

Ingegneria del software

DOCUMENTO DI SVILUPPO

MOUNTAIN WONDERS
Gruppo G24

Anno accademico 2023/2024

Indice

1 Scopo del documento	4
2 User flow	5
3 Resource diagram	6
4 API diagram	7
5 Sviluppo applicazione	10
5.1 Struttura progetto	10
5.2 Dipendenze	11
5.3 Modelli	12
5.3.1 Modello User	12
5.3.2 Modello Mountain	12
5.3.3 Modello Refuge	13
5.4 API	15
5.4.1 API modello User	15
5.4.2 API modello Mountain	16
5.4.3 API modello Refuges	16
5.5 API documentation - Swagger	18
5.6 Pagine frontend	20
5.6.1 Home page	20
5.6.2 Pagina login	21
5.6.3 Pagina registrazione	22
5.6.4 Pagina montagne	23
5.6.5 Pagina rifugi di una montagna	24
5.6.6 Pagina rifugi	25
5.7 Testing	27

5.7.1	Risultati del testing	28
5.8	GitHub repository e informazioni sul deployment	30
5.8.1	Nota per la documentazione	30

1 Scopo del documento

Questo documento riporta la parte di sviluppo del software. In particolare contiene i seguenti diagrammi:

- User flow diagram
- Resource diagram
- API diagram

Questi saranno la base per la corretta implementazione del codice per l'applicazione web di Mountain Wonders.

Segue poi la parte di sviluppo del codice dell'applicazione, in cui verranno mostrati:

- Struttura del progetto
- Modelli utilizzati
- Presetazione API
- Presentazione front end
- Testing
- Informazioni sul deploy

2 User flow

In questa sezione viene inserito lo user flow diagram, diagramma che viene utilizzato per visualizzare le possibili azioni eseguite da un utente, in modo da comprendere poi le eventuali operazioni da eseguire sul database. Viene di seguito inserita una legenda per una migliore comprensione del diagramma:



Figura 1: User flow diagram legend

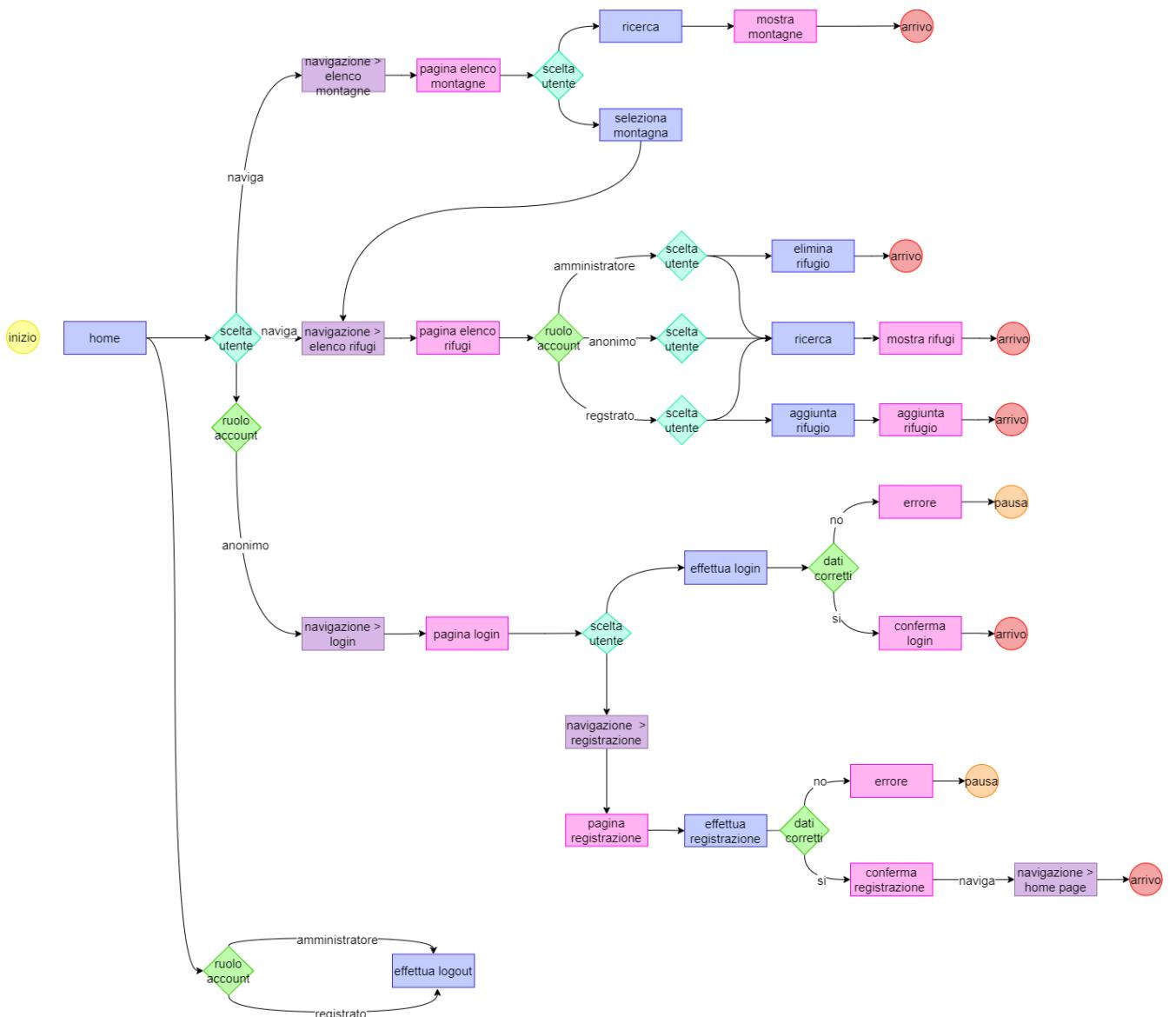


Figura 2: User flow diagram

3 Resource diagram

Questo diagramma viene utilizzato per comprendere gli oggetti presenti all'interno del database, come ad esempio gli utenti, e definire quali operazioni sono possibili fare per ognuno di essi.

Partendo dal class diagram, sono state individuate le risorse principali su cui si basa la nostra applicazione:

- Utente
- Rifugio
- Montagna

Da queste abbiamo poi sviluppato altre risorse partendo dai metodi delle risorse iniziali. Per ogni risorsa vengono evidenziati i metodi necessari per interagire con il database e se si tratta di richieste di tipo GET, POST oppure DELETE.

Vengono poi specificati anche i parametri che verranno utilizzati, inserendo body qualora siano richiesti tutti i parametri della risorsa.

Per ogni metodo di una risorsa viene inoltre mostrato se il metodo vada ad applicare modifiche nella parte di frontend o di backend dell'applicazione.

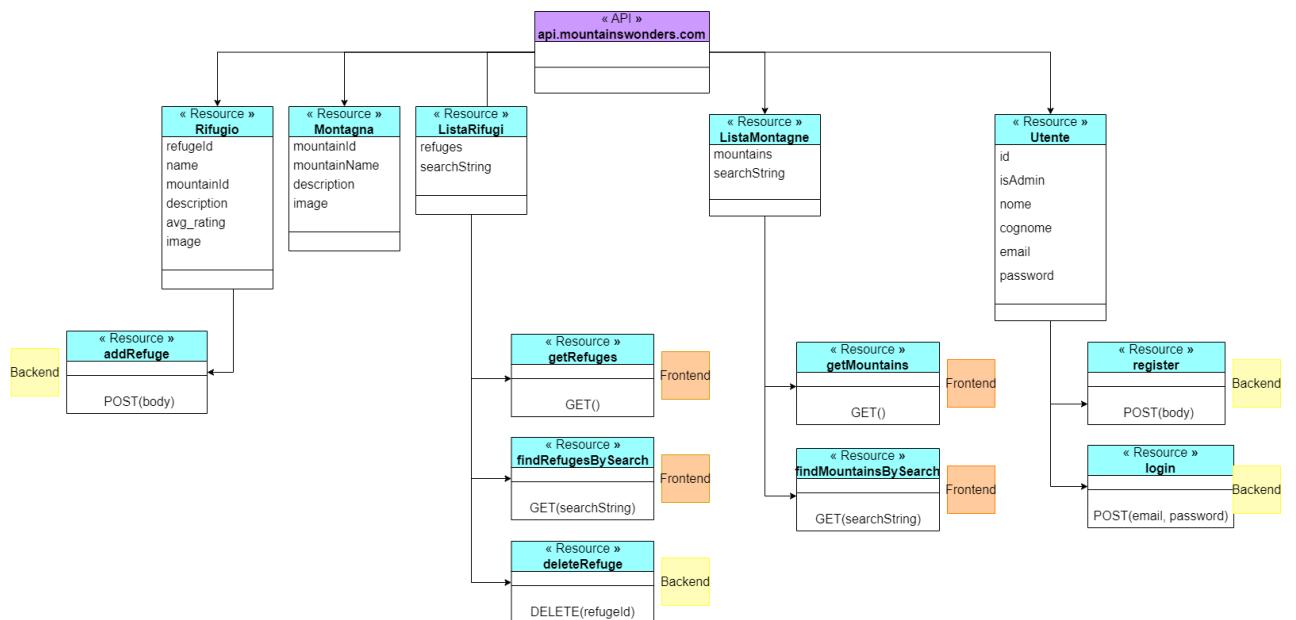


Figura 3: Resource diagram

4 API diagram

In questo diagramma viene specificato per ogni operazione che deve essere eseguita su una risorsa all'interno del database di quali input e quali output necessita.

Queste informazioni saranno poi utilizzate per la creazione delle API in seguito.

Ogni API può tornare un diverso errore a seconda del comportamento ottenuto dal database, che può essere:

- 200: azione eseguita correttamente
- 201: elemento creato correttamente
- 401: operazione non permessa perchè non si è fatto l'accesso
- 403: operazione non permessa perchè non si hanno i permessi adeguati
- 404: elemento non trovato
- 409: dato già presente nel database
- 415: formato del file non supportato

Per maggior chiarezza, il diagramma delle API è stato suddiviso.

- API diagram 1

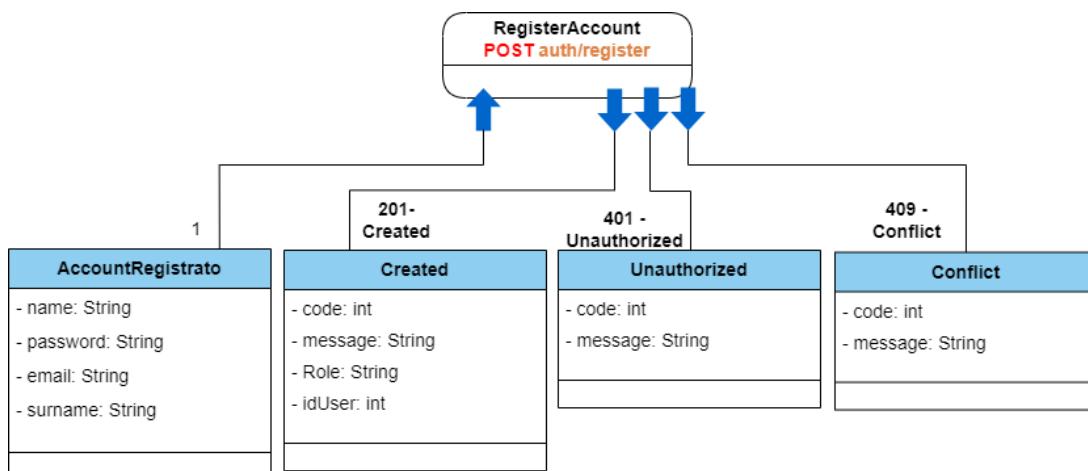


Figura 4: Register api diagram

- Api diagram 2

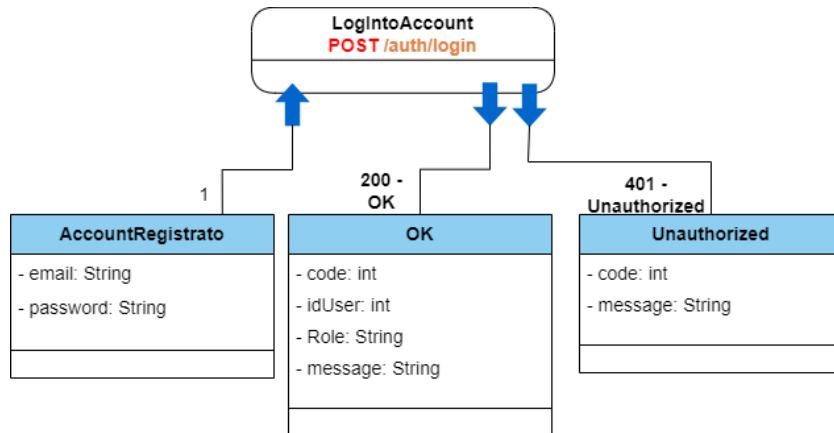


Figura 5: Login api diagram

• Api diagram 3

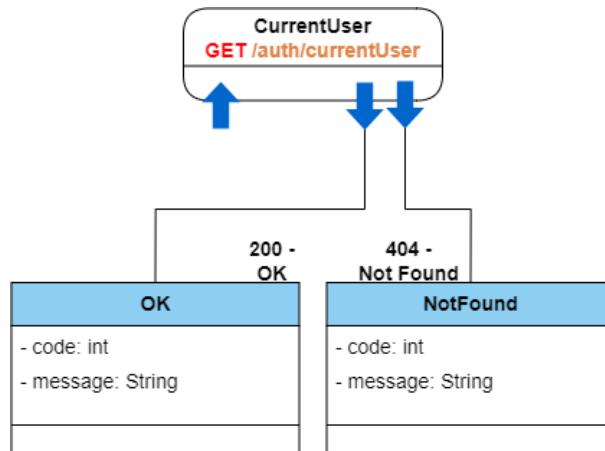


Figura 6: Current user api diagram

• Api diagram 4

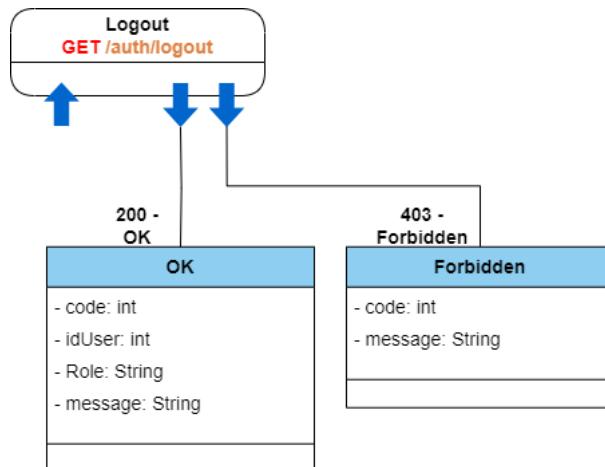


Figura 7: Logout api diagram

4 API DIAGRAM

- Api diagram 5

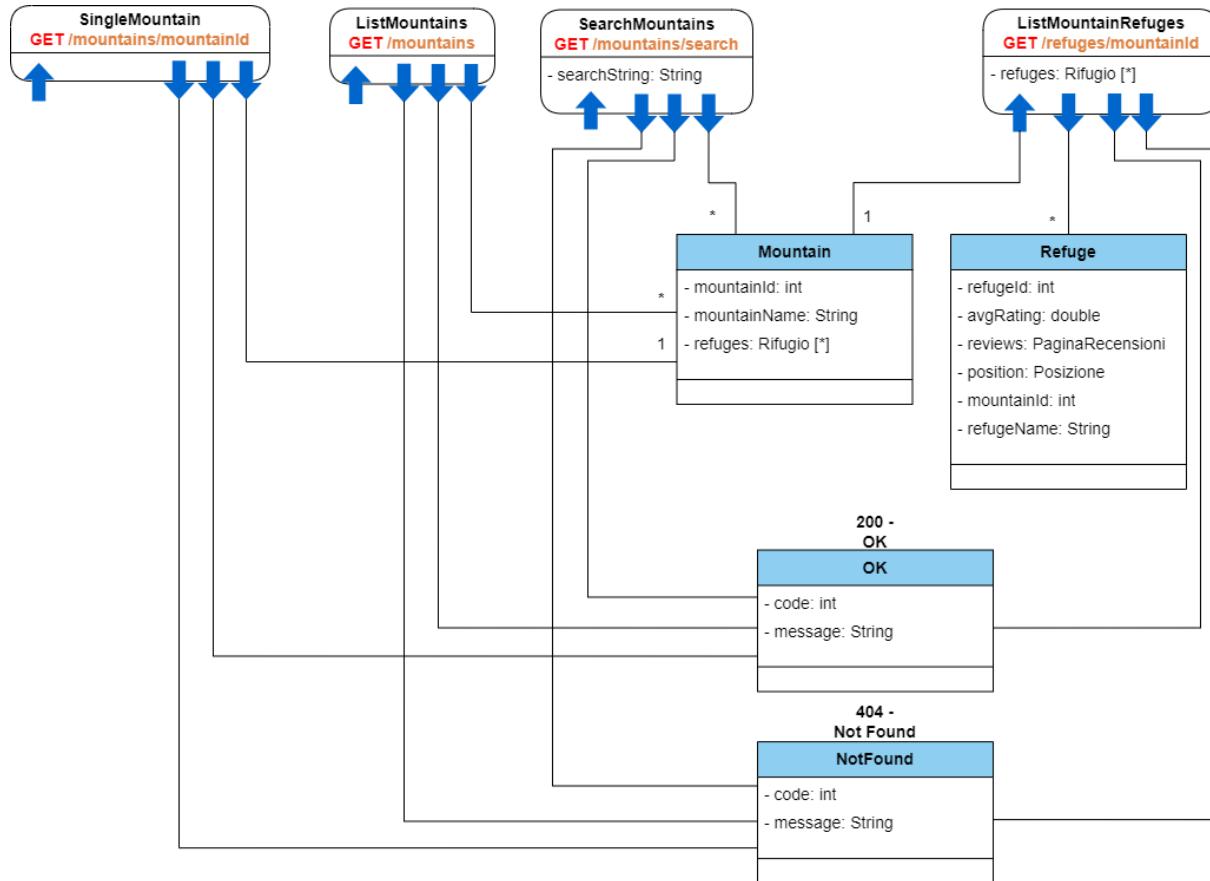


Figura 8: Mountains api diagram

- Api diagram 6

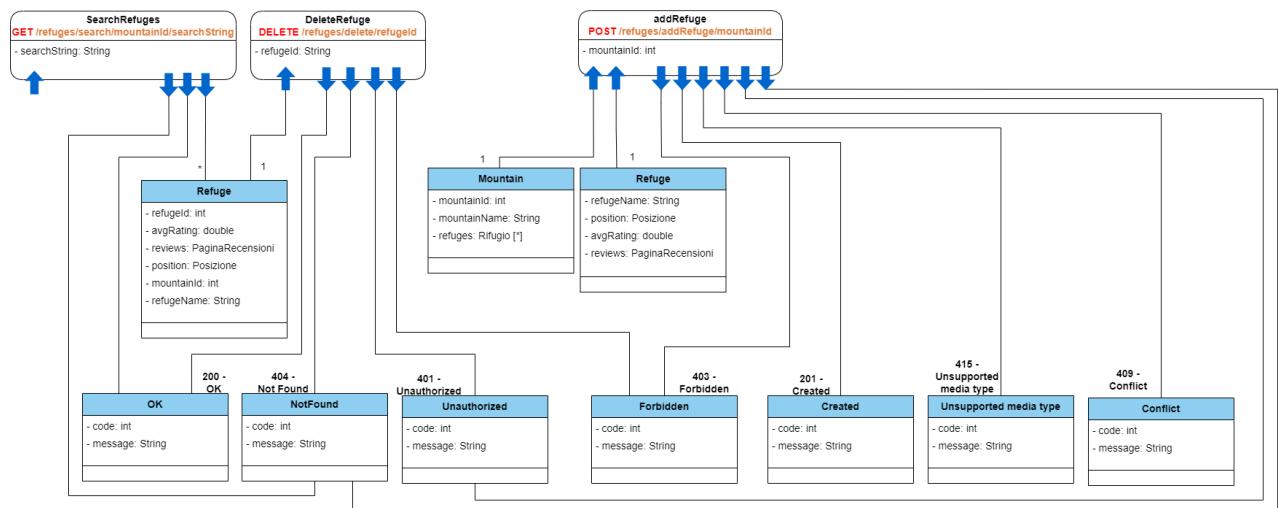


Figura 9: Refuges api diagram

5 Sviluppo applicazione

Verrà di seguito presentato come è stata sviluppata la nostra applicazione web, mostrando prima i modelli utilizzati all'interno del database per salvare le informazioni necessarie, proseguendo poi con la descrizione delle API per poi concludere con la descrizione del frontend.

5.1 Struttura progetto

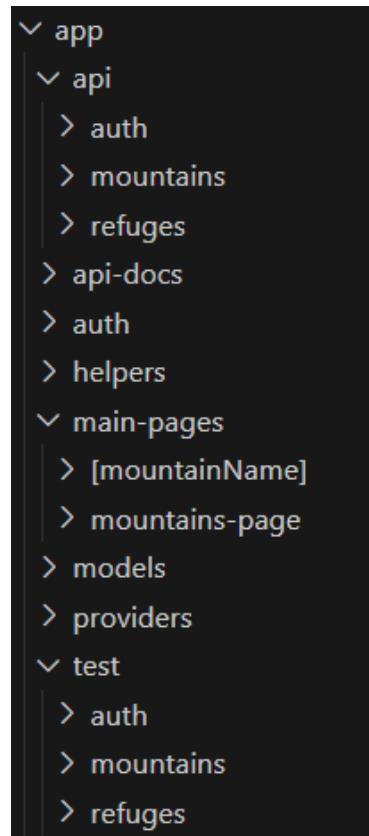


Figura 10: Stuttura del progetto

La cartella principale è app, che contiene le sottocartelle per:

- api
- pagine front-end
- test

La cartella api contiene le api necessarie per l'applicazione, che verranno descritte in dettaglio nella sezione "Swagger".

La cartella auth contiene lo sviluppo frontend per le pagine di login e registrazione di un utente.

La cartella main-pages contiene lo sviluppo frontend per le pagine principali dell'applicazione, ossia la pagina delle montagne e quella dei rifugi.

Nella cartella modelli sono contenuti i modelli utilizzati dall'applicazione, ossia il modello utente, montagna e rifugi. Questi verranno descritti meglio in seguito nella sezione modelli.

Nella cartella test sono contenuti tutti i test eseguiti sulle api.

In ogni cartella, i file sono stati organizzati poi in sottocartelle, in modo da permettere un facile recupero dei dati necessari per ricontrrollare.

5.2 Dipendenze

Elenchiamo di seguito le dipendenze esterne utilizzate dall'applicazione Mountain Wonders:

- mongoose from mongoose utilizzato per la connessione al database
- nextRequest, nextResponse from next/server per una continua interazione tra database e applicazione
- jwt from jsonwebtoken per la criptatura dei dati lato client
- cookies from next/headers per il salvataggio dello stato al login di un utente
- bcrypt from bcryptjs per la protezione dei dati lato server
- axios from axios per la comunicazione con il server
- useRouter from next/navigation per la redirezione delle api
- React from react per la creazione dell'interfaccia utente
- swagger-ui-react/swagger-ui.css per la documentazione delle api

5.3 Modelli

Verranno di seguito mostrati i modelli utilizzati per lo sviluppo dell'applicazione, estrapolati a partire dal diagramma delle risorse.

5.3.1 Modello User

Il primo modello è quello dell'utente che si registra nella nostra applicazione, che è stato chiamato User.

```
export const userSchema = new mongoose.Schema(  
  {  
    name: {  
      type: String,  
      required: true,  
    },  
    email: {  
      type: String,  
      required: true,  
    },  
    password: {  
      type: String,  
      required: true  
    },  
    isActive: {  
      type: Boolean,  
      default: true,  
      required: true,  
    },  
    isAdmin: {  
      type: Boolean,  
      default: false,  
      required: false,  
    },  
    {  
      timestamps: true,  
    }  
  }  
);
```

Figura 11: Modello user

I dati nome, cognome sono necessari per l'identificazione di un utente, mentre gli attributi email e password saranno poi utilizzati per permettere all'utente di effettuare il login all'interno dell'applicazione web.

L'attributo isAdmin viene utilizzato per verificare quali permessi ha l'utente loggato e permettergli di effettuare determinate operazioni o meno.

5.3.2 Modello Mountain

Segue poi il modello della montagna.

```
export const mountainSchema = new mongoose.Schema(  
  {  
    id: {  
      type: Number,  
      required: true,  
    },  
    name: {  
      type: String,  
      required: true,  
    },  
    image: {  
      type: String,  
      required: true,  
    },  
    description: {  
      type: String,  
      required: true,  
    }  
  }  
)
```

Figura 12: Modello montagna

Questo modello è dotato di parametri necessari per il riconoscimento da parte degli utenti. In questo modo l'utente prima di aggiungere un rifugio ad una montagna, può verificare se la montagna a cui vuole aggiungere il rifugio è quella scelta.

5.3.3 Modello Refuge

L'ultimo modello dell'applicazione è quello che rappresenta i rifugi.

```
export const refugeSchema = new mongoose.Schema(  
  {  
    name: {  
      type: String,  
      required: true,  
    },  
    avgRating: {  
      type: Number,  
      required: false,  
    },  
    description: {  
      type: String,  
      required: true,  
    },  
    mountainId: {  
      type: Number,  
      required: false,  
    },  
    image: {  
      type: String,  
      required: true,  
    },  
    __v: {  
      type: Number,  
      default: 0,  
      required: false,  
    }  
  }  
)
```

Figura 13: Refuge model

Ogni rifugio è dotato di nome, descrizione, mountainId e image che forniscono dei dettagli sul rifugio agli utenti che lo visitano all'interno del sito.

Ha inoltre il parametro avgRating che sarà settato alla creazione del rifugio e poi rimarrà sempre uguale dal momento che il team ha deciso di non implementare la parte di applicazione necessaria alla modifica di questo attributo.

Il parametro __v verrà utilizzato per mostrare tutti i rifugi presenti all'interno del database dal momento che usando vercel per il deployment il team ha riscontrato dei problemi durante il fetching.

5.4 API

Vengono di seguito descritte le API utilizzate per lo scambio di informazioni tra applicazione e database.

5.4.1 API modello User

Verranno descritte di seguito le API utilizzate per un utente:

- currentUser:

questa API viene utilizzata per identificare il ruolo dello user attualmente loggato all'interno del sito.

Si tratta di una richiesta di tipo GET.

Non riceve alcun parametro in input dal momento che gli attributi di cui ha bisogno per verificare il ruolo di un utente li recupera dai cookie, utilizzando il package cookies di next/headers.

In output viene ritornata una NextResponse contenente lo stato della risposta o dell'errore e il relativo messaggio.

- login:

questa API viene utilizzata per permettere ad un utente di loggarsi all'interno del sito ed eseguire azioni specifiche permesse soltanto ad un utente loggato.

Si tratta di una richiesta di tipo POST essendo che ha bisogno di inviare dei dati al database per confrontarli con quelli già presenti.

Come parametro in input contiene i dati inseriti nel form di login da parte dell'utente, ossia email e password.

In output viene ritornata una NextResponse contenente lo stato della risposta o dell'errore e il relativo messaggio.

- logout:

questa API viene utilizzata per permettere ad un utente di scollegarsi dal sito una volta terminato l'utilizzo.

Si tratta di una richiesta di tipo GET.

Non riceve alcun parametro in input dal momento che gli attributi di cui ha bisogno per verificare il ruolo di un utente li recupera dai cookie, utilizzando il package cookies di next/headers.

In output viene ritornata una NextResponse contenente lo stato della risposta o dell'errore e il relativo messaggio.

- register:

questa API viene utilizzata per permettere ad un utente di effettuare la registrazione all'interno del sito per eseguire azioni specifiche permesse soltanto ad un utente loggato.

Si tratta di una richiesta di tipo POST essendo che ha bisogno di inviare dei dati al database per confrontarli con quelli già presenti.

Come parametro in input contiene i dati inseriti nel form di registrazione da parte dell'utente, ossia nome, cognome, email e password.

In output viene ritornata una NextResponse contenente lo stato della risposta qualora l'utente sia stato inserito correttamente o dell'errore e il relativo messaggio.

5.4.2 API modello Mountain

Verranno descritte di seguito le API utilizzate per una montagna:

- mountains:

questa API viene utilizzata per recuperare tutte le montagne presenti all'interno del database.

Si tratta di una richiesta di tipo GET perchè è necessario recuperare dei dati già presenti all'interno del database.

In input non prende alcun parametro.

In output viene ritornata una NextResponse contenente lo stato della risposta o dell'errore e il relativo messaggio.

- mountainId:

questa API viene utilizzata per ritornare le informazioni di una montagna in modo da mostrarle sul sito.

Si tratta di una richiesta di tipo GET.

Riceve in input un solo parametro, tramite path, ossia l'id della montagna da trovare.

In output viene ritornata una NextResponse contenente lo stato della risposta o dell'errore e il relativo messaggio.

- search:

questa API viene utilizzata per effettuare una ricerca sul nome delle montagne presenti all'interno del database, che corrispondano parzialmente o in modo totale alla stringa inserita.

Si tratta di una richiesta di tipo GET perchè è necessario recuperare dei dati già presenti all'interno del database.

Come parametro in input riceve la stringa di ricerca inserita da parte dell'utente che verrà confrontata con i nomi delle montagne presenti all'interno del database.

In output viene ritornata una NextResponse contenente lo stato della risposta o dell'errore e il relativo messaggio.

5.4.3 API modello Refuges

Verranno descritte di seguito le API utilizzate per una montagna:

- mountainId:

questa API viene utilizzata per recuperare tutti i rifugi di una montagna presenti all'interno del database. Nel caso la montagna sia 0, vengono recuperati tutti i rifugi all'interno del database

Si tratta di una richiesta di tipo GET perchè è necessario recuperare dei dati già presenti all'interno del database.

In output viene ritornata una NextResponse contenente lo stato della risposta o

dell'errore e il relativo messaggio.

- addRefuge:

questa API viene utilizzata per permettere ad un utente loggato all'interno del sito di aggiungere un rifugio al database. Prima di inserirlo, controlla se un rifugio con gli stessi parametri esiste già.

Si tratta di una richiesta di tipo POST essendo che ha bisogno di inviare dei dati al database per confrontarli con quelli già presenti.

Come parametro in input riceve il rifugio da inserire con i parametri inseriti dall'utente nell'apposito form per l'aggiunta.

In output viene ritornata una NextResponse contenente lo stato della risposta o dell'errore e il relativo messaggio.

- delete:

questa API viene utilizzata per permettere all'utente admin di eliminare un rifugio nel caso non rispetti alcune delle norme citate all'interno del documento dei requisiti.

Si tratta di una richiesta di tipo DELETE perchè è necessario eliminare dei dati

presenti all'interno del database.

Come parametro in input riceve l'id del rifugio da eliminare dal database.

In output viene ritornata una NextResponse contenente lo stato della risposta o dell'errore e il relativo messaggio.

- search:

questa API viene utilizzata per effettuare una ricerca sul nome dei rifugi presenti all'interno del database, che corrispondano parzialmente o in modo totale alla stringa inserita.

Si tratta di una richiesta di tipo GET perchè è necessario recuperare dei dati già presenti all'interno del database.

Come parametro in input riceve la stringa di ricerca inserita da parte dell'utente che verrà confrontata con i nomi di rifugi presenti all'interno del database.

In output viene ritornata una NextResponse contenente lo stato della risposta o dell'errore e il relativo messaggio.

5.5 API documentation - Swagger

Le API sviluppate per l'applicazione e descritte nel dettaglio precedentemente sono state documentate utilizzando Swagger. Swagger permette di avere una paronamica di tutte le API e di avere più informazioni su di esse. Il link per poter visitare la documentazione delle API è il seguente:

<https://mountain-wonders.vercel.app/api-docs>

Le API presenti per i rifugi sono le seguenti:

The screenshot shows the Swagger UI for the 'refuges' endpoint. It lists four operations: GET /refuges/{mountainId} (Find refuges by mountain), DELETE /refuges/delete/{refugoid} (Delete a refuge), POST /refuges/addRefuge/{mountainId} (Add a refuge), and GET /refuges/search/{mountainId}/{searchString} (Find refuges by name). Each operation is shown with its method, URL, and a brief description.

Figura 14: API rifugi

Le API presenti per le montagne:

The screenshot shows the Swagger UI for the 'mountains' endpoint. It lists three operations: GET /mountains/{mountainId} (Find mountain by ID), GET /mountains (Find all mountains), and GET /mountains/search/{searchString} (Find mountains by name). Each operation is shown with its method, URL, and a brief description.

Figura 15: API montagne

Le API presenti per l'autenticazione

The screenshot shows the Swagger UI for the 'auth' endpoint. It lists five operations: GET /auth/currentUser (Retrieve the currentUser), POST /auth/login (Login), POST /auth/register (Register), and GET /auth/logout (Logout). Each operation is shown with its method, URL, and a brief description.

Figura 16: API autenticazione

E' presente una breve descrizione per ogni API, e una volta selezionata, è possibile visualizzare più dettagli, come il tipo di API, i parametri presenti e tutte le possibili risposte.

The screenshot shows a REST API documentation interface. At the top, a blue header bar displays the method **GET** and the endpoint **/refuges/{mountainId}**. To the right of the endpoint, the description "Find refuges by mountain" is visible. Below the header, there is a section titled "Parameters" which lists a single parameter: **mountainId** (required, integer type). A "Try it out" button is located in the top right corner of this section. The next section is "Responses", which contains two entries: a successful response (Code 200) and an error response (Code 404). The 200 response is described as "successful operation" and includes a "Media type" dropdown set to "application/json". A note below the dropdown says "Controls Accept header.". An "Example Value" button is available, and the schema is shown as a JSON array:

```
[ { "name": "string", "avgRating": "string", "description": "string", "mountainId": 0, "image": "string", "_v": 1 } ]
```

. The 404 response is described as "No refuges found". Both responses have a "Links" column which is currently empty, indicated by the text "No links".

Figura 17: Esempio API

5.6 Pagine frontend

Di seguito andremo a descrivere in modo dettagliato ciascuna pagina frontend del sito sviluppata dal team e le operazioni possibili per ogni tipologia di utente che visita il sito.

5.6.1 Home page

Questa è la pagina in cui verranno reindirizzati gli utenti che vogliono visitare il sito.



Figura 18: Home page

Le possibili operazioni per ciascuna tipologia di utente sono visualizzare tutti i rifugi presenti oppure le montagne presenti all'interno del database, utilizzando l'apposito bottone.

In alto è presente una navbar, comune ad ogni pagina del sito. I bottoni che possono essere cliccati sono:

- home
- profile
- logout

Qualora venga cliccato il bottone home, l'utente (sia anonimo, che registrato, che amministratore) viene reinidirizzato alla home page.

Nel caso in cui venga invece cliccato il bottone Profile:

- se il bottone viene cliccato da un utente registrato all'interno del sito, esso verrà ridirezionato alla home page

- se il bottone viene cliccato da un utente anonimo, esso verrà ridirezionato alla pagina di login.

Il bottone logout viene mostrato solamente qualora un utente abbia effettuato l'accesso al sito: premento quindi questo bottone, l'applicazione provvederà ad eliminare i cookie salvati nel browser e disconnettere l'utente.

5.6.2 Pagina login

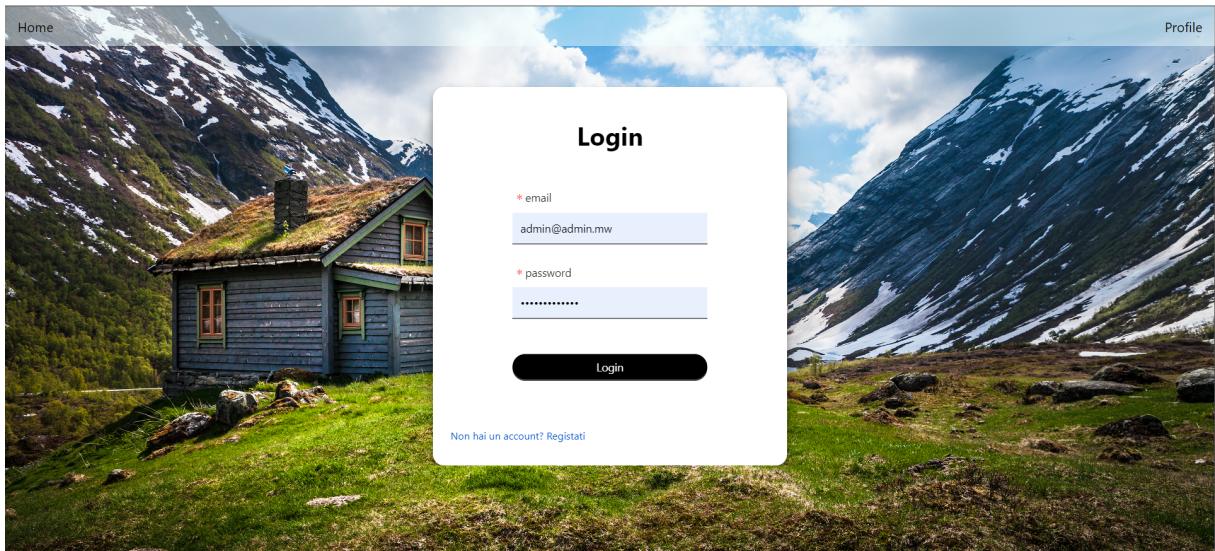


Figura 19: Pagina login

Questa pagina permette ad un utente anonimo di loggarsi all'interno del sito per effettuare operazioni concesse solamente agli utenti registrati.

Nel caso in cui l'utente abbia già precedentemente creato un account all'interno del database, sarà sufficiente inserire la mail e la password utilizzate durante la registrazione per poi collegarsi.

Il database provvederà di conseguenza a verificare le credenziali con quelle presenti all'interno del database e a mostrare all'utente il messaggio nel caso il login sia stato eseguito correttamente o meno.

5.6.3 Pagina registrazione

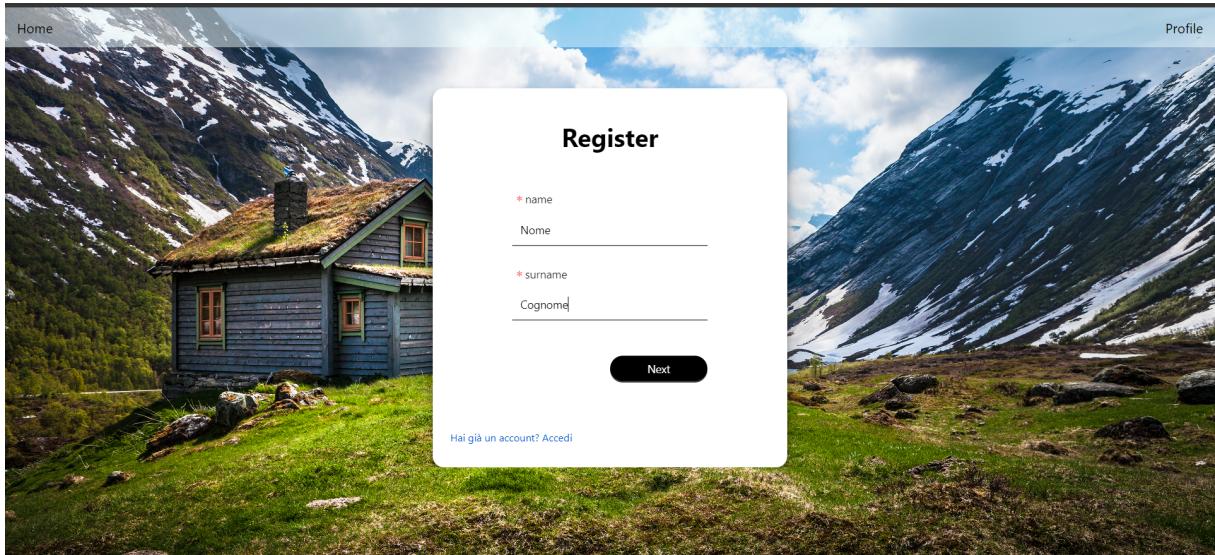


Figura 20: Pagina registrazione 1

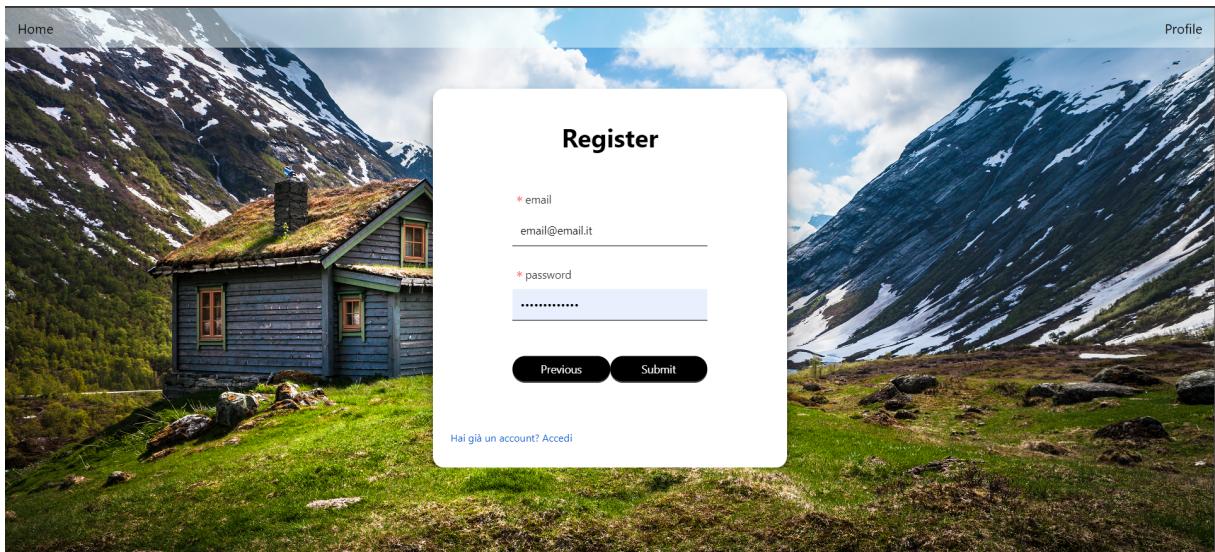


Figura 21: Pagina registrazione 2

Questa pagina permette ad un utente anonimo di loggarsi all'interno del sito per effettuare operazioni concesse solamente agli utenti registrati.
Nel caso in cui l'utente non abbia ancora creato un account all'interno del database, dovrà creare uno inserendo gli appositi dati all'interno del form di registrazione.
Il database provvederà quindi a verificare che non sia già presente un account con la stessa mail e a mostrare all'utente il messaggio nel caso la registrazione sia stata eseguita correttamente o meno.

5.6.4 Pagina montagne

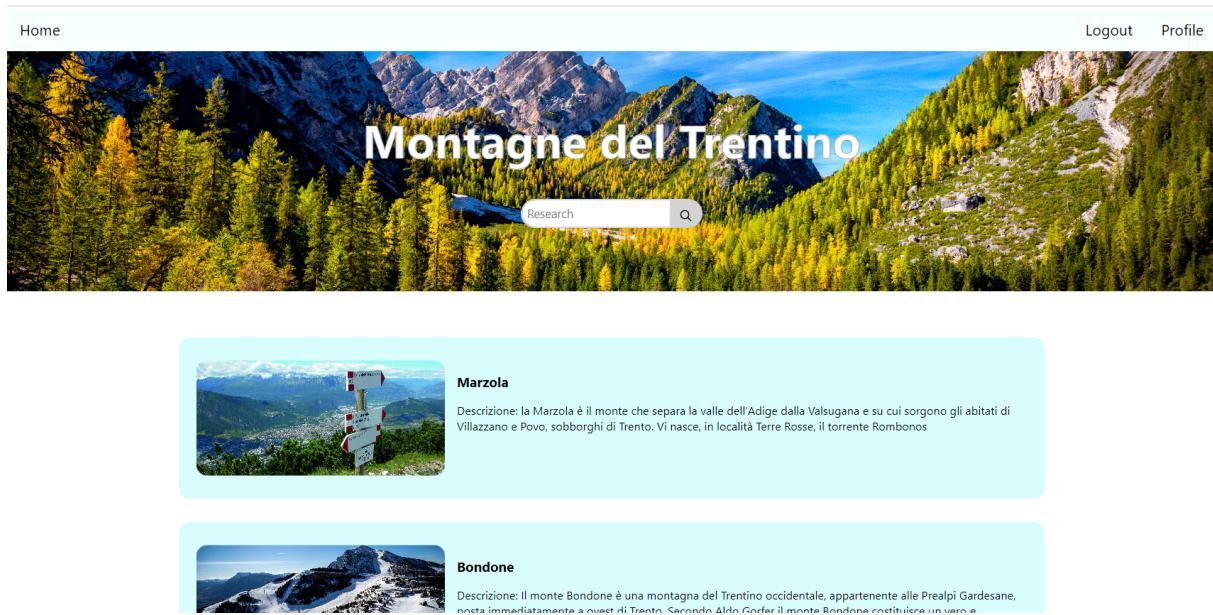


Figura 22: Pagina montagne

Questa pagina mostra a qualsiasi utente che visita il sito, l’elenco delle montagne presenti all’interno del database.

Dal momento che in questa versione nè utenti registrati nè admin nè utenti anonimi possono aggiungere una montagna, la visualizzazione della pagina sarà uguale per tutti gli utenti. Le operazioni possibili all’interno della pagina sono:

- ricerca di una montagna
- selezione di una montagna

Quando si ricerca una montagna, è possibile inserire all’interno della barra di ricerca una qualsiasi stringa: se la stringa è contenuta in almeno uno dei nomi delle montagne, la ricerca mostrerà queste montagne, con i relativi dettagli.

Altrimenti se non sono presenti montagne il cui nome contiene la sottostringa inserita dall’utente, verranno mostrate tutte le montagne presenti nel database e verrà mostrato un errore che indica che non sono state trovate montagne.

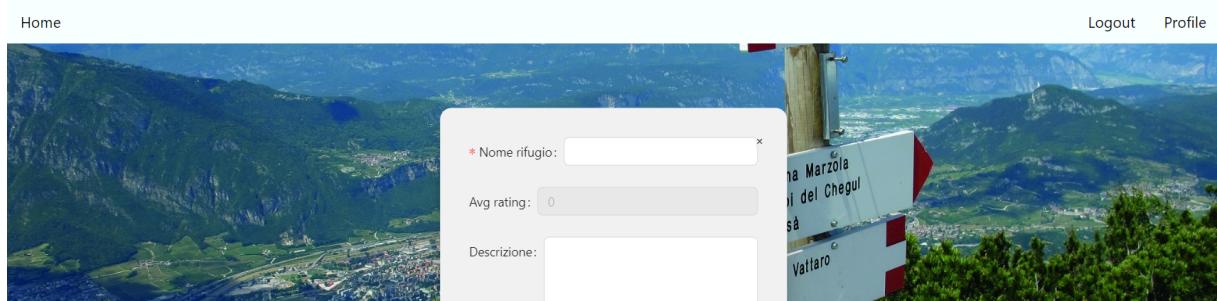
Se invece l’utente seleziona una montagna, viene ridirezionato alla pagina seguente.

5.6.5 Pagina rifugi di una montagna



The screenshot shows a search interface with a placeholder 'Research' and a magnifying glass icon. Below it are three filter buttons labeled 'Filtro1', 'Filtro2', and 'Filtro3'. To the right is a directional signpost with text: 'Marzola', 'Cima Marzola', 'Stoi del Chegul', 'Susà', and 'Vigolo Valtaro'. Below the signpost, there are two cards: one for 'Rifugio da Giuseppe' (described as the best mountain refuge) and another for 'Ciao'.

Figura 23: Pagina rifugi di una montagna



A modal window is open for adding a new refuge. It contains fields for 'Nome rifugio' (Name of refuge), 'Avg rating' (Average rating), 'Descrizione' (Description), 'Mountain' (Mountain), 'Immagine' (Image), and 'Valutazione' (Evaluation). A 'Valutazione' section shows a 5-star rating. At the bottom is a large black 'Aggiungi' (Add) button. The background shows a view of the Marzola mountain range.

Figura 24: Aggiunta rifugio in una montagna

Questa pagina permette a qualsiasi utente di visualizzare tutti i rifugi di una montagna e i relativi dettagli.

Per gli utenti anonimi che visitano il sito, l'unica operazione che possono eseguire è la ricerca di un rifugio tra quelli presenti all'interno della montagna selezionata in precedenza.

E' possibile inserire all'interno della barra di ricerca una qualsiasi stringa: se la stringa è contenuta in almeno uno dei nomi dei rifugi, la ricerca mostrerà queste montagne con i relativi dettagli.

Altrimenti se non sono presenti rifugi il cui nome contiene la sottostringa inserita dall'utente, verranno mostrati tutti i rifugi della montagna selezionata e verrà mostrato un errore che indica che non sono stati trovati rifugi.

Nel caso in cui l'utente sia amministratore, oltre alla ricerca avrà anche il permesso per poter eliminare un rifugio qualora non rispettasse alcune delle norme citate all'interno del documento dei requisiti: dopo aver cliccato sul bottone per l'eliminazione di un rifugio, nella pagina verrà mostrato il messaggio di risposta, nel caso l'operazione sia andata a buon fine o meno.

Nel caso in cui l'utente sia un utente registrato, oltre alla ricerca avrà anche il permesso per poter aggiungere un rifugio all'interno del database: dopo aver cliccato sul bottone per l'aggiunta di un rifugio, nella pagina verrà mostrato un form in cui l'utente dovrà inserire i dettagli del nuovo rifugio.

Dopo aver cliccato il bottone "aggiungi" del form, verrà mostrato il messaggio di risposta, nel caso l'operazione sia andata a buon fine o meno.

5.6.6 Pagina rifugi

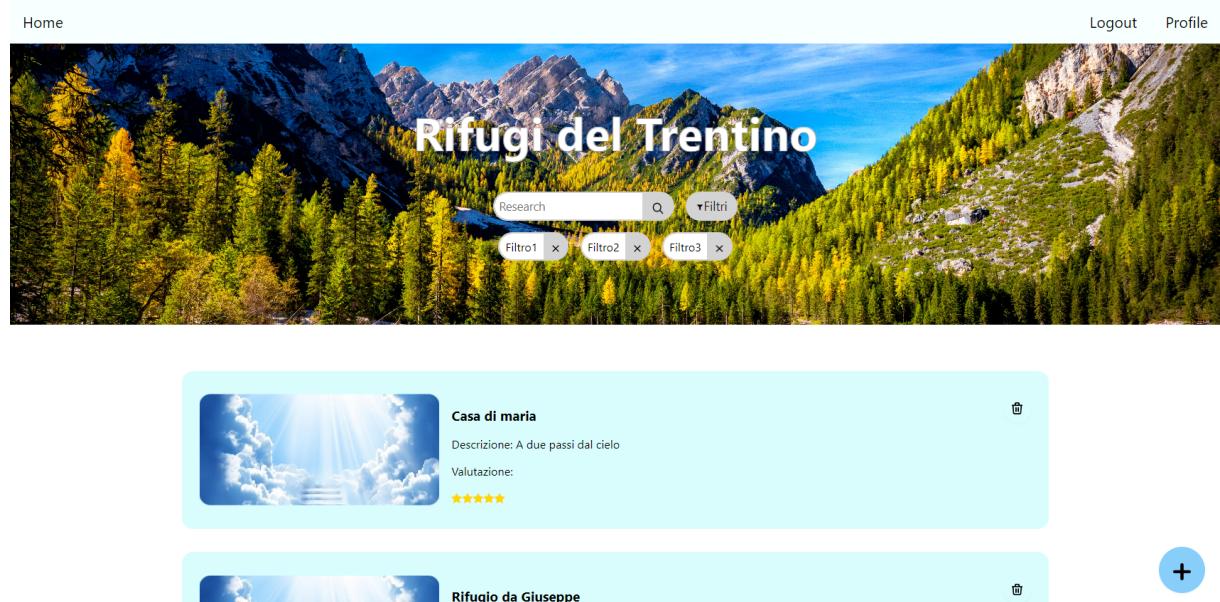


Figura 25: Pagina rifugi

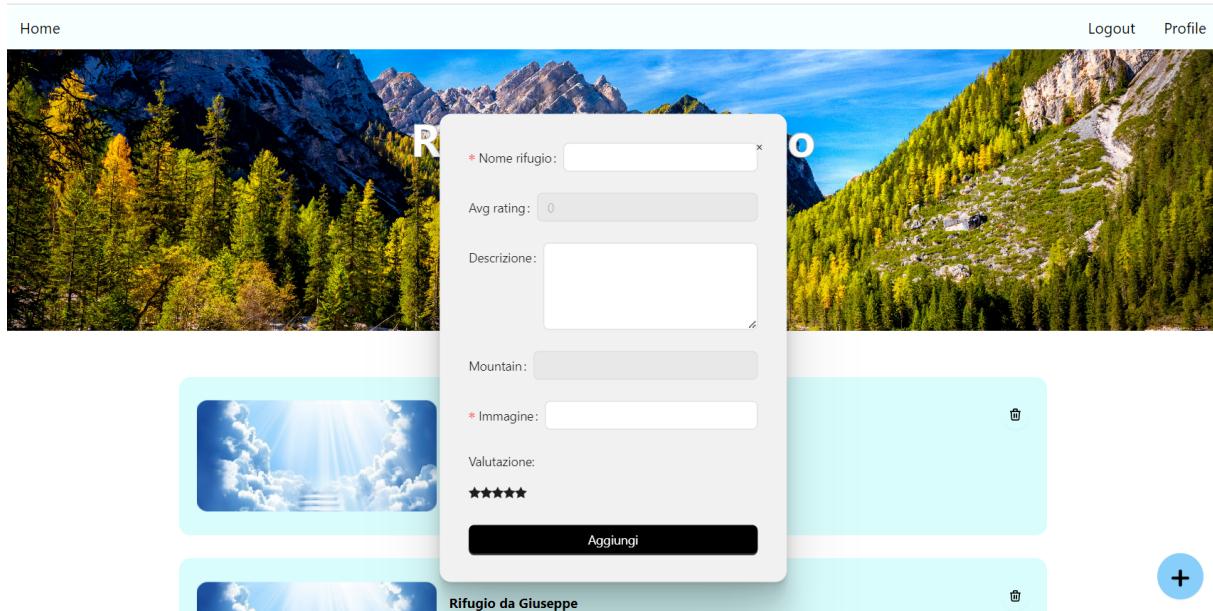


Figura 26: Aggiunta rifugio alla lista

Questa pagina permette a qualsiasi utente di visualizzare tutti i rifugi presenti all'interno del database e i relativi dettagli.

Per gli utenti anonimi che visitano il sito, l'unica operazione che possono eseguire è la ricerca di un rifugio tra quelli presenti all'interno del database.

E' possibile inserire all'interno della barra di ricerca una qualsiasi stringa: se la stringa è contenuta in almeno uno dei nomi dei rifugi, la ricerca mostrerà questi rifugi con i relativi dettagli. Altrimenti se non sono presenti rifugi il cui nome contiene la sottostringa inserita dall'utente, verranno mostrati tutti i rifugi della montagna selezionata e verrà mostrato un errore che indica che non sono stati trovati rifugi.

Nel caso in cui l'utente sia amministratore, anche in questo caso avrà il permesso per poter eliminare un rifugio.

Nel caso in cui l'utente sia un utente registrato, oltre alla ricerca avrà anche il permesso per poter aggiungere un rifugio all'interno del database, come descritto in precedenza.

5.7 Testing

Per condurre le operazioni di testing, abbiamo utilizzato la libreria Jest in combinazione con il metodo fetch per verificare e chiamare le API. Tutti i test svolti sono accuratamente organizzati nella directory 'test/[nomeCartellaDaTestare]', fornendo una struttura chiara e ordinata.

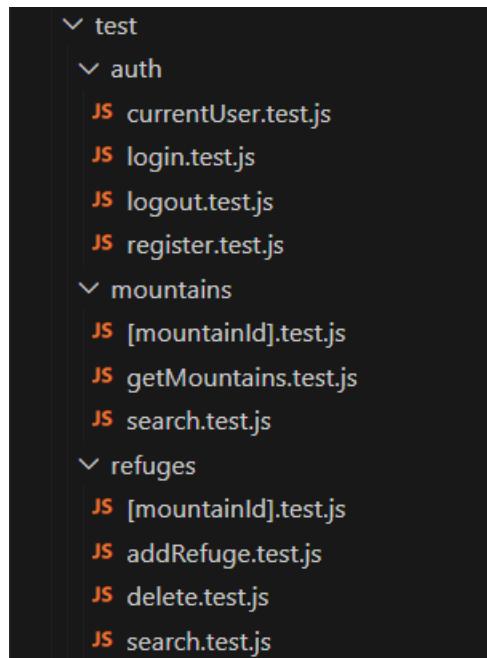


Figura 27: Cartella test

All'interno di questa directory sono presenti tre cartelle che contengono complessivamente undici file con estensione ".test.js", i quali rappresentano i file dedicati all'esecuzione dei test per il nostro sistema. Per ciascun modello che dispone di API definite, esiste un file specifico in cui sono dettagliati tutti i casi di test corrispondenti a tali API.

Analizziamo la tipica struttura di un file .test.js dedicato al testing delle API, prendendo il file login.test.js come esempio. Tale struttura è uniforme tra tutti i file di test.

```

app > test > auth > JS login.test.js > ...
1  const { describe, before } = require("node:test");
2  const url = "http://localhost:3000/api/auth/login/";
3  const correctPSW = "accountprova"; // String not in the database, empty response expected
4  const wrongPSW = "accountprovolta"; // String in the database, non-empty response expected (at least one result)
5  const mongoose = require('mongoose');
6  require("dotenv").config();
7
8  describe("POST /api/auth/login", () => {
9    beforeAll(async () => {
10      const timeout = 10000;
11      // Promise to connect to MongoDB
12      const connectPromise = new Promise((resolve, reject) => {
13        const timeoutId = setTimeout(() => {
14          reject(new Error(`Timed out after ${timeout}ms while connecting to MongoDB`));
15        }, timeout);
16        mongoose.connect(process.env.ATLAS_URI).then(() => {
17          clearTimeout(timeoutId);
18          resolve();
19        }).catch((error) => {
20          clearTimeout(timeoutId);
21          reject(error);
22        });
23      });
24    });
25    afterAll(async () => {
26      await mongoose.connection.close(true);
27    });
28    test("POST login with correct credentials", async () => {
29      const res = await fetch(url, {
30        method: 'POST',
31        body: JSON.stringify({
32          email: 'account@prova.it',
33          password: correctPSW}),
34      });
35      expect(await res.json()).status.toEqual(200);
36    });
37  });

```

Figura 28: Struttura file test

Nel file .test.js, iniziamo importando i moduli per il testing delle API, tra il link del server in locale e mongoose per stabilire una connessione al database.

All'interno del metodo `beforeAll()`, definiamo le operazioni da eseguire all'inizio dell'esecuzione del codice, ovvero la connessione al database con un timeout di 10 secondi.

Il metodo `afterAll()` viene chiamato alla fine del file e si occupa di chiudere la connessione al database. Questi due metodi sono presenti in ogni file di test, consentendo la connessione al database per chiamare effettivamente le varie API.

Ogni API è testata all'interno della corrispondente suite. Viene utilizzato un metodo `describe()` per ogni API, all'interno del quale sono presenti diversi metodi `test()` che implementano i casi di test effettivi. Ogni API è testata più volte, verificando tutti i possibili status code di ritorno, alcuni anche più volte.

5.7.1 Risultati del testing

Per eseguire i test, abbiamo prima incluso il seguente script nel file package.json:

```
"test": "jest -coverage"
```

Così facendo eseguendo il comando npm test dalla directory principale del progetto, verranno eseguiti tutti i file .test.js che abbiamo definito.

Di seguito i risultati dei test:

```
Test Suites: 11 passed, 11 total
Tests:       31 passed, 31 total
Snapshots:   0 total
Time:        12.519 s
Ran all test suites.
```

Figura 29: Test

Tutte le test suites sono state eseguite e tutti i 31 casi di test (definiti dai metodi test()) risultano passati.

Non è stato possibile effettuare il coverage dell'applicazione a causa di un problema di jest. Avendo jest un contesto di esecuzione diverso rispetto a quello delle API non è possibile effettuare un coverage.

5.8 GitHub repository e informazioni sul deployment

Al seguente URL è presente la repository del progetto

<https://github.com/MountainWonders-G24/code>

Il deployment del progetto è stato effettuato su vercel (<https://vercel.com/>) al seguente link:

<https://mountain-wonders.vercel.app/>

Nel caso si voglia eseguire il progetto in locale basterà scaricare la cartella nel link citato sopra, in seguito si dovrà creare il file .env nella directory principale che deve essere formattato come segue:

```
*****  
ATLAS_URI =  
jwt_secret = *chiave segreta*  
*****
```

In seguito basterà eseguire il comando npm install nella directory principale del progetto per scaricare tutti i moduli utilizzati. Successivamente, eseguire il comando npm start nella directory principale del progetto. Non appena verranno visualizzate le seguenti righe sulla console:

```
*****  
> mountainwonders@0.1.0 dev  
> next dev  
*****
```

Next.js 14.0.3

- Local: <http://localhost:3000>
- Environments: .env

Ready in 4.3s

```
*****  
Ora collegandosi all'URL http://localhost:3000/ si riuscirà a navigare nel sito.  
*****
```

5.8.1 Nota per la documentazione

La documentazione delle API sarà disponibile a <http://localhost:3000/api-docs> .