



PROGETTO DI INGEGNERIA DEL SOFTWARE

SKI ONLINE

Sviluppo applicazione

Indice dei contenuti

INDICE DEI CONTENUTI	2
SCOPO DEL DOCUMENTO	3
USER FLOWS	3
UTENTE ANONIMO	3
UTENTE REGISTRATO	4
UTENTE OPERATORE/GESTORE.....	4
APPLICATION IMPLEMENTATION AND DOCUMENTATION	6
PROJECT STRUCTURE	6
PROJECT DEPENDENCIES.....	6
PROJECT DATA OR DB.....	7
PROJECT APIS	8
Resources extraction from the class diagram.....	8
Resources models.....	9
SVILUPPO API	12
Dati di un utente.....	12
Creazione nuovo utente	13
Rimozione di un utente	13
Autenticazione di un utente	14
Modifica della password di un utente	14
Elenco degli impianti memorizzati.....	15
Modifica dello stato di apertura di un impianto	15
Elenco dei log di accesso di un utente	15
Creazione di un log di accesso.....	16
Numero di utenti con almeno un log nell'ultima ora.....	16
Elenco delle percentuali di occupazione di impianto nell'ultima mezz'ora	16
API DOCUMENTATION	17
FRONT-END IMPLEMENTATION	18
HOME PAGE.....	18
STATO IMPIANTI.....	19
PROFILO	20
GESTIONE IMPIANTI.....	21
GITHUB REPOSITORY AND DEPLOYMENT INFO	22
TESTING	22

Scopo del documento

Il presente documento riporta tutte le informazioni necessarie per lo sviluppo di una parte dell'applicazione Ski Online. In particolare, presenta tutti gli artefatti necessari per realizzare i servizi di gestione *degli utenti e degli impianti di risalita dell'applicazione Ski Online*. Partendo dalla descrizione degli user flow legate *alle azioni eseguibili dagli utenti anonimi, dagli utenti registrati e dagli utenti di sistema (operatore e gestore)*, il documento prosegue con la presentazione delle API necessarie (tramite l'API Model e il Modello delle risorse) per *effettuare il login, registrazione, cancellazione di un account, visualizzare lo stato di affollamento degli impianti e lo storico dei log degli accessi agli stessi* necessari all'applicazione Ski Online.

Per ogni API realizzata, oltre ad una descrizione delle funzionalità fornite, il documento presenta la sua documentazione e i test effettuati. Infine una sezione è dedicata alle informazioni del Git Repository e il deployment dell'applicazione stessa.

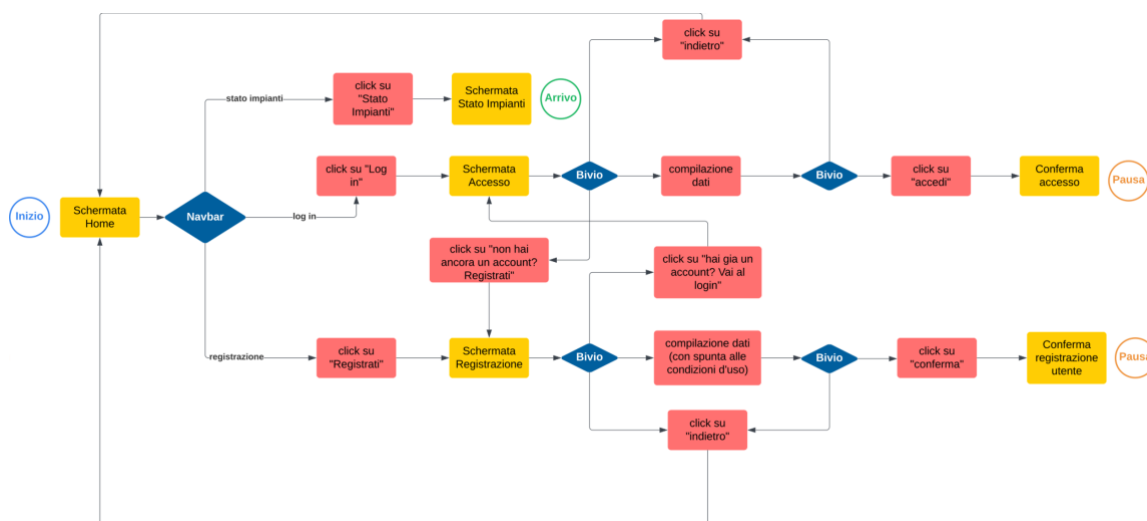
User flows

In questa sezione del documento di sviluppo riportiamo gli "user flows" per il ruolo sia dell'utente registrato che per quello anonimo.

La prima figura descrive gli user flows relativi alle operazioni di log-in e registrazione da parte dell'utente anonimo. Diversamente, la seconda si riferisce all'utente registrato e alle azioni che può effettuare a partire dalla "Schermata Utente" a lui dedicata; mentre, la terza descrive lo stato di apertura degli impianti gestita dall'Utente Gestore o Operatore.

Utente Anonimo

Nel seguente User Flow l'Utente Anonimo, a meno di selezionare la voce "indietro", tramite la corretta compilazione dei campi richiesti può proseguire con il log-in oppure la registrazione di un nuovo utente.

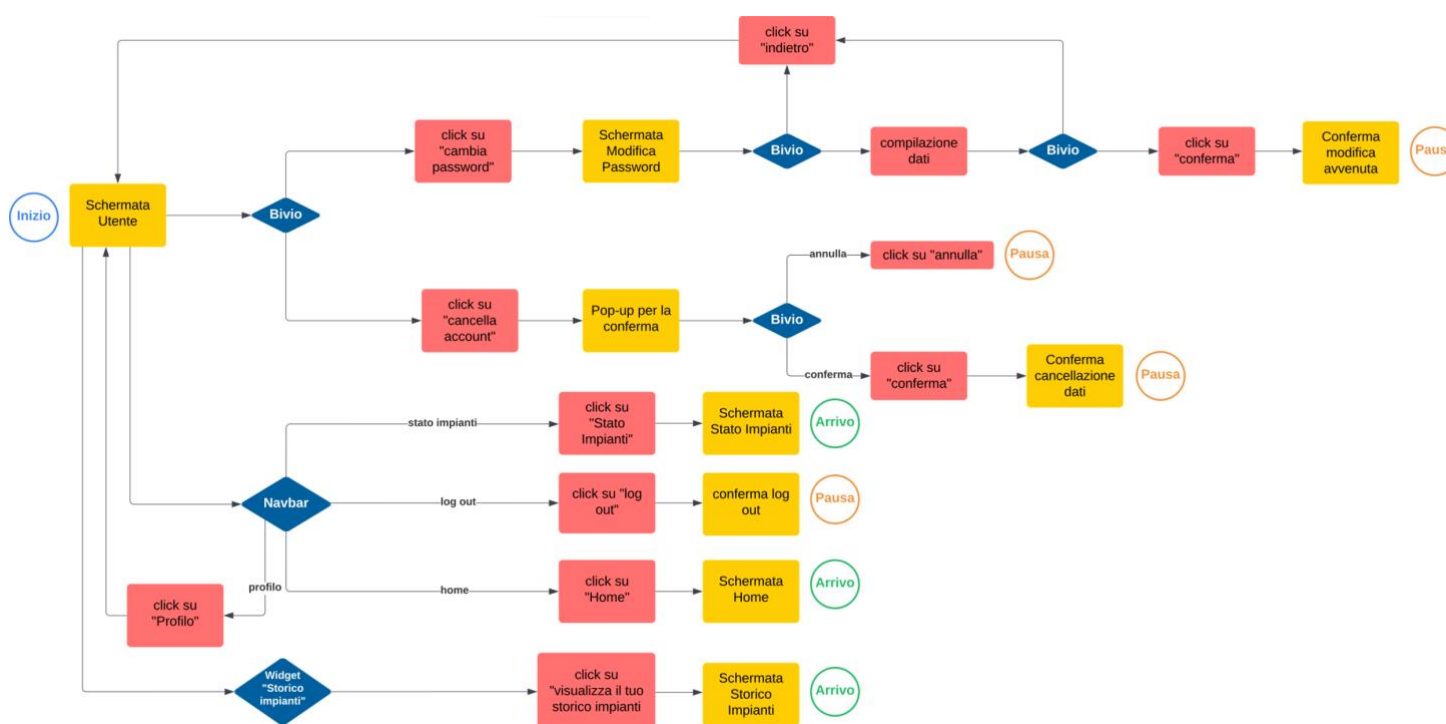


Utente Registrato

Nel seguente User Flow sono descritte le principali funzionalità a cui può accedere, dalla schermata dedicata alla gestione dell'account, un utente di livello "Registrato", "Gestore" oppure "Operatore".

Tramite due bottoni "Cancella Account" e "Cambia Password" può, come suggeriscono i nomi, eliminare definitivamente l'utenza oppure modificare la propria password.

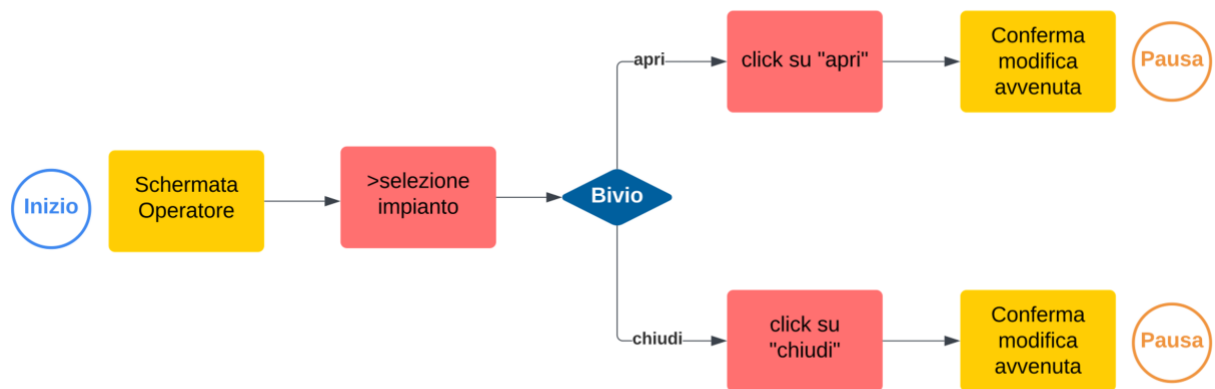
Tramite una navbar. In aggiunta, da quest'ultima è possibile effettuare il log-out ed essere reindirizzati alla pagina "Home".



Utente Operatore/Gestore

Nel seguente User Flow è descritta una delle funzionalità principali dell'Utente di livello "Operatore" e "Gestore" a cui solo loro possono accedere.

Tramite una schermata apposita l'utente, in seguito alla selezione di un impianto, può modificarne lo stato di apertura o chiusura.

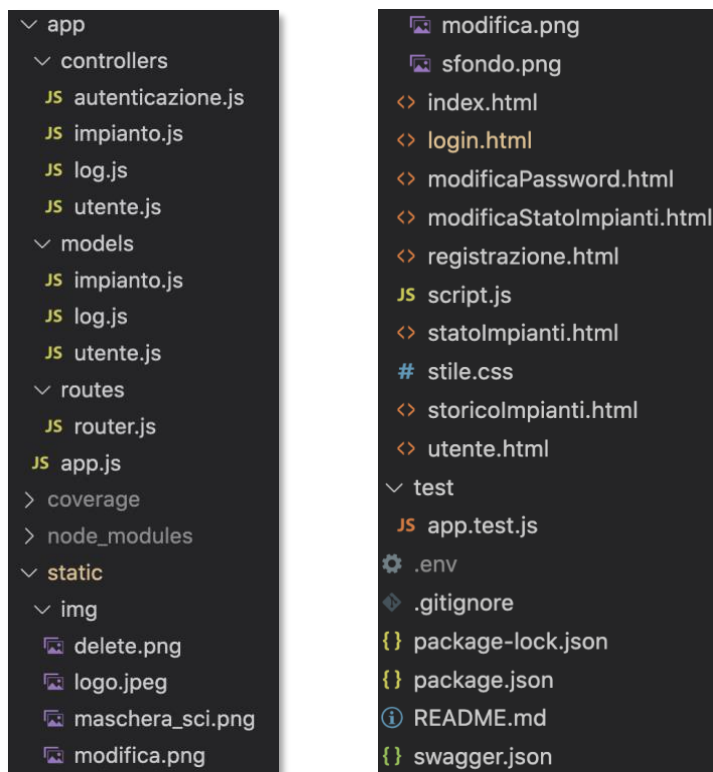


Application Implementation and Documentation

Project Structure

La struttura del progetto è presentata in figura ed è composta di:

- una cartella app per la gestione delle API locali;
- una cartella static per la parte del front-end;
- una cartella test per il testing del progetto;
- dei file di sistema come, ad esempio, i file .env, .gitignore e package.json e file swagger.json per la documentazione delle API che verrà discussa in seguito.



Project Dependencies

I seguenti moduli Node sono stati utilizzati e aggiunti al file package.json

- Bcrypt
- Dotenv
- Express
- Jsonwebtoken
- Mongoose
- Mongoose-express-api
- Swagger-ui-express

Inoltre, per quanto riguarda la fase di development del progetto, sono stati utilizzati i seguenti moduli:

- Jest
- Supertest

Project Data or DB

Per la gestione dei dati utili all'applicazione abbiamo definito tre principali strutture dati come illustrato in figura:

- una collezione di "Utenti";
- una collezione di "Impianti";
- una collezione di "Log" che lega il passaggio di un Utente per un Impianto.

LOGICAL DATA SIZE: 2.86KB		STORAGE SIZE: 108KB		INDEX SIZE: 108KB		TOTAL COLLECTIONS: 3		CREATE COLLECTION
Collection Name	Documents	Logical Data Size	Avg Document Size	Storage Size	Indexes	Index Size	Avg Index Size	
impiantos	3	256B	86B	36KB	1	36KB	36KB	
logs	16	1.97KB	126B	36KB	1	36KB	36KB	
utentes	3	658B	220B	36KB	1	36KB	36KB	

Per rappresentare gli Utenti, Impianti e Log abbiamo definito i seguenti tipi di dati di esempio:

```
_id: ObjectId('639c64292f351b13f99e0137')
nome: "Mario"
cognome: "Rossi"
email: "mariorossi@gmail.com"
password: "$2b$05$K1za2US7NwGZ0CwAIXrvl0.r2Eqy7.fzgD6V5gSiFS0taQjUY3xrG"
livelloUtente: "Base"
__v: 0
```

```
_id: ObjectId('63a4de9eec8ee30dee706855')
nome: "Piz Boe"
portataOraria: 1500
statoApertura: false
__v: 0
```

```
_id: ObjectId('63a44607994b249035e12c57')
impianto: "63987e3b498173d611a5c667"
utente: "63972a8231b4484c9fe6eb97"
timestamp: 1671710215700
__v: 0
```

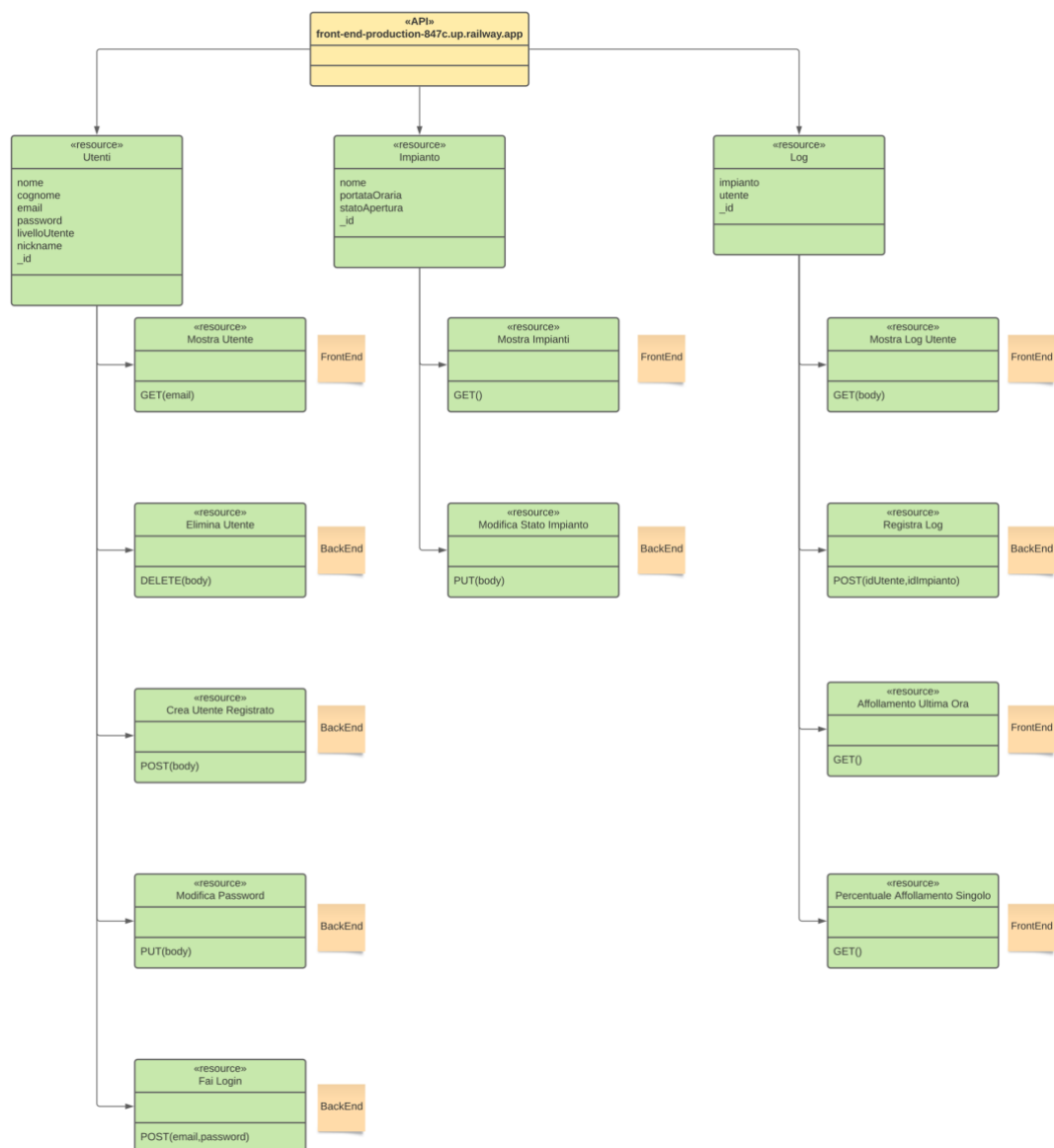
Project APIs

Resources extraction from the class diagram

Nel seguente digramma sono presenti le risorse estratte dal Class diagram realizzato nel documento precedente.

A partire dal root del diagramma, dove è presente l'URL del nostro sito front-end-production-847c.up.railway.app/, si diramano tutte le relazioni che gestiamo a livello di API, ovvero le seguenti tre risorse:

- Utenti;
- Impianto;
- Log.



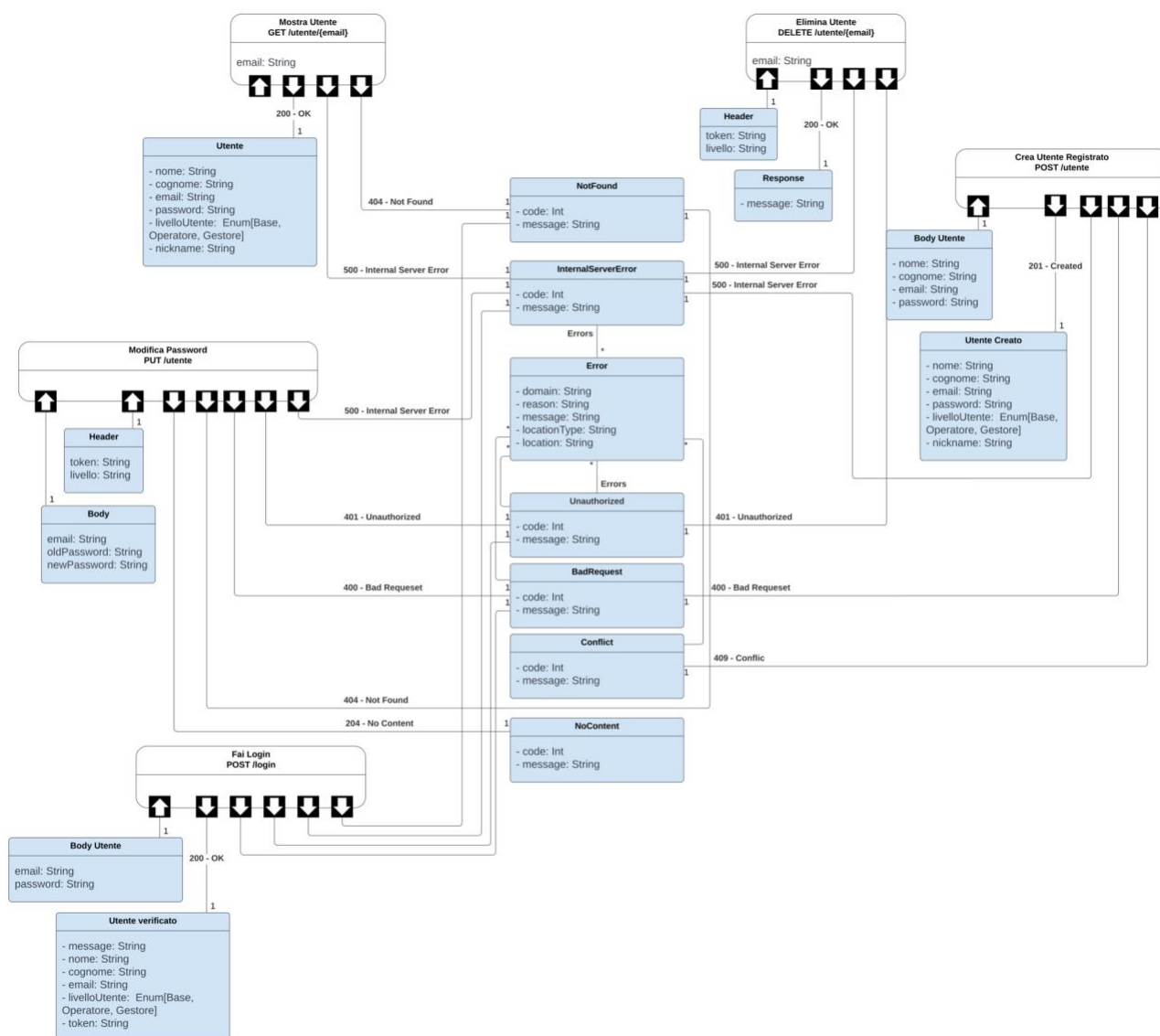
Nota bene: il parametro `_id` indicato nelle risorse si riferisce all'identificativo univoco generato da MongoDB all'atto di registrazione dei dati

Resources models

Basandoci sulla sezione precedente sono stati realizzati quattro Resources Models suddivisi per risorsa.

Utenti

Nel seguente diagramma sono presenti le risorse precedentemente descritte per “Utenti” attraverso l’API resource. Ognuna, oltre al nome e al metodo, presenta anche l’URI tramite il quale è possibile accedere al servizio specifico.

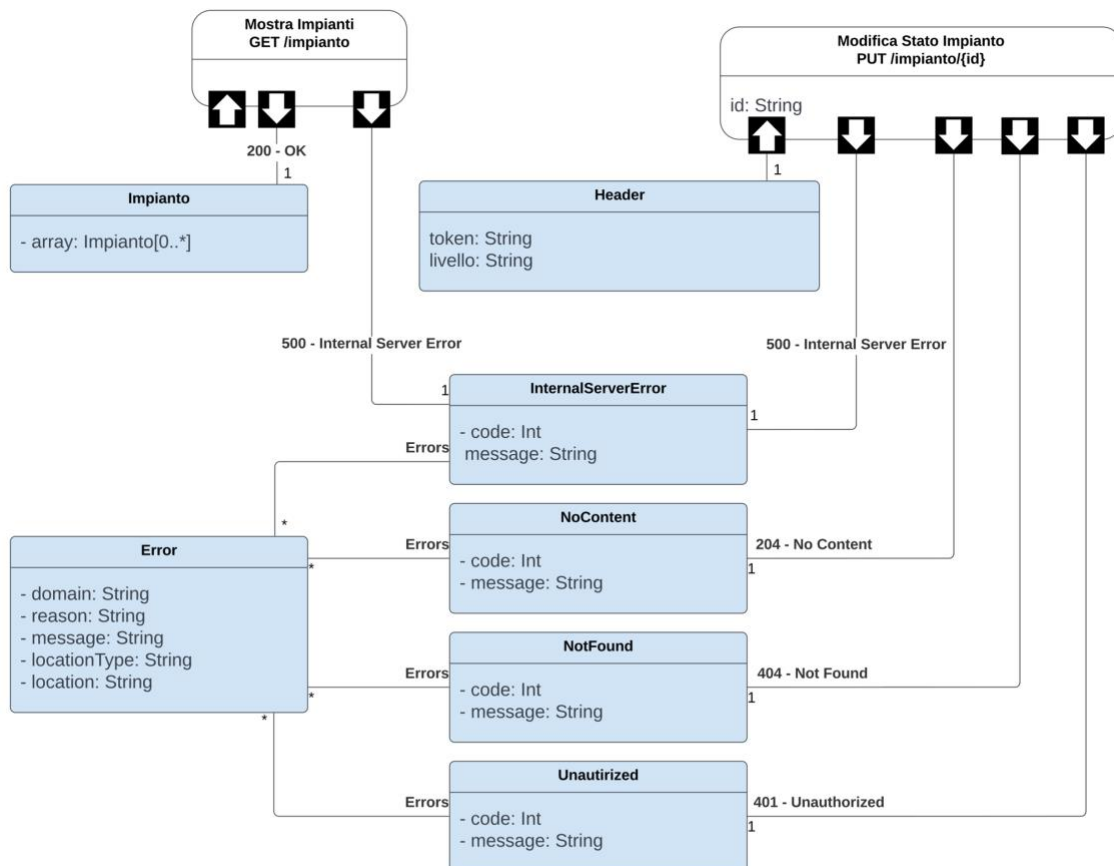


In questo caso abbiamo:

Mostra Utente	GET	/utente/{email}
Elimina Utente	DELETE	/utente/{email}
Crea Utente Registrato	POST	/utente
Modifica Password	PUT	/utente
Fai Login	POST	/login

Impianti

Nel seguente diagramma sono presenti le risorse precedentemente descritte per “Impianti” attraverso l’API resource. Ognuna, oltre al nome e al metodo, presenta anche l’URI tramite il quale è possibile accedere al servizio specifico.

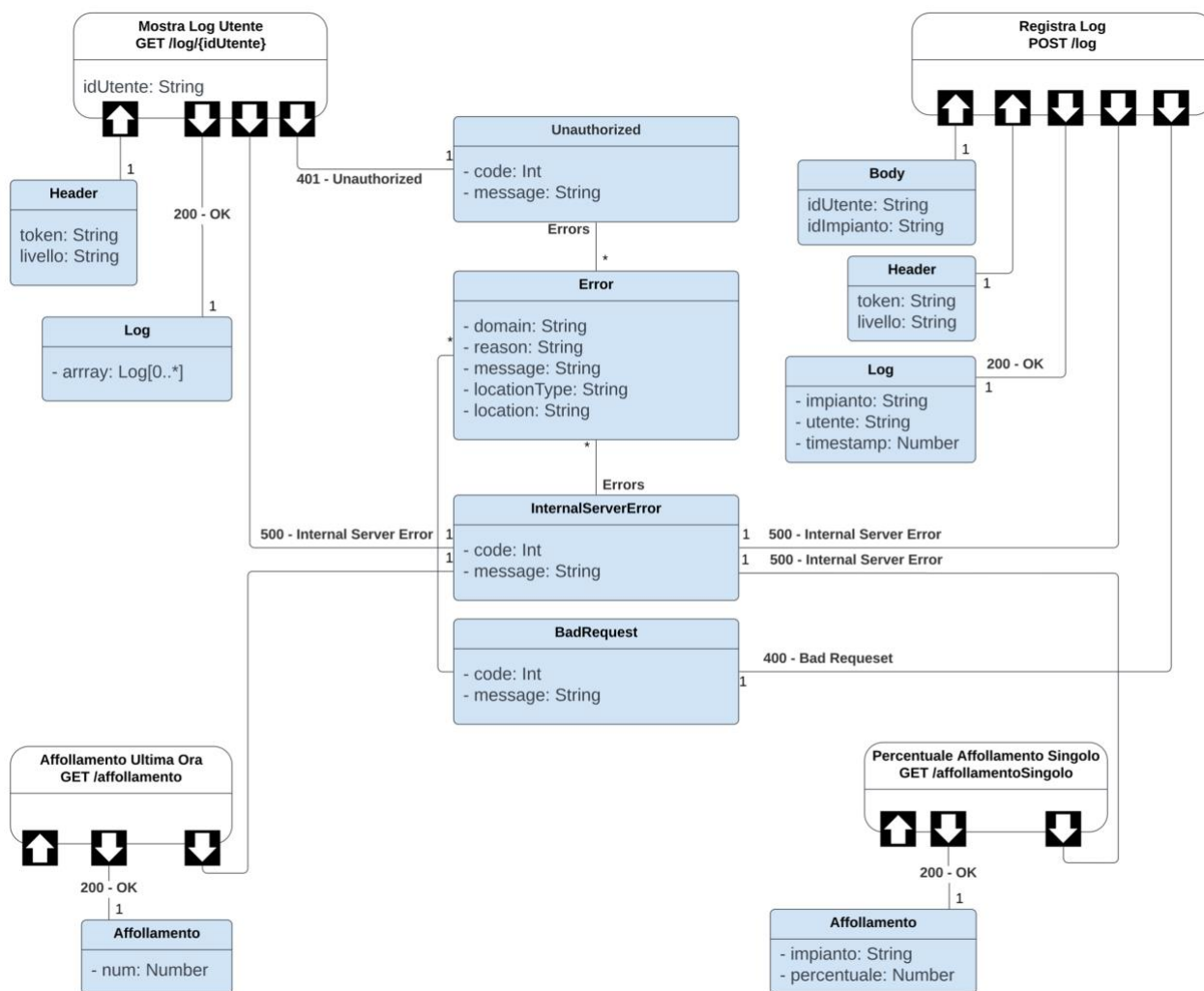


In questo caso abbiamo:

Mostra Impianti	GET	/impianto
Modifica Stato Impianto	PUT	/impianto/{id}

Log

Nel seguente diagramma sono presenti le risorse precedentemente descritte per “Log” attraverso l’API resource. Ognuna, oltre al nome e al metodo, presenta anche l’URI tramite il quale è possibile accedere al servizio specifico.



In questo caso abbiamo:

Mostra Log Utente	GET	/log/{idUtente}
Regista Log	POST	/log
Affollamento Ultima Ora	GET	/affollamento
Percentuale Affollamento Singolo	GET	/affollamentoSingolo

Sviluppo API

Di seguito vengono presentate le varie API sviluppate a partire dai precedenti resources models.

Tutte le API che necessitano di essere eseguite, solo da utenti autenticati, utilizzano la seguente funzione prima della loro esecuzione:

```
const tokenCheck = (req, res, next) => {  
  if(!(req.headers.token && req.headers.livello)){  
    return res.status(401).json({Error: 'Token o livello non presenti'})  
  }  
  jwt.verify(req.headers.token, process.env.TOKEN_KEY, (err, decoded) => {  
    if(err){  
      return res.status(401).json({Error: "Token fornito non valido"})  
    }  
    next()  
  })  
}
```

Tutte le API che di seguito ne richiederanno l'esecuzione presentano, ad inizio descrizione, una parentesi con la dicitura "Richiede autenticazione".

Dati di un utente

La seguente API viene utilizzata per ottenere tutte le informazioni relative ad un utente specificando, nell'url della richiesta, l'indirizzo email che lo identifica.

```
const getUtente = (req, res) => {  
  Utente.findOne({email: req.params.email}, (err, data) => {  
    if(err){  
      return res.status(500).json({Error: err})  
    }else if(!data){  
      return res.status(404).json({message: "Utente non trovato"})  
    }  
    return res.json(data)  
  })  
}
```

Creazione nuovo utente

La seguente API viene utilizzata per creare un nuovo utente nel sistema specificando, nel body della richiesta: il nome, il cognome, l'indirizzo email, la password e il nickname scelti. Prima di registrare l'utenza ci si assicura che non esista già un'utenza associata all'indirizzo email inserito.

```
const nuovoUtente = (req, res) => {
  if(!(req.body.nome && req.body.cognome && req.body.email && req.body.password && req.body.nickname)){
    return res.status(400).json({Error: 'dati inseriti nel formato sbagliato'})
  }
  Utente.findOne({email: req.body.email}, (err, data) => {
    if(!data){ //se il dato non esiste ancora
      bcrypt.hash(req.body.password, 5, (err, hash) => {
        if(err)
          return res.status(500).json({Error: err})
        const newUtente = new Utente({
          nome: req.body.nome,
          cognome: req.body.cognome,
          email: req.body.email,
          password: hash,
          nickname: req.body.nickname
        })
        newUtente.save((err, data) =>{
          if(err)
            return res.status(500).json({Error: err})
          return res.status(201).json(data)
        })
      })
    }else{
      if(err)
        return res.status(500).json({Error: `Something went wrong, please try again. ${err}`})
      return res.status(409).json({Error: "Indirizzo email già utilizzato da un utente"})
    }
  })
}
```

Rimozione di un utente

(Richiede autenticazione) La seguente API viene utilizzata per cancellare un utente dal sistema specificando, nell'url della richiesta, l'indirizzo email che lo identifica. Prima di eliminare l'utenza ci si assicura che il suo livello non corrisponda a quello di Operatore o Gestore, perché in tal caso è compito dell'admin provvedere alla cancellazione (parte non implementata).

```
const rimuoviUtente = (req, res) => {
  if(req.headers.livello !== 'Base'){
    return res.status(401).json({Error: "Azione non permessa ad un Utente di livello " + req.headers.livello})
  }
  Utente.deleteOne({email: req.params.email}, (err, data) => {
    if(err){
      return res.status(500).json({Error: `Something went wrong, please try again. ${err}`})
    }
    return res.json({message: "Utente eliminato correttamente"})
  })
}
```

Autenticazione di un utente

La seguente API viene utilizzata per autenticare un utente specificando, nel body della richiesta, l'email e la password per effettuare il login. Come risposta, in caso di autenticazione correttamente conclusa, vengono indicati i dati dell'utente ed il token, valido per un giorno, che il front-end utilizzerà per invocare API che richiedono l'autenticazione.

```
const login = (req, res) => {
  if(!(req.body.email && req.body.password)){
    return res.status(400).json({Error: 'dati inseriti nel formato sbagliato'})
  }
  Utente.findOne({email: req.body.email}, (err, data) => {
    if(!data){
      return res.status(404).json({Error: "Utente non trovato"})
    } else if(err){
      return res.status(500).json({Error: err})
    } else{
      bcrypt.compare(req.body.password, data.password, (err, result) => {
        if(err){
          return res.status(500).json({Error: err})
        } else if(result){
          let token = jwt.sign({email: data.email, id: data._id}, process.env.TOKEN_KEY, {expiresIn: 86400}/* one day*/)
          return res.json({nome: data.nome, cognome: data.cognome, email: data.email, livelloUtente: data.livelloUtente, token: token})
        } else{
          return res.status(401).json({Error: "Password errata"})
        }
      })
    }
  })
}
```

Modifica della password di un utente

(Richiede autenticazione) La seguente API viene utilizzata per modificare la password di un utente del sistema specificando, nel body della richiesta, la vecchia password e la nuova password che si desidera utilizzare. Prima di modificare la password ci si assicura che la vecchia password inserita corrisponda a quella memorizzata nel sistema.

```
const modificaPassword = (req, res) => {
  if(!(req.body.email && req.body.oldPassword && req.body.newPassword)){
    return res.status(400).json({Error: 'dati inseriti nel formato sbagliato'})
  }
  Utente.findOne({email: req.body.email}, (err, data) => {
    if(!data){
      return res.status(404).json({Error: "Utente non trovato"})
    } else if(err){
      return res.status(500).json({Error: err})
    } else{
      bcrypt.compare(req.body.oldPassword, data.password, (err, result) => {
        if(err){
          return res.json({Error: err})
        } else if(result){
          bcrypt.hash(req.body.newPassword, 5, (err, hash) => {
            if(err)
              return res.status(500).json({Error: err})
            Utente.updateOne({email: req.body.email}, {password: hash}, (err, update) => {
              if(err)
                return res.status(500).json({Error: err})
              return res.status(204).json({})
            })
          })
        } else{
          return res.status(401).json({Error: "Password errata"})
        }
      })
    }
  })
}
```

Elenco degli impianti memorizzati

La seguente API viene utilizzata per ottenere l'elenco degli impianti memorizzati nel sistema.

```
const getAllImpianti = (req, res) => {
  Impianto.find({}, (err, data) => {
    if(err){
      return res.status(500).json({Error: err})
    }
    return res.json(data)
  })
}
```

Modifica dello stato di apertura di un impianto

(Richiede autenticazione) La seguente API viene utilizzata per modificare lo stato di apertura di un impianto, da aperto a chiuso o vice versa specificando, nell'url della richiesta, l'id che identifica l'impianto. Prima di modificare lo stato ci si assicura che l'utente che sta utilizzando l'API sia o un utente Operatore oppure un utente Gestore.

```
const modificaStatoImpianto = (req, res) => {
  if(req.headers.livello !== 'Operatore' && req.headers.livello !== 'Gestore'){
    return res.status(401).json({Error: "Azione non permessa ad un utente di livello " + req.headers.livello})
  }
  Impianto.findById(req.params.id, (err, data) => {
    if(!data){
      return res.status(404).json({message: "Impianto non trovato"})
    }
    Impianto.updateOne({_id: req.params.id}, {statoApertura: !data.statoApertura}, (err, data) => {
      if(err)
        return res.status(500).json({Error: err})
      return res.status(204).json({})
    })
  })
}
```

Elenco dei log di accesso di un utente

(Richiede autenticazione) La seguente API viene utilizzata per ottenere l'elenco dei log di accesso agli impianti di un utente specificando, nell'url della richiesta, l'id che lo identifica.

```
const getAllLogsUtente = (req, res) => {
  Log.find({utente: req.params.idUtente}, (err, data) => {
    if(err){
      return res.status(500).json({Error: err})
    }
    return res.json(data)
  })
}
```

Creazione di un log di accesso

(Richiede autenticazione) La seguente API viene utilizzata per creare un log di accesso agli impianti da parte di un utente (quello che nella realtà implementativa corrisponde ad un passaggio al tornello di accesso all'impianto) specificando, nel body della richiesta, l'id dell'impianto e dell'utente.

```
const createLog = (req, res) => {
  if(!(req.body.idImpianto && req.body.idUtente)){
    return res.status(400).json({Error: 'dati inseriti nel formato sbagliato'})
  }
  const newLog = new Log({
    impianto: req.body.idImpianto,
    utente: req.body.idUtente,
    timestamp: Date.now()
  })
  newLog.save((err, data) => {
    if(err)
      return res.status(500).json({Error: err})
    return res.json(data)
  })
}
```

Numero di utenti con almeno un log nell'ultima ora

La seguente API viene utilizzata per ottenere il numero di utenti con almeno un log nell'ultima ora (quello che nella realtà implementativa corrisponde alla stima del numero di persone presenti sulle piste da sci).

```
const utentiNellUltimaOra = (req, res) => {
  Log.distinct("utente", {timestamp: {$gte : (Date.now() - (60 * 60 * 1000))}}, (err, data) => {
    if(err){
      return res.status(500).json({Error: err})
    }
    return res.json({"num": data.length})
  })
}
```

Elenco delle percentuali di occupazione di impianto nell'ultima mezz'ora

La seguente API viene utilizzata per ottenere l'elenco della percentuale di occupazione per ogni impianto. Come risposta, nel caso l'impianto sia chiuso, al posto del normale valore percentuale viene indicato il valore -1.

```
async function logNegliUltimi30Minuti(req, res){
  let impianti = await Impianto.find()
  let list = []
  for(const impianto of impianti){
    let val
    if(impianto.statoApertura){
      let result = await Log.find({impianto: impianto._id, timestamp: {$gte : (Date.now() - (30 * 60 * 1000))}}).count()
      val = result*100/impianto.portataOraria
    }else{
      val = -1
    }
    list.push({"impianto": impianto.nome, "perc": val})
  };
  res.json(list)
}
```


API documentation

Le API Locali fornite dall'applicazione Ski Online e descritte nella sezione precedente sono state documentate utilizzando il modulo NodeJS chiamato Swagger UI Express. In questo modo la documentazione relativa alle API è direttamente disponibile a chiunque veda il codice sorgente. Per poter generare l'endpoint dedicato alla presentazione delle API abbiamo utilizzato Swagger UI in quanto crea una pagina web dalle definizioni delle specifiche OpenAPI. In particolare, di seguito mostriamo la pagina web relativa alla documentazione che presenta le 11 API (GET, POST, DELETE e PUT) per la gestione dei dati della nostra applicazione. La GET viene utilizzata per visualizzare i dati in una pagina HTML. La POST per inserire un nuovo dato nel nostro sistema o per l'operazione di login. La DELETE per cancellare un dato dal nostro sistema. La PUT per modificare un dato già presente nel nostro sistema. L'endpoint da invocare per raggiungere la seguente documentazione è: <https://front-end-production-847c.up.railway.app/api-docs>

Swagger
Supported by SMARTBEAR

Ski Online project ^{1.0.0}

[Base URL: localhost:8080/]

Documentazione delle API Ski Online project

MIT

Schemes
HTTP

Utenti

API per gli utenti che utilizzano il sistema

- GET** `/utente/{email}` Mostra l'utenza associata ad uno specifico indirizzo email
- DELETE** `/utente/{email}` Elimina l'utenza associata all'indirizzo email specificato
- POST** `/utente` Crea un nuovo Utente di livello Utente Registrato nel sistema
- PUT** `/utente` Modifica la password di un utente, richiedendo prima la password precedente alla modifica
- POST** `/login` Effettua l'autenticazione di una utenza

Impianti

API per la gestione degli impianti sciistici nella località sciistica

- GET** `/impianto` Fornisce la lista di tutti gli impianti memorizzati
- PUT** `/impianto/{id}` Modifica lo stato di apertura di un impianto da aperto a chiuso o vice versa

Log

API per la gestione dei log degli accessi degli utenti agli impianti sciistici

- GET** `/log/{idUtente}` Fornisce la lista di tutti i log di accesso ad un impianto effettuati da un utente
- POST** `/log` Registra un nuovo log di passaggio di un utente per un impianto
- GET** `/affollamento` Fornisce il numero di utenti con almeno un accesso ad un impianto nell'ultima ora
- GET** `/affollamentoSingolo` Fornisce per ogni impianto la percentuale di affollamento (num accessi/portata max nell'ultima mezz'ora)

Front-End Implementation

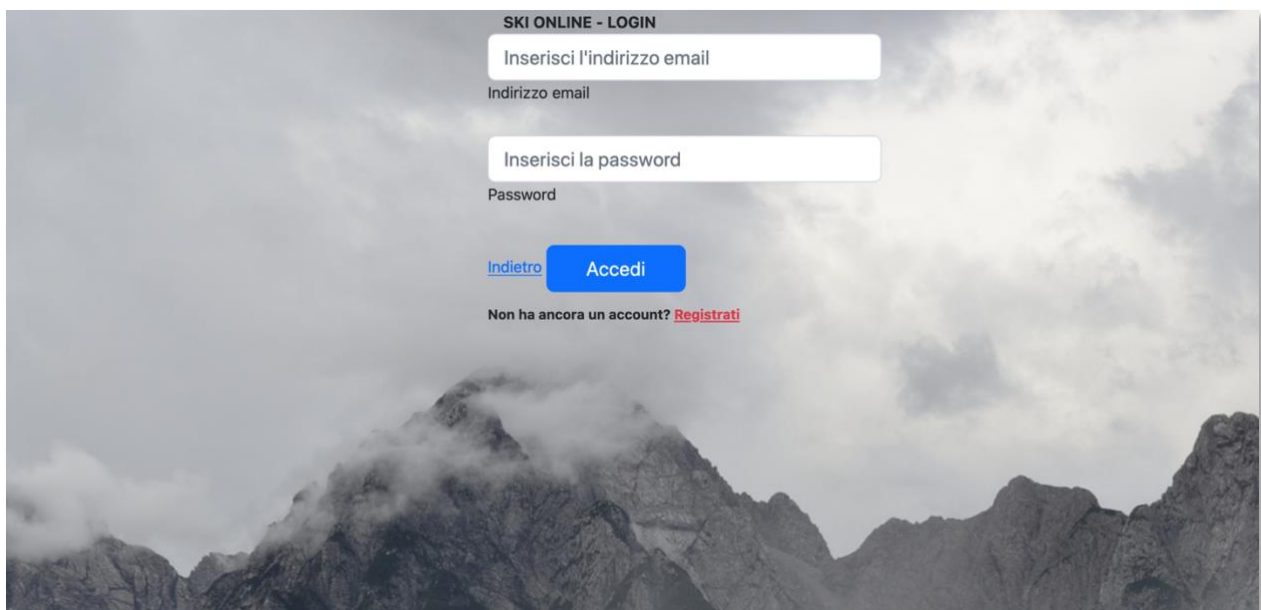
Il front-end fornisce tutte le funzionalità descritte in precedenza rese disponibili tramite le corrispondenti API. In particolare, l'applicazione è strutturata su quattro pagine principali e da altre quattro di supporto descritte, nel dettaglio, di seguito. È stato sviluppato con l'ausilio del framework "Bootstrap".

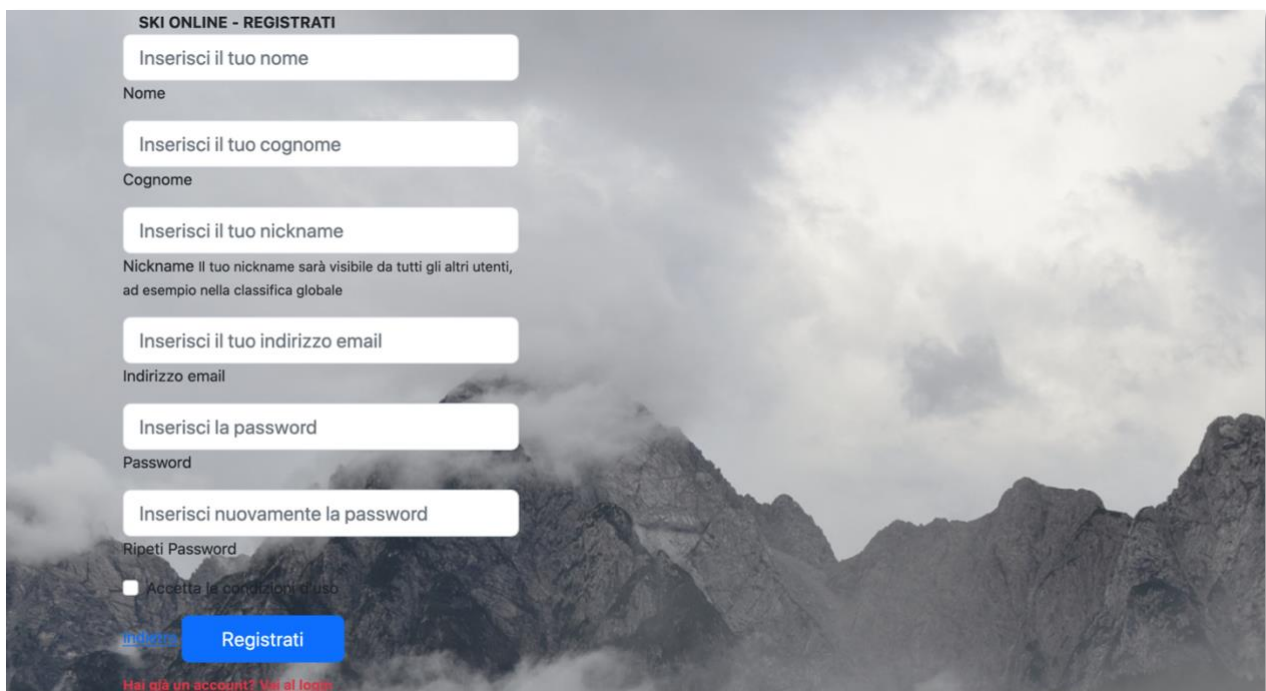
Home page

Nella home page è presente una navbar, comune a tutte le altre pagine principali, che permette di raggiungere velocemente quest'ultima. Inoltre, nel corpo della pagina è presente un elenco puntato delle funzioni principali dell'applicazione.



All'interno della navbar sono presenti due bottoni per eseguire il login e la registrazione al sistema; se premuti, conducono alle due pagine di supporto mostrate di seguito.





SKI ONLINE - REGISTRATI

Inserisci il tuo nome

Nome

Inserisci il tuo cognome

Cognome

Inserisci il tuo nickname

Nickname Il tuo nickname sarà visibile da tutti gli altri utenti, ad esempio nella classifica globale

Inserisci il tuo indirizzo email

Indirizzo email

Inserisci la password

Password

Inserisci nuovamente la password

Ripeti Password

☐ Accetta le condizioni d'uso

[Informativa](#) **Registrati**

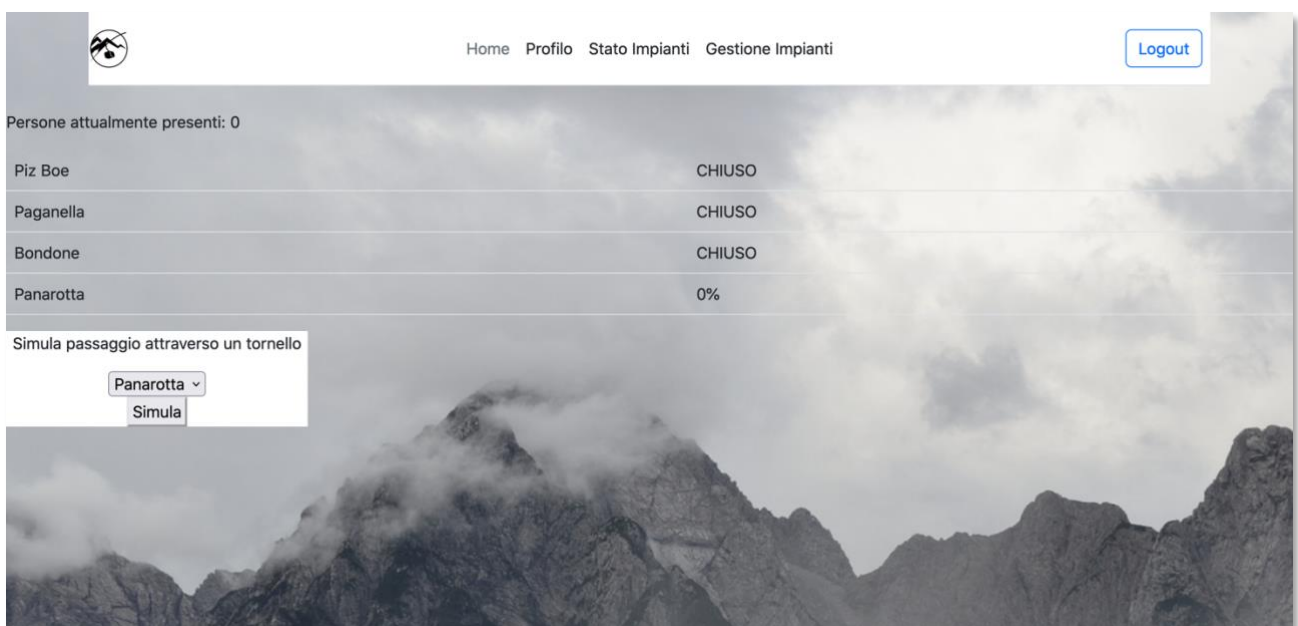
[Hai già un account? Vai al login](#)


Nel caso in cui si sia già effettuata la registrazione comparirà, al posto dei bottoni precedentemente descritti, un bottone per eseguire il logout.

Stato Impianti

Nella schermata Stato Impianti è possibile visualizzare il numero di utenti attualmente presenti e, per ogni impianto, la percentuale di occupazione con eventualmente lo stato di chiusura.

Informazioni per il testing: Se un utente è attualmente autenticato, comparirà in basso un form tramite il quale è possibile simulare il passaggio dell'utente attualmente loggato, tramite uno degli impianti, in modo da verificare la conseguente modifica dei dati precedentemente mostrati.



 Home Profilo Stato Impianti Gestione Impianti [Logout](#)

Persone attualmente presenti: 0

Piz Boe	CHIUSO
Paganella	CHIUSO
Bondone	CHIUSO
Panarotta	0%

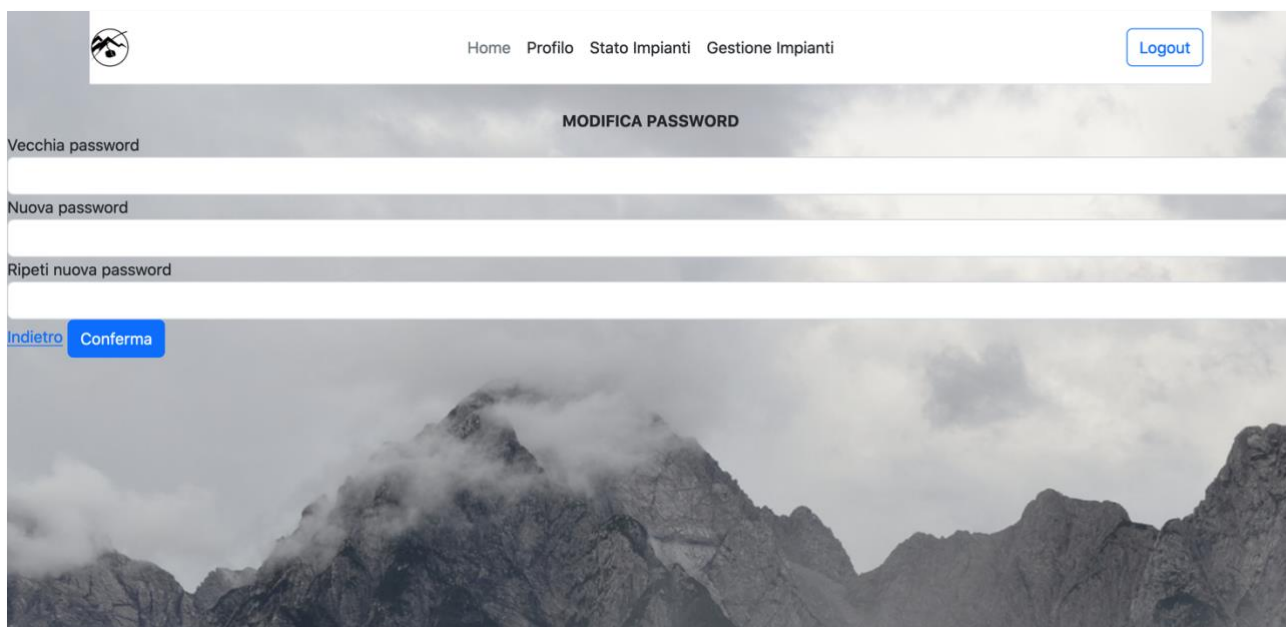
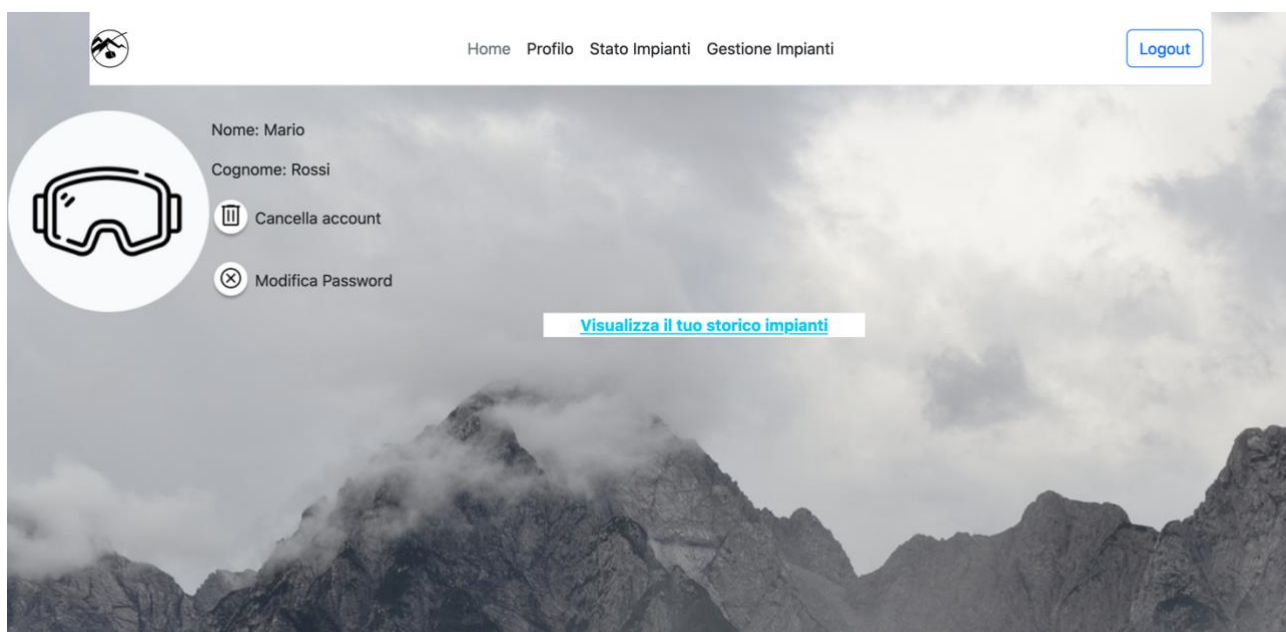
Simula passaggio attraverso un tornello

Panarotta ▾

Simula

Profilo

Nella schermata profilo è possibile visualizzare le proprie informazioni personali come nome e cognome e accedere alle due pagine di supporto relative alla modifica della password e visualizzazione dello storico degli impianti. Per poter accedere alla pagina è necessario essere già autenticati mentre, l'opzione cancella account è accessibile solamente a utenti di livello "Registrato".



Informazioni per il testing: nella schermata dello storico degli impianti è presente, come per la schermata Stato Impianti, un form per simulare un passaggio per un impianto, per verificare l'aggiunta dei passaggi così generati all'elenco presente sotto. Ogni simulazione di passaggio influenza sia il dato dello storico impianti che quello relativo allo stato degli impianti.

Impianto	Data
Piz Boe	Fri Dec 30 2022 12:46:49 GMT+0100 (Ora standard dell'Europa centrale)
Piz Boe	Fri Dec 30 2022 13:13:24 GMT+0100 (Ora standard dell'Europa centrale)
Piz Boe	Fri Dec 30 2022 15:57:20 GMT+0100 (Ora standard dell'Europa centrale)
Piz Boe	Fri Dec 30 2022 16:01:40 GMT+0100 (Ora standard dell'Europa centrale)
Pinarotta	Fri Dec 30 2022 21:18:53 GMT+0100 (Ora standard dell'Europa centrale)

Gestione Impianti

Nella schermata gestione impianti è possibile modificare lo stato di apertura di ogni impianto da aperto a chiuso e vice versa. Per poter accedere alla pagina è necessario essere già autenticati e essere un utente di livello "Operatore" o "Gestore".

Informazioni per il testing: per poter provare la seguente funzionalità è possibile effettuare l'accesso con un account di livello gestore utilizzando le seguenti credenziali nel login: email: gestore@mail.com, password: Password.0

Seleziona impianto

L'impianto è attualmente CHIUSO

APRI

GitHub Repository and Deployment info

Il github repository in cui si è proceduto alla progettazione e sviluppo dell'applicazione è raggiungibile al seguente link: <https://github.com/orgs/ski-online/repositories> ed è composto da quattro repositories:

- Documents, contenente la documentazione del progetto, mano a mano che è stata prodotta;
- Back-end, contenente il codice in Node.js relativo ai servizi di back-end, mano a mano che è stato sviluppato;
- Front-end, contenente il codice HTML e JavaScript relativo alle pagine di front-end e all'utilizzo delle API fornite dal back-end;
- Deliverable, contenente cinque file PDF relativi alle versioni finali dei cinque deliverable.

È stato fatto il deployment dell'applicazione sulla piattaforma railway.app ed è raggiungibile al seguente url: <https://front-end-production-847c.up.railway.app>

Testing

È stato effettuato il testing delle API sviluppate nel back-end (unit testing) tramite le librerie Node.js jest e supertest ottenendo i seguenti risultati:

```
PASS test/app.test.js (6.849 s)
API test
  ✓ test API GET /utente esistente (2113 ms)
  ✓ test API GET /utente non esistente (163 ms)
  ✓ test API POST /utente vuoto (8 ms)
  ✓ test API POST /utente già esistente (145 ms)
  ✓ test API POST /utente corretta (217 ms)
  ✓ test API PUT /utente non trovato (95 ms)
  ✓ test API PUT /utente vecchia password errata (119 ms)
  ✓ test API PUT /utente formato sbagliato (8 ms)
  ✓ test API PUT /utente corretta (228 ms)
  ✓ test API POST /login formato errato (11 ms)
  ✓ test API POST /login corretta (126 ms)
  ✓ test API POST /login non esistente (96 ms)
  ✓ test API POST /login password errata (102 ms)
  ✓ test API DELETE /utente token non presente (4 ms)
  ✓ test API DELETE /utente token non valido (3 ms)
  ✓ test API DELETE /utente non permessa (5 ms)
  ✓ test API DELETE /utente corretta (104 ms)
  ✓ test API POST /log formato sbagliato (4 ms)
  ✓ test API POST /log corretta (90 ms)
  ✓ test API GET /log (96 ms)
  ✓ test API GET /affollamento (115 ms)
  ✓ test API GET /affollamentoSingolo (304 ms)
  ✓ test API GET /impianto (102 ms)
  ✓ test API PUT /impianto non permessa (5 ms)
  ✓ test API PUT /impianto non esistente (7 ms)
  ✓ test API PUT /impianto corretta (207 ms)
```

All files

90.65% Statements 165/182 79.26% Branches 65/82 100% Functions 32/32 91.16% Lines 165/181

Press *n* or *j* to go to the next uncovered block, *b*, *p* or *k* for the previous block.

Filter:

File	Statements	Branches	Functions	Lines
app	95.45%	21/22	50%	1/2
app/controllers	87.5%	112/128	80%	64/80
app/models	100%	12/12	100%	0/0
app/routes	100%	20/20	100%	0/0

Si può notare dalla seconda immagine che non è stata effettuata una copertura del 100% del codice sviluppato, questo a causa dei branches di controllo in caso di errore del database oppure delle librerie utilizzate che non è stato possibile provare in quanto dipendenti da fattori esterni alla nostra applicazione. Ad ogni modo, in tutti i casi di questo tipo viene ritornato un codice di errore 500 "Internal Server Error".