



UNIVERSITÀ
DI TRENTO

Dipartimento di Ingegneria
e Scienza dell'Informazione

Ingegneria del Software

Grab a Book

Gruppo G23

Documento di Sviluppo

Anno Accademico 2023/2024

Indice

Scopo del documento	3
User Flow	4
Implementazione e documentazione delle API	5
Struttura del backend	5
Dipendenze del progetto	6
Modellazione dati nel database	7
Estrazione delle risorse	8
Modello delle risorse	9
<i>Annunci Filtrati</i>	10
<i>Visualizza Annuncio</i>	10
<i>Inserisci Annuncio</i>	10
<i>Elimina Annuncio</i>	10
<i>Registrazione</i>	11
<i>Login</i>	11
<i>Logout</i>	11
<i>GetUser</i>	11
<i>Modifica profilo</i>	12
Sviluppo delle API	13
<i>Annunci filtrati</i>	13
<i>Visualizza annuncio</i>	15
<i>Inserisci annuncio</i>	15
<i>Elimina annuncio</i>	15
<i>Registrazione</i>	16
<i>Login</i>	16
<i>Logout</i>	18
<i>GetUser</i>	18
<i>Modifica profilo</i>	19
Documentazione delle API	20
Implementazione del frontend	23
GitHub repository e deployment	28
Testing delle API	29

Scopo del documento

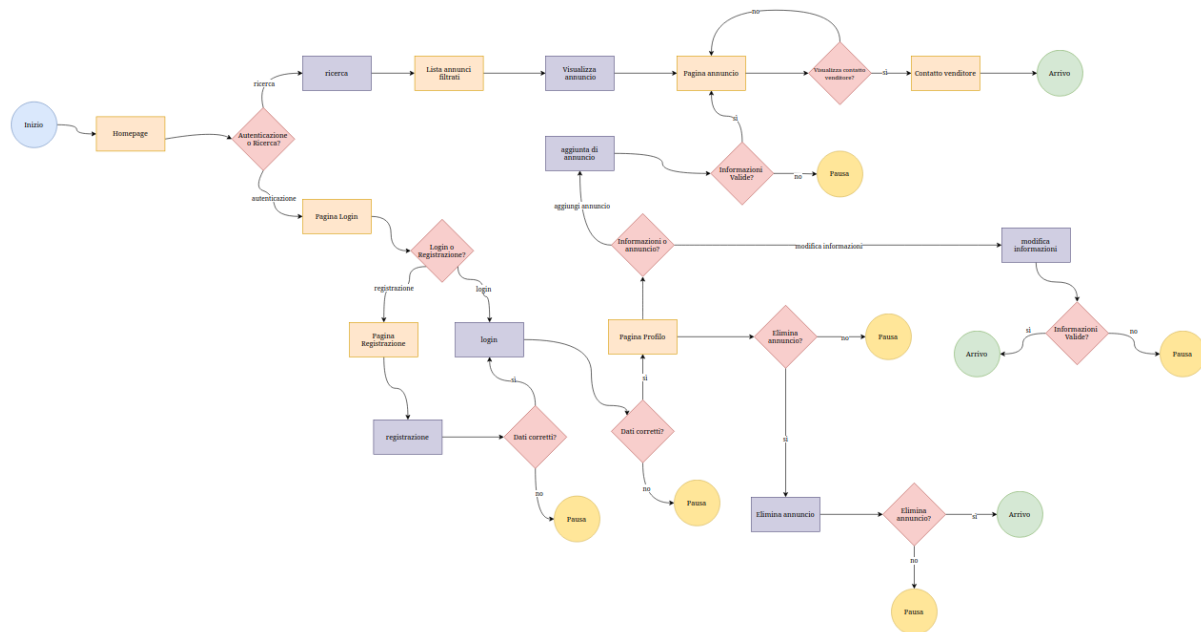
Il presente documento riporta le informazioni necessarie allo sviluppo di una parte dell'applicazione *Grab a Book*. In particolare presenta tutti gli strumenti e i servizi necessari per realizzare una pagina di login o registrazione, la ricerca degli annunci, l'aggiunta di un nuovo annuncio e la sua eliminazione, la possibilità di modificare le informazioni dell'utente quali *username*, *password* e *email*.

User Flow

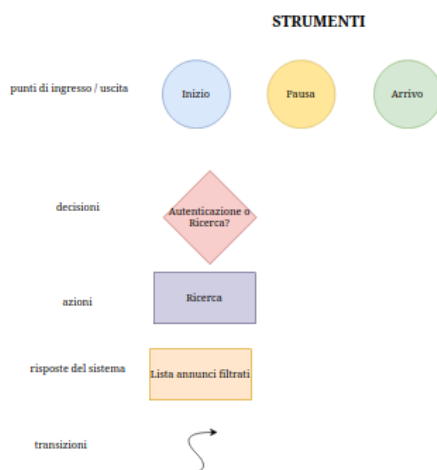
In questa sezione del documento si riporta l' "User Flow" dell'utente.

Un utente può visualizzare la Homepage ed effettuare ricerche indipendentemente se sia o meno registrato all'interno del sistema.

L'accesso alla pagina profilo avviene solo dopo registrazione o accesso con le proprie credenziali, una volta effettuato l'accesso verrà visualizzata la pagina profilo in cui sarà possibile modificare le proprie informazioni quali username, email e password o creare un nuovo annuncio. Inoltre dalla pagina del profilo sarà visibile l'elenco degli annunci creati, con la possibilità di eliminarli.



L'immagine è consultabile anche a questo link: [D4-Flow.drawio.png](#)



Implementazione e documentazione delle API

In questa sezione viene descritta l'implementazione delle API e del backend come descritto nei precedenti Deliverables

Struttura del backend

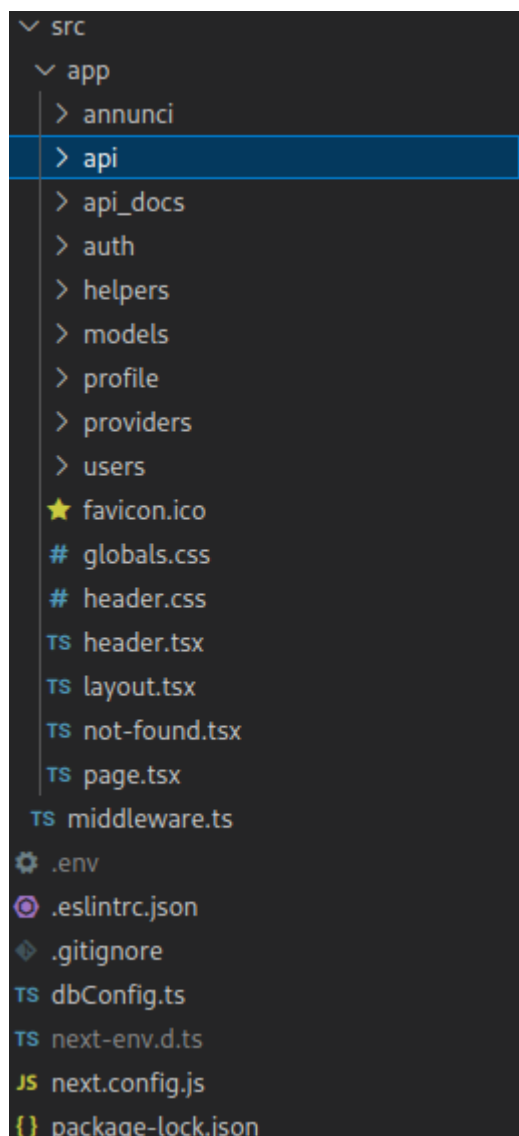
La struttura del backend è riportata nella figura sottostante. All'interno della cartella *api/* sono presenti tutte le API divise per categoria e accessibili tramite il percorso della cartella stesso.

Abbiamo utilizzato un *middleware* per gestire l'accesso alle pagine.

Nella cartella *models* sono presenti i modelli dei dati nel database.

Inoltre le varie pagine del frontend sono divise nelle loro cartelle in modo da essere accessibili tramite il così definito percorso.

Nel file *.env* sono presenti l'*url* per l'accesso al database di mongodb e il *secret* necessario per la creazione e decrittazione dei token tramite *jwt*.



Dipendenze del progetto

Segue ora l'elenco delle dipendenze Node utilizzate nel nostro progetto con una breve descrizione del loro scopo:

- **antd**: per il design dei componenti grafici
- **axios**: per la gestione delle richieste http alle API
- **bcryptjs**: per la gestione del token di autenticazione
- **jsonwebtoken**: per la gestione dei token di autenticazione
- **dotenv**: per la lettura dei secret da file .env
- **mongoose**: per il collegamento con il database mongodb
- **next**: il framework su cui si basa la nostra web-app
- **swagger-ui-react**: per la visualizzazione della documentazione delle API direttamente da web-app (/api_docs)
- **react**: il framework su cui si basa next
- **reactjs-popup**: per la creazione di popup
- **react-tooltip**: per la creazione di tooltip
- **jest**: per il testing

Modellazione dati nel database

Per la gestione dei dati abbiamo utilizzato MongoDB, un database non relazionale. In particolare abbiamo rappresentato le seguenti strutture dati:

grab-a-book

LOGICAL DATA SIZE: 1.52KB STORAGE SIZE: 72KB INDEX SIZE: 72KB TOTAL COLLECTIONS: 2

CREATE COLLECTION

Collection Name	Documents	Logical Data Size	Avg Document Size	Storage Size	Indexes	Index Size	Avg Index Size
announces	5	1010B	202B	36KB	1	36KB	36KB
users	3	544B	182B	36KB	1	36KB	36KB

`announces` contiene le informazioni relative agli annunci pubblicati dagli utenti mentre `users` contiene le informazioni degli utenti.

Ognuna di esse è a sua volta composta in questo modo:

Un esempio di `announces`:

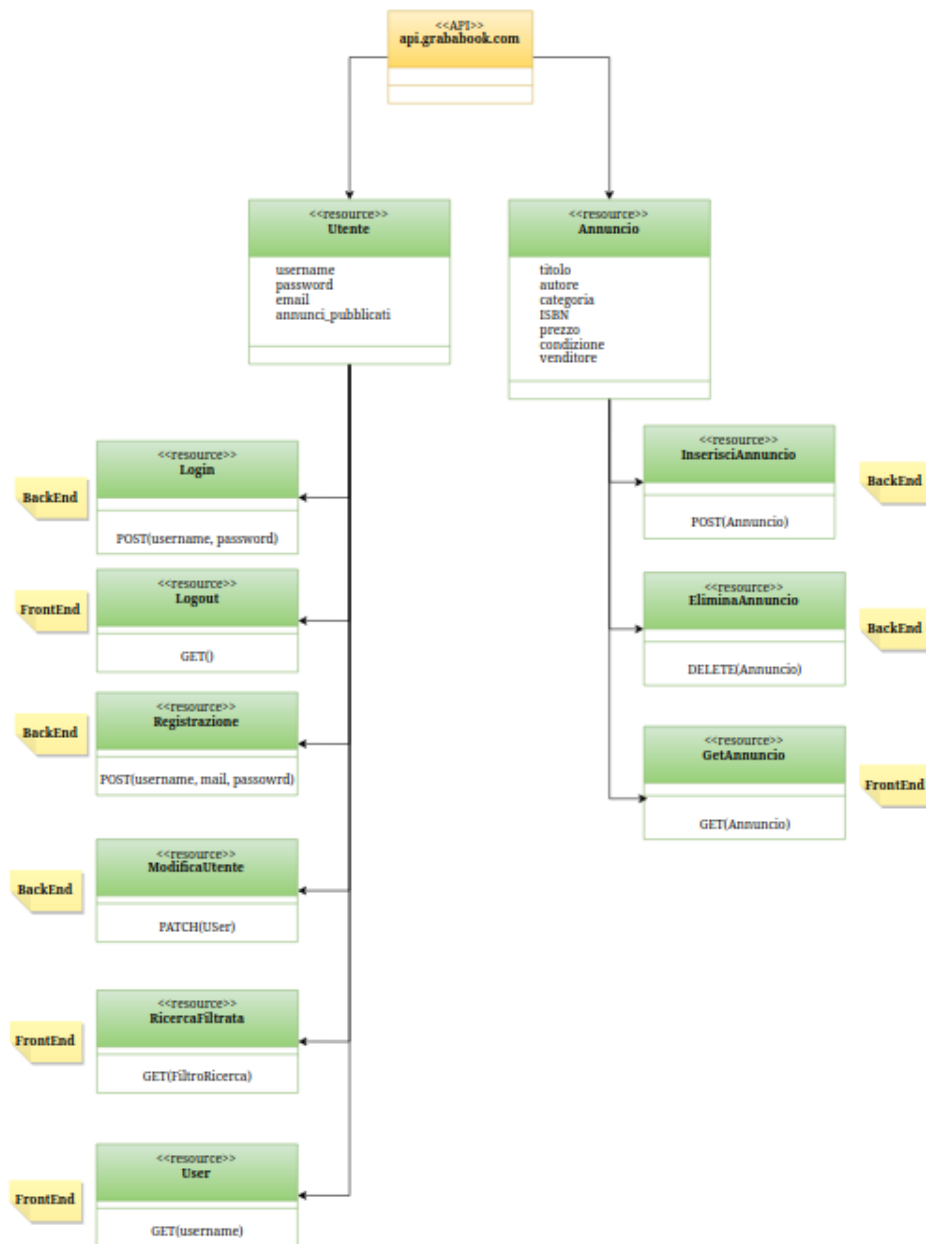
```
_id: ObjectId('6585bcc10e672d2016e18d54')
title: "myBook"
author: "autore"
category: "university"
ISBN: "123"
price: 50
condition: "new"
seller: "idUtente"
createdAt: 2023-12-22T16:43:45.493+00:00
updatedAt: 2023-12-22T16:43:45.493+00:00
__v: 0
```

Un esempio di `users`:

```
_id: ObjectId('65843a80fe683ad816348bca')
username: "a"
email: "a@gmail.com"
password: "$2a$10$tW6BYc0AZr0iE97F.qfkZeWq3HTVhhtq.lDtwfRDx5Kag.tGNT80S"
createdAt: 2023-12-21T13:15:44.225+00:00
updatedAt: 2023-12-21T13:15:44.225+00:00
__v: 0
```

Estrazione delle risorse

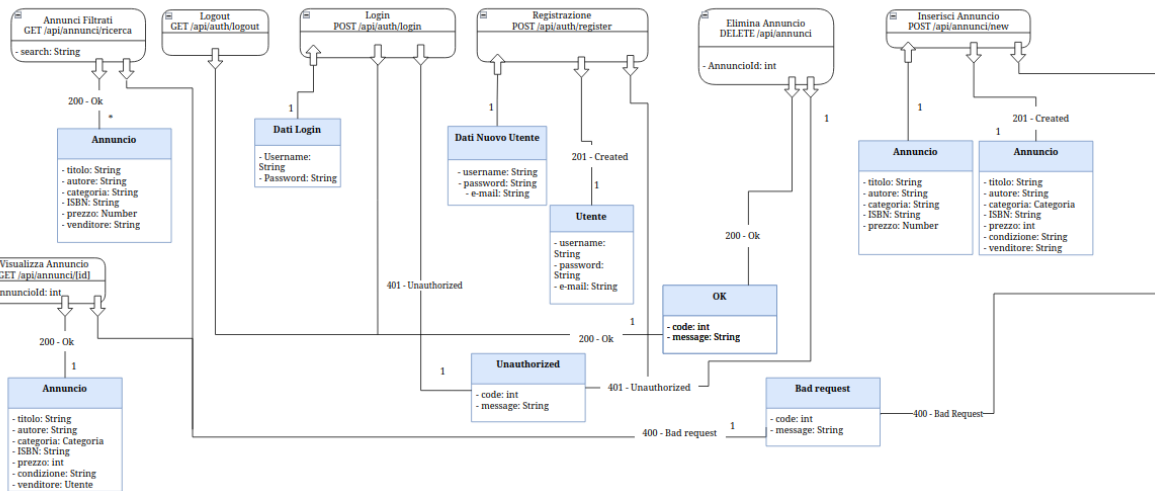
In questo paragrafo viene allegato il diagramma di estrazione delle risorse.



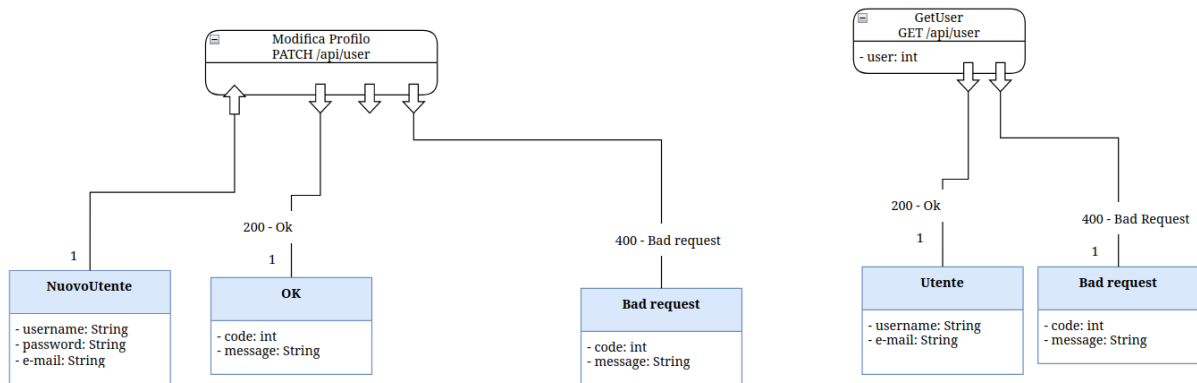
L'immagine è consultabile anche a questo link: [D4-risorse.png](#)

Modello delle risorse

In questo paragrafo viene allegato il modello delle risorse e una descrizione delle API con i relativi URI, metodi HTTP e possibili risposte che possono essere restituite.



L'immagine è consultabile anche a questo link: [D4-Modello delle risorse.png](#)



L'immagine è consultabile anche a questo link: [D4-Modello delle risorse.png](#)

Annunci Filtrati

Questa API, raggiungibile all'indirizzo `/api/annuncio/ricerca` permette di ottenere tutti gli annunci che corrispondono ad una determinata ricerca. In particolare è necessario fare una richiesta GET all'API passando come parametro un *FiltroRicerca*, se il filtro è ben formattato dall'API si otterrà il codice *200* e il messaggio *Ricerca effettuata con successo*, restituendo l'elenco di gli annunci che corrispondono al filtro.

Se invece il *FiltroRicerca* non è corretto si otterrà il codice di errore *400* e il messaggio di errore *Bad request*.

Visualizza Annuncio

Questa API, raggiungibile all'indirizzo `/api/annuncio/(id)` permette di ottenere tutti i dettagli di un annuncio tramite il suo *id*.

In particolare è necessario fare una richiesta GET passando come parametro l'*id* e se si ottiene il codice *200* la richiesta è andata a buon fine, si otterrà l'annuncio con tutti i suoi dettagli qui elencati:

- titolo
- categoria
- ISBN
- prezzo
- condizione
- venditore

In caso di richiesta malformata si riceve un codice di errore *400*.

Inserisci Annuncio

Questa API, raggiungibile all'indirizzo `/api/annuncio/new` permette di inserire un nuovo annuncio.

In particolare è necessario fare una richiesta POST passando nel body un nuovo annuncio con tutte le informazioni necessarie.

In caso di successo verrà restituito il messaggio *Annuncio creato con successo* e il codice *201* insieme all'annuncio appena creato.

In caso l'annuncio passato nel body sia malformato verrà ritornato il codice *400* e il messaggio *Bad Request*.

Elimina Annuncio

Questa API, raggiungibile all'indirizzo `/api/annuncio/(id)` permette di eliminare un annuncio tramite il suo *id*.

In particolare è necessario fare una richiesta DELETE passando come parametro l'*id* e se si ottiene il codice *200* e il messaggio *Annuncio eliminato con successo*, la richiesta è andata a buon fine e l'annuncio è eliminato.

Se non si è il proprietario dell'annuncio non è possibile eliminarlo e quindi utilizzare questa API senza autenticazione restituirà il codice *401* e il messaggio *Unauthorized: Non puoi eliminare questo annuncio*.

Registrazione

Questa API, raggiungibile all'indirizzo `/api/auth/register` permette di registrarsi alla piattaforma.

In particolare è necessario fare una richiesta POST passando nel body i dati del nuovo utente con tutte le informazioni necessarie.

In caso di successo verrà restituito il messaggio *Utente creato con successo* e il codice *201* insieme all'oggetto Utente appena creato.

In caso le informazioni dell'utente siano già presenti (vengono controllate solo username e email) verrà ritornato il codice *401* e il messaggio *Unauthorized* impedendo a più utenti di registrarsi con la stessa email o username.

Login

Questa API, raggiungibile all'indirizzo `/api/auth/login` permette di accedere alla piattaforma.

In particolare è necessario fare una richiesta POST passando nel body l'username e la password.

In caso di successo verrà restituito il messaggio *Login effettuato con successo* e il codice *200*.

In caso il nome utente inserito non corrispondesse a nessun utente nel database oppure la password collegata all'utente inserito non corrispondesse, verrà ritornato il codice *401* e il messaggio *Unauthorized*.

Logout

Questa API, raggiungibile all'indirizzo `/api/auth/logout` permette di disconnettersi dalla piattaforma.

In particolare tramite una richiesta GET l'utente verrà disconnesso (il token memorizzato nel cookie del browser verrà eliminato) e l'API ritornerà il codice *200* e il messaggio *Logout effettuato con successo*.

Nel caso in cui non sia presente il token al momento del logout, l'API ritornerà comunque il codice *200* con il messaggio *Nessun profilo collegato*.

GetUser

Questa API, raggiungibile all'indirizzo `/api/users/user` permette di ottenere tutti i dettagli di un utente tramite il suo *id*.

In particolare è necessario fare una richiesta GET passando come query parameter l'*id* e se si ottiene il codice *200* e il messaggio *Ok*, la richiesta è andata a buon fine e si ottiene l'utente con tutti i suoi dettagli qui elencati:

- username
- email

Per ottenere i dettagli di se stessi è possibile non passare nessun parametro e se si è autenticati l'API utilizzerà il token di autenticazione per ottenere l'id utente e restituire le informazioni richieste.

In caso di richiesta mal formattata si otterrà il codice di errore *400* e il messaggio *Bad Request*.

Modifica profilo

Questa API, raggiungibile all'indirizzo `/api/user/user` permette di modificare le informazioni dell'utente, ovvero password, username e email.

In particolare è necessario fare una richiesta PATCH passando nel body le seguenti informazioni:

- username
- email

La password si può omettere, oppure se presente verrà aggiornata (criptandola)

In caso di successo verrà restituito il messaggio *Account aggiornato correttamente* e il codice *200* insieme.

In caso le informazioni passate nel body siano malformate o corrispondano a quelle di un altro utente, verrà ritornato il codice *400* e il messaggio *Bad Request*.

Sviluppo delle API

Abbiamo sviluppato le 9 API rappresentate dal diagramma e dalle descrizioni precedenti. Segue ora una breve descrizione e gli screenshot del codice sviluppato per ogni API.

Annunci filtrati

Esegue la ricerca degli annunci, è possibile solo un numero per ricercare quel numero negli ISBN, solo una stringa per cercare la stringa nel titolo oppure usare dei filtri su autore, titolo, ISBN, categoria.

Per farlo basta cercare la parola chiave seguita dal carattere ":" e dalla stringa che si vuole cercare tra virgolette (es: autore:"Matteo Bordone").

Con la parola chiave *last* si ottengono gli ultimi 3 annunci pubblicati; con la parola chiave *user* si ottengono gli annunci pubblicati dall'utente loggato.

```
export async function GET(request: NextRequest) {
  try {
    let AnnuncelList = [];
    const searchUrl = request.nextUrl.searchParams.get("search");
    let search:string;
    if (searchUrl == null) {
      search = "";
    } else {
      search = searchUrl;
    }
    const isNum = /^d+$/i.test(search);
    if (isNum) {
      AnnuncelList = await Annunce.find({ ISBN: search });
    } else {
      if (search.includes(':')) {
        const splitted = search.split(':');
        console.log(splitted);
        const typeOfSearch = splitted[0].split(':')[0];
        const searchTerm = splitted[1];
        if (typeOfSearch.toLowerCase() === "titolo") {
          AnnuncelList = await Annunce.find({ title: { $regex: new RegExp(`.*${searchTerm}.*`, "i") } });
          if (splitted.length > 3) {
            AnnuncelList = multiOptionFilter(splitted, AnnuncelList);
          }
        } else if (typeOfSearch.toLowerCase() === "autore") {
          AnnuncelList = await Annunce.find({ author: { $regex: new RegExp(`.*${searchTerm}.*`, "i") } });
          if (splitted.length > 3) {
            AnnuncelList = multiOptionFilter(splitted, AnnuncelList);
          }
        } else if (typeOfSearch.toLowerCase() === "categoria") {
          AnnuncelList = await Annunce.find({ category: { $regex: new RegExp(`.*${searchTerm}.*`, "i") } });
        }
      }
    }
  }
}
```

```

        if (splitted.length > 3) {
            AnnuncelList = multiOptionFilter(splitted, AnnuncelList);
        }
    } else if (typeOfSearch.toLowerCase() === "isbn") {
        AnnuncelList = await Annunce.find({ ISBN: searchTerm });
        if (splitted.length > 3) {
            AnnuncelList = multiOptionFilter(splitted, AnnuncelList);
        }
    }
} else {
    if (search === "last") { // this is a search for the last 3 annunces used only on the home page
        console.log(AnnuncelList = await Annunce.find());
        AnnuncelList = await Annunce.find().sort({ _id: -1 }).limit(3);
        return NextResponse.json({
            message: "Search completed successfully",
            data: AnnuncelList,
            status: 200
        });
    } else if (search === "user") { // this is a search by userId used only for the profile page so doesn't need to be f
        const userId = await validateJWT(request);
        console.log(userId);
        AnnuncelList = await Annunce.find({ seller: userId });
        console.log(AnnuncelList);
        return NextResponse.json({
            message: "Search completed successfully",
            data: AnnuncelList,
            status: 200
        });
    }
    // console.log("searching for: " + search);
    AnnuncelList = await Annunce.find({ title: { $regex: new RegExp(`.*${search}.*`, "i") } });
}

return NextResponse.json({
    message: "Search completed successfully",
    data: AnnuncelList,
    status: 200
});
} catch (error: any) {
    console.log(error);
    return NextResponse.json({
        message: "Bad request",
        status: 400, {status: 400});
    });
}

function multiOptionFilter(splitted: string[], AnnuncelList: any[]): any[] {
    for (let i = 2; i < splitted.length - 1; i += 2) {
        const searchType = splitted[i].split(':')[0].toLowerCase().trim();
        const search = splitted[i + 1];
        if (searchType === "autore") {
            AnnuncelList = AnnuncelList.filter((annunce) => {
                return annunce.author.toLowerCase().includes(search.toLowerCase());
            });
        } else if (searchType === "categoria") {
            AnnuncelList = AnnuncelList.filter((annunce) => {
                return annunce.category.toLowerCase().includes(search.toLowerCase());
            });
        } else if (searchType === "isbn") {
            console.log("isbn");
            AnnuncelList = AnnuncelList.filter((annunce) => {
                return annunce.ISBN === search;
            });
        }
        // console.log(AnnuncelList);
    } else if (searchType === "titolo") {
        AnnuncelList = AnnuncelList.filter((annunce) => {
            return annunce.title.toLowerCase().includes(search.toLowerCase());
        });
    }
}
return AnnuncelList;
}

```

Visualizza annuncio

Permette di ottenere un annuncio dato il suo id

```
export async function GET(request: NextRequest, {params}: {params:{id:string}}){
  try{
    const ad = await Ad.findOne({_id: params.id});
    if(!ad) {
      throw new Error("Annuncio non trovato")
    }
    return NextResponse.json({ad, status: 200});
  }catch (error: any) {
    return NextResponse.json({ message: "Bad request: "+error.message, status: 400 }, {status: 400});
  }
}
```

Inserisci annuncio

Permette di inserire un nuovo annuncio nel sistema

```
export async function POST(request: NextRequest) {
  try {
    const userId = await validateJWT(request);
    const reqBody = await request.json();
    const newAd = new Ad(reqBody);
    newAd.seller = userId;
    await newAd.save();

    return NextResponse.json({
      message: "Annuncio creato con successo",
      data: newAd,
      status: 201, {status: 201}
    })
  } catch (error: any) {
    return NextResponse.json({
      message: "Bad request: " + error.message,
      status: 400
    }, {status: 400});
  }
}
```

Elimina annuncio

Elimina un annuncio, controlla se si è il proprietari di quell'annuncio altrimenti ritorna un errore

```
export async function DELETE(request: NextRequest, {params}: {params:{id:string}}){
  try{
    const userId = await validateJWT(request);
    const ad = await Ad.findOne({_id: params.id});
    if (ad.seller !== userId) {
      // console.log("ad.seller: " + ad.seller);
      return NextResponse.json({message:'Unauthorized: Non puoi eliminare questo annuncio!', status: 401})
    } else {
      await Ad.findOneAndDelete({_id: params.id});
      return NextResponse.json({message:'Annuncio eliminato con successo!', status: 200});
    }
  }catch (error: any) {
    return NextResponse.json({ message: "Bad request: "+error.message, status: 400 }, {status: 400});
  }
}
```

Registrazione

Permette la registrazione, verificando che *username* e *email* non siano già nel database.

```
export async function POST(request: NextRequest) {
  try {
    const reqBody = await request.json();
    //check if the user already exists
    const emailExists = await User.findOne({ email: reqBody.email });
    const userExists = await User.findOne({ username: reqBody.username });

    if (emailExists) {
      throw new Error("Email già esistente");
    }
    if (userExists) {
      throw new Error("Nome utente già esistente");
    }

    // create new user
    // random string
    const salt = await bcrypt.genSalt(10);
    // hashing the pwd
    const hashedPassword = await bcrypt.hash(reqBody.password, salt);
    reqBody.password = hashedPassword;
    const newUser = new User(reqBody);
    await newUser.save();
    return NextResponse.json({
      message: "Utente creato con successo",
      data: newUser,
      status: 201, {status: 201}});
  } catch (error: any) {
    return NextResponse.json({message: "Unauthorized: " + error.message, status: 401}, {status: 401});
  }
}
```

Login

Permette di effettuare il login e aggiunge il token di autenticazione nei cookie.

```
export async function POST(request: NextRequest) {
  try {
    const reqBody = await request.json();
    // check if user exists in the DB or not
    const user = await User.findOne({ username: reqBody.username });
    if (!user) {
      throw new Error("User does not exist");
    }
    // password match
    const passwordMatch = await bcrypt.compare(reqBody.password, user.password);

    if (!passwordMatch) {
      throw new Error("Invalid credentials");
    }
    // create token
    const token = jwt.sign({ id: user._id }, process.env.jwt_secret!, { expiresIn: "7d" });

    const response = NextResponse.json({
      message: "Login successfull",
      status: 200})
    response.cookies.set("token", token, {
      httpOnly: true,
      path: "/",
    });
    return response;
  } catch (error: any) {
    return NextResponse.json({
      message: "Unauthorized: " + error.message,
      status: 401, {status: 401}});
  }
}
```


Logout

Permette di effettuare il logout, eliminando il token di autenticazione dai cookie.

```
export async function GET(request: NextRequest){
  if(request.cookies.get("token") != null){
    const response = NextResponse.json({
      message: "Logout effettuato con successo!",
      status:200});
    // Remove the cookie
    response.cookies.delete("token");

    return response;
  }
  else {
    const response = NextResponse.json({
      message: "Nessun profilo collegato",
      status:200});
    return response;
  }
}
```

GetUser

Ottiene le informazioni di un utente, se nessun utente viene passato come parametro cerca di ottenere le informazioni dell'utente loggato.

```
export async function GET(request: NextRequest) {
  try {
    // const userId = await validateJWT(request);
    var userId = request.nextUrl.searchParams.get("user");
    // console.log(userId);
    if (!userId) {
      userId = await validateJWT(request);
    }
    // retrieve the user without the password
    const user = await User.findById(userId).select("-password");
    return NextResponse.json({
      data: user,
      message: "Ok",
      status:200});
  } catch (error: any) {
    return NextResponse.json({
      message: "Bad request",
      status: 400,, {status: 400});
  }
}
```

Modifica profilo

Permette di modificare il profilo, controlla che username e password non siano già presenti nel database, il controllo viene effettuato solo se l'utente ha effettivamente cambiato email o username per evitare falsi positivi dati dal fatto che quei dati già presenti nel database dato che sono i dati dell'utente stesso.

Se passata come dato viene aggiornata anche la password altrimenti essa rimane invariata.

```
export async function PATCH(request: NextRequest) {
  try {
    const reqBody = await request.json();

    const userId = await validateJWT(request);
    // retrieve the user without the password
    const user = await User.findById(userId).select("-password");

    if (user.email !== reqBody.email) {
      const userExists = await User.findOne({ email: reqBody.email });

      if (userExists) {
        throw new Error("La nuova email è già collegata ad un altro account");
      } else {
        user.email = reqBody.email;
      }
    }

    if (user.username !== reqBody.username) {
      const userExists = await User.findOne({ username: reqBody.username });

      if (userExists) {
        throw new Error("Il nuovo username è già collegato ad un altro account");
      } else {
        user.username = reqBody.username;
      }
    }

    if (reqBody.password !== '') {
      const salt = await bcrypt.genSalt(10);
      // hashing the pwd
      const hashedPassword = await bcrypt.hash(reqBody.password, salt);
      user.password = hashedPassword;
    }

    await user.save();
    return NextResponse.json({
      message: "Account aggiornato correttamente!",
      status: 200
    });
  } catch (error: any) {
    return NextResponse.json({
      message: error.message,
      status: 400
    }, { status: 400 });
  }
}
```

Documentazione delle API

Le API presentate nei paragrafi precedenti sono state documentate utilizzando l'editor online di *swagger* e seguendo lo standard OpenAPI.

Per poter avere una facile e comoda visualizzazione è possibile accedere alla pagina dove è presente l'interfaccia **Swagger UI**, generata grazie al modulo *swagger-ui-react*, che permette la visualizzazione di tutte le principali e più importanti informazioni relative alle API oltre a consentire il test al volo.

La documentazione è raggiungibile tramite l'endpoint `/api_docs`.

Per alcune API come descritto nella documentazione di *swagger* è necessario essere autenticati.

Se si utilizza l'endpoint tramite la web-app, sarà sufficiente effettuare il login o nella web-app o tramite l'api, per settare il token necessario.

Se per qualche motivo non fosse possibile, è presente l'opzione per attivare l'autenticazione inserendo manualmente il token.

Di seguito riportiamo lo screenshot della pagina principale e l'esempio di quattro API che usano metodi tutti diversi: GET, POST, DELETE, PATCH. Per dare un'idea generale della documentazione.

Ogni API ha un esempio per far comprendere facilmente come funziona.

Interfaccia principale

Grab a Book 1.0.0 OAS 3.0

API della web-app Grab a Book realizzate per il progetto di ingegneria del software dell'anno 2023/24 dal gruppo G23. Alcune API funzionano solo se si è loggati:

- Modifica un utente (PATCH `/api/user`)
- Crea un nuovo annuncio (POST `/api/annunci/new`)
- Elimina un annuncio (DELETE `/api/annunci/{id}`)
- Ottenere le informazioni su se stessi (GET `/api/user`)

Normalmente se si accede a questa pagina dalla web-app, dato che le richieste sono fatte dallo stesso dominio, una volta eseguito il login l'autenticazione è automatica.

Nel caso in cui questo non avvenga è possibile inserire il valore del token presente nei cookie quando si effettua il login per utilizzare queste API

Authorize 

Autenticazione API per la gestione della registrazione e dell'accesso

Utenti API relative alla gestione degli utenti

Annunci API relative alla gestione degli annunci

POST `/api/annunci/new` Crea un nuovo annuncio

GET `/api/annunci/ricerca` Cerca tra gli annunci

Esempio richiesta GET

GET /api/annunci/ricerca Cerca tra gli annunci

Parameters [Try it out](#)

Name	Description
search string (query)	usa il dato filtro di ricerca, se si passa 'last' si ottengono gli ultimi 3 annunci pubblicati, se si passa user si ottengono gli annunci pubblicati da quell'utente <input type="text" value="Le cronache di Narnia"/>

Responses

Code	Description	Links
200	Lista annunci Media type <input type="text" value="application/json"/> Controls Accept header Example Value Schema <pre>[{ "titolo": "Le cronache di Narnia", "autore": "C. S. Lewis", "categoria": "Fantasy", "ISBN": "8022264753845", "prezzo": 15.6, "condizione": "buono", "venditore": "658d9e3cf36dc1419f4a5b5d" }]</pre>	No links
400	La richiesta non è valida	No links

Esempio richiesta POST

POST /api/auth/login Permette ad un'utente di accedere

Parameters [Try it out](#)

No parameters

Request body **required**

Dati utente
Example Value | Schema

```
{
  "username": "username",
  "password": "NuovaPassword"
}
```

Responses

Code	Description	Links
200	Accesso eseguito con successo	No links
401	I dati inseriti non corrispondono a nessun utente nel database e quindi non è possibile accedere	No links

Esempio richiesta DELETE

DELETE `/api/annuncio/{id}` Elimina un annuncio

Parameters Try it out

Name	Description
id <small>★ required</small>	è l'id di un annuncio
string (path)	<input type="text" value="658d9f02f36dc1419f4a5b68"/>

Responses

Code	Description	Links
200	Annuncio eliminato con successo	No links
401	L'annuncio che si tenta di eliminare non è stato pubblicato dall'utente attualmente collegato	No links

Esempio richiesta PATCH

PATCH `/api/user` Permette di modificare il profilo di un'utente in uno o tutti i suoi campi

Parameters Try it out

No parameters

Request body required application/json

Dati utente

Example Value Schema

```
{
  "email": "nuova_email@email.com",
  "username": "username",
  "password": "NuovaPassword"
}
```

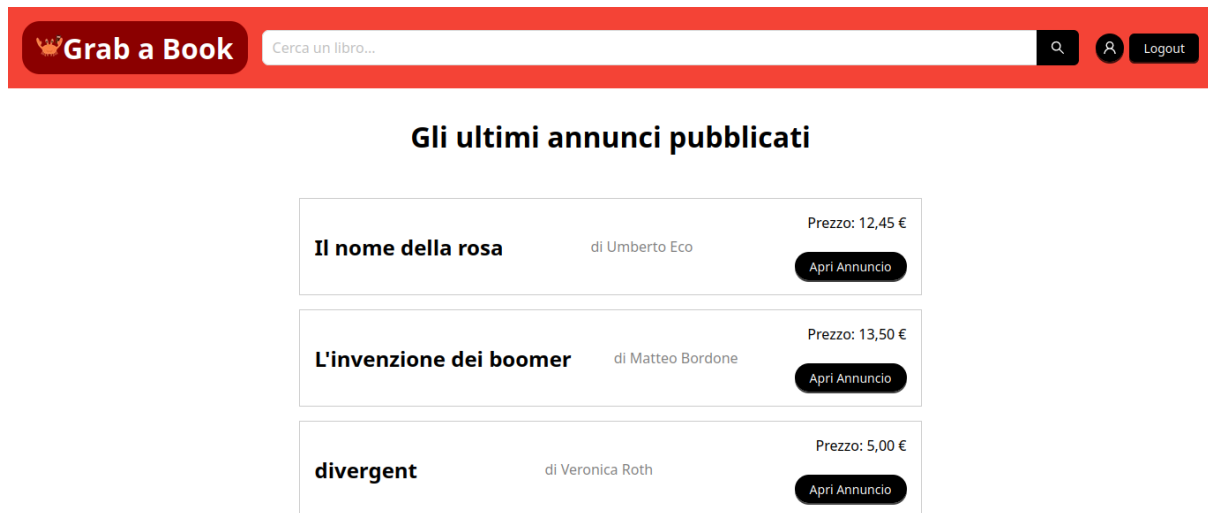
Responses

Code	Description	Links
200	Account aggiornato correttamente	No links
400	I dati inseriti corrispondono a un utente nel database e quindi non è possibile modificare il profilo con quei dati	No links

Implementazione del frontend

L'applicazione ha due diversi tipi di pagine: pubbliche e private.
L'accesso a quelle private è possibile solo tramite autenticazione.

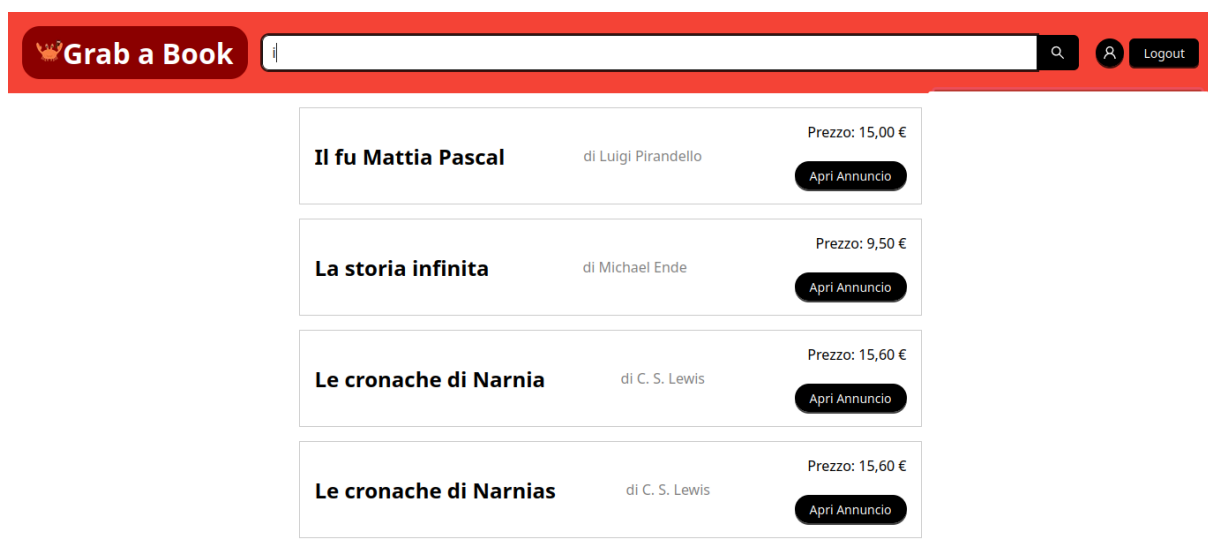
Segue ora la descrizione del frontend con la spiegazione dei vari componenti creati.



Appena entrati nel sito l'homepage presenta l'header che permette di effettuare una ricerca o di accedere al proprio profilo.

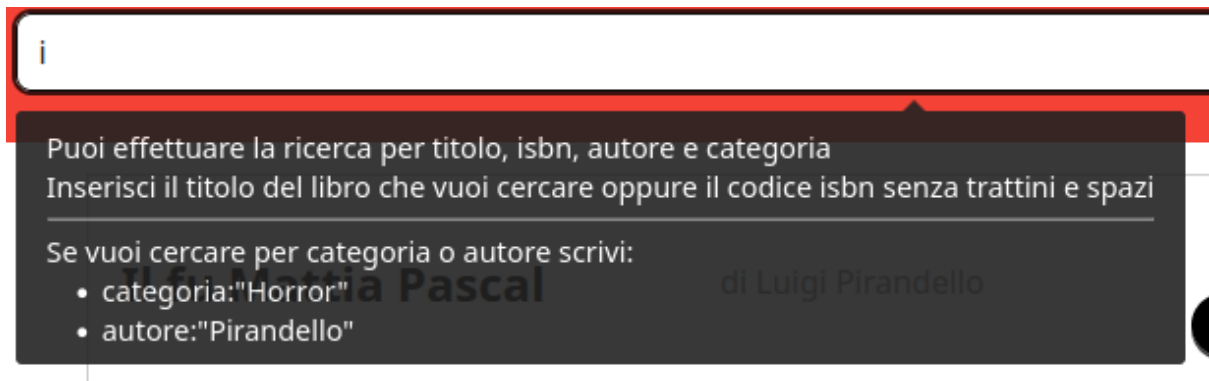
Sono poi mostrati gli ultimi **tre** annunci pubblicati nel sistema.

Effettuando una ricerca si ottiene questa schermata:

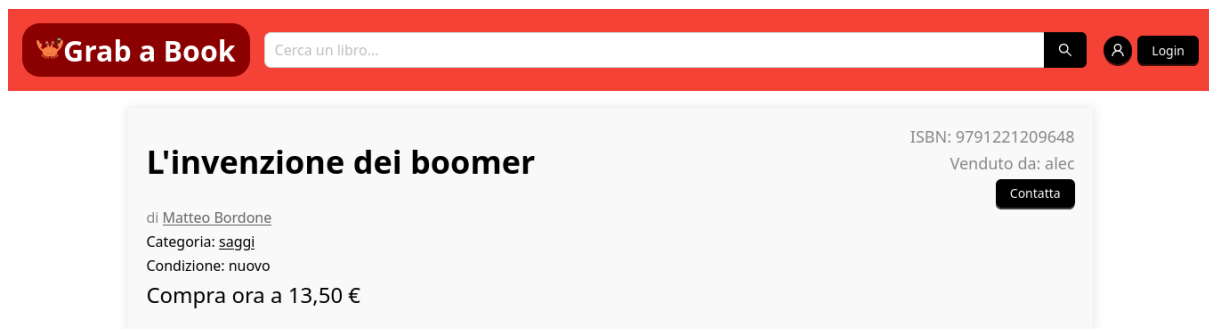


In questo caso non si è applicato nessun filtro ma si è cercata la stringa "i" che ha prodotto i risultati mostrati.

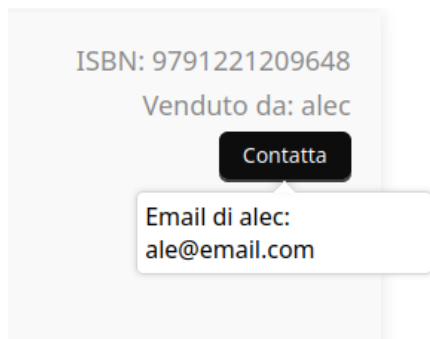
È possibile cliccare sul pulsante *Apri Annuncio* per visualizzare i dettagli dell'annuncio. Inoltre se si seleziona la ricerca compare un tooltip che spiega come funziona la ricerca:



Se preme il pulsante *Apri Annuncio* si accede alla pagina dei dettagli dell'annuncio:



In questa schermata vengono visualizzati i dettagli dell'annuncio. Inoltre troviamo il pulsante *Contatta* che apre un pop-up con all'interno la mail dell'utente che ha postato l'annuncio in modo che possa avvenire la contrattazione per l'acquisto del libro.



È possibile cliccare l'autore e la categoria per far partire una ricerca con filtro l'autore o la categoria in modo da poter esplorare annunci con libri dello stesso genere o scritti dallo stesso autore.

L'invenzione dei boomer

di [Matteo Bordone](#)

Link cliccabili

Categoria: [saggi](#)

Condizione: nuovo

Compra ora a 13,50 €

Tramite l'header è possibile passare alla pagina profilo cliccando il pulsante con l'icona di una persona:



Una volta cliccato il pulsante se non è ancora effettuato l'accesso verrà richiesto di accedere, cosa che si può fare anche premendo il pulsante *Login*:

A screenshot of a login form. At the top is a red header bar with the 'Grab a Book' logo, a search bar, and navigation icons. Below the header is a section titled 'Accedi'. It contains two input fields: one for 'Username' and one for 'password'. Below these fields is a black button with the text 'Login'. At the bottom of the form is a link that says 'Non hai ancora un account? Registrati.'

Se non si possedesse ancora un account è possibile passare al form di registrazione cliccando il link in basso:

The screenshot shows the 'Grab a Book' website header with a search bar and a 'Login' button. Below the header is a 'Registrati' (Register) form. The form has three input fields: 'Username', 'Email', and 'Password', each preceded by an asterisk. Below these fields is a 'Register' button. At the bottom of the form, there is a link: 'Hai già un account? Accedi.'

Una volta compilato il form di registrazione verrà chiesto di effettuare il login attraverso la schermata vista precedentemente.

Una volta terminata questa operazione si avrà accesso alla pagina del profilo:




The screenshot shows the user profile page. The header is the same as the previous page. Below the header, the text 'Ciao alec,' is displayed. The page is divided into two main sections: 'Profilo' (Profile) on the left and 'I tuoi annunci' (Your announcements) on the right. The 'Profilo' section contains input fields for 'Username' (with the value 'alec'), 'Email' (with the value 'ale@email.com'), and 'Password'. Below these fields are two buttons: 'Modifica profilo' (Edit profile) and 'Conferma modifica' (Confirm modification). The 'I tuoi annunci' section displays a list of four announcements. Each announcement has a title, the author's name, and two buttons: 'Apri Annuncio' (Open announcement) and 'Elimina' (Delete). The announcements are: 'Le cronache di Narnia' by C. S. Lewis, 'Le cronache di Narnias' by C. S. Lewis, 'L'invenzione dei boomer' by Matteo Bordone, and 'Il nome della rosa' by Umberto Eco. At the bottom of the 'I tuoi annunci' section is a button: 'Aggiungi un nuovo annuncio' (Add a new announcement).

All'interno della pagina del profilo è possibile modificare le proprie informazioni usando il form a sinistra, eliminare un annuncio che si ha pubblicato o pubblicarne uno nuovo.

Si può anche visualizzare un annuncio pubblicato tramite il pulsante apposito.

Se si decide di eliminare un'annuncio, comparirà un pop-up che chiederà conferma prima di procedere all'eliminazione vera e propria.

Se si decide di pubblicare un nuovo annuncio tramite il pulsante *Aggiungi un nuovo annuncio* si accederà alla pagina di creazione di un nuovo annuncio:

   [Login](#)

Aggiungi un annuncio

* Titolo

* Autore

* Categoria

Altro

* ISBN

* Prezzo

* Condizione

Nuovo

Inserisci l'annuncio

In questa pagina sarà necessario compilare il form con tutti i dati richiesti, quando tutti i campi saranno compilati correttamente sarà possibile pubblicare l'annuncio.

Una volta pubblicato la web-app reindirizzerà automaticamente alla pagina del dettaglio dell'annuncio appena pubblicato.

GitHub repository e deployment

Il progetto è disponibile su GitHub e raggiungibile tramite questo [link](#).

È suddiviso in due repository: una contenente i 5 deliverable chiamata Deliverables, l'altra contenente il codice sviluppato chiamata Code.

Il deployment è stato effettuato tramite **Vercel** e dovrebbe essere possibile accedere all'applicazione tramite questo url: <https://code-three-khaki.vercel.app>.

Ci si può registrare e navigare il sito come nuovi utenti ma in caso non lo si desiderasse fare è possibile accedere alla piattaforma anche tramite le seguenti credenziali:

- username: alec
- password: ale

In caso nel momento della lettura del seguente documento l'applicazione non risultasse disponibile al link fornito precedentemente è possibile seguire le istruzioni fornite nel **README** per eseguire l'intero progetto in locale.

Per utilizzare un database, sarà necessario creare un file .env con all'interno le seguenti stringhe:

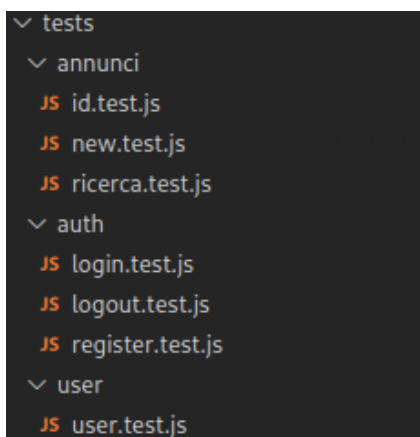
```
mongo_url = [your mongodb connection string]  
jwt_secret = [a secret string]
```

Testing delle API

Il testing delle API è stato realizzato tramite la libreria *Jest*.

Jest permette di dividere i test in suite che contengono i vari unit test. La libreria fornirebbe anche automaticamente il report sulla test coverage, purtroppo non siamo stati in grado di far funzionare quest'ultima funzionalità.

I test sono stati raccolti nella cartella */tests* e divisi in Test Suite:



Questo è un esempio di test:

```
describe("POST /api/auth/login", () => {
  beforeAll( async () => {
    jest.setTimeout(8000);
    await mongoose.connect(process.env.mongo_url);
  });
  afterAll( () => {
    mongoose.connection.close(true);
  });

  test('POST /api/auth/login with Account not registered', async () => {
    var response = await fetch(url, {
      method: 'POST',
      body: JSON.stringify({username: 'accountnotregistered', password: 'password'})
    });
    expect((await response.json()).message).toEqual('Unauthorized: User does not exist');
  });
});
```

Presentiamo ora i risultati dei test:

```
PASS src/app/tests/annunci/new.test.js
PASS src/app/tests/annunci/id.test.js
PASS src/app/tests/auth/login.test.js
PASS src/app/tests/auth/register.test.js
PASS src/app/tests/user/user.test.js
PASS src/app/tests/auth/logout.test.js
PASS src/app/tests/annunci/ricerca.test.js
```

```
Test Suites: 7 passed, 7 total
Tests:       22 passed, 22 total
Snapshots:   0 total
Time:        31.093 s
Ran all test suites.
```