UNIVERSITY OF TRENTO

Dipartimento di Ingegneria e Scienza dell'Informazione

Progetto:

REVHUB

Titolo del documento:

Analisi del Programma

Scopo del documento

Il seguente documento si porta il compito di spiegare il procedimento di come è stato realizzato REVHUB.
Come prima sezione si vede lo User Flow, ovvero una rappresentazione visiva di tutte le decisioni possibili sotto forma di timelines delle funzionalità sviluppate.
Successivamente è presente la struttura delle cartelle e dei file presenti, con una spiegazione delle librerie esterne usate per la programmazione,
Per poi continuare con un'analisi delle API sviluppate, fatta attraverso il diagramma delle risorse estrapolato dal diagramma delle classi presente nel deliverable 3.
Nel capitolo successivo è presente la documentazione delle API con Swagger.
Procedendo avanti è presente una breve descrizione delle pagine utilizzabili dagli utenti.
Successivamente una sezione con la spiegazione della repository di GitHub e di come provare la piattaforma dal punto di vista di un utente.
Per concludere un report della fase di testing e dei risultati.

1. User Flow

1.1 Legenda

In questo Capitolo viene presentato lo user-flow dell'applicazione, ovvero una descrizione grafica del flusso di azioni che qualsiasi tipologia di utente può seguire all'interno della piattaforma.
Di seguito una didascalia dei componenti utilizzati.

Figura 1.1 Legenda dello User Flow

Ora a seguire varie parti dello User Flow con una loro descrizione per poi mostrare tutto lo User Flow completo.

1.2 Sezioni dello User Flow

da modificare, togliere comm

1.2.1 Ricerca

Questa sezione rappresenta un frangente riguardante il momento in cui un qualsiasi utente esegue una ricerca, per poi ricavare una lista di risultati e dopo una selezione di uno dei risultati si ottiene la pagina.
Pagina profilo se nella barra di ricerca si è digitato lo username preceduto da "@", pagina della recensione se si è digitato nella barra di ricerca "#" (eseguirà una ricerca nel database per tag) o sempre di una pagina della recensione con testo senza particolare formattazione nella barra di ricerca (eseguirà una ricerca nel database per titolo).

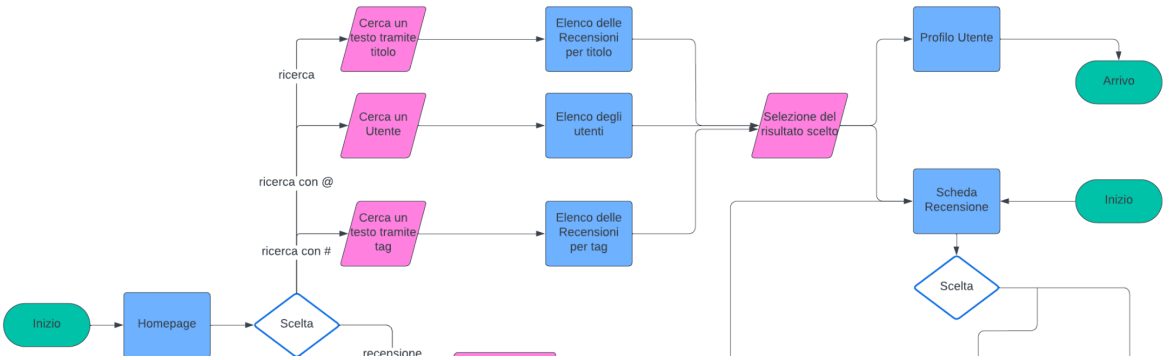


Figura 1.2.1 Ricerca

1.2.2 Menù

Qui è presente la sequenza di azioni possibili per un utente autenticato.
Quindi per lei/lui è possibile selezionare la visualizzazione del profilo personale, fare il logout o di creare una recensione.

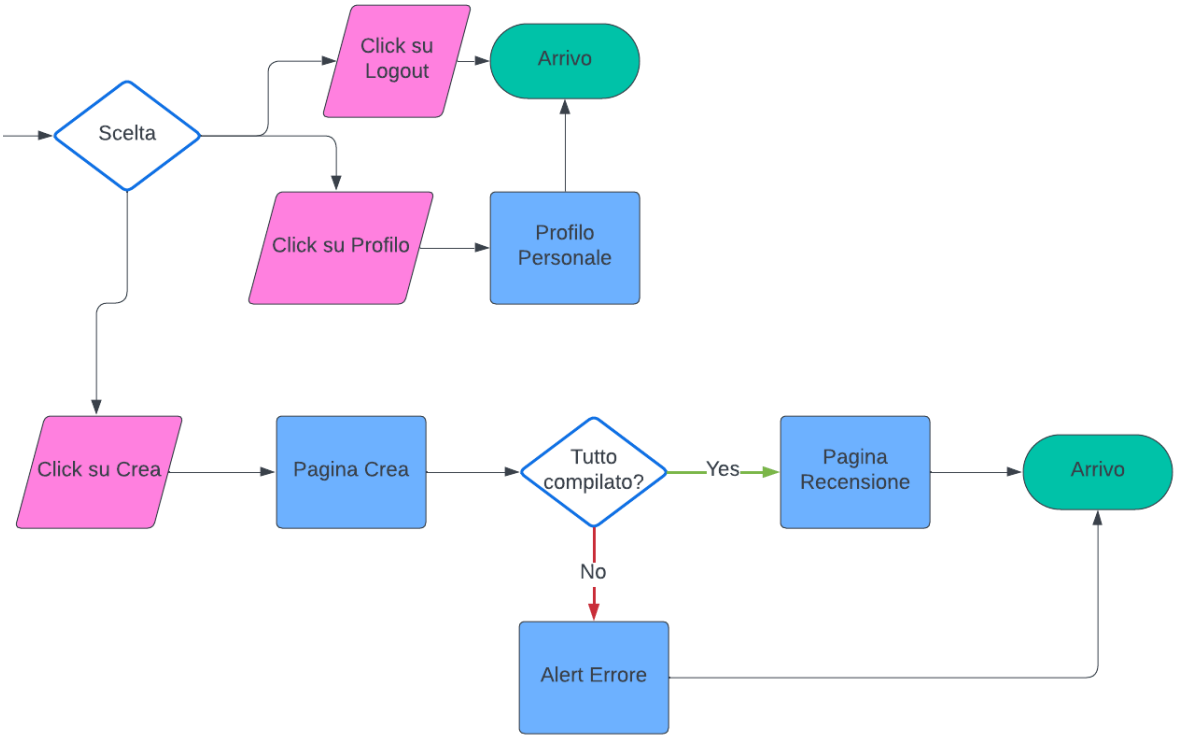


Figura 1.2.2 Menù

1.2.3 Like & Commenti

Qui un utente non autenticato se cercasse di mettere like o di commentare riceverebbe un errore, mentre ciò non avviene per quelli autenticati.

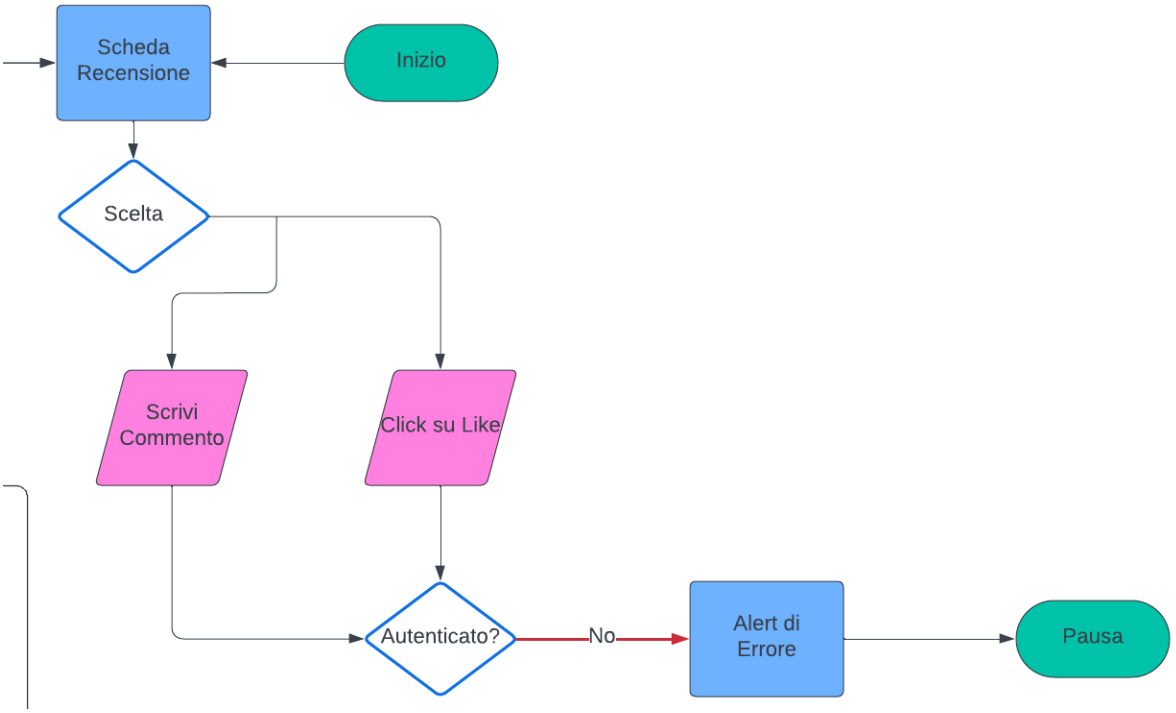


Figura 1.2.3 Like & Commenti

1.2.4 Login

Per concludere è presente il frangente di azioni disponibili durante la creazione di un profilo tramite credenziali già esistenti di Ateneo o Google.
Nello specifico l'utente non autenticato può andare su Login e inserire il profilo che dovrebbe già possedere, se non fosse così può andare nella pagina di registrazione dove deve scegliere una password, username e scegliere se usare le credenziali di Ateneo o di Google con relativa email associata.

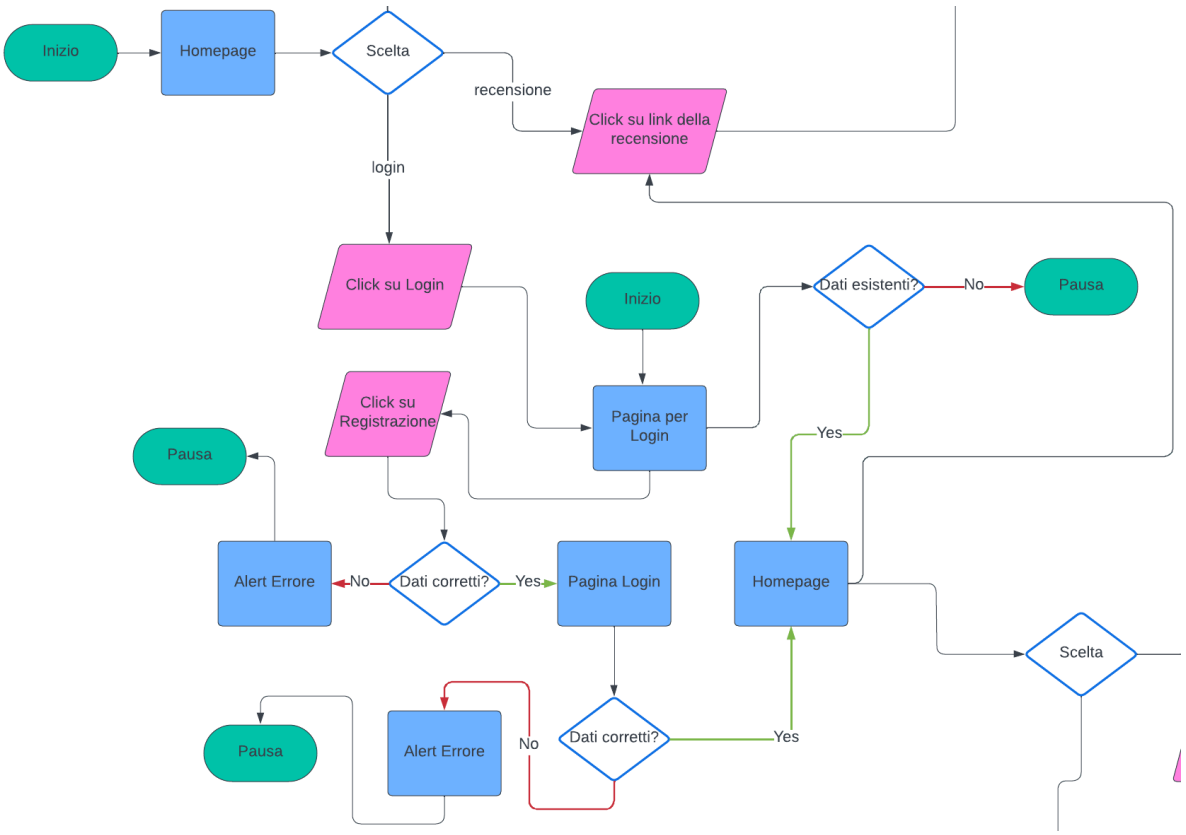


Figura 1.2.4 Login

1.3 User Flow

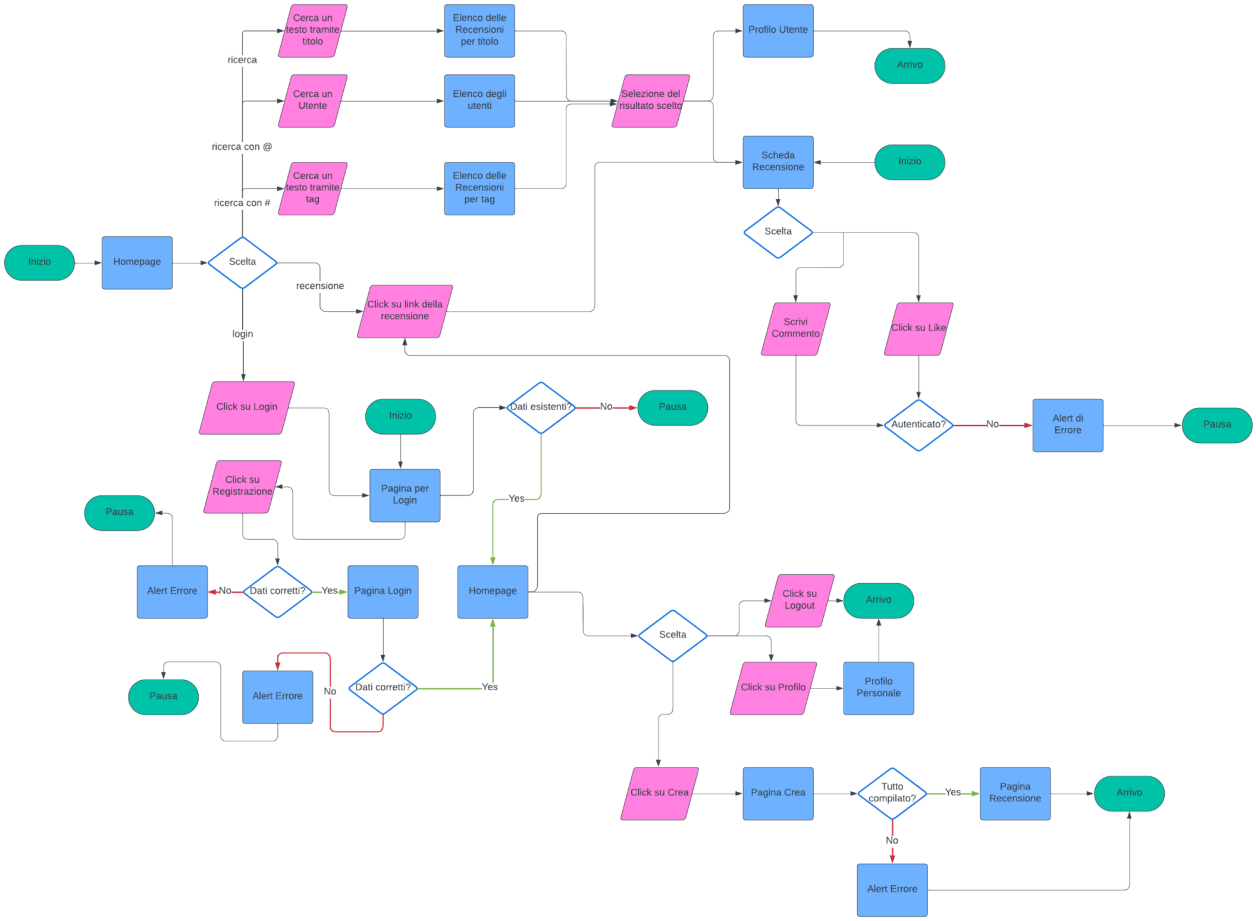
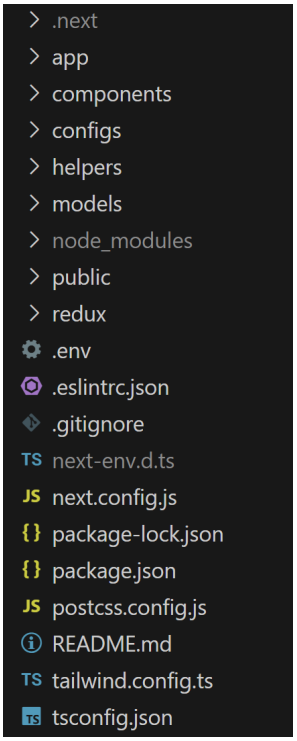


Figura 1.3 User Flow

2. Struttura del Codice

L'applicazione RevHub è stata sviluppata utilizzando per la parte di frontend NodeJS, e a livello di backend è stato usato MongoDB.
Come si può vedere dallo User Flow, sono state implementate tutte le componenti in esso descritte, come la creazione di un profilo con credenziali esterne, il login, la ricerca di un utente o recensione, la pagina profilo, visualizzazione di una recensione e la creazione di una recensione e infine la homepage.

La struttura del progetto è rappresentata come segue.
da aggiungere foto con "test e swagger"

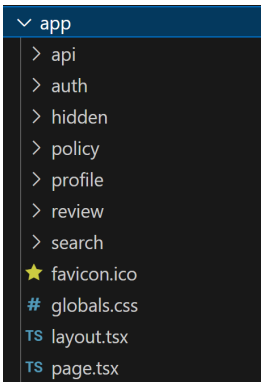


La directory principale è **rev_hub** ed è formata dai seguenti files:

- file swagger.json dove sono documentate le API sviluppate;
- package.json, file per la configurazione generale del progetto;
- file .gitignore, per non rendere pubblici il file .env e la cartella node_modules;
- file .env, dove sono definite le variabili d'ambiente (credenziali di connessione al db MongoDB);
- **MOLTI ALTRI CHE NON SO**
- cartella **INESISTENTE** /test dove sono definiti i test cases per le api;
- cartella /node_modules dove sono presenti le dipendenze nel sistema;
- cartella /redux **PER NON NE HO IDEA**;
- cartella /public per le icone svg;
- cartella /models dove sono definiti i modelli utilizzati dopo per strutturare il db;
- cartella /helpers dove si delega la validazione di un token salvato nei cookies per mantenere il login;
- cartella /configs, dove avviene la connessione al db;
- cartella /components, dove sono definiti la barra di ricerca, del menù, il footer;

- cartella /app dove sono definite le pagine e quindi frontend e backend

La cartella /app è definita come segue:



- la cartella /api, dove sono definite tutte le API;
- la cartella /auth, dove sono definite le pagine di login e registrazione;
- cartella /hidden per quanto riguarda la creazione di credenziali fruibili successivamente per la creazione di un profilo;
- cartella /policy contenente la pagina coi nostri termini di servizio per adempiere a una parte dei requisiti non funzionali;
- cartella /profile, dove è possibile la visualizzazione del profilo di un utente;
- cartella /review, contente il codice per creare e visualizzare una recensione;
- cartella /search, contenente i risultati della ricerca

2.1 Dipendenze

I moduli Node presenti nel file package.json nel campo dependencies sono:

```
"dependencies": {
  "@headlessui/react": "^1.7.17",
  "@heroicons/react": "^2.0.18",
  "@reduxjs/toolkit": "^2.0.1",
  "@types/bcrypt": "^5.0.2",
  "@types/jsonwebtoken": "^9.0.5",
  "axios": "^1.6.2",
  "bcrypt": "^5.1.1",
  "date-fns": "^2.30.0",
  "headlessui": "^0.0.0",
  "heroicons": "^2.0.18",
  "jsonwebtoken": "^9.0.2",
  "mongoose": "^8.0.2",
  "next": "14.0.3",
  "react": "^18",
  "react-dom": "^18",
  "react-redux": "^9.0.2",
  "sass": "^1.69.5"
```

```
  },
```

- **MANCA JEST e swagger**
- jsonwebtoken: modulo per la gestione dei token di accesso;
- mongoose: per le interazioni con MongoDB;
- jest: per il t**esting delle API;**
- **swagger-ui-express: per la documentaz**ione delle API;
- date-fns: per la formattazione e la visualizzazione dell'orario di creazione di una recensione;
- axois: promise-based HTTP Client, per elaborazione delle richieste e il fetch dei dati;
- bcrypt: per l'hashing delle password
- headlessui: Un set di componenti dell'interfaccia utente completamente privi di stile e completamente accessibili per React, **poi boh non so**
- heroicons: per la creazione di icone
- next:
- react:
- sass: per l'utilizzo di css

2.2 Modelli nel Database

Per la gestione dei dati nel progetto sono stati definiti modelli a partire dal diagramma delle classi presente nel deliverable_3. Di conseguenza le risorse necessarie da gestire hanno fatto sì che si sviluppassero cinque modelli con ognuno una collezione nel database.

Modello Comment

Il seguente modello è stato sviluppato a livello di design ma non è stato usato nello sviluppo del codice, come mostrato da user flow. Esso rappresenta i dati necessari per la memorizzazione di un commento fatto da un utente autenticato.

```
export const Comment = new mongoose.Schema (
    {
        id: { type: Number, required: true },
        review_id: { type: Number, required: true },
        author_id: { type: Number, required: true },
        text: { type: String, required: true },
        date: { type: Date, required:true }
    }
)
```

review_id si riferisce all'identificativo della recensione a cui fa riferimento, mentre author_id il medesimo ma si riferisce all'utente autenticato che ha compilato il commento.

Modello Credential

```
export const Credential = new mongoose.Schema(
```

```
    {
        organization: { type: String, required: true },
        email: { type: String, required: true },
        password: { type: String, required: true },
        name: { type: String, required: true },
        surname: { type: String, required: true }
    }
)
```

Modello Rate

Questo particolare modello serve per poter memorizzare nel db di MongoDB l'associazione di ogni singola valutazione in termini di like e dislike. Infatti viene memorizzato con un booleano rate se stiamo parlando di un like (con true) o dislike (con false), poi chi ha messo tale valutazione e a quale recensione.

```
export const Rate = new mongoose.Schema (
    {
        author_id: { type: Number, required: true },
        review_id: { type: Number, required: true },
        rate: { type: Boolean, required: true }
    }
)
```

Modello Review

Per memorizzare le recensioni sono stati usati i seguenti valori, di cui tags, title e text possono essere compilati dall'utente e solo gli ultimi due sono obbligatori. Gli altri dati vengono ottenuti o generati automaticamente dal sistema.

```
export const Review = new mongoose.Schema(
    {
        id: { type: Number, required: true, unique: true },
        title: { type: String, required: true },
        author_id: { type: Number, required: true },
        date: { type: Date, default: Date.now, required: true, },
        tags: { type: [String], default: [], required: false },
        text: { type: String, required: true },
        views: { type: Number, default: 0, required: false }
    }
)
```

Modello User

Per memorizzare i dati sensibili dell'utente ed effettuare la sua registrazione e la verifica delle sue credenziali, come anche la visualizzazione dei suoi dati nella pagina profilo è disponibile il seguente schema usato.

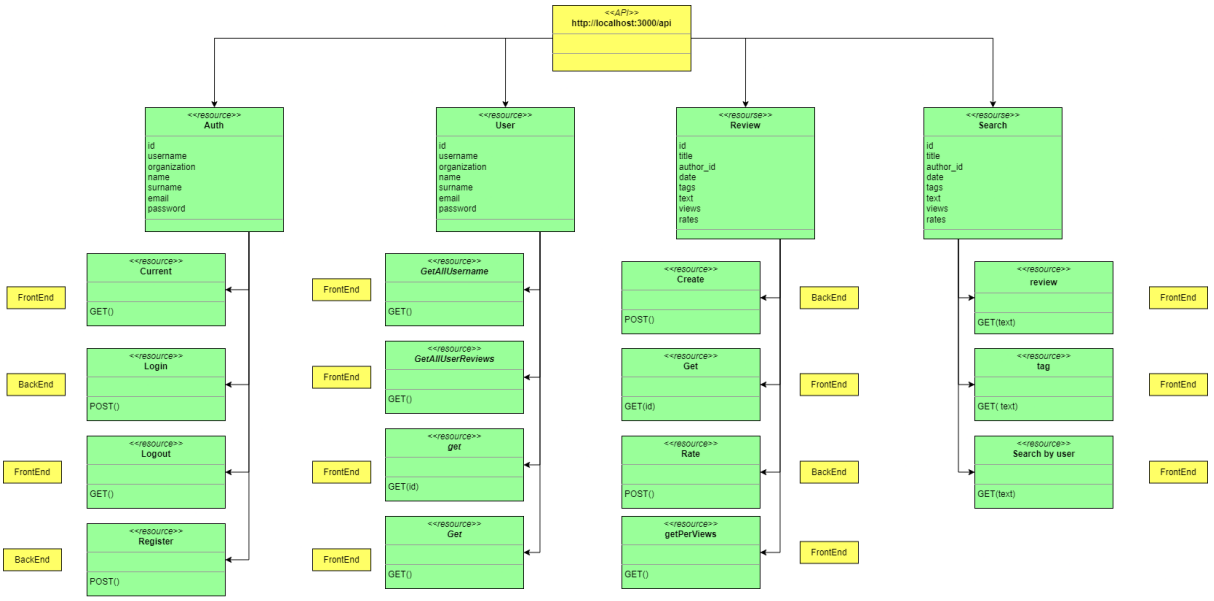
```
export const User = new mongoose.Schema(
    {
        id: { type: Number, required: true, unique: true },
        name: { type: String, required: true },
        surname: { type: String, required: true },
        username: { type: String, required: true, unique: true },
        email: { type: String, required: true, unique: true },
        password: { type: String, required: true },
        followers: { type: [Number], default: [], required: false },
        isAdmin: { type: Boolean, default: false, required: false, }
    }
)
```

Da notare come sia l'id che lo username sono etichettati come unique, ciò è stato scelto per garantire l'unicità dello username ma è stato scelto durante l'implementazione del codice e relativa struttura di usare un identificativo numerico per le ricerche nel db. Poteva essere scelto di fare tutto solo con lo username come chiave primaria, ma è stato scelto per il design di utilizzare l'id.
Successivamente sono stati raccolti i dati sensibili come name, surname, email e password.
followers è un vettore contenente gli identificativi degli utenti che seguono la persona. Inoltre isAdmin è un booleano che identifica se un utente autenticato è anche admin. Questi due ultimi dati con le relative funzionalità non sono state sviluppate (come da user flow) ma sono state incluse nel modello (come da diagramma delle classi).

3. Estratto delle risorse

In questa sezione del documento verranno descritte le API sviluppate dal diagramma delle classi del documento deliverable_3.
Di seguito è presente una estrazione delle risorse dal diagramma delle classi. Questo diagramma mostra come sono state estratte le “risorse” a partire dal diagramme delle classi.
Si parte dalla directory /api per poi avere quattro sotto cartelle contenenti le proprie API: Auth, User, Reviews e Search (con ognuno i dati che utilizza presi dai modelli nel db precedentemente descritti).
Sono stati omesse le sottocartelle /example, /hidden e /testing in quanto, rispettivamente, servono solo per fare una prova di creazione di un nuovo utente, la creazione dei profili per l’accesso con credenziali esterne, e per leggere tutti i dati dal DB.

DEVO CAMBIARE I DATI DI SEARCH?



4. Diagramma delle risorse

Di seguito una rappresentazione in sezioni del diagramma delle risorse per poi avere tutto il diagramma. In tutte le parti del diagramma sono state specificate le componenti in input e in output. Ovviamente le varie API possono ritornare un file json con il risultato desiderato, oppure un errore etichettato; perciò è stata creata una unica classe Error a cui abbiamo collegato tutto per evitare di duplicare classi e complessità.

4.1 /auth

In questa sezione di diagramma le possibilità di output sono un 200 - ok per rappresentare che l'azione è stata effettuata con successo, oppure un 400 - invalid request tutti collegati alla classe Error per identificare una risposta invalida e quindi un errore. E' possibile ricevere come output la classe User che rappresenta la risorsa User analizzata nell'estratto delle risorse precedenti. Da notare la differenza tra User e User_credential, in quanto la prima si riferisce a tutti i dati che si possono estrarre dal db di un utente autenticato, mentre il secondo si riferiscono ai soli dati inseriti dall'utente non autenticato per ottenere l'autorizzazione ad accedere con tali credenziali.
FOTO

4.2 /user

Qui è possibile ricevere in output la classe User, le recensioni associate ad un utente oppure la classe standard Error.

4.3 /review

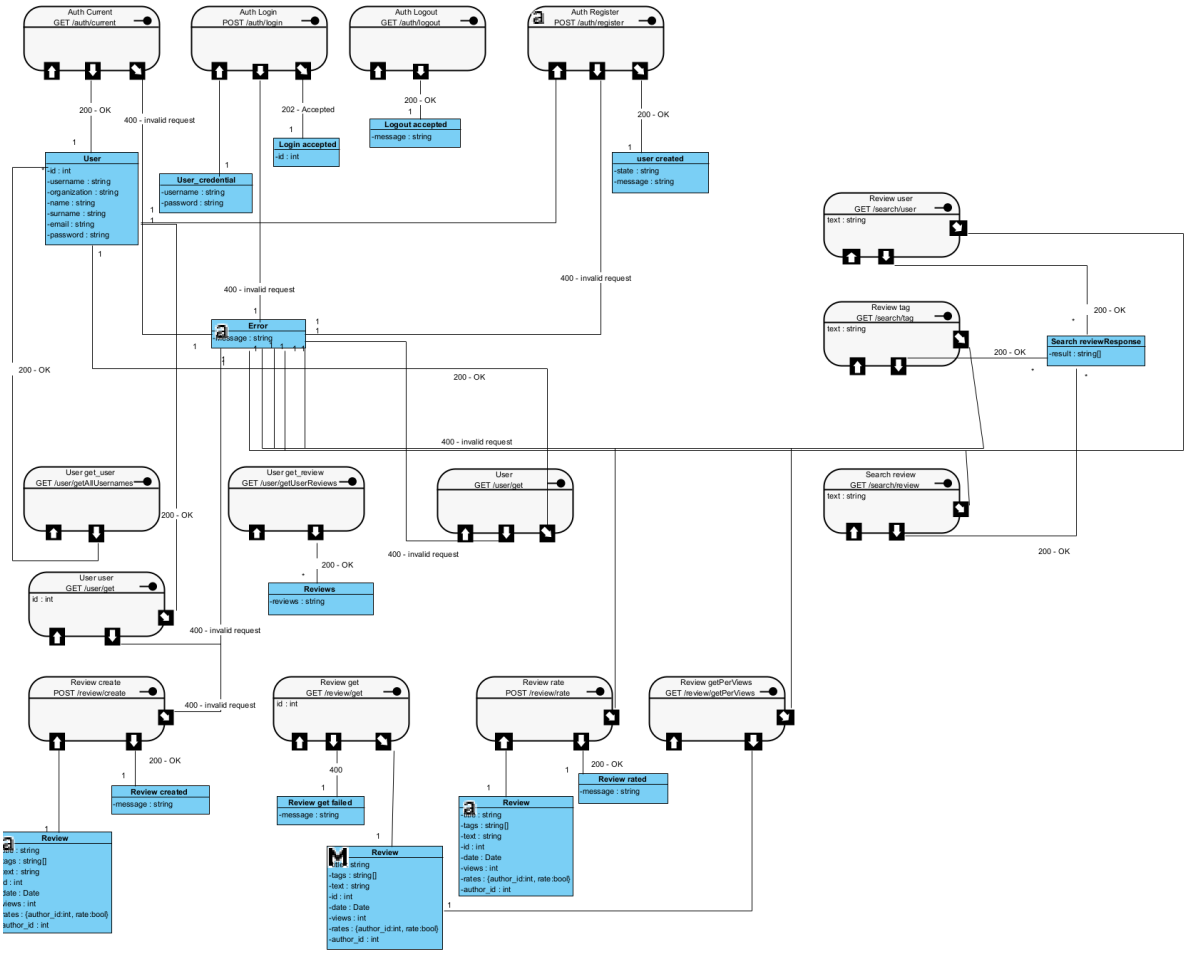
Qui si può ottenere come input e anche output la classe Review oppure ricevere un 200 - ok o un errore 400.

4.4 /search

Infine questa sezione per la ricerca di una recensione tramite titolo o tag, oppure in un utente si può ricevere in input solo il parametro text compilato dall'utente e in output una lista coi risultati coerenti.

4.5 Diagramma delle risorse

da modificare, forse devo collegare i risultati di /search con la classe Reviews?
E collegare search e review ad un unico reviews? e portare tutto ad error?


Auth Current
GET /auth/current
Auth Login
POST /auth/login
Auth Logout
GET /auth/logout
Auth Register
POST /auth/register
200 - OK
400 - invalid request
202 - Accepted
200 - OK
User
-id : int
-username : string
-organization : string
-name : string
-surname : string
-email : string
-password : string
User_credential
-username : string
-password : string
Login accepted
-id : int
Logout accepted
-message : string
user created
-state : string
-message : string
Review user
GET /search/user
text : string
Review tag
GET /search/tag
text : string
Search reviewResponse
-result : string[]
Error
-message : string
User get_user
GET /user/getAllUsernames
User get_review
GET /user/getUserReviews
User
GET /user/get
Search review
GET /search/review
User user
GET /user/get
id : int
Reviews
-reviews : string
Review create
POST /review/create
Review get
GET /review/get
id : int
Review rate
POST /review/rate
Review getPerViews
GET /review/getPerViews
Review created
-message : string
Review get failed
-message : string
Review rated
-message : string
Review
-id : int
-author_id : int
Review
-id : int
-tags : string[]
-text : string
-id : int
-date : Date
-views : int
-rates : [author_id:int, rate:bool]
-author_id : int

5. API Documentation

Qui di seguito verrà descritto il funzionamento delle API sviluppate.
Tali API consentono di interagire con il database, consentendo di aggiungere, modificare e visionare il suo contenuto.

Le API locali fornite da RevHub e i parametri da esse coinvolti sono stati documentati in modo approfondito tramite il file *revhub_api.yaml* contenuto nel progetto all'interno della cartella *api_documentation*.
È possibile osservare l'output di tale file recandosi su Swagger Editor e importando il file yaml.

5.1 API di /auth

Usate per permettere all'utente di registrarsi, fare login e logout e per verificare se l'utente ha fatto o meno il login.

5.1.1 /current

Controlla se nei cookies della request è contenuto un token che corrisponde ad un id chiave di un utente registrato.

5.1.2 /login

Tramite una query al database, viene controllato se le credenziali inserite nel login sono valide. Se lo sono viene inserito nella response un token associato all'utente a cui appartengono tali credenziali.

5.1.3 /logout

Per effettuare il logout si elimina, se esiste, il token dell'utente contenuto nella request.

5.1.4 /register

Permette di creare un nuovo utente collegato alle credenziali UniTn o Google inserite.

5.2 API di /review

Sono le api usate per creare, visualizzare ed interagire con le recensioni.

5.2.1 /create

Crea una nuova recensione usando i dati passati nel corpo della request.

5.2.2 /get/{id}

Restituisce la recensione con ID pari all'ID passato come parametro.

5.2.3 /getPerViews

Restituisce le recensioni con il maggior numero di visualizzazioni.

5.2.4 /rate

Gestisce la votazione degli utenti, aggiornando la raccolta di Rate nel database in base ai dati ricevuti in modo da rispecchiare gli input degli utenti.

5.3 API di /search

Sono le api che consentono di cercare utenti e recensioni.

5.3.1 /review/{text}

Restituisce quelle recensioni che hanno un match tra il testo usato per la ricerca ed il loro titolo.

5.3.2 /tag/{text}

Restituisce quelle recensioni che possiedono tra i tag quello inserito nella ricerca.

5.3.3 /user/{text}

Restituisce quegli utenti che hanno un match tra il testo usato per la ricerca ed il loro username.

5.4 API di /user

Sono le api usate per ottenere dei dati riguardanti gli utenti.

5.4.1 /get/{id}

Restituisce l'utente con ID pari all l'ID passato come parametro.

5.4.2 /get

Restituisce l'utente corrente, utilizzando il token contenuto nella request.

5.4.3 /getAllUsernames

Restituisce una lista con tutti gli username di tutti gli utenti esistenti.

5.4.4 /getUserReviews/{id}

Restituisce le recensioni dell'utente il cui ID è stato passato come parametro

6. Descrizione pagine

In questa sezione vengono descritte le schermate disponibili del frontend e di come possono essere utilizzate.

Homepage (autenticato)
Partendo dalla pagina principale, visibile a tutti gli utenti, è possibile in prima azione e senza fare nulla visionare la classifica delle recensioni più visualizzate. Se si volesse interagire con la barra di navigazione si possono aprire delle scelte per gli utenti autenticati, per quelli non autenticati verrà trattato successivamente.
Se si è autenticati si possono fare cinque azioni col menù.
Cliccare in alto a sinistra il logo per ritornare alla homepage.
Cliccare la barra di ricerca e dopo aver scritto qualcosa premere “Cerca” per ottenere i risultati desiderati. Si possono vedere i risultati delle ricerche solo se si compila l’input della barra di ricerca.
Cliccare “Profilo” per visionare i propri dati di profilo e vedere le recensioni scritte associate a quel profilo. Per visionare il profilo di altri utenti autenticati, bisogna digitare “@”+<nome_utente> nella barra di ricerca.
Cliccare “Crea” per accedere alla pagina per creare una recensione.
Oppure cliccare “logout” per effettuare il logout.

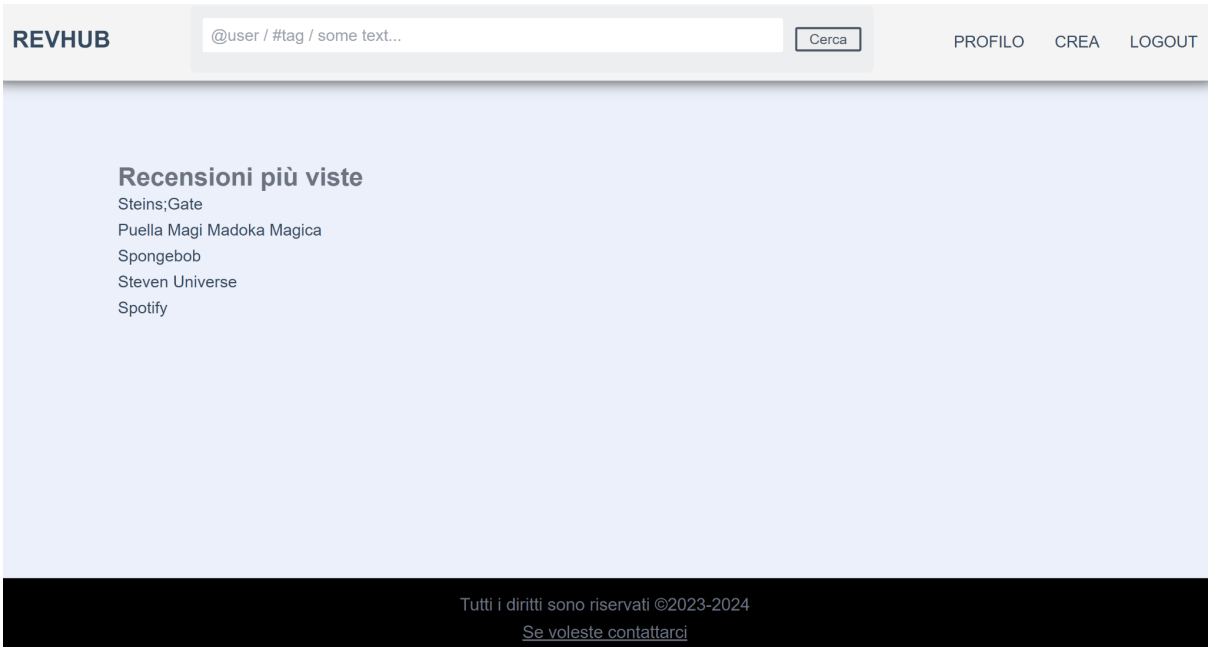


Figura 6.1 - schermata home di un utente loggato

Visualizzazione di una recensione
Dopo aver scritto un campo di testo nella barra di ricerca e aver selezionato una delle scelte ottenute per le recensioni, oppure dopo aver cliccato una delle recensioni esistenti nella pagina profilo di qualcuno si arriva alla sua visualizzazione.
In questa schermata, fruibili per tutti gli utenti, è possibile visualizzare la recensione e mettere like o dislike (questa opzione è disponibile dopo essersi autenticati).

Figura 6.2 - schermata visualizzazione di una recensione

Pagina profilo

Dopo aver cliccato “Profilo” nella barra di navigazione, o aver selezionato uno dei risultati nella pagina di ricerca, si possono vedere i dati associati a quel profilo e tutte le recensioni che ha scritto. Qui si possono cliccare i titoli delle recensioni per andare alla pagina descritta precedentemente e visualizzare il contenuto.

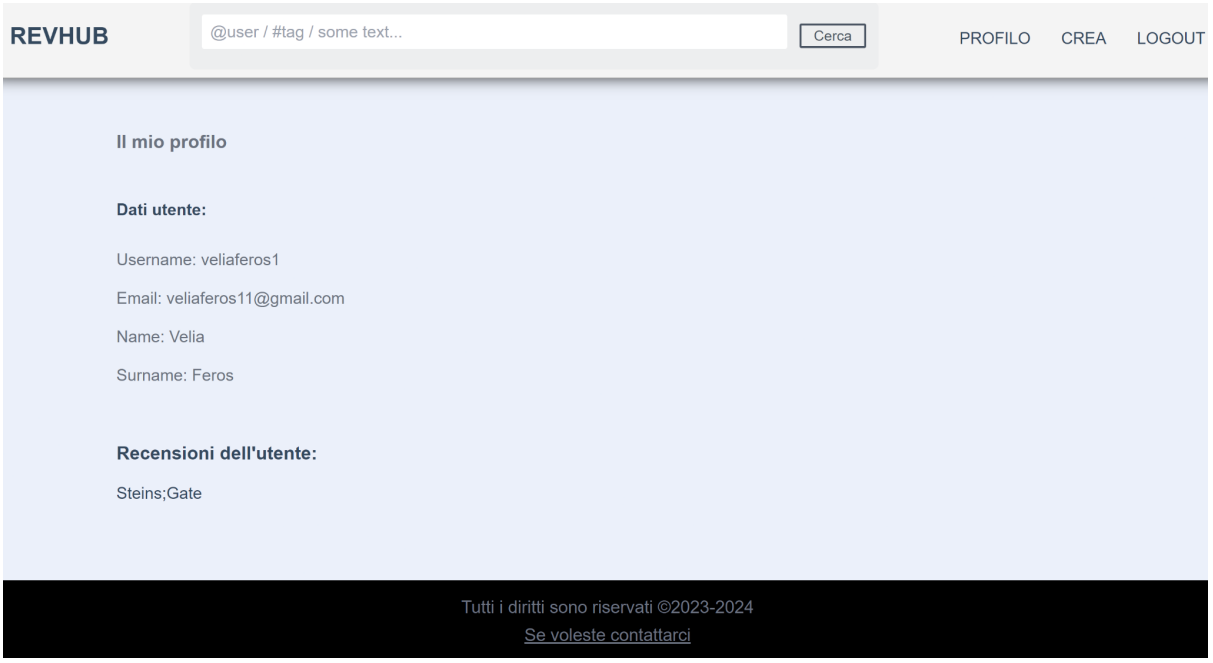


Figura 6.3 - schermata di profilo

Pagina Creazione

Dopo aver cliccato “Crea” nel menù si accede a questa pagina con un form da compilare per creare la propria recensione, si può accedere qui solo se autenticati.
Il campo Titolo e Testo sono obbligatori, viene specificato con un placeholder quali sono i caratteri accettati e non. Per aggiungere tags bisogna scrivere una singola parola o più unite da “_” e poi cliccare su “+”.

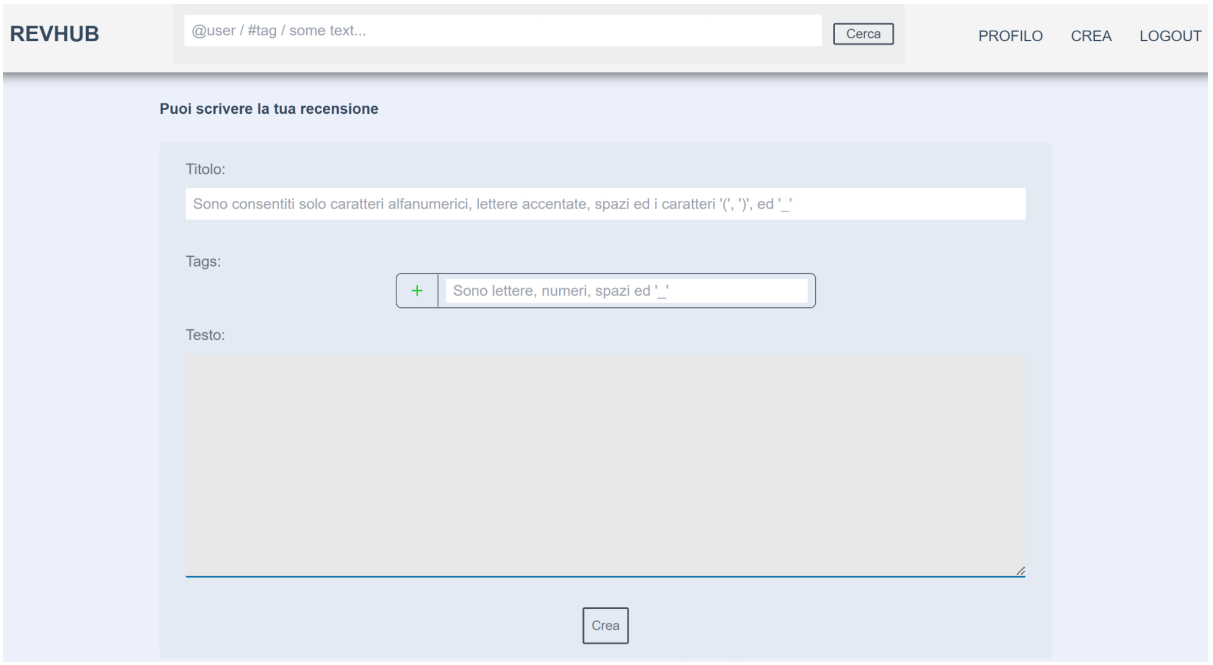


Figura 6.4 - schermata creazione di una recensione

Homepage (non autenticato)
Questa è la stessa schermata tranne per la barra di navigazione dove compare solo il logo, la barra di ricerca e il pulsante “Login” che porta alla pagina di ricerca, successivamente descritta.

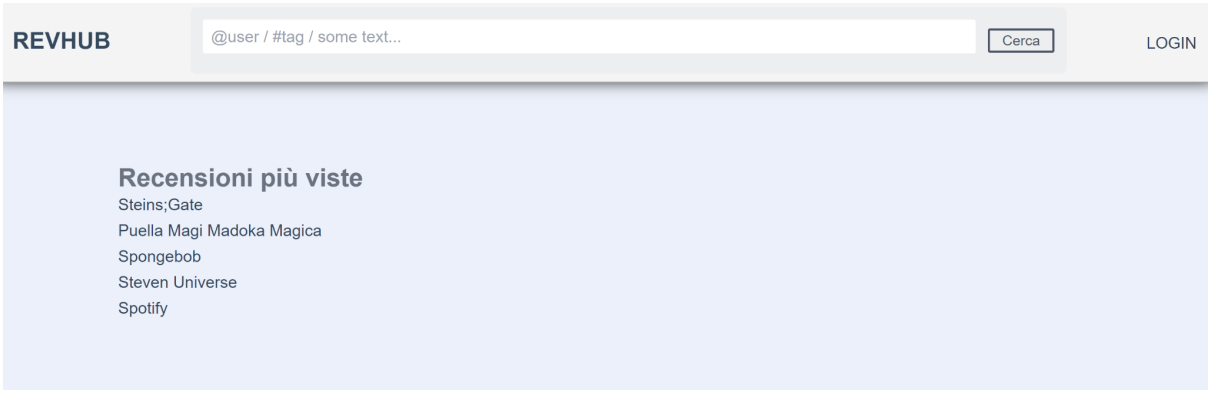


Figura 6.5 - schermata home di un utente non loggato

Pagina Login

In questa schermata è possibile effettuare il login con le credenziali già create nel form di registrazione che verrà spiegato in seguito. Se non si dovessero possedere le credenziali, si possono usare le credenziali di ateneo o Google da inserire durante la registrazione.

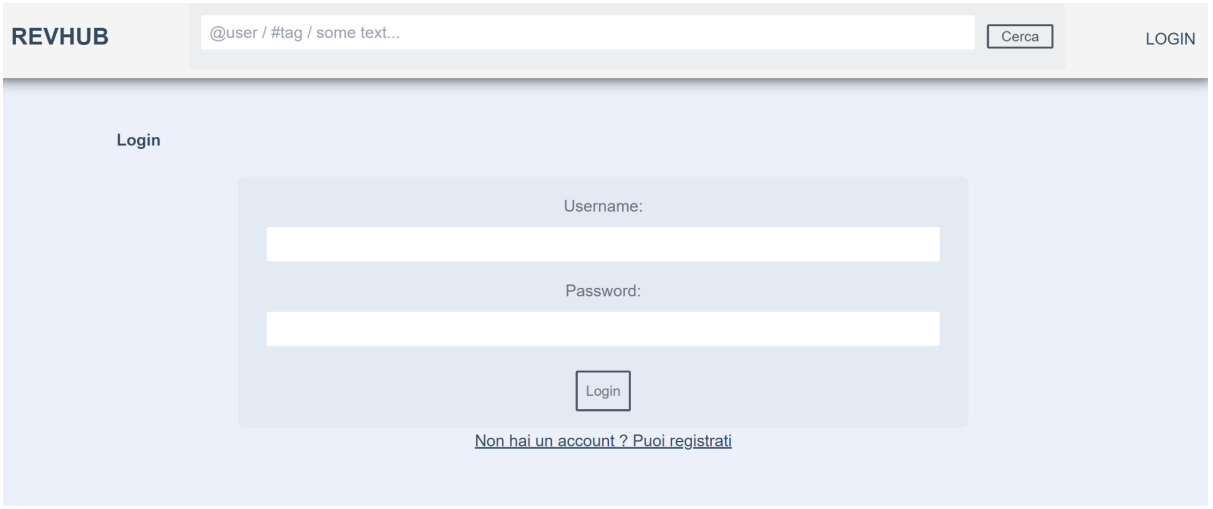


Figura 6.6 - schermata di login

Pagina di Registrazione

Questa è la schermata per gli utenti non autenticati che non possiedono credenziali RevHub o per chi volesse fare un altro profilo. Per compilarla servono credenziali di Ateneo o di Google proprie, quindi si seleziona su “Organizzazione” quale tipologia di credenziali si userà, poi si inserisce la email associata alla tipologia di credenziali terze che si è deciso di utilizzare, poi una password nuova a piacere (nota bene: non è richiesto di inserire la password delle credenziali terze, ma una nuova che verrà associata al nuovo profilo RevHub che si sta creando) e infine un username unico.

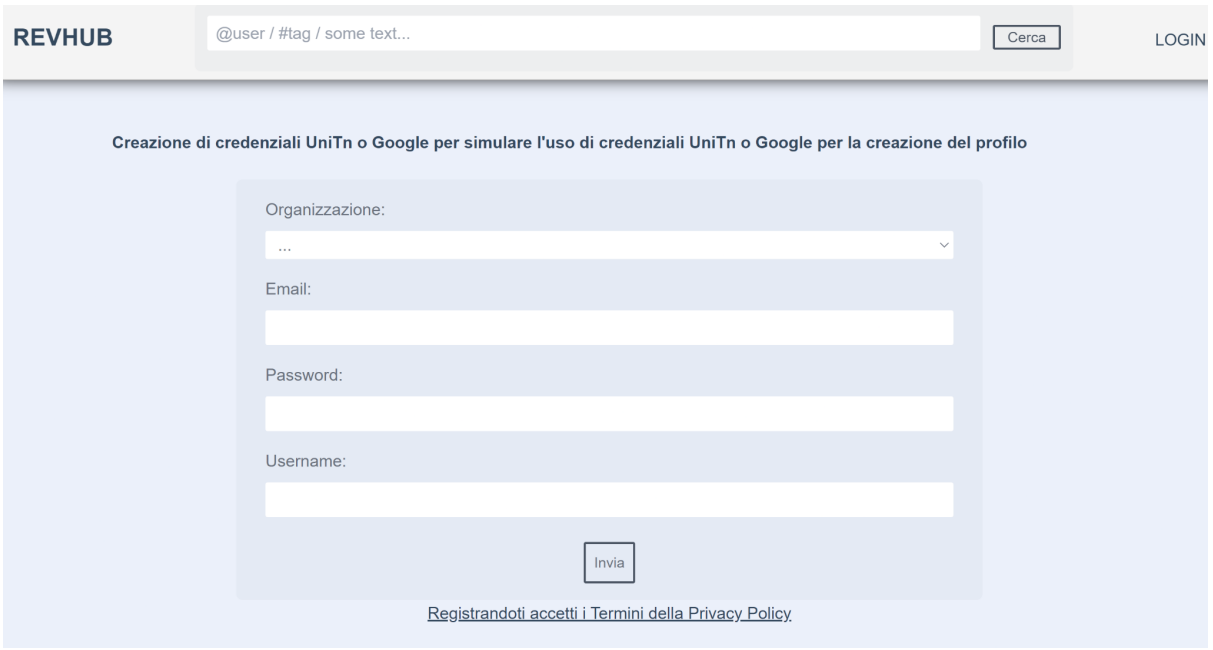


Figura 6.7 - schermata di creazione account

Pagina Ricerca

Per concludere è presente la pagina di ricerca a cui si può accedere esclusivamente dopo aver scritto qualcosa nel campo di ricerca della barra di ricerca presente nel menù e aver premuto “Cerca”. Si possono cercare utenti, tramite il loro username, e recensioni, tramite i loro tags associati o tramite titolo.

Per cercare un utente si può digitare tutto il suo username o anche una sua parte tramite il formato “@” + <username>, mentre per le recensioni; serve formattare il testo in modo preciso (pena nessun risultato).

Per cercare una recensione tramite tags bisogna scrivere nel campo testo della barra di ricerca “#” + <tag>, infine per cercare tramite titolo non serve alcuna formattazione.

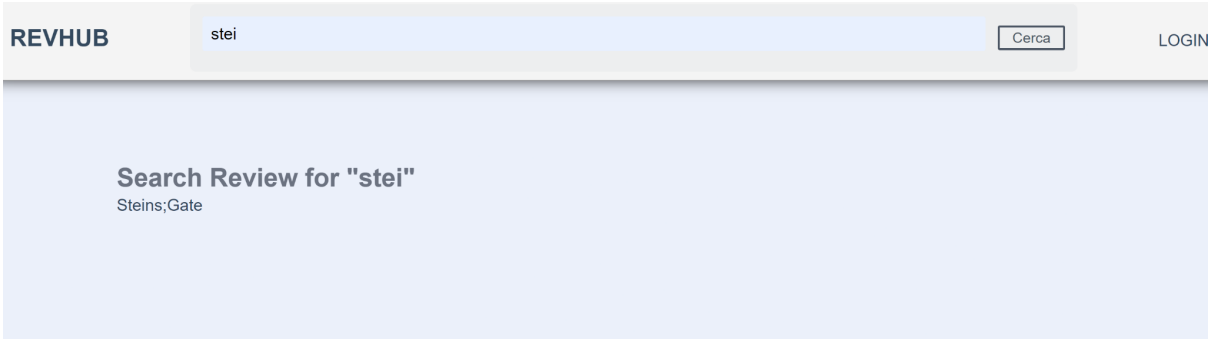


Figura 6.8 - schermata di visualizzazione dei risultati di una ricerca

7. Repository

Nel seguente link è presente tutto il progetto sotto forma di codice open source: https://github.com/G13-RevHub/RevHub

Per eseguire il codice in locale bisogna seguire i passaggi seguenti, descritti anche nel file README.md:

- clonare il progetto tramite il link https://github.com/G13-RevHub/RevHub.git

Successivamente:

```
# eseguire sul terminale, aperto sulla directory del progetto clonato
npm run dev
# oppure, se si dispone di sufficiente RAM
npm run dev --turbo

oppure si può creare una build e lanciare quest'ultima, ottenendo una
versione più veloce ma che richiede un rebuild per essere modificata,
tramite i comandi:
npm run build
npm run start
```

- Quindi apri il seguente indirizzo http://localhost:3000 nel suo browser.

La preghiamo di non utilizzare la connessione di rete di Ateneo durante l'esecuzione del codice, perché è presente un firmware che non ci consente di eseguire la connessione al database di MongoDB, di conseguenza falliranno tutte le chiamate ad api che interagiscono col database.

8. Testing

Per effettuare il testing è stata usata la libreria Jest, come citato nella struttura del codice, e **il modulo superjest**.