req: NextApiRequest, res: NextApiResponse) {  
  if (req.method !== "GET") {  
    return res.status(405).send({  
      success: false,  
      error: "HTTP method not valid, GET accepted.",  
    });  
  }  
  
  try {  
    const q = query(collection(firestore, "problems"), orderBy("order", "asc"));  
    const querySnapshot = await getDocs(q)  
    const tmp: DBproblem[] = [];  
    querySnapshot.forEach((doc) => {  
      tmp.push({ ...doc.data() } as DBproblem);  
    });  
    return res.status(200).json(tmp);  
  } catch (error) {  
    return res.status(500).send((error as Error).message);  
  }  
}
```

Questa API restituisce i problemi disponibili al momento della richiesta e tutti i dati relativi ad essi (ordine, titolo, difficoltà, videoId, categoria ecc). Dopo aver ricevuto una richiesta manderemo una query al Firestore per ottenere i dati relativi a tutti i problemi presenti nella collection problems. In caso la richiesta è malformata avremo HTTP 405, in caso la richiesta incontri qualsiasi problema durante la query avremo HTTP 500, in caso di successo HTTP 200.

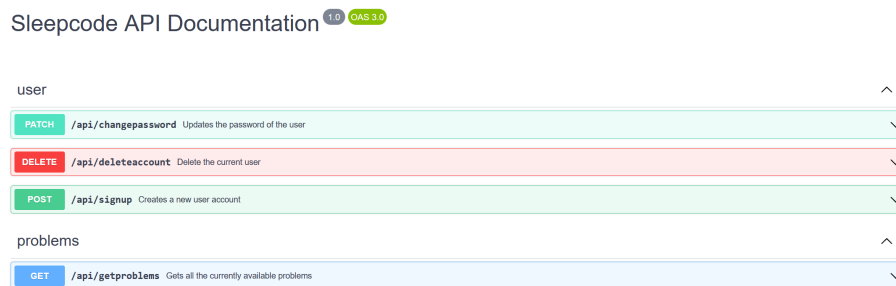
3.6 Documentazione API

Le API fornite dall'applicativo che sono state presentate nella sezione precedente sono state documentate utilizzando il pacchetto NPM [Swagger](#), grazie a questo pacchetto siamo riusciti a generare una pagina web dedicata alla definizione delle specifiche OpenAPI, la quale è disponibile attraverso questo link: [api-doc](#). In caso la documentazione è disponibile nel codice sorgente.

Durante lo sviluppo delle API abbiamo utilizzato diversi metodi:

- GET: utilizzato per ottenere dati da un server, non contiene body.
- POST: utilizzato per creare risorse su un server.
- PATCH: utilizzato per modificare una risorsa su un server.
- DELETE: utilizzato per eliminare risorse da un server.

Di seguito riportiamo un'immagine della pagina della documentazione:



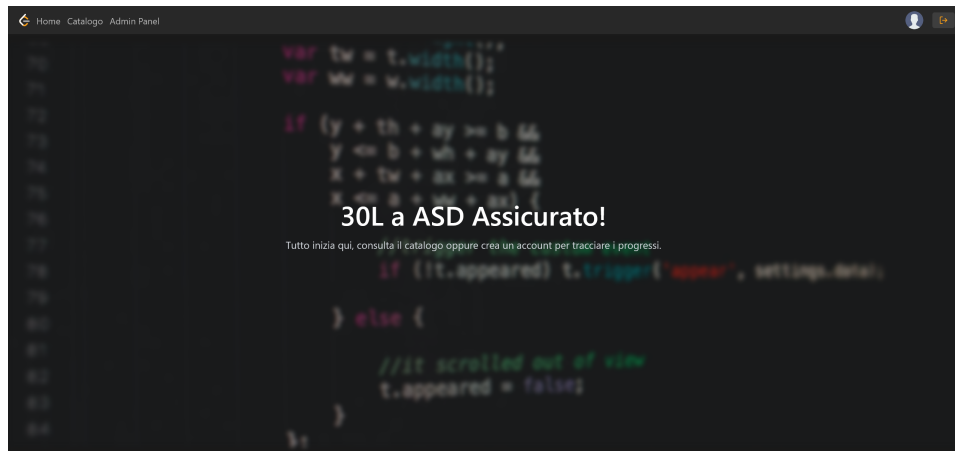
4 Frontend

In questa sezione del documento mostreremo una parte vitale dell'applicazione, il **FrontEnd**, ovvero ciò con cui l'utente interagisce con l'applicazione. Per ogni componente forniremo una breve descrizione delle azioni disponibili all'utente

4.1 Home

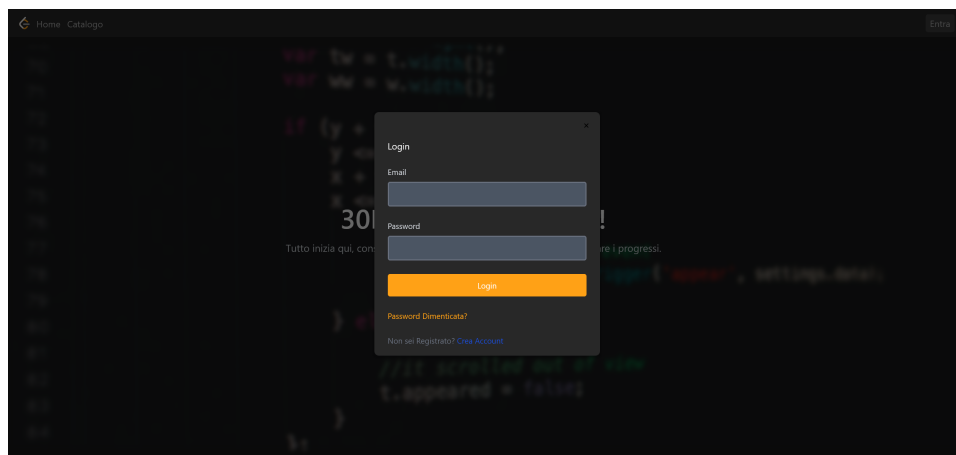
La Home è la prima schermata che un utente vede appena si connette al sito dato l'url del sito, essa ha una NavBar (presente in ogni pagina) che contiene diversi bottoni:

- Home: Questo bottone, se cliccato riporterà l'utente alla pagina principale.
- Catalogo: Questo bottone, se cliccato porterà l'utente alla pagina del catalogo contenente tutti i problemi.
- Admin Panel: Questo bottone appare solo se nel Database l'utente ha come ruolo "Administrator", ricordiamo che come precedentemente illustrato il ruolo di ogni utente è "User" e per promuovere un utente si dovrà interagire con la **CLI** di Firebase per modificare il ruolo del singolo utente, questo può essere fatto solo da persone connesse al progetto sul sito di Firebase.
- Pagina Profilo: Questo Bottone appare solo se l'utente è autenticato, se si appoggia il mouse sopra verrà fornita la mail dell'utente attualmente collegato, in caso cliccato si verrà portati alla pagina del profilo utente.
- Bottone di Login: Questo bottone compare solo se l'utente non è attualmente autenticato, e se cliccato apre il modello di login.
- Bottone di Logout: Questo bottone compare solo se l'utente è autenticato, e se cliccato fa uscire l'utente dall'account.



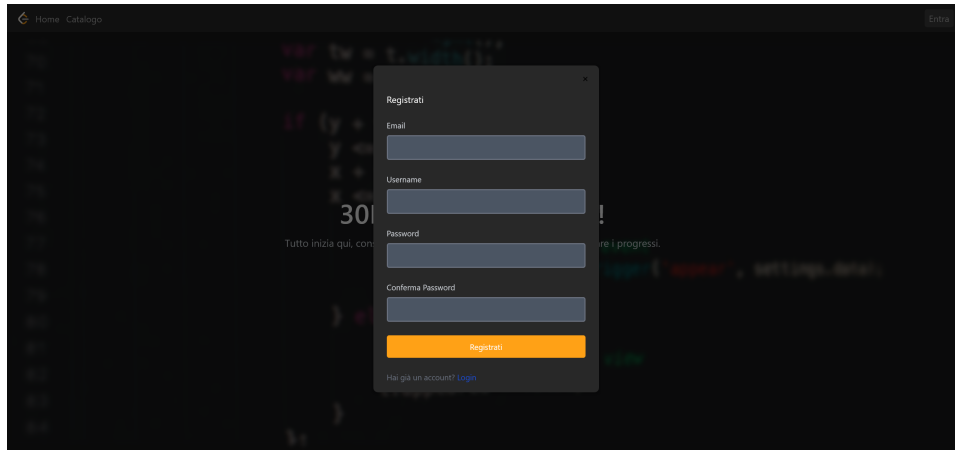
4.2 Login

Questo componente permette ad un utente non ancora autenticato ma registrato di entrare con le proprie credenziali, a causa di regole da firebase non modificabili, la password che viene data non viene controllata se conforme dato che il modello di recupero password di Firebase non permette l'impostazione di regole per la password strength nel piano gratuito. Se l'utente ha inserito le proprie credenziali e sono corrette allora verrà autenticato con messaggio di conferma, altrimenti avrà un messaggio di errore.



4.3 Signup

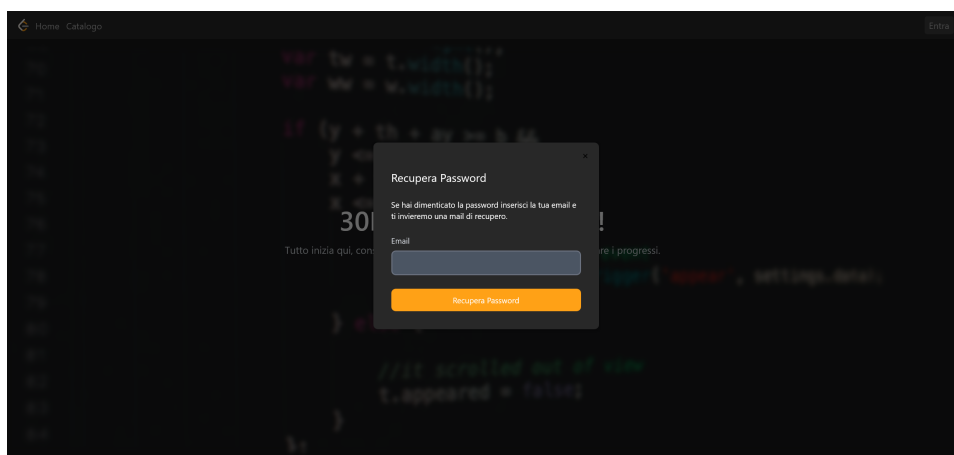
Questo componente permette ad un utente non ancora autenticato di registrarsi, l'utente dovrà fornire un'email, username e password, username e password verranno controllati attraverso **Yup** se rispettano i criteri imposti, in caso contrario l'utente verrà avvertito tramite messaggio di errore. In caso i dati inseriti sono corretti allora l'utente verrà rimandato al componente di Login con messaggio di successo.



The screenshot displays a dark-themed web application interface. A modal window titled "Registrati" is centered on the screen. The modal contains four input fields labeled "Email", "Username", "Password", and "Conferma Password". Below these fields is a prominent orange button labeled "Registrati". At the bottom of the modal, there is a link that says "Hai già un account? [login](#)". The background of the application is dark with faint, colorful code snippets visible.

4.4 Recupero Password

Questo componente permette ad un utente non ancora autenticato di richiedere una mail per recupero password, anche se la mail non è registrata verrà comunque dato un messaggio di conferma di invio, per ragioni di sicurezza. Come già scritto precedentemente a causa di limitazioni col piano Firebase non possiamo impostare regole per le password "recuperate" in quanto gestite da componenti interni di Firebase non disponibili al nostro piano gratuito.



4.5 Pannello Admin

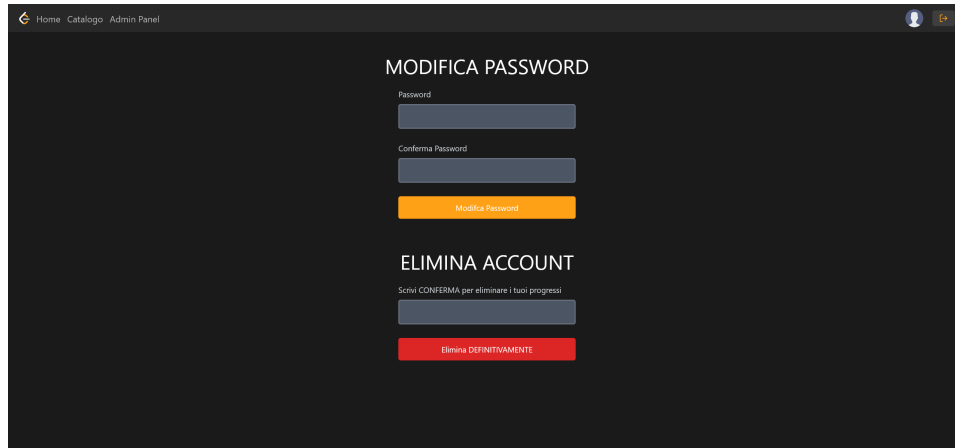
Questo componente (non completamente sviluppata) permette ad un utente amministratore di aggiungere problemi al database, purtroppo per mancanza di tempo non siamo riusciti a sviluppare la funzione in tempo, ci scusiamo per l'inconvenienza.



The screenshot shows a web application interface with a dark theme. At the top, there is a navigation bar with links for 'Home', 'Catalogo', and 'Admin Panel'. A message in the center states: 'Non siamo riusciti a sviluppare in tempo questa funzione, ci scusiamo per eventuali disagi'. Below this message is a form with the following fields: 'id', 'title', 'category', 'difficulty', 'order', and 'videoId?'. Each field has a small red error icon to its left. At the bottom of the form is a button labeled 'Aggiungi al DB'.

4.6 Pagina Profilo

Questa pagina permette ad un utente autenticato di accedere alle funzioni di modifica password e di eliminazione account attraverso appositi form, in caso un utente cerca di accedersi senza essere autenticato verrà riportato alla home.



The screenshot displays a dark-themed web interface for a user profile. At the top, a navigation bar includes links for 'Home', 'Catalogo', and 'Admin Panel', along with a user profile icon and a settings gear. The main content area is divided into two sections. The first section, titled 'MODIFICA PASSWORD', contains two input fields labeled 'Password' and 'Conferma Password', followed by an orange button labeled 'Modifica Password'. The second section, titled 'ELIMINA ACCOUNT', includes a warning message 'Scrivi CONFERMA per eliminare i tuoi progressi' above an input field, and a red button labeled 'Elimina DEFINITIVAMENTE'.

4.7 Catalogo

Questa pagina raccoglie tutti i problemi disponibili, la loro difficoltà, categoria, ordine, titolo, e videoId, e gli offre a tutti gli utenti, in caso l'utente sia autenticato, verrà mostrato un componente che traccia i progressi del singolo utente, e i problemi risolto e/o aggiunti ai preferiti. Quando un qualsiasi utente si connette verranno mostrati degli "Scheletri" per mostrare all'utente che la pagina sta aspettando informazioni dal server, e che di conseguenza dovrà aspettare.

STATUS	TITOLO	DIFFICOLTÀ	CATEGORIA	SOLUZIONE
✓	Two Sum	Facile	Array	▶
♥	Reverse Linked List	Difficile	Linked List	▶
	Jump Game	Medio	Dynamic Programming	▶
	Valid Parentheses	Facile	Stack	▶
	Search a 2D Matrix	Difficile	Binary Search	▶

1/5 Problemi Risolti

4.8 Problema

Questa pagina esiste per ogni problema al momento disponibile è suddivisa in 3 parti:

- **Descrizione:** Contiene tutte le informazioni e/o immagini relative al problema, oltre ad indicare e permettere di aggiungere un problema ai preferiti e visualizzare il numero di "like" e capire se il problema è già stato risolto. Oltre a questo abbiamo un numero variabile di esempi per aiutare l'utente nella risoluzione del problema.
- **Console:** Dove l'utente può scrivere codice e selezionare il proprio linguaggio preferito, al momento l'unico linguaggio disponibile è Javascript, siccome altri linguaggi richiederebbero server dedicati per la compilazione del file. Ricordiamo di non modificare la firma della funzione.
- **Testcase:** Dove l'utente è in grado di selezionare quale test case vedere e sottomettere o provare a risolvere il problema attraverso appositi bottoni. (**Sottometti o Runna**)

Oltre a questo ogni utente (anche non autenticato) è in grado di cliccare sull'icona a forma di orologio e cronometrare il proprio tempo di risoluzione del problema, il fermare o resettare il cronometro è compito dell'utente.

In caso l'utente connesso alla pagina non è autenticato esso non potrà tracciare i propri progressi né aggiungere problemi ai preferiti, in caso si prova a eseguire quest'ultima azione verrà riportato un messaggio di errore.

The screenshot shows the SleepCode interface for the 'Two Sum' problem. The interface is divided into three main sections: Description, Console, and Testcase.

Description: The problem is titled '1. Two Sum'. It includes a description in Italian: 'Dato un array di integer nums e un integer target, ritorna gli indici di due numeri tali che la loro somma sia target. Puoi assumere che ogni input abbia esattamente una soluzione, e che tu non possa utilizzare lo stesso elemento più di una volta.' It provides three examples (Esempio 1, 2, 3) with input, output, and explanation. Constraints are listed at the bottom:

- $2 \leq \text{nums.length} \leq 10$
- $-10 \leq \text{nums}[i] \leq 10$
- $-10 \leq \text{target} \leq 10$

Console: The console shows the function signature: `1 v function TwoSum(nums,target) {`, `2 //Scrivi qui il tuo codice`, and `3 }`.

Testcase: The Testcase section shows three cases (Case 1, Case 2, Case 3). Case 1 is selected, showing the input: `nums = [2,7,11,15], target = 9` and the output: `[0,1]`. There are buttons for 'Console', 'Runna', and 'Sottometti'.

5 Github Repository e Deployment Info

La repository del progetto è disponibile al seguente link:

[G17-UniTn](#)