



**UNIVERSITÀ
DI TRENTO**

Dipartimento di Ingegneria e Scienza dell'Informazione

Nome progetto :

BellHotel

Nome documento :

Documento di Architettura

Indice

| | |
|--|----|
| <i>Scopo del documento</i> | 3 |
| <i>Userflow Diagram</i> | 4 |
| <i>Implementazione delle API</i> | 5 |
| <i>Specifica ed estrazione delle risorse</i> | 8 |
| <i>Modello delle risorse</i> | 10 |
| <i>Sviluppo delle API</i> | 19 |
| <i>Documentazione delle API</i> | 27 |
| <i>Implementazione del Front-End</i> | 30 |
| <i>Testing delle API</i> | 33 |
| <i>Repository e Deployment</i> | 35 |

Scopo del documento

Il presente documento riporta lo sviluppo di alcune parti importanti del nostro sito. Nello specifico andremo a progettare tutte le possibili azioni di utenti con l'account cliente oppure con l'account gestore: nel primo caso ci concentreremo sulla possibilità di prenotare stanze di hotel, mentre nel secondo caso si potrà aggiungere nuovi hotel sul sito.

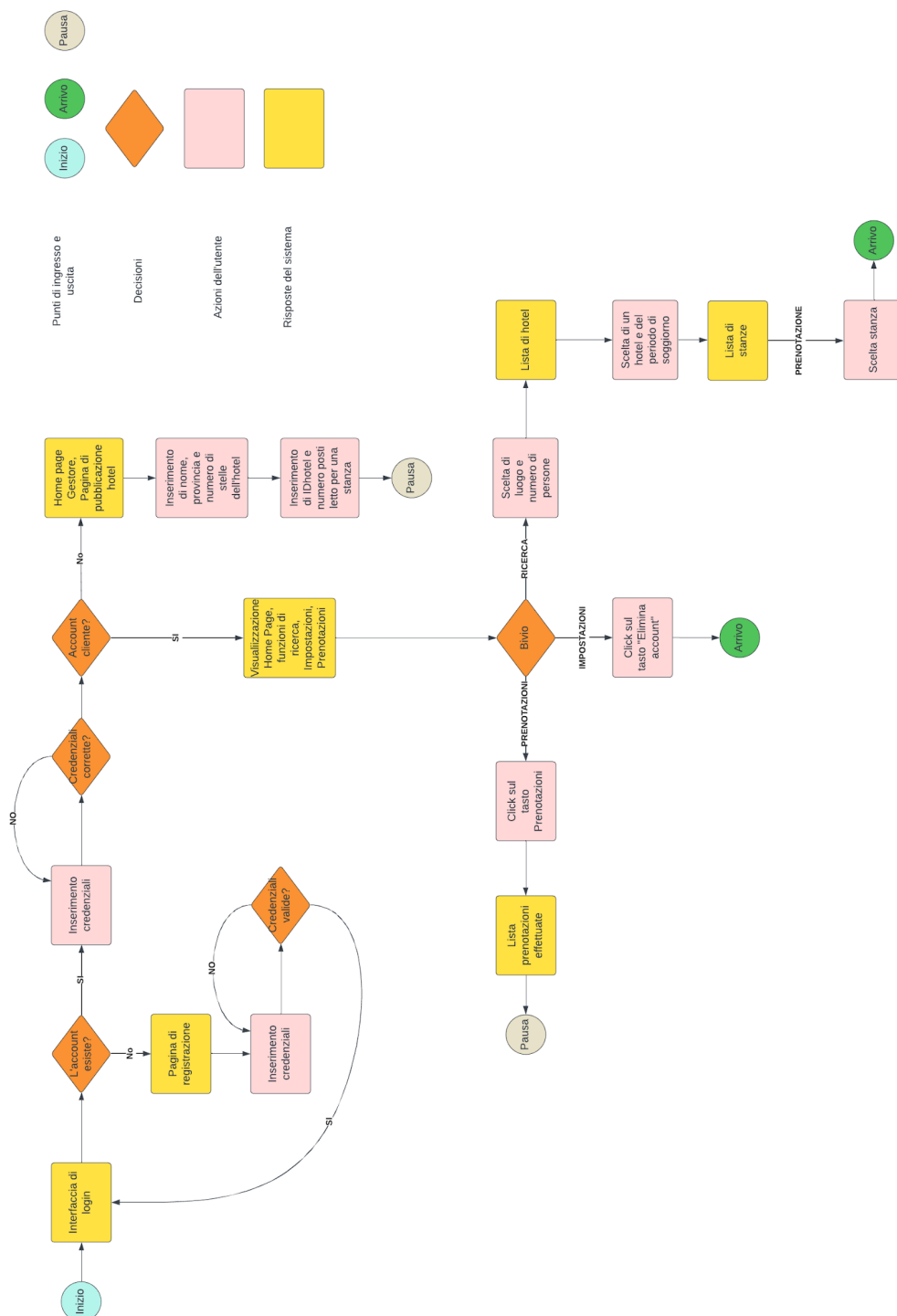
Le sezioni principali di questo documento sono le seguenti:

- UserFlow Diagram
- Modello delle risorse
- Implementazione delle API con annessa documentazione
- Testing delle API
- Implementazione del Front-End

Userflow Diagram

In questo capitolo viene presentato lo Userflow Diagram riguardante tutte le possibili azioni del cliente.

Di seguito vi sarà inoltre una legenda per comprendere il significato di tutti i simboli utilizzati.



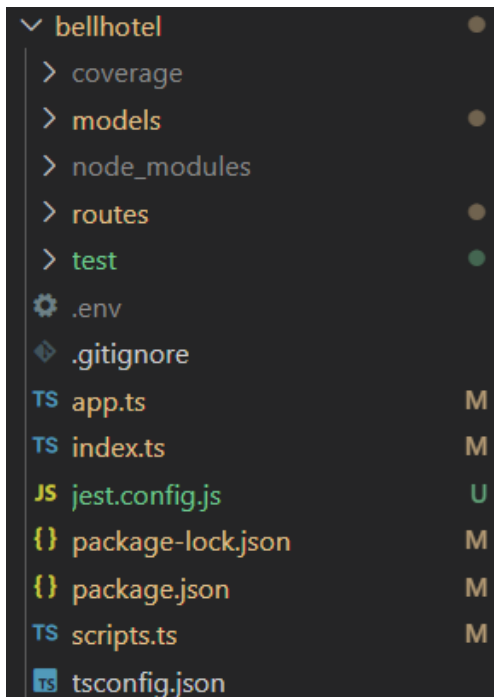
Implementazione delle API

In questa sezione andiamo a sviluppare il backend del nostro progetto, analizzando tutte le API coinvolte nelle azioni descritte dallo Userflow Diagram.

Struttura del Back-End

La struttura del backend è riportata nella figura sottostante. All'interno della cartella sono presenti i file che implementano le API nella cartella **/routes** e i modelli dei dati che fanno da ponte tra il backend e il database sul quale verranno salvate le informazioni nella cartella **/models**. La cartella **/test** contiene invece i file per il testing, che verrà approfondito nella prossima sezione.

Il file **index.ts** è il file principale del backend, in quanto si occupa di istanziare l'oggetto **app.ts** e di effettuare la connessione al database. Il file **app.ts** consiste nell'applicazione stessa, alla quale verranno aggiunte le rotte per le API e la documentazione. Il file **scripts.ts** contiene funzioni utilizzate in più punti del codice. I restanti file sono di configurazione.



Dipendenze del progetto

Il nostro progetto dipende da diverse librerie di NodeJS per il suo funzionamento. Queste sono :

- **dotenv** per le variabili d'ambiente
- **mongoose** come ponte tra il backend e il database MongoDB
- **jest** e **supertest** per il testing
- **typescript** e vari **@types** come aiuto allo sviluppo
- **express** come framework per la creazione del server
- **cors** per permettere ad un server esterno di accedere alle risorse presenti nel database dal backend

Collezione di dati nel database

E' stata utilizzata come base di dati MongoDB, cioè un database non relazionale orientato ai documenti. Abbiamo creato 4 Collections, cioè cioè gruppi di documenti di tipo diverso, ognuna per il tipo di dato che è necessario da memorizzare. Questi sono **Utente**, **Prenotazione**, **Hotel**, **Stanza**.

Figura 1 : Collezioni presenti nel database

| Collection Name | Documents | Logical Data Size | Avg Document Size | Storage Size | Indexes | Index Size | Avg Index Size |
|-----------------|-----------|-------------------|-------------------|--------------|---------|------------|----------------|
| Hotel | 5 | 428B | 86B | 36KB | 1 | 36KB | 36KB |
| Prenotazione | 2 | 310B | 155B | 36KB | 1 | 36KB | 36KB |
| Stanza | 3 | 192B | 64B | 36KB | 1 | 36KB | 36KB |
| Utente | 3 | 327B | 109B | 36KB | 1 | 36KB | 36KB |

Figura 2: Modello tipo di dato Utente

| | | |
|---|--|----------|
| 1 | <code>_id: ObjectId('6627d9dbe35700ef9a16c9b5')</code> | ObjectId |
| 2 | <code>email: "giorgioBianchi@gmail.com"</code> | String |
| 3 | <code>password: "giorgio89"</code> | String |
| 4 | <code>tipoAccount: "cliente"</code> | String |

Figura 3 : Modello tipo di dato Prenotazione

```
1  _id: ObjectId('662e0b7e40365e8c5c16c9b5')      ObjectId
2  numeroPersone : 5                             Int32
3  IDhotel : 66309965e135aebced256141             ObjectId
4  IDstanza : 66309a40e135aebced256142            ObjectId
5  inizioSoggiorno : 2024-06-15T00:00:00.000+00:00 Date
6  fineSoggiorno : 2024-06-20T00:00:00.000+00:00 Date
7  IDutente : 6627d9dbe35700ef9a16c9b5           ObjectId
```

Figura 4: Modello tipo di dato Hotel

```
1  _id: ObjectId('66309965e135aebced256141')
2  provincia : "Mantova/"
3  nome : "HotelRigoletto/"
4  numeroStelle : 3
```

Figura 5 : Modello tipo di dato tipo Stanza

```
1  _id: ObjectId('66d22d9ebf00a1438745b0cb')      ObjectId
2  reserved : false                               Boolean
3  numeroPostiLetto : 4                           Int32
4  hotelAppartenenza : 66d0e22a4c6e943d12f2090d   ObjectId
5  __v : 0                                         Int32
```

Specifica ed estrazione delle risorse dal Class Diagram

In questo paragrafo vengono descritte le risorse estratte dal Class Diagram: nello specifico, vi saranno delle risorse principali e, di seguito, tutte le API che la risorsa in questione può utilizzare.

«resource» Utente

La risorsa Utente ha come attributi:

- IDutente
- email
- password
- tipoAccount

«resource» Prenotazione

La risorsa Prenotazione ha come attributi:

- IDprenotazione
- IDutente
- IDhotel
- IDstanza
- numeroPersone
- inizioSoggiorno
- fineSoggiorno

«resource» Hotel

La risorsa Hotel ha come attributi:

- IDhotel
- IDgestore
- provincia
- nome
- numeroStelle

«resource» Stanza

La risorsa Stanza ha come attributi:

- IDstanza
- hotelAppartenenza
- reserved
- numeroPostiLetto

APIs

«resource» signUp: API che permette ad un utente di registrarsi al sito. Ciò che viene svolta è una POST che ha, come parametri di ingresso, email, password e TipoAccount.

«resource» login: API che permette all'utente di effettuare il login al sito. Viene svolta una GET che preleva l'utente dal database, in base ai parametri inseriti in ingresso quali email e password.

«resource» ricerca: API che permette all'utente di cercare un hotel esistente nel database. Viene svolta una GET che prende come parametri in ingresso numeroPersone e provincia.

«resource» getIDs: API che svolge due funzioni differenti a seconda della tipologia di account dell'utente.

Nel caso di un cliente, questa API permette di ottenere l'ID di tutte le prenotazioni effettuate, mentre, nel caso di un gestore, consente l'ottenimento degli ID di tutti gli hotel da lui inseriti.

Viene svolta una GET che prende come parametro in ingresso IDutente.

«resource» prenotazione: API che permette all'utente di prenotare un hotel precedentemente selezionato.

Viene svolta una POST che prende come parametri in ingresso inizioSoggiorno, fineSoggiorno e numeroPersone.

«resource» inserisciHotel: API che permette ad un utente con l'account gestore di inserire sul sito il proprio hotel.

Viene svolta una POST che prende in ingresso IDgestore, provincia, nome e numeroStelle.

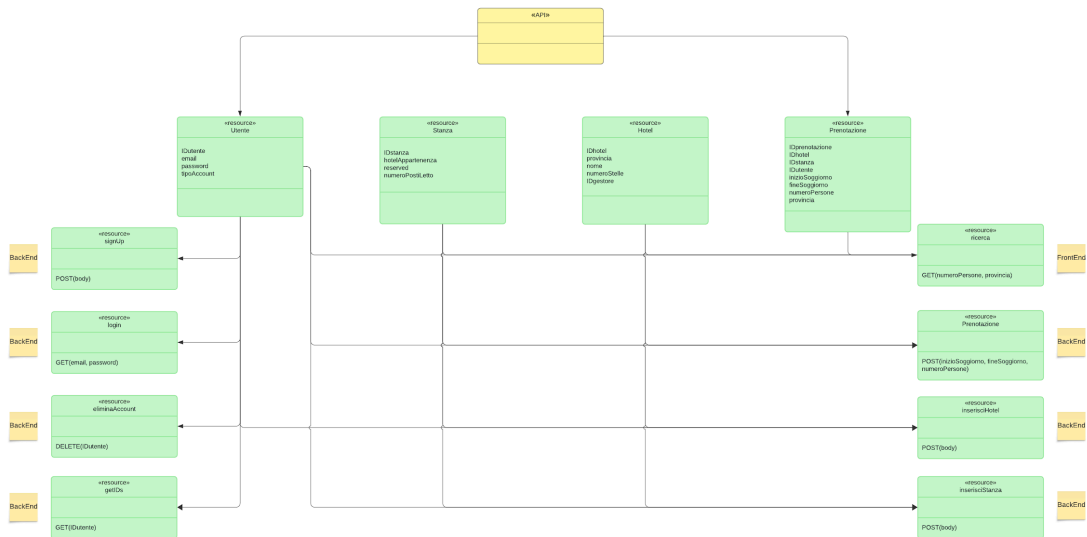
«resource» inserisciStanza: API che permette ad un utente con l'account gestore di inserire una stanza in uno dei suoi hotel.

Viene svolta una POST che prende in ingresso IDgestore e IDhotel.

«resource» eliminaAccount: API che permette all'utente di eliminare il proprio account, il quale verrà rimosso dal database tramite una DELETE.
Il parametro che invece viene chiesto in ingresso per tale operazione è l'IDutente.

Modello delle risorse

Di seguito vi sono i modelli delle risorse, i quali vanno a descrivere nel dettaglio ciascuna API precedentemente elencata.



SignUp

Questa API, all'indirizzo `/api/signUp`, utilizza un metodo POST per inserire un nuovo utente nel database del sito.

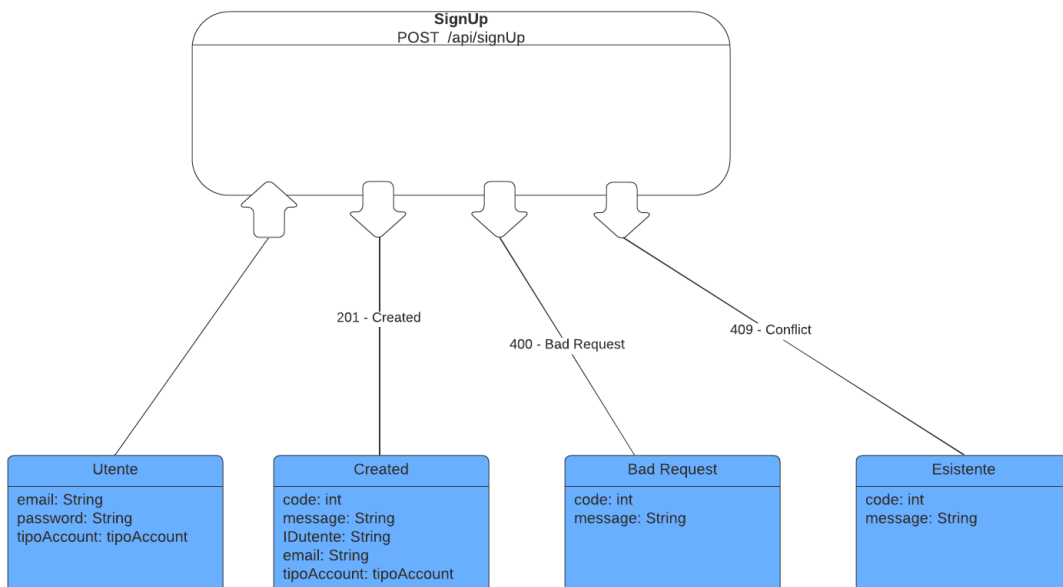
Nella request body vi è la risorsa `Utente` con i seguenti attributi: *email*, *password* e *tipoAccount*.

Ci sono poi tre tipologie di response body in base alla riuscita o meno dell'operazione.

Una di queste è data dal `code = 201` e `message = "Created"`, dove l'operazione è effettivamente riuscita e dunque viene restituito l'*IDutente*, l'*email* e il *tipoAccount*, dunque l'utente riesce a creare un proprio account con il quale registrarsi al sito.

Se invece nella response body è presente il `code = 400` e `message = "Bad Request"`, allora l'operazione non è riuscita in quanto l'email o la password non sono corretti, o non conformi alla regolazione del sito.

Infine, nel caso di `code = 409` e `message = "Conflict"`, l'operazione non è riuscita in quanto l'email inserita è già presente nel database.



Login

Questa API, all'indirizzo `/api/login`, utilizza un metodo GET per permettere ad un utente di effettuare il login.

La request body è rappresentata dalla risorsa Utente, i quali attributi sono *email* e *password*.

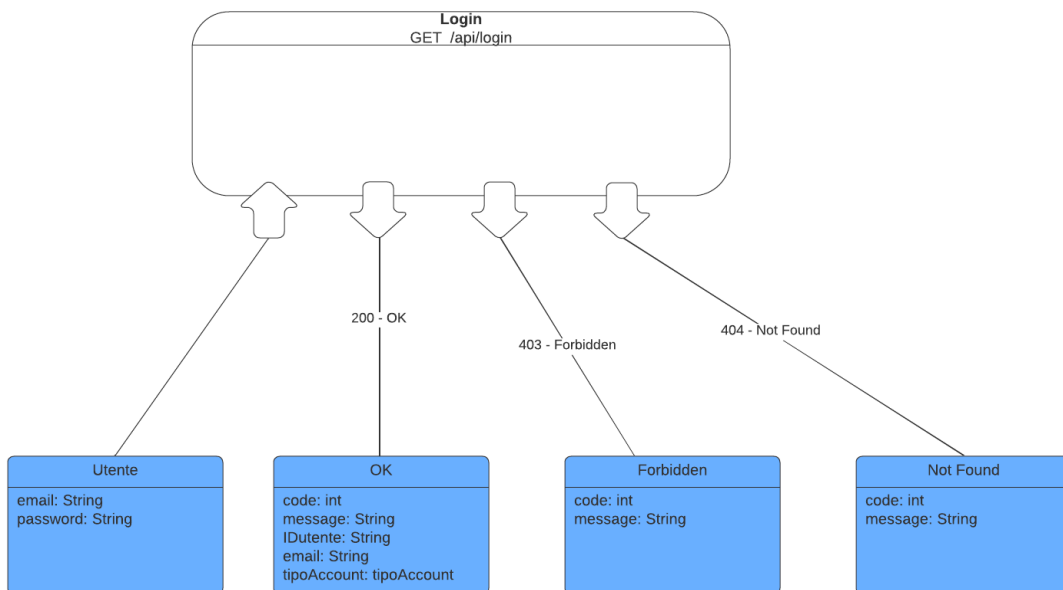
Possono poi esserci tre tipologie di response body a seconda della riuscita o meno dell'operazione.

Nel primo caso, se `code = 200` e `message = "OK"`, allora l'operazione è riuscita e nella response body viene restituito l'*IDutente*, l'*email* e il *tipoAccount*.

Questa casistica corrisponde quindi all'evento in cui l'utente riesce ad effettuare l'accesso al sito con il proprio account.

Se invece `code = 403` e `message = "Forbidden"`, allora l'operazione non è riuscita a causa dell'incorrettezza della password inserita.

Nell'ultima casistica invece si ha `code = 404` e `message = "Not Found"`: questo evento corrisponde ad un'operazione non riuscita in quanto l'email inserita non era presente nel database.



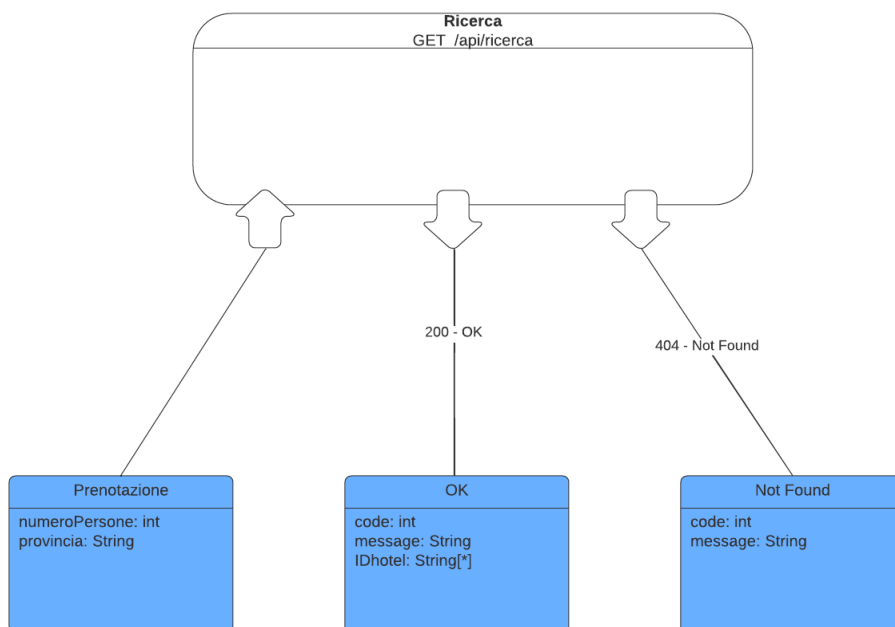
Ricerca

Questa API, all'indirizzo `/api/ricerca`, utilizza un metodo GET per permettere ad un utente di cercare degli hotel.

La request body è rappresentata dalla risorsa Prenotazione, che ha come attributi *numeroPersone* e *provincia*.

Possono poi esserci due tipologie di response body a seconda della riuscita o meno dell'operazione.

Nel primo caso, se *code* = 200 e *message* = "OK", allora l'operazione è riuscita e vengono restituiti gli hotel disponibili in base ai filtri precedentemente selezionati. Se invece *code* = 404 e *message* = "Not Found", allora la ricerca non è andata a buon fine in quanto non esistono hotel che rispecchiano i filtri selezionati.



Prenotazione

Questa API, all'indirizzo `/api/prenotazione`, utilizza un metodo POST per permettere ad un utente di prenotare un hotel.

La request body è rappresentata dalla risorsa Prenotazione, che ha come attributi *inizioSoggiorno*, *fineSoggiorno* e *numeroPersone*.

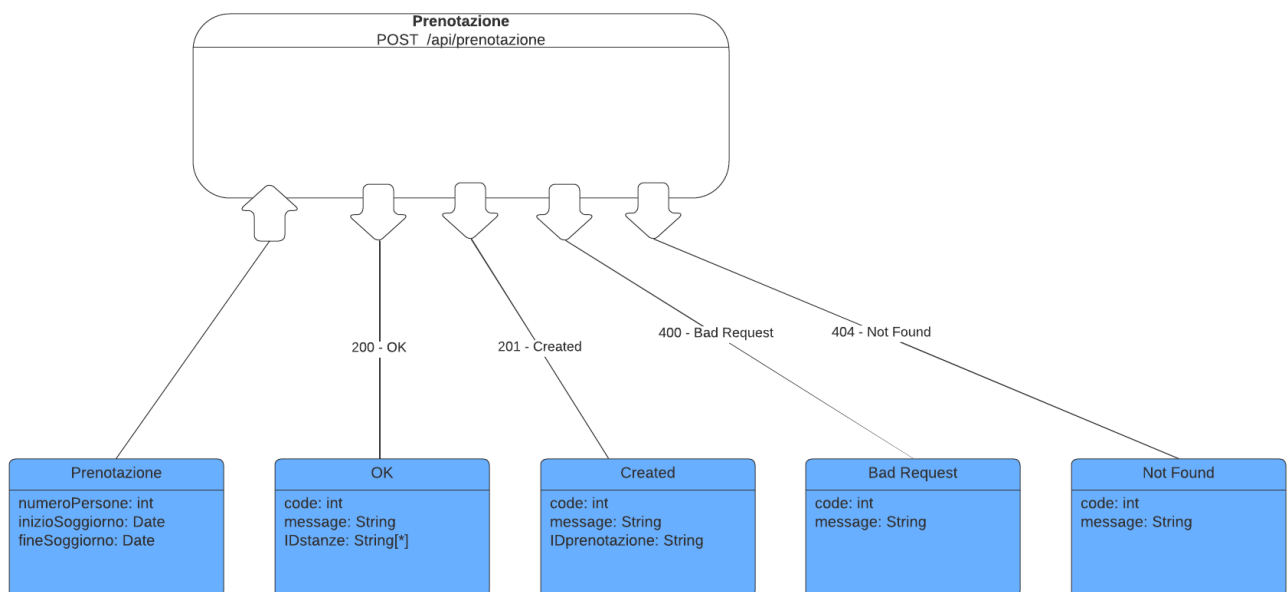
Possono poi esserci due tipologie di response body a seconda della riuscita o meno dell'operazione.

Nel primo caso, se `code = 200` e `message = "OK"`, allora l'operazione è riuscita e vengono restituite le stanze di un hotel selezionato.

Se invece `code = 201` e `message = "Created"`, allora la prenotazione è andata a buon fine e la stanza è stata prenotata.

Se `code = 400` e `message = "Bad Request"`, allora non è andata a buon fine, in quanto l'ID dell'hotel scelto non è valido oppure il numero di persone per cui si vuole prenotare è minore di 1.

Se infine `code = 404` e `message = "Not Found"`, allora la prenotazione non è andata a buon fine in quanto non sono state trovate stanze per l'hotel selezionato.



Inserisci Hotel

Questa API, all'indirizzo `/api/hotel`, utilizza un metodo POST per permettere ad un utente con l'account gestore di inserire il proprio hotel sul sito.

La request body è rappresentata dalle risorse Utente e Hotel e gli attributi richiesti sono *IDgestore*, *provincia*, *nome* e *numeroStelle*.

Possono poi esserci quattro tipologie di response body a seconda della riuscita o meno dell'operazione.

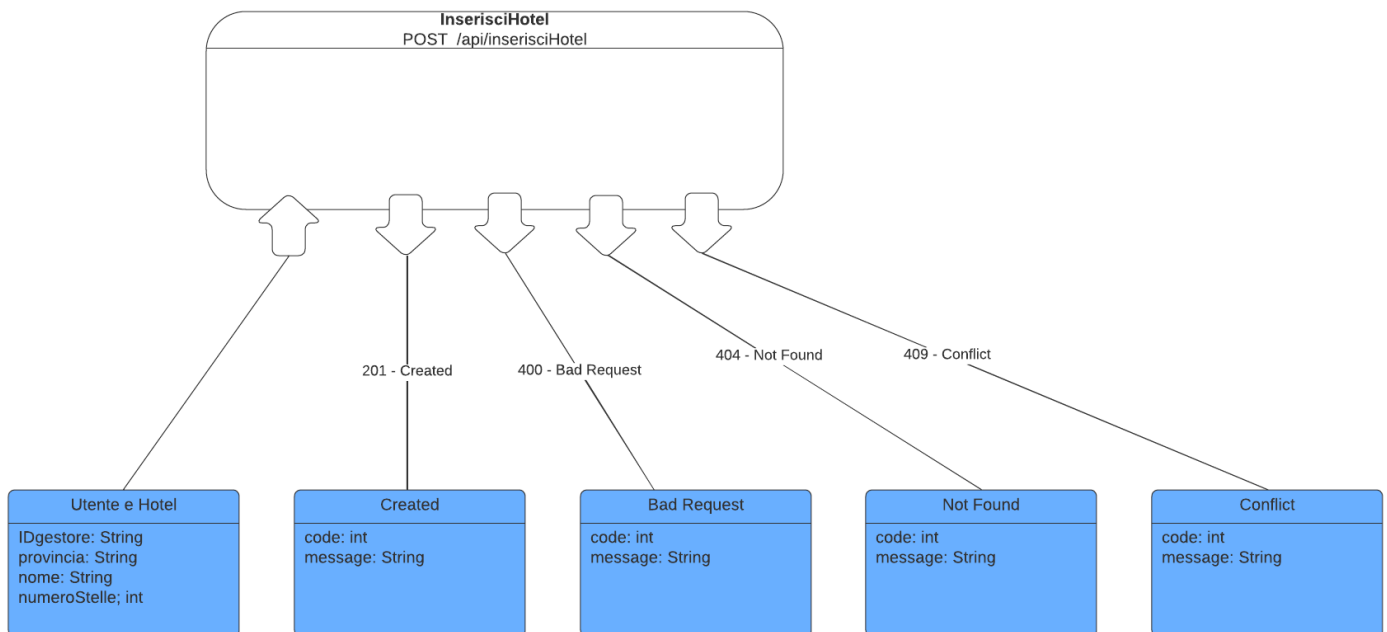
Nel primo caso, se *code* = 201 e *message* = "Created", allora l'operazione è riuscita e non viene restituito nient'altro.

Questa casistica corrisponde quindi all'evento in cui il gestore riesce ad inserire con successo il proprio hotel sul sito.

Se invece *code* = 400 e *message* = "Bad Request", allora l'operazione non è riuscita a causa dell'incorrettezza di almeno uno dei parametri inseriti, o nel caso in cui l'IDutente non sia l'identificativo di un utente con l'account gestore.

Se invece si ha *code* = 404 e *message* = "Not Found", significa che l'operazione non è riuscita in quanto l'IDutente inserito non è valido.

Nell'ultima casistica si ha *code* = 409 e *message* = "Conflict": ciò avviene quando il gestore prova a caricare un hotel che ha già caricato precedentemente sul sito.



GetIDs

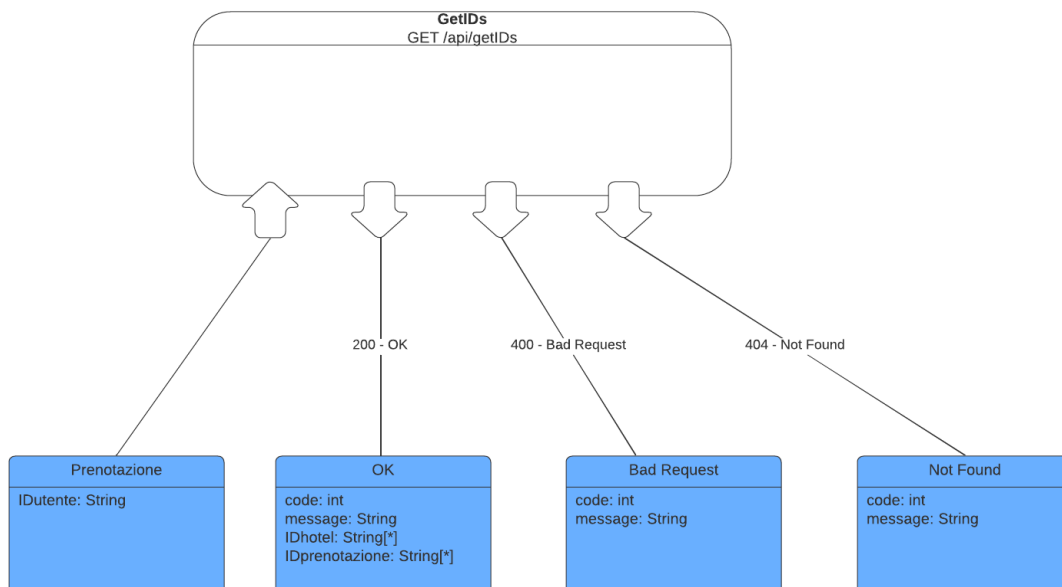
Questa API, all'indirizzo `/api/getIDs`, utilizza un metodo GET per permettere ad un utente con l'account cliente di ottenere gli ID di tutte le prenotazioni da lui effettuate. Se invece ad usufruirne è un utente con l'account gestore, questi otterrà gli ID di tutti gli hotel precedentemente inseriti.

La request body è rappresentata dalla risorsa Utente l'attributo richiesto è `IDutente`. Possono poi esserci tre tipologie di response body a seconda della riuscita o meno dell'operazione.

Nel primo caso, se `code = 200` e `message = "OK"`, allora l'operazione è riuscita e vengono restituiti gli ID delle prenotazioni oppure gli ID degli hotel, rispettivamente per utenti con account cliente e gestore.

Se invece `code = 400` e `message = "Bad Request"`, allora l'operazione non è riuscita a causa dell'incorrettezza del parametro inserito.

Nell'ultima casistica si ha `code = 404` e `message = "Not Found"`: ciò significa che l'operazione non è riuscita in quanto l'`IDutente` inserito non è valido.



Inserisci Stanza

Questa API, all'indirizzo `/api/hotel`, utilizza un metodo POST per permettere ad un utente con l'account gestore di inserire una nuova stanza in un hotel precedentemente inserito sul sito.

La request body è rappresentata dalle risorse Utente, Hotel e Stanza e gli attributi richiesti sono *IDgestore*, *hotelAppartenenza*, e *numeroPostiLetto*.

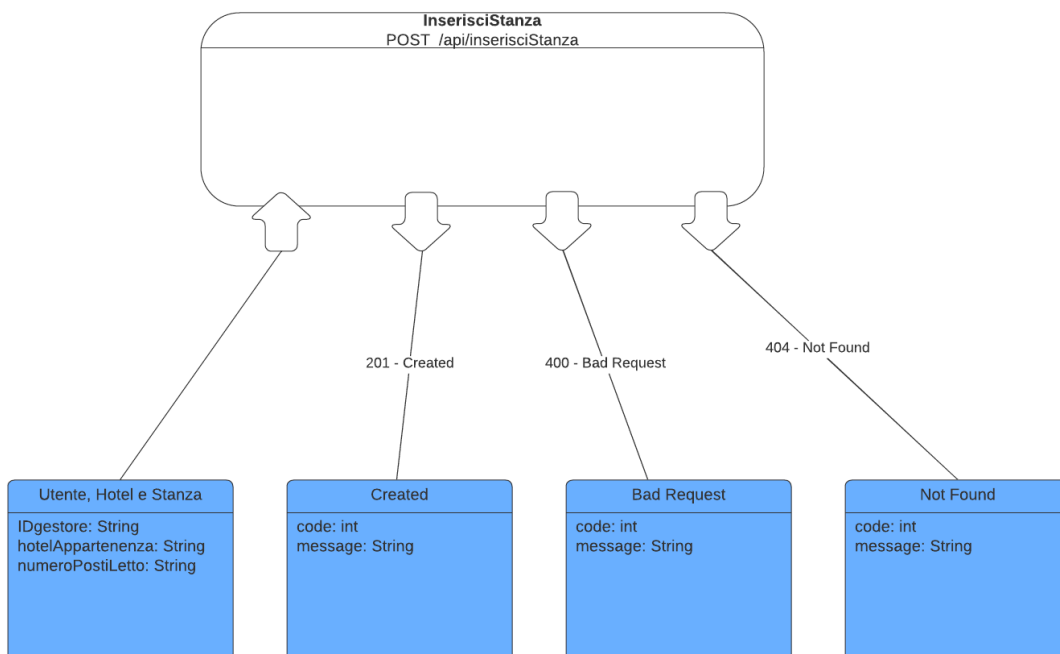
Possono poi esserci tre tipologie di response body a seconda della riuscita o meno dell'operazione.

Nel primo caso, se *code* = 201 e *message* = "Created", allora l'operazione è riuscita e non viene restituito nient'altro.

Questa casistica corrisponde quindi all'evento in cui il gestore riesce ad inserire con successo una stanza in un hotel.

Se invece *code* = 400 e *message* = "Bad Request", allora l'operazione non è riuscita a causa dell'incorrettezza di almeno uno dei parametri inseriti, o nel caso in cui l'IDutente non sia l'identificativo di un utente con l'account gestore.

Nell'ultima casistica si ha *code* = 404 e *message* = "Not Found": ciò significa che l'operazione non è riuscita in quanto l'IDutente inserito non è valido.



Elimina Account

Questa API, all'indirizzo `/api/eliminaAccount`, utilizza un metodo DELETE per permettere ad un utente di eliminare il proprio account.

La request body è rappresentata dalla risorsa Utente, che ha come attributo *IDutente*.

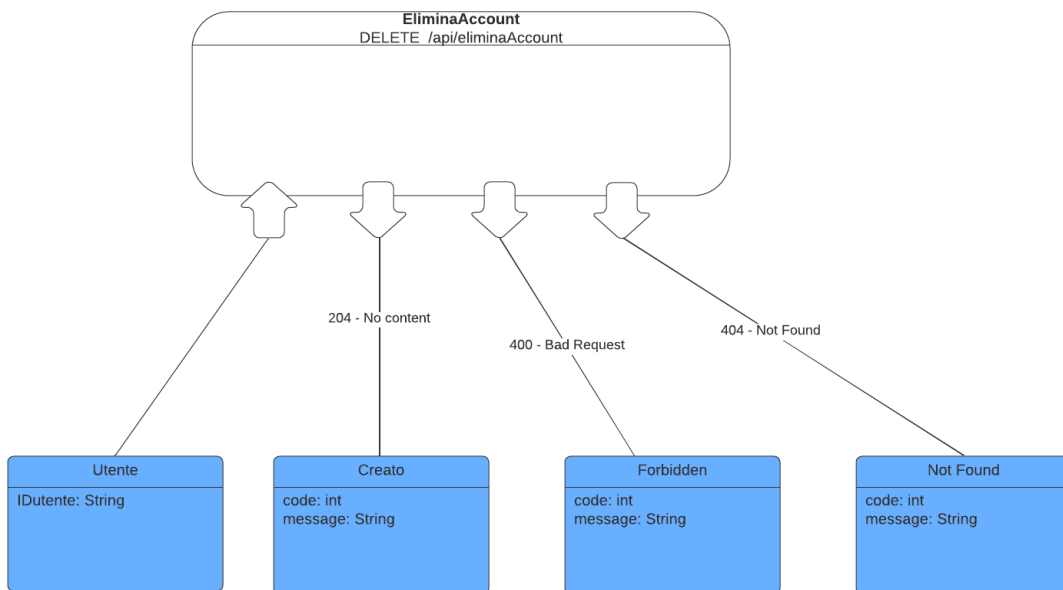
Possono poi esserci tre tipologie di response body a seconda della riuscita o meno dell'operazione.

Nel primo caso, se `code = 204` e `message = "No Content"`, allora l'operazione è riuscita e non viene restituito nient'altro.

Questa casistica corrisponde quindi all'evento in cui l'utente riesce ad eliminare con successo il proprio account.

Se invece `code = 400` e `message = "Bad Request"`, allora l'operazione non è riuscita a causa dell'incorrettezza della password inserita, richiesta per confermare l'eliminazione dell'account.

Nell'ultima casistica invece si ha `code = 404` e `message = "Not Found"`: questo evento corrisponde ad un'operazione non riuscita in quanto l'email inserita per la cancellazione non era presente nel database.



Sviluppo delle API

In questo capitolo vediamo le 8 API che abbiamo sviluppato a partire dai modelli delle risorse visti precedentemente.

Sono presenti dunque le API per la registrazione e l'accesso al sito, quelle per cercare un hotel e per gestire le proprie prenotazioni, e infine quella per eliminare il proprio account.

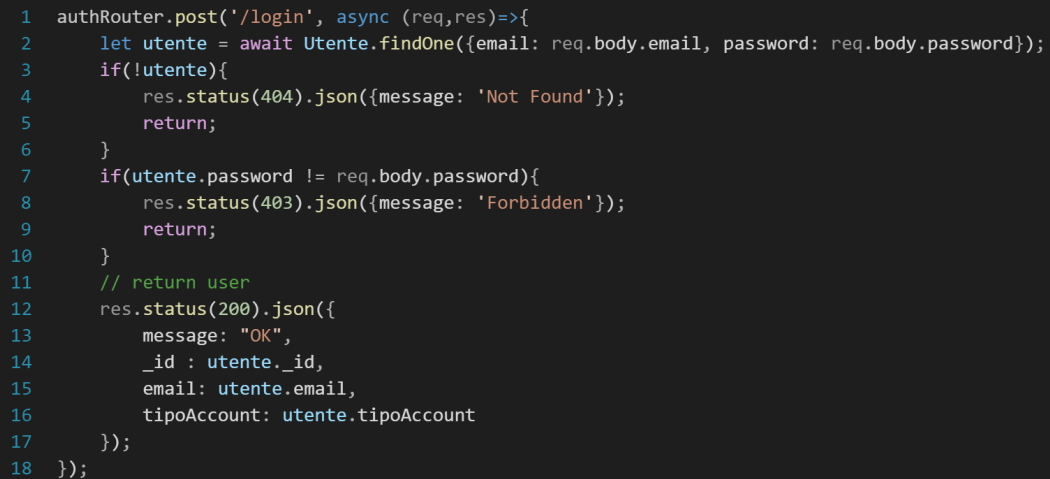
SignUp:

Questa API viene utilizzata per inserire nel database l'account di un nuovo utente. Dopo aver ricevuto la richiesta, l'input viene validato per verificare che i dati inseriti siano corretti. Nel caso in cui l'utente inserito non sia registrato, i dati vengono assegnati ad un nuovo oggetto creato basandosi sullo schema Utente. Questo oggetto viene poi salvato nel database, e alcuni dati vengono restituiti all'utente.

```
1 //add new user
2 authRouter.post('/signUp', async (req, res) => {
3   //request validation
4   if (!(validateEmail(req.body.email) && validatePsw(req.body.password))) {
5     res.status(400).json({message: 'Bad Request'});
6     return;
7   }
8   //account type validation
9   if (req.body.tipoAccount !== 'gestore' && req.body.tipoAccount !== 'cliente') {
10    res.status(400).json({message: 'Bad Request'})
11    return;
12  }
13  //check for email address in database (maybe the email is already present)
14  let utente = await Utente.findOne({email: req.body.email});
15  if (utente) {
16    res.status(409).json({message: 'Conflict'});
17    return;
18  }
19  //create a new user
20  let newUtente = new Utente({
21    email: req.body.email,
22    password: req.body.password,
23    tipoAccount: req.body.tipoAccount
24  });
25  newUtente = await newUtente.save();
26  res.status(201).json({
27    message: "Created",
28    _id: newUtente._id,
29    email: newUtente.email,
30    tipoAccount: newUtente.tipoAccount
31  });
32  });
33
34  });
```

Login:

Questa API permette ad un utente già registrato di accedere al sito utilizzando il suo account. Si verifica che l'utente esista sul database e, in caso affermativo, si procede a verificare la correttezza della password inserita. In caso di esito positivo viene restituita la risposta.



```
1  authRouter.post('/login', async (req,res)=>{
2    let utente = await Utente.findOne({email: req.body.email, password: req.body.password});
3    if(!utente){
4      res.status(404).json({message: 'Not Found'});
5      return;
6    }
7    if(utente.password != req.body.password){
8      res.status(403).json({message: 'Forbidden'});
9      return;
10   }
11   // return user
12   res.status(200).json({
13     message: "OK",
14     _id : utente._id,
15     email: utente.email,
16     tipoAccount: utente.tipoAccount
17   });
18 });
```

Ricerca:

Questa API permette ad un utente di cercare un hotel grazie a dei filtri che permettono di scegliere la provincia e il numero di persone per cui si intende prenotare.

```
1  ricercaRouter.get('/ricerca', async (req,res)=>{
2
3      let hotels = await Hotel.find({provincia : req.query.provincia});
4      if(!hotels){
5          res.status(404).json({message: "Not Found"});
6          return;
7      }
8      else{
9          let availableHotels: { _id: any; hotel: any; numeroStelle: any; }[] = [];
10         // waiting for operations in map function to complete. map iterates over hotels and for each hotel searches for available rooms
11         await Promise.all(hotels.map(async (hotel:any)=>{
12             const stanze = await Stanza.find({IDhotel: hotel._id, reserved: false, numeroPostiletto: {$eq: req.query.numeroPersone} });
13             if(stanze.length > 0){
14                 availableHotels.push({_id: hotel._id, hotel: hotel.nome, numeroStelle: hotel.numeroStelle});
15             }
16         }));
17
18         if(availableHotels.length > 0){
19             res.status(200).json({
20                 message: "OK",
21                 hotels: availableHotels
22             });
23         }
24         else{
25             res.status(404).json({message: "Not Found"});
26         }
27     }
28 });
```

Prenotazione:

Questa API permette ad un utente con l'account cliente di prenotare una stanza di un hotel.

In particolare, il cliente sceglie un hotel ottenuto tramite ricerca e poi seleziona una delle stanze disponibili per la prenotazione.

```
1  prenotazioneRouter.post('/prenotazione', async (req,res)=>{
2    if(req.body.part == 'hotel'){
3
4      if(!mongoose.isValidObjectId(req.body.IDutente)){
5        res.status(400).json({message: "Bad Request"});
6        return;
7      }
8      let user = await Utente.findOne({_id: req.body.IDutente});
9      if(!user){
10        res.status(404).json({message: "Not Found"});
11        return;
12      }
13      let hotelChoice = req.body._id;
14      if(!mongoose.isValidObjectId(hotelChoice) || req.body.numeroPersone < 1 || user.tipoAccount != "cliente"){
15        res.status(400).json({message: "Bad Request"});
16        return;
17      }
18      let hotels = await Hotel.findOne({_id: hotelChoice});
19      if(!hotels){
20        res.status(404).json({message: "Not Found"});
21        return;
22      }
23      else{
24        let stanze = await Stanza.find({hotelAppartenenza: hotelChoice, reserved: false});
25        console.log(stanze);
26        let availableStanze: {_id: any; numeroPostiletto: any;}[] = [];
27        await Promise.all(stanze.map(async (stanza:any)=>{
28          if(stanza.reserved == false){
29            availableStanze.push({_id: stanza._id, numeroPostiletto: stanza.numeroPostiletto});
30          }
31        }));
32        if(availableStanze.length > 0){
33          res.status(200).json({
34            message: "OK",
35            stanze: availableStanze
36          });
37        }
38        else{
39          res.status(404).json({
40            message: "Not Found"
41          });
42          return;
43        }
44      }
45    }
46    else if(req.body.part == 'stanza'){
47      let newPrenotazione = new Prenotazione({
48        numeroPersone: req.body.numeroPersone,
49        IDhotel: req.body.IDhotel,
50        IDstanza : req.body.IDstanza,
51        inizioSoggiorno: req.body.inizioSoggiorno,
52        fineSoggiorno: req.body.fineSoggiorno,
53        IDutente: req.body.IDutente
54      });
55      newPrenotazione = await newPrenotazione.save();
56
57      let filter = {_id: req.body.IDstanza};
58      let update = {reserved: true};
59      await Stanza.updateOne(filter, {$set: update});
60
61      res.status(201).json({
62        numeroPersone: newPrenotazione.numeroPersone,
63        inizioSoggiorno: newPrenotazione.inizioSoggiorno,
64        fineSoggiorno: newPrenotazione.fineSoggiorno
65      });
66    }
67  }
68  });
```

Elimina Account:

Questa API permette ad un utente registrato di eliminare il proprio account dal database del sito. Una volta verificata la presenza dell'utente nel database si chiede una conferma effettiva dell'eliminazione. In caso positivo si procede all'eliminazione delle eventuali prenotazioni associate all'utente oppure, nel caso gestore, degli hotel e stanze ad esso associate.

Infine, viene eliminato l'account.

```
1 deleteRouter.delete('/eliminaAccount', async (req,res)=>{
2
3   if(!Types.ObjectId.isValid(req.body.IDutente)){
4     res.status(400).json({message: "Bad Request"});
5     return;
6   }
7   let user = await Utente.findOne({_id: req.body.IDutente});
8   if(!user){
9     res.status(404).json({message: 'Not Found'});
10    return;
11  }
12  // Delete user from database -- client scenario
13  if(user.tipoAccount == 'cliente'){
14
15    let prenotazioni = await Prenotazione.find({IDutente: user._id});
16    if(!prenotazioni){
17      res.status(404).json({message: 'Not Found'});
18      return;
19    }
20    else{
21      await Promise.all(prenotazioni.map(async (prenotazione:any)=>{
22        let filter = {_id: prenotazione.IDstanza};
23        let update = {reserved: false};
24        await Stanza.updateOne(filter, {$set: update});
25        await Prenotazione.deleteOne({_id: prenotazione._id});
26      }));
27    }
28    await Utente.deleteOne({_id: user._id});
29    res.status(204).json({message: 'No Content'});
30    return;
31  }
32  }
33  // Delete user from database -- gestore scenario
34  else{
35    let hotels = await Hotel.find({IDgestore: user._id});
36    if(!hotels){
37      res.status(404).json({message: 'Not Found'});
38      return;
39    }
40    else{
41      //Delete every room with specific id hotel; then delete the hotel itself
42      await Promise.all(hotels.map(async (hotel:any)=>{
43        await Stanza.deleteMany({hotelAppartenenza: hotel._id});
44        await Hotel.deleteOne({_id: hotel._id});
45      }));
46      await Utente.deleteOne({_id: user._id});
47      res.status(204).json({message: 'No Content'});
48      return;
49    }
50  }
51  });
```

Inserisci hotel:

Questa API permette ad un utente registrato come “gestore” di inserire un nuovo hotel nel database. Una volta verificato che l’utente sia effettivamente registrato come gestore e che alcuni campi della richiesta siano validi, si procede a verificare che l’hotel inserito non sia presente nel database. In caso positivo viene creato un nuovo oggetto basato sullo schema hotel con i dati della richiesta che viene poi inserito nel database. Viene poi restituita una risposta.

```
1  inserisciHotel.post('/inserisciHotel', async (req,res)=>{
2    let id_utente = req.body.IDgestore;
3    if(!mongoose.isValidObjectId(id_utente)){
4      res.status(400).json({message: "Bad Request"});
5      return;
6    }
7    let utente = await Utente.findOne({_id: id_utente});
8    if(!utente){
9      res.status(404).json({message: "Not Found"});
10     return;
11   }
12   else{
13     if(utente.tipoAccount != "gestore" || (req.body.numeroStelle <1 || req.body.numeroStelle>5)){
14       res.status(400).json({message: "Bad Request"});
15       return;
16     }
17     let hotel = await Hotel.findOne({nome: req.body.nome});
18     if(!hotel){
19       let newHotel = new Hotel({
20         IDgestore: id_utente,
21         nome: req.body.nome,
22         numeroStelle: req.body.numeroStelle,
23         provincia: req.body.provincia
24       });
25       await newHotel.save();
26       res.status(201).json({message: "Created"});
27     }
28     else{
29       res.status(409).json({message: "Conflict"});
30       return;
31     }
32   }
33 });
```


GetIDs:

Questa API compie una duplice funzione: nel caso in cui l'utente abbia l'account cliente, essa restituisce all'utente gli ID di tutte le prenotazioni da lui effettuate. Se invece l'utente ha un account gestore, allora l'API gli restituisce tutti gli ID degli hotel da lui inseriti.

```
1 getIDsrouter.get('/getIDs/:IDutente', async (req,res)=>{
2   const IDutente = req.params.IDutente;
3   if(!mongoose.isValidObjectId(IDutente)){
4     res.status(400).json({message: "Bad Request"});
5     return;
6   }
7   let utente = await Utente.findOne({_id: IDutente});
8   if(!utente){
9     res.status(404).json({message: "Not Found"});
10  }
11  else{
12    if(utente.tipoAccount == 'cliente'){
13      //
14      let prenotazioniUtente: {_id: any; inizioSoggiorno: any; fineSoggiorno: any}[] = [];
15      let prenotazioni = await Prenotazione.find({IDutente: IDutente});
16      await Promise.all(prenotazioni.map(async (prenotazione: any)=>{
17        prenotazioniUtente.push({_id: prenotazione._id, inizioSoggiorno: prenotazione.inizioSoggiorno, fineSoggiorno: prenotazione.fineSoggiorno});
18      }));
19      if(prenotazioniUtente.length > 0){
20        res.status(200).json({
21          prenotazioniUtente: prenotazioniUtente
22        });
23      }
24      else{
25        res.status(404).json({
26          message: "Not Found"
27        });
28      }
29    }
30    else if(utente.tipoAccount == 'gestore'){
31      let hotelsGestore: {_id: any; nomeHotel: any}[] = [];
32      let hotels = await Hotel.find({IDgestore: IDutente}); // controllare nomi
33      await Promise.all(hotels.map(async (hotel: any)=>{
34        hotelsGestore.push({_id: hotel._id, nomeHotel: hotel.nome});
35      }));
36      if(hotelsGestore.length > 0){
37        res.status(200).json({
38          hotelsGestore: hotelsGestore
39        });
40      }
41      else{
42        res.status(404).json({
43          message: "Not Found"
44        });
45      }
46    }
47  }
48 });
```

Inserisci stanza:

Questa API permette ad un utente registrato come “gestore” di inserire una nuova stanza appartenente ad un hotel all’interno del database. Una volta verificato che l’utente sia effettivamente registrato come gestore, viene verificata l’esistenza dell’hotel inserito. In caso positivo si procede alla creazione di un nuovo oggetto basato sullo schema stanza i cui dati sono quelli presenti nella richiesta. L’oggetto viene poi salvato sul database.

```
1 inserisciStanza.post('/inserisciStanza', async (req, res) => {
2   let id_utente = req.body.IDutente;
3   let id_hotel = req.body.IDhotel;
4   if(!mongoose.isValidObjectId(id_utente) || !mongoose.isValidObjectId(id_hotel)){
5     res.status(400).json({message: 'Bad Request'});
6     return;
7   }
8
9   let utente = await Utente.findOne({_id: id_utente});
10  let hotel = await Hotel.findOne({_id: id_hotel});
11
12  if(!utente || !hotel){
13    res.status(404).json({message: 'Not Found'});
14    return;
15  }
16  if(utente.tipoAccount !== 'gestore'){
17    res.status(400).json({message: 'Bad Request'});
18    return;
19  }
20  let newStanza = new Stanza({
21    reserved: false,
22    numeroPostiLetto: req.body.numeroPostiLetto,
23    hotelAppartenenza: hotel._id
24  });
25  await newStanza.save();
26  res.status(201).json({message: 'Created'});
27  return;
28 });
```

Documentazione delle API

In questa sezione è possibile vedere la documentazione delle API tramite interfaccia **SwaggerUI**, con la quale si può avere una visione chiara di ogni tipologia di API che caratterizza il nostro sito.

Di seguito vi sono la schermata completa delle API e alcuni esempi più nel dettaglio.

- Documentazione API su Swagger

The screenshot displays the Swagger UI for the BellHotel API. At the top, the title 'BellHotel' is followed by version indicators '1.0.0' and 'OAS 2.0'. Below this, the base URL is specified as 'localhost/v1', and the project is identified as 'Progetto del gruppo G11 di Ingegneria del Software anno 2023/24'. A 'Schemas' dropdown menu is set to 'HTTP 1.1'. The main content is organized into four expandable sections: 'Utente', 'Hotel', 'Stanza', and 'Prenotazioni'. Each section lists its endpoints with their respective HTTP methods and descriptions. The 'Utente' section includes endpoints for signing up, logging in, deleting an account, and getting IDs. The 'Hotel' section includes endpoints for inserting a hotel and searching for one. The 'Stanza' section includes an endpoint for inserting a room. The 'Prenotazioni' section includes an endpoint for booking a room. At the bottom, a 'Models' section lists the data models: 'Utente', 'Hotel', 'Stanze', and 'Prenotazioni', each with a right-pointing arrow indicating further details.

BellHotel 1.0.0 OAS 2.0
[Base URL: localhost/v1]
Progetto del gruppo G11 di Ingegneria del Software anno 2023/24

Schemas
HTTP 1.1

Utente Gestione di account "cliente" e "gestore" ^

- POST** /api/signup Aggiungi un nuovo utente nel database v
- POST** /api/login Effettua l'accesso al sito v
- DELETE** /api/eliminaAccount Rimuovi un account dal database v
- GET** /api/getIDs(IDutente) Ottieni gli ID delle prenotazioni se sei cliente, altrimenti ottieni ID dagli hotel se sei gestore v

Hotel Prenotazioni e inserimento di hotel ^

- POST** /api/inserisciHotel Inserisci un hotel sul sito v
- GET** /api/ricerca Cerca un hotel v

Stanza Prenotazioni e inserimento di stanze ^

- POST** /api/inserisciStanza Inserisci una stanza in un hotel v

Prenotazioni Gestione delle prenotazioni ^

- POST** /api/prenotazione Prenota una stanza di un hotel v

Models ^

- Utente > <-i
- Hotel > <-i
- Stanze > <-i
- Prenotazioni > <-i

- Esempio di API di tipo POST

POST /api/login Effettua l'accesso al sito

Parameters Try it out

| Name | Description |
|--|--|
| Utente registrato * required | Example Value Model |
| object (body) | <pre>{ "email": "giorgioBianchi@gmail.com", "password": "giorgio89"}</pre> |
| Parameter content type application/json | |

Responses Response content type application/json

| Code | Description |
|------|---|
| 200 | Accesso eseguito Example Value Model <pre>{ "idUser": "6627d9dbe35700ef9a16c9b5", "email": "giorgioBianchi@gmail.com", "tipoAccount": "cliente"}</pre> |
| 403 | Forbidden |
| 404 | Not Found |

- Esempio di API di tipo DELETE

DELETE /api/eliminaAccount Rimuovi un account dal database

Parameters Try it out

| Name | Description |
|--|---|
| ID utente * required | Example Value Model |
| object (body) | <pre>{ "IDutente": "6627d9dbe35700ef9a16c9b5"}</pre> |
| Parameter content type application/json | |

Responses Response content type application/json

| Code | Description |
|------|-------------------|
| 204 | Account eliminato |
| 400 | Bad Request |
| 404 | Not Found |

- Esempio API di tipo GET

GET

/api/getIDs{IDutente} Ottieni gli ID delle prenotazioni se sei cliente, altrimenti ottieni ID degli hotel se sei gestore

Try it out

| Name | Description |
|-------------------------------|---|
| IDutente * required (path) | <input type="text" value="6627d9dbe35700ef9a16c9b5"/> |

Responses

Response content type application/json

| Code | Description |
|------|--|
| 200 | OK Example Value Model <pre>{ "IDprenotazione": "662e0b7e40365e8c5c16c9b5" }</pre> |
| 400 | Bad Request |
| 404 | Not Found |

Implementazione del Front-End

Il frontend rappresenta un'interfaccia grafica grazie alla quale è possibile usufruire delle API sviluppate a livello del backend.

Il framework utilizzato per lo sviluppo del frontend è **VueJS**.

Il sito si presenta con una schermata nella quale è possibile accedere al sito inserendo email e password: nel caso in cui l'utente non abbia ancora un account, potrà registrarsi nella pagina di SignUp.

Un utente può avere due tipologie di account, ovvero "cliente" e "gestore": a seconda del tipo, si potranno vedere due schermate differenti.

Nella prima schermata, ovvero quella riservata al cliente, sarà possibile cercare un hotel e prenotarne una stanza, mentre nella seconda schermata il gestore potrà creare un nuovo hotel e aggiungere delle stanze.

Di seguito sono mostrate le varie interfacce del sito:

- **Login**

Accesso al sito: login signUp

BellHotel

Login

Email: MarioRossi@gmail.com

Password: Abcdef1!

Accedi

- SignUp

Accesso al sito:

BellHotel

Sign Up

Email:

Password:

Conferma Password:

Tipologia Account: ▼

- Schermata cliente:

Benvenuto MarioRossi@gmail.com

Ricerca

Provincia:

Numero persone:

Risultati:

- Nome: Hotel3 Stelle: 4
- Nome: Hotel5 Stelle: 4

Stanze disponibili:

- Stanza: 66d8657caa5e135b7aedc6b6

Impostazioni

Elimina Account

Elimina Account

Ottieni ID delle prenotazioni

- **Schermata gestore:**

Benvenuto LuigiVerdi@gmail.com

[Logout](#)

Inserisci un nuovo hotel

Nome hotel:

Provincia:

Numero di stelle:

[Inserisci hotel](#)

Ottieni ID degli hotel

[Mostra](#)

Inserisci una nuova stanza

Numero di posti letto:

ID hotel:

[Inserisci stanza](#)

Impostazioni

Elimina Account

Elimina Account [Elimina](#)

Testing delle API

Il testing delle API è stato effettuato usando due librerie di NodeJS, ossia **Jest** e **supertest**. Jest è il framework principale, il quale gestisce il testing effettuando una divisione in suite di test, l'esecuzione dei vari unit test e infine la produzione di un report sul test coverage. supertest è stato usato per gestire le richieste di test ai vari endpoint.

All'interno della cartella **/test** sono salvati i file di test. Sono presenti in totale 7 suite di test, ognuna per gruppo di API simili. Corrispondono ai file presenti in **/routes** che si occupano delle API.

Dentro la suite test sono presenti tanti unit test quanti sono le possibili risposte che un' API può restituire. A seguire il report sul coverage dei test e un esempio di suite di test.

All files bellhotel/routes

94.22% Statements 212/225 88.7% Branches 55/62 100% Functions 30/30 93.83% Lines 198/211

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 |
|--------------------|-------------------------------------|-----------------------------------|---------------------------------|-------------------------------------|
| autenticazione.ts | <div><div></div></div> 93.93% 31/33 | <div><div></div></div> 88.88% 8/9 | <div><div></div></div> 100% 6/6 | <div><div></div></div> 93.54% 29/31 |
| eliminaAccount.ts | <div><div></div></div> 90.47% 38/42 | <div><div></div></div> 75% 6/8 | <div><div></div></div> 100% 6/6 | <div><div></div></div> 89.74% 35/39 |
| getIDs.ts | <div><div></div></div> 94.11% 32/34 | <div><div></div></div> 80% 8/10 | <div><div></div></div> 100% 6/6 | <div><div></div></div> 93.54% 29/31 |
| inserisciHotel.ts | <div><div></div></div> 88.46% 23/26 | <div><div></div></div> 88.88% 8/9 | <div><div></div></div> 100% 2/2 | <div><div></div></div> 88% 22/25 |
| inserisciStanza.ts | <div><div></div></div> 100% 26/26 | <div><div></div></div> 100% 7/7 | <div><div></div></div> 100% 2/2 | <div><div></div></div> 100% 25/25 |
| prenotazione.ts | <div><div></div></div> 100% 44/44 | <div><div></div></div> 100% 14/14 | <div><div></div></div> 100% 4/4 | <div><div></div></div> 100% 42/42 |
| ricerca.ts | <div><div></div></div> 90% 18/20 | <div><div></div></div> 80% 4/5 | <div><div></div></div> 100% 4/4 | <div><div></div></div> 88.88% 16/18 |

```
1 import request from 'supertest';
2 import dotenv from 'dotenv';
3 import mongoose from 'mongoose';
4 import app from '../app';
5 import { deleteHotelTest, deleteStanzaTest, deleteUtenteTest, generaHotelTest, generaStanzaTest, generaUtenteTest } from '../scripts';
6
7 dotenv.config();
8 let connection;
9 let IDgestore: string;
10 let IDhotel: string;
11 let IDstanza: string;
12 let provinciaEsistente: string;
13 let provinciaNonEsistente: string;
14 let numeroPersone: number;
15 let numeroPersoneNonCorrispondente: number;
16
17 beforeAll(async () =>{
18   mongoose.set('strictQuery',true);
19   connection = await mongoose.connect(process.env.mongodb_uri || "");
20   IDgestore = <string> (<unknown> await generaUtenteTest('emailGestore@gmail.com','Password123!','gestore'));
21   provinciaEsistente = 'ProvinciaTest';
22   provinciaNonEsistente = 'ProvinciaNonEsistenteTest';
23   numeroPersone = 4;
24   numeroPersoneNonCorrispondente = 99999999;
25   IDhotel = <string> (<unknown> await generaHotelTest(IDgestore,'nome',provinciaEsistente,3));
26   IDstanza = <string> (<unknown> await generaStanzaTest(IDhotel,numeroPersone,false));
27 });
28 afterAll(async () =>{
29   await deleteHotelTest(IDhotel);
30   await deleteUtenteTest(IDgestore);
31   await deleteStanzaTest(IDstanza);
32   mongoose.connection.close(true);
33 });
34
35 describe('Testing ricerca', () => {
36
37   test('Hotel non trovati, provincia non esistente ', async () =>{
38     const response = await request(app)
39       .get('/api/ricerca?provincia='+ provinciaNonEsistente + '&numeroPersone='+ numeroPersone.toString())
40       .set('Accept','application/json')
41
42     expect(response.statusCode).toBe(404);
43     expect(response.body).toEqual({message: 'Not Found'});
44   });
45
46   test('Hotel non trovati, numero di persone non corrisponde a nessuna capienza delle stanze ', async () =>{
47     const response = await request(app)
48       .get('/api/ricerca?provincia='+ provinciaEsistente + '&numeroPersone='+ numeroPersoneNonCorrispondente.toString())
49       .set('Accept','application/json')
50
51     expect(response.statusCode).toBe(404);
52     expect(response.body).toEqual({message: 'Not Found'});
53   });
54
55   test('Ricerca ha prodotto risultati', async () =>{
56     const response = await request(app)
57       .get('/api/ricerca?provincia='+ provinciaEsistente + '&numeroPersone='+ numeroPersone.toString())
58       .set('Accept','application/json')
59
60     expect(response.statusCode).toBe(200);
61     expect(response.body).toEqual(expect.objectContaining({
62       message: 'OK',
63       hotels: expect.any([])
64     }));
65   });
66 });
```

Repository e Deployment

Il progetto si può trovare al seguente link: <https://github.com/G11-Hotel>

Per quanto riguarda il deployment di frontend e backend abbiamo usufruito di due servizi differenti, rispettivamente di **Netlify** e **Railway**.

Su GitHub sono presenti tre repositories principali: nelle prime 2 sono disposti tutti i file che compongono backend e frontend del progetto, mentre nell'ultima sono presenti tutti i Deliverables.

Link backend su Railway: <https://backend-production-08dc.up.railway.app>

Link frontend su Netlify: <https://bellhotel-frontend.netlify.app>

In caso backend o frontend non siano online si possono seguire le istruzioni presenti sulla cartella nel file README.md per avviare il progetto in locale.