



UNIVERSITÀ  
DI TRENTO

DIPARTIMENTO DI INGEGNERIA  
E SCIENZA DELL'INFORMAZIONE

# SleepCode

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

---

## Documento di Sviluppo

---

*Descrizione:* descrizione dello sviluppo e della realizzazione di una parte dell'applicazione SleepCode.

*Numero documento:* D4

*Versione documento:* 1.0

*Membri del gruppo:*

Raffaele CASTAGNA

Alberto ROVESTI

Zeno SALETTI

*Numero gruppo:* G17

*Ultima revisione:* 6 febbraio 2024

## Indice

<b>1</b>	<b>Introduzione</b>	<b>3</b>
1.1	Scopo del documento . . . . .	3
1.2	Soddisfacimento dei requisiti . . . . .	3
<b>2</b>	<b>User Flows</b>	<b>4</b>
2.1	Azioni riguardanti l'autenticazione . . . . .	5
2.2	Azioni riguardanti l'utilizzo del sito . . . . .	6
<b>3</b>	<b>Documentazione e implementazione dell'applicazione</b>	<b>7</b>
3.1	Struttura del Progetto . . . . .	7
3.1.1	Directory: __mock__ . . . . .	8
3.1.2	Directory: __test__ . . . . .	8
3.1.3	Directory: public . . . . .	8
3.1.4	Directory: src . . . . .	8
3.1.5	.env.local . . . . .	9
3.1.6	.gitignore . . . . .	9
3.1.7	jest.config.js . . . . .	9
3.1.8	package.json . . . . .	9
3.1.9	README.md . . . . .	9
3.1.10	postcss.config.js . . . . .	9
3.1.11	tailwind.config.js . . . . .	9
3.1.12	tsconfig.json . . . . .	9
3.2	Dependencies del progetto . . . . .	10
3.3	Dati del Progetti e Database . . . . .	11
3.3.1	Database . . . . .	11
3.4	Project API . . . . .	14
3.4.1	Estrazione delle risorse dal class diagram . . . . .	14
3.4.2	API riguardanti l'utente . . . . .	14
3.4.3	API riguardanti i problemi . . . . .	14
3.4.4	Resource Models . . . . .	15
3.5	Sviluppo API . . . . .	17
3.5.1	Signup . . . . .	18
3.5.2	deleteAccount . . . . .	19
3.5.3	changePassword . . . . .	20
3.5.4	getProblems . . . . .	21
3.6	Documentazione API . . . . .	22
<b>4</b>	<b>Frontend</b>	<b>23</b>
4.1	Home . . . . .	23
4.2	Login . . . . .	24
4.3	Signup . . . . .	24
4.4	Recupero Password . . . . .	25

4.5	Pannello Admin . . . . .	26
4.6	Pagina Profilo . . . . .	26
4.7	Catalogo . . . . .	27
4.8	Area di esercitazione . . . . .	27
<b>5</b>	<b>Github Repository &amp; Deployment Information</b>	<b>29</b>
<b>6</b>	<b>Testing</b>	<b>30</b>
6.1	Test API . . . . .	30
6.1.1	test api/signup . . . . .	30
6.1.2	test api/changepassword . . . . .	31
6.1.3	test api/deleteAccount . . . . .	31
6.1.4	test api/getProblems . . . . .	31
6.2	Code Coverage . . . . .	32

---

**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 Introduzione

## 1.1 Scopo del documento

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

- user flow diagrams relativi alle operazioni che l'utente può effettuare sull'applicazione;
- documentazione delle API attraverso API Model e Modello delle Risorse;
- API fornite per interagire con l'applicazione e loro descrizione;
- presentazione del design finale del front-end;
- informazioni su GitHub repositories e deployment;
- risultati delle test suite applicata sulle API realizzate.

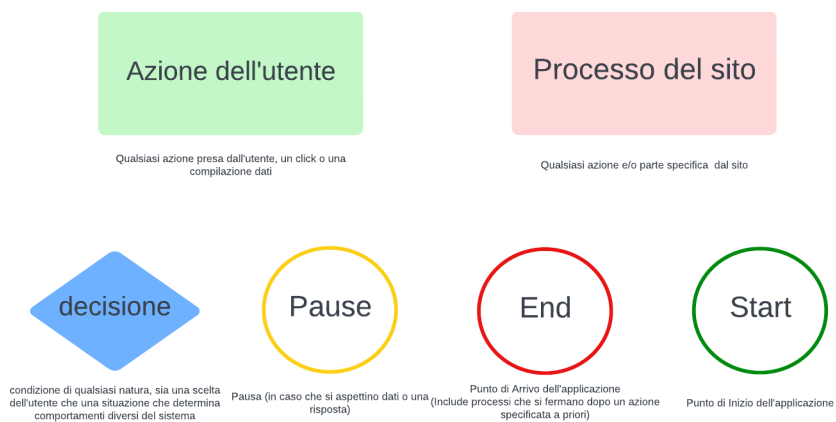
## 1.2 Soddisfacimento dei requisiti

Vengono specificati di seguito i requisiti funzionali (RF) e non funzionali (RNF) implementati interamente nell'applicazione finale. Essi fanno riferimento ai requisiti presentati nel documento D1:

## 2 User Flows

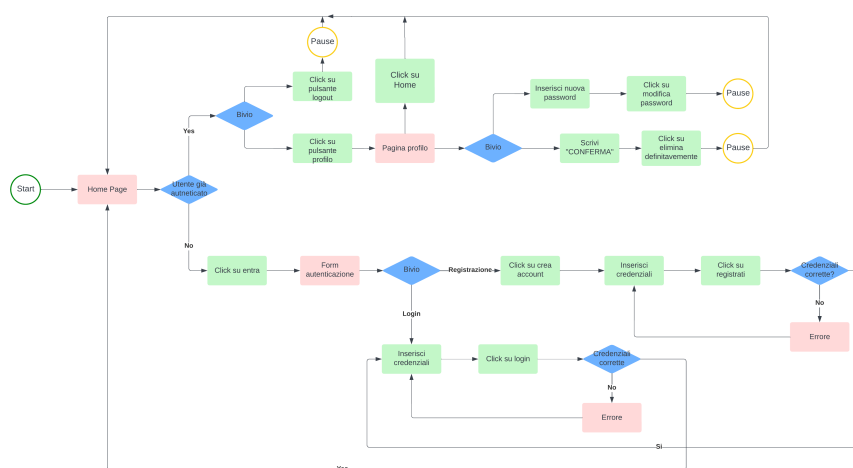
In questa sezione del documento di sviluppo vengono riportati gli User Flows. Lo scopo degli User Flows è quello di poter specificare le azioni disponibili all'utente e le conseguenze di esse.

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



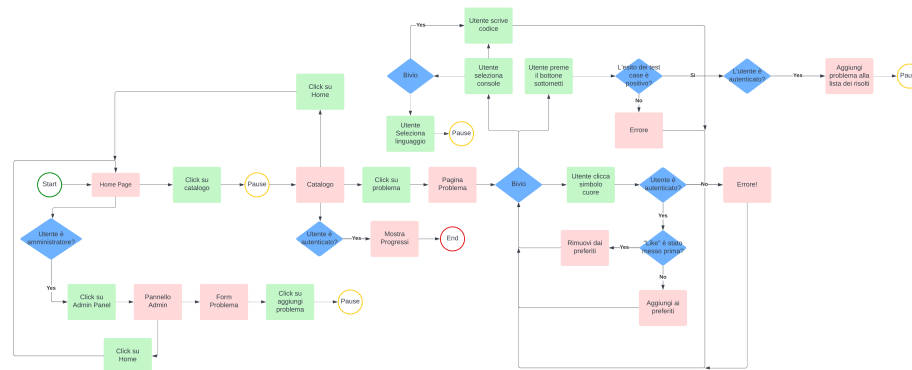
## 2.1 Azioni riguardanti l'autenticazione

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.

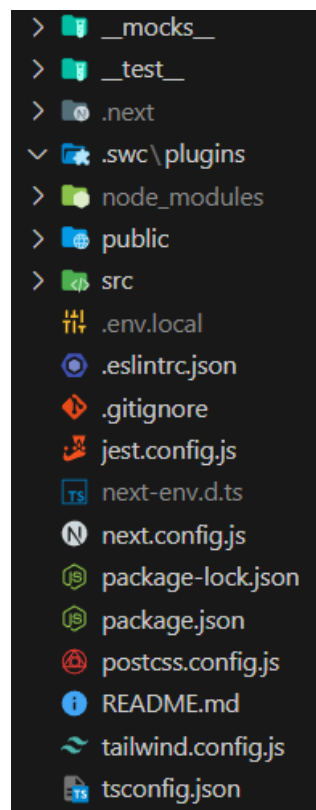


### 3 Documentazione e implementazione dell'applicazione

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](#) (versione 14.0.3), un framework Javascript free-open source basato su [React](#).

#### 3.1 Struttura del Progetto

Il software utilizzato per il version control è [Git](#); come remote repository è stato impiegato [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 della repository, 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__`

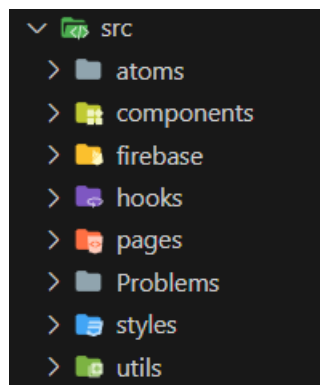
Questa cartella contiene tutti i test case e test suite dedicate al testing.

### 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:** È stata utilizzata 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 la più importante dal momento che essa 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 **dependencies** 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 **palette** 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, conferisce uno stile simile all'interfaccia grafica di VS Code durante la scrittura del codice.
- **Split**: Utilizzata nel front-end per rendere dinamica l'area di esercitazione (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**: Sono state impiegate 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.

### 3.3 Dati del Progetti e Database

Per il corretto funzionamento del sito, l'applicazione ha bisogno di consultare alcuni file locali. In seguito elencheremo alcune strutture dati utilizzate.

#### 3.3.1 Database

Il database scelto per memorizzare i dati è **Firestore**, 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 memorizza; questo è dovuto sia per motivi di semplicità che limiti sui dati che possiamo mantenere nel cloud. Ci sono alcuni campi la cui funzione non è immediatamente chiara, quindi verrà descritta 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**: Contiene tutti gli esempi da 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.
- **starterCode**: 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 indica come deve essere strutturato un esempio all'interno del problema. Esso include due campi opzionali (**explanation**, **img**) che possono non essere presenti in 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 intende promuovere l’utente al livello di amministratore.

```
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 sono presentate le API dell'applicazione SleepCode. Esse saranno prima descritte e, successivamente, verrà mostrato il loro codice.

#### 3.4.1 Estrazione delle risorse dal class diagram

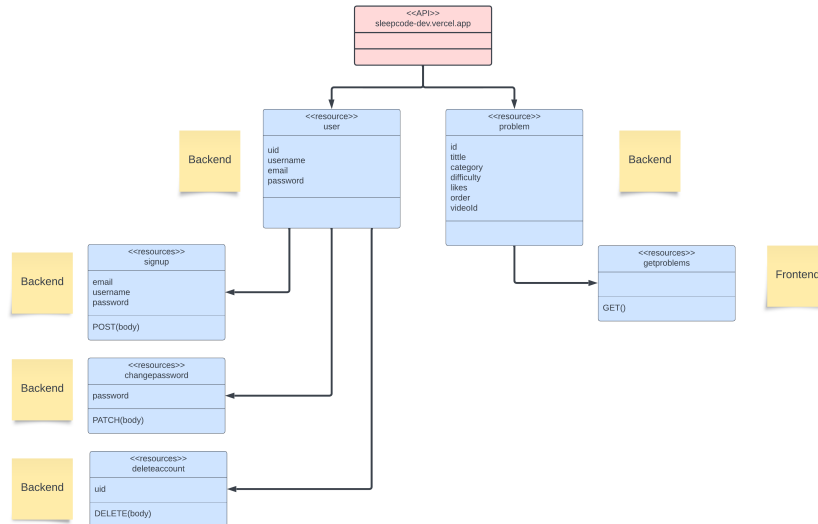
Analizzando il diagramma delle classi sono state individuate due risorse principali: l'utente e i problemi. Di seguito riportiamo le API:

#### 3.4.2 API riguardanti l'utente

- **signUp (POST)**: Questa API permette all'utente non ancora autenticato di creare un account sulla piattaforma per tracciare i progressi e i preferiti. Se tutte le informazioni sono valide, l'account verrà creato normalmente, altrimenti l'utente riceverà una notifica di errore.
- **deleteAccount (DELETE)**: Questa API permette all'utente autenticato di eliminare il proprio account e tutte le informazioni associate ad esso. (I *preferiti* contrassegnati dall'utente e collezionati dai problemi rimarranno nel database in modo da avere uno storico dei problemi, e il loro ammontare di "likes", più accurato).
- **changePassword (PATCH)**: Questa API permette all'utente autenticato di poter cambiare la propria password, effettuando tale operazione sul sito stesso. Se le informazioni sono valide, la password verrà cambiata, altrimenti l'utente riceverà una notifica di errore.

#### 3.4.3 API riguardanti i problemi

- **getProblems (GET)**: Questa API viene chiamata quando un utente (autenticato o non) si connette alla pagina del catalogo. Essa ritorna la lista di problemi disponibili in quel momento.

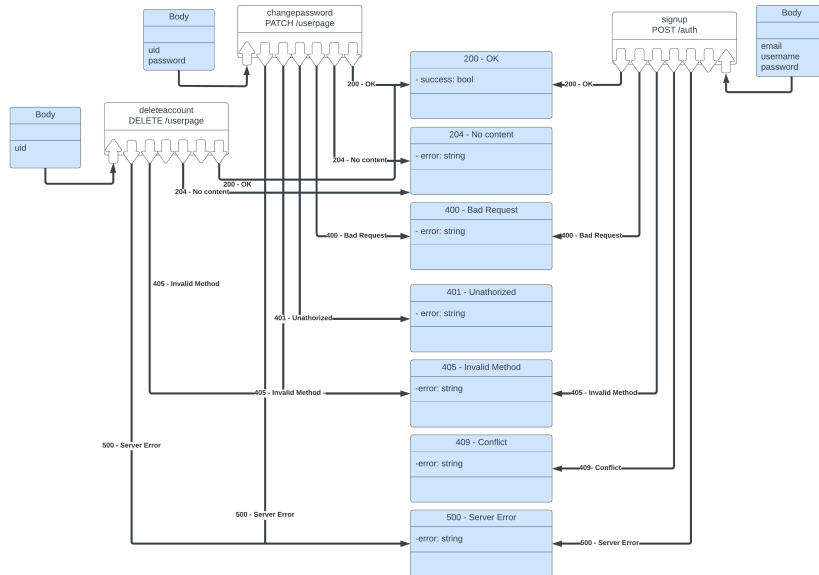


### 3.4.4 Resource Models

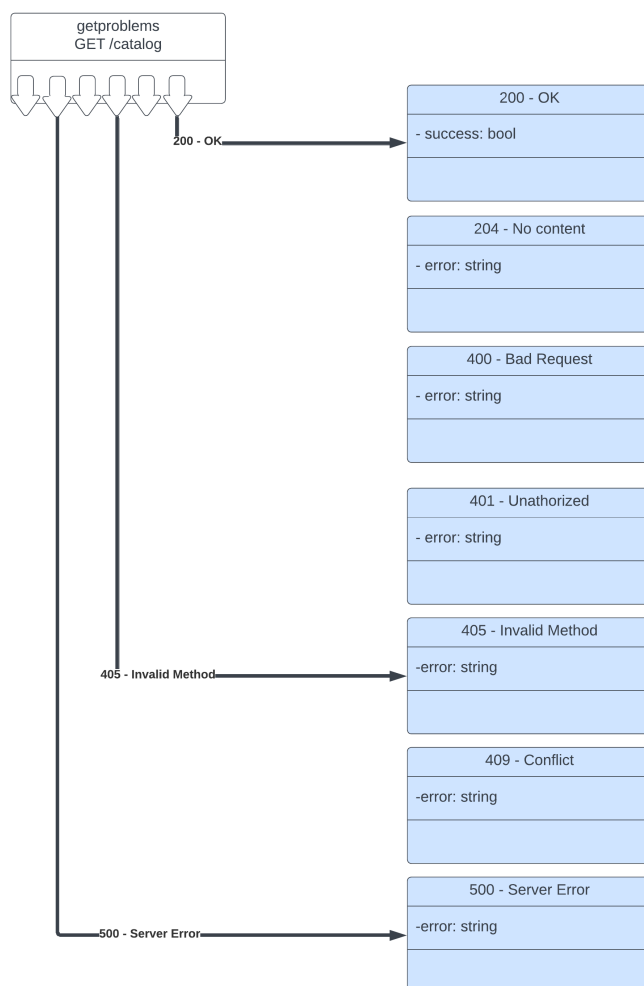
Il Resource model esprime, per ogni API, le diverse risposte, come le richieste sono strutturate e come ci si può accedere. Ogni API dovrà elaborare il body (se lo richiede), ovvero le informazioni necessarie per il corretto funzionamento, e risponderà appropriatamente con uno [Status Code](#). Il body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

- **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: essa richiede un'email, un username e una password. Attraverso la libreria

Yup viene verificato che le informazioni inserite dall'utente siano accettabili; se accettabili, il sistema proverà a registrare l'utente. In caso vi siano problemi di conflitto nella creazione dell'utente, l'HTTP response avrà 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
    }).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 Reloggati");
    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 una password forte. Dopo aver ricevuto una richiesta, si controllerà che la password rispetti i requisiti per essere tale, dopodichè verrà controllata l'esistenza dell'utente e, se esso esiste, la password verrà cambiata con quella appena fornita. In caso la richiesta sia malformata, lo status code sarà HTTP 405, in caso la password fornita sia malformata avremo HTTP 400, in caso di utente inesistente HTTP 204, in caso qualcosa vada storto con l'operazione di modifica, per altre ragioni, lo status code corrisponderà a 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);  
  }  
}
```

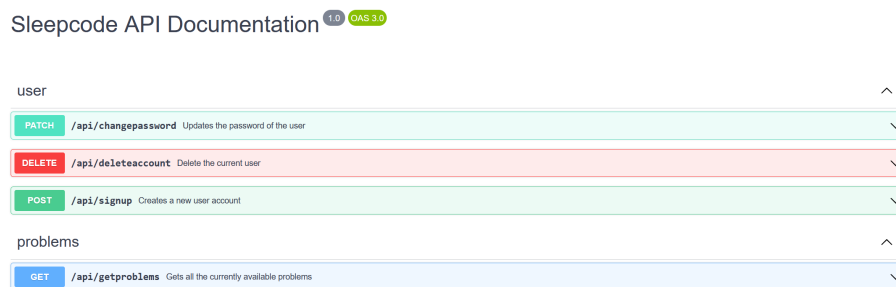
Questa API restituisce i problemi disponibili al momento della richiesta e tutti i dati relativi ad essi (ordine, titolo, difficoltà, videoId, categoria ecc). Dopo aver ricevuto una richiesta, una query verrà inviata al Firestore per ottenere i dati relativi a tutti i problemi presenti nella collection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

### 3.6 Documentazione API

Le API fornite dall'applicativo, presentate nella sezione precedente, sono state documentate utilizzando il pacchetto NPM [Swagger](#). Grazie a questo pacchetto, è stato possibile generare una pagina web dedicata alla definizione delle specifiche OpenAPI (Figura 1), la quale è disponibile attraverso questo link: [api-doc](#). La documentazione è reperibile anche nel codice sorgente.

Durante lo sviluppo delle API sono stati impiegati 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.



**Figura 1:** Un'immagine che mostra la pagina contenente la documentazione

## 4 Frontend

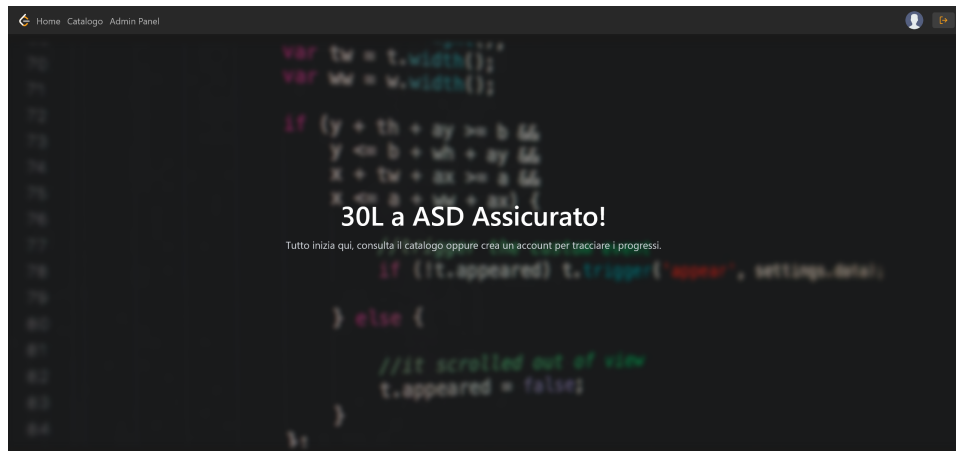
In questa sezione del documento verrà mostrata una parte vitale dell'applicazione: il **front-end**, ovvero la parte dell'applicazione con cui l'utente finale interagisce. Per ogni componente viene fornita una breve descrizione delle azioni disponibili all'utente.

### 4.1 Home

La Home è la prima schermata che un utente vede appena si connette al sito. Essa contiene una NavBar (presente in ogni pagina) che contiene diversi bottoni:

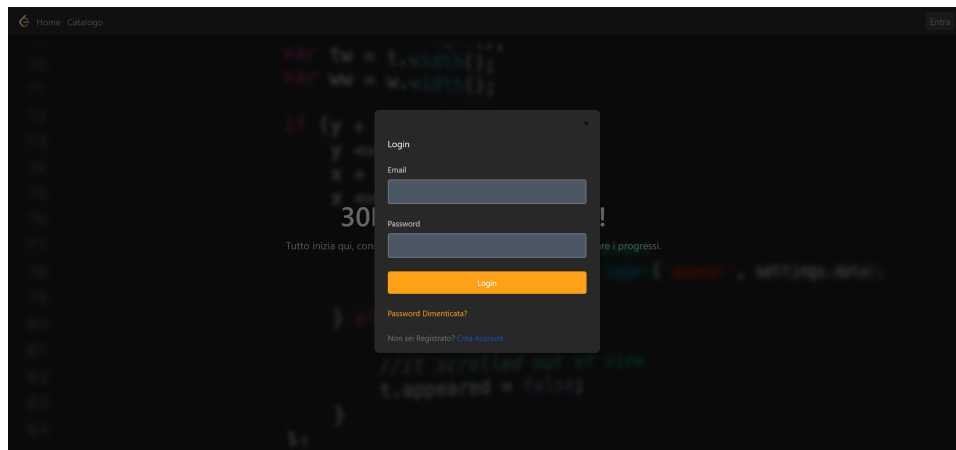
- Home: Questo bottone, se cliccato, riporterà l'utente alla pagina principale.
- Catalogo: Questo bottone, se cliccato, porterà l'utente alla pagina del catalogo contenente tutti i problemi.
- Admin Panel: Questo bottone appare solo se nel Database l'utente ha come ruolo "Administrator", ricordiamo 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 posiziona il cursore del mouse al di sopra, verrà visualizzata l'email dell'utente attualmente collegato; se cliccato, si arriverà alla pagina del profilo utente.
- Bottone di Login: Questo bottone compare solo se l'utente non è attualmente autenticato; ha lo scopo di aprire il form di login.
- Bottone di Logout: Questo bottone compare solo se l'utente è autenticato, e se cliccato effettua l'operazione di logout.





## 4.2 Login

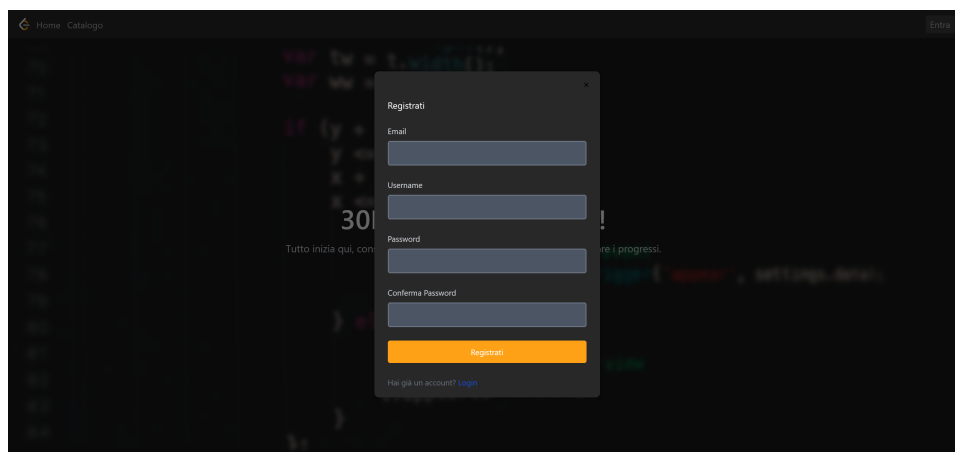
Questo componente permette ad un utente non ancora autenticato ma registrato di entrare con le proprie credenziali. A causa di regole imposte da firebase e non modificabili, la password che viene data non viene controllata se conforme dato che il modello di recupero password di Firebase non permette 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

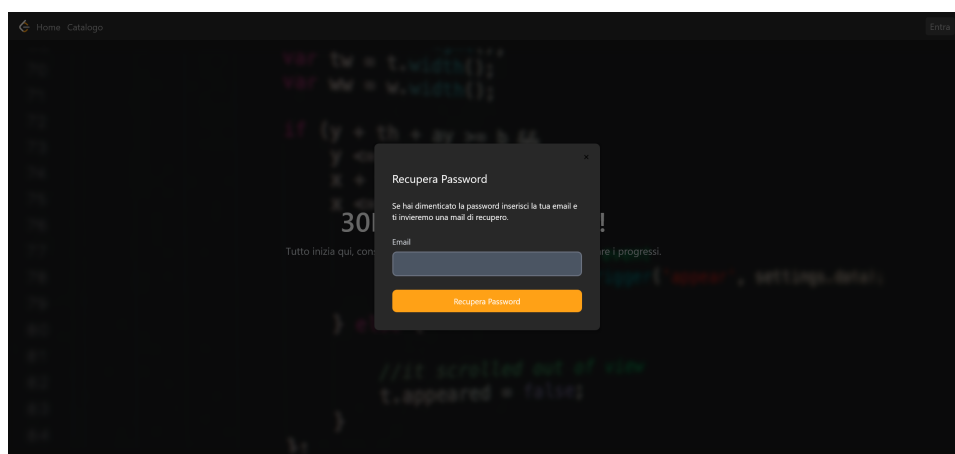
Questo componente permette ad un utente non ancora autenticato di registrarsi, l'utente dovrà fornire un'email, username e password, username e password verranno controllati attraverso **Yup** se rispettano i criteri imposti, in caso contrario l'utente verrà avvertito tramite messaggio di errore. In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

i dati inseriti sono corretti allora l'utente verrà rimandato al componente di Login con messaggio di successo.



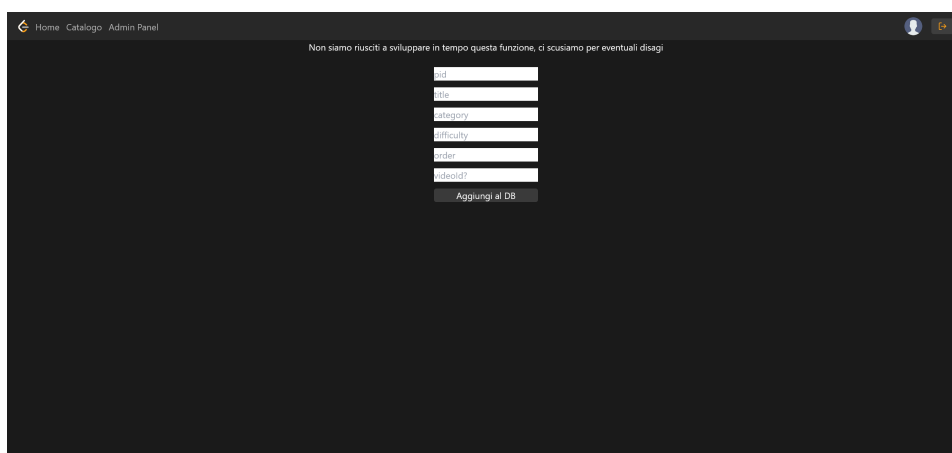
#### 4.4 Recupero Password

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

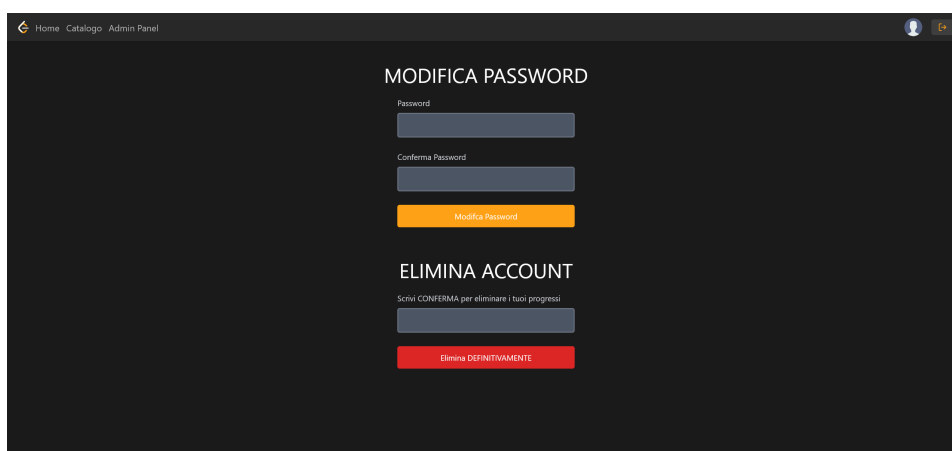
Questo componente (non completamente sviluppato) permette ad un utente amministratore di aggiungere problemi al database, purtroppo per mancanza di tempo non siamo riusciti a sviluppare la funzione in tempo, ci scusiamo per l'inconvenienza.



The screenshot shows the 'Admin Panel' section of the application. At the top, there is a navigation bar with 'Home', 'Catalogo', and 'Admin Panel'. A message at the top states: 'Non siamo riusciti a sviluppare in tempo questa funzione, ci scusiamo per eventuali disagi'. Below this, there is a form with the following fields: 'id', 'title', 'category', 'difficulty', 'order', and 'sold?'. Each field has a corresponding input box. At the bottom of the form is a button labeled 'Aggiungi al DB'.

## 4.6 Pagina Profilo

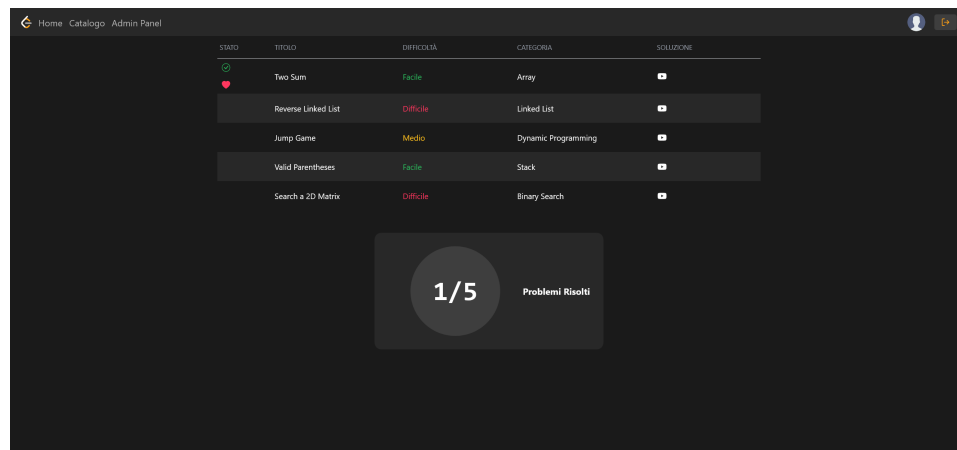
Questa pagina permette ad un utente autenticato di accedere alle funzioni di modifica password e di eliminazione account attraverso appositi form, in caso un utente cerca di accedersi senza essere autenticato verrà riportato alla home.



The screenshot shows the 'Profile Page' of the application. At the top, there is a navigation bar with 'Home', 'Catalogo', and 'Admin Panel'. The page is divided into two main sections. The first section is titled 'MODIFICA PASSWORD' and contains two input fields: 'Password' and 'Conferma Password'. Below these fields is a button labeled 'Modifica Password'. The second section is titled 'ELIMINA ACCOUNT' and contains a single input field. Below this field is a 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. Oltre a questo un qualsiasi utente cliccando sull'icona nella colonna **"Soluzione"** verrà aperta una overlay per visualizzare un video che offre suggerimenti su come risolvere il problema e la soluzione in linguaggio Javascript. 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.



## 4.8 Area di esercitazione

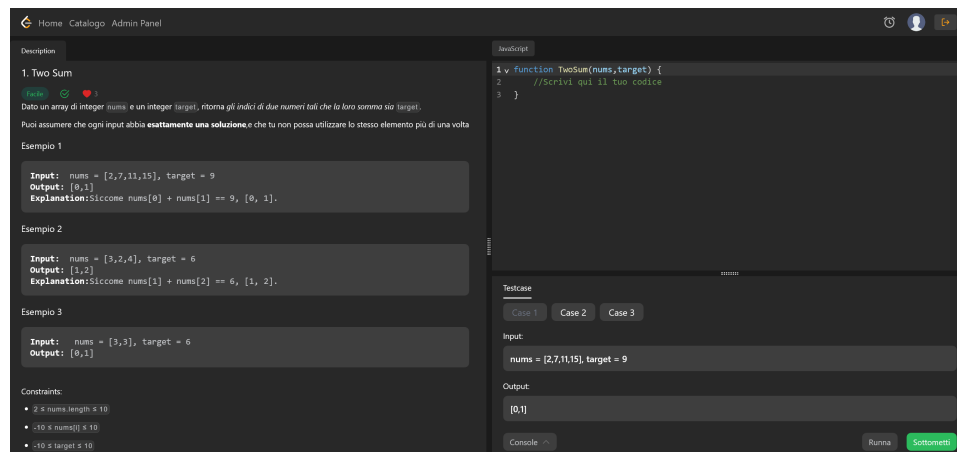
Questa pagina esiste per ogni problema al momento disponibile ed è suddivisa in 3 parti:

- **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 linguaggio con cui risolvere il problema, 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 funziona.

- 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.



## 5 Github Repository & Deployment Information




La repositories del progetto sono disponibili all'interno dell'organizzazione G17-UniTn su GitHub, al seguente link:

[G17-UniTn](#)

L'organizzazione è suddivisa in tre repositories:

- **Deliverables:** contiene tutte le risorse impiegate per realizzare in  $\text{\LaTeX}$  i documenti D1, D2, D3, D4 e D5, inclusi i file `.pdf`, i sorgenti `.tex` e le immagini.
- **Documents:** raccoglie solamente i file PDF dei cinque deliverables, identici a quelli presenti in Deliverables e a quelli consegnati al termine del progetto.
- **CodeBase:** contiene tutto il codice relativo al front-end ed al back-end.

Il gruppo che ha sviluppato il progetto all'interno dell'organizzazione G17-UniTn, sul sito GitHub, è composto dai seguenti membri:

- Raffaele Castagna 
- Zeno Saletti 
- Alberto Rovesti 

Il sito è attualmente attivo ed hostato sulla piattaforma Vercel. Di seguito è riportato il link per visualizzare il sito web:

<https://sleepcode-dev.vercel.app/>

Per poter testare il sito è stato creato un account che gode dei privilegi di amministratore:

Email: `admin@gmail.com`, password: `PasswordAdmin2024!`

Ricordiamo che pultroppo non siamo stati in grado di dare tutte le funzionalità descritte nel D2 all'amministratore, tuttavia il pannello accessibile solo ad utenti amministratore è funzionante.

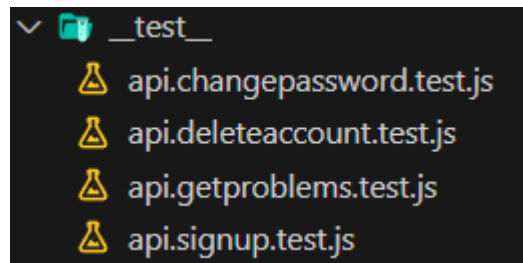
Ricordiamo anche che il processo di invio email per recupero password e la pagina dove immettere la nuova password è gestito da terze parti, e le regole per la password non possono essere utilizzate se si richiede un recupero password.

## 6 Testing

Per eseguire il testing, è stato utilizzato il pacchetto [Jest](#) ed è stato inoltre integrato con Firebase con il pacchetto [Jest-firebase-mock](#). Per simulare le richieste API sono stati utilizzati i pacchetti [node-mocks-http](#).

Sono state definite due cartelle `__mocks__` e `__test__`: la prima è stata creata per utilizzare le funzioni di “mock” di Jest che permettono a quest’ultimo di “imitare” connessioni al database e/o credenziali, molto utile nel testing di API che richiedono funzioni disponibili solo alla [SDK admin](#) di firebase.

Di seguito riportiamo le diverse test suite create



### 6.1 Test API

Di seguito sono mostrati i risultati delle test suite applicate sulle diverse API.

#### 6.1.1 test api/signup

```
PASS __test__/api.signup.test.js
/api/changepassword test
  ✓ Should return 200, Operation Completed (2807 ms)
  ✓ Should return 400, Invalid Password. (16 ms)
  ✓ Should return 400, Ivalid Username (2 ms)
  ✓ Should return 405, HTTP Method not valid,POST Accepted

Test Suites: 1 passed, 1 total
Tests:       4 passed, 4 total
Snapshots:   0 total
Time:        3.769 s, estimated 6 s
Ran all test suites matching /api.signup.test.js/i.
```

### 6.1.2 test api/changepassword

```
PASS __test__/api.changepassword.test.js
/api/changepassword test
  ✓ Should return 200, Operation Completed (50 ms)
  ✓ Should return 405, HTTP method not valid, PATCH accepted. (1 ms)
  ✓ Should return 401, Invalid Password (1 ms)

Test Suites: 1 passed, 1 total
Tests:       3 passed, 3 total
Snapshots:   0 total
Time:        0.969 s, estimated 3 s
Ran all test suites matching /api.changepassword.test.js/i.
```

### 6.1.3 test api/deleteAccount

```
PASS __test__/api.deleteaccount.test.js
/api/changepassword test
  ✓ Should return 200, Operation Completed (2376 ms)
  ✓ Should return 405, HTTP method not valid, DELETE accepted.
  ✓ Should return 500, Firebase cannot use UserID (1 ms)

Test Suites: 1 passed, 1 total
Tests:       3 passed, 3 total
Snapshots:   0 total
Time:        3.506 s, estimated 6 s
Ran all test suites matching /api.deleteaccount.test.js/i.
```

### 6.1.4 test api/getProblems

```
PASS __test__/api.getproblems.test.js
/api/getproblems test
  ✓ Should return list of problems (1329 ms)
  ✓ Should return 405, HTTP method not valid, GET accepted.

Test Suites: 1 passed, 1 total
Tests:       2 passed, 2 total
Snapshots:   0 total
Time:        2.32 s, estimated 4 s
Ran all test suites matching /api.getproblems.test.js/i.
```



## 6.2 Code Coverage

Di seguito riportiamo il code coverage generato da Jest.

```
PASS __test__/api.deleteaccount.test.js (5.401 s)
A worker process has failed to exit gracefully and has been force exited. This is likely caused by this, ensure that .unref() was called on them.
```

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Line #s
All files	85.21	60	60	89.32	
firebase	81.81	50	100	100	
firebase-admin-config.ts	81.81	50	100	100	15
firebase.ts	81.81	50	100	100	24
pages/api	85	66.66	60	85.33	
changepassword.ts	76.19	50	33.33	78.94	70,75,79-80
deleteaccount.ts	93.75	100	66.66	93.33	66
getproblems.ts	93.33	100	100	92.85	57
signup.ts	82.14	50	50	81.48	82-83,86,103-104
utils	92.3	100	100	100	
yupSchemas.js	92.3	100	100	100	

```
Test Suites: 4 passed, 4 total
Tests:       12 passed, 12 total
Snapshots:   0 total
Time:        6.472 s
Ran all test suites.
```