



Dipartimento di Ingegneria e Scienza dell'Informazione
Anno formativo 2022/2023

Percorso di studio: Scienze e Tecnologie Informatiche
Attività didattica: Ingegneria del Software



Progetto: “All About Trento”
Titolo del documento: Sviluppo Applicazione
Gruppo: T07

Allievi: Manuel Vettori, Emanuele Nardelli, Zappacosta Lorenzo

INDICE

Scopo del Documento	3
1. User Flows	3
2. Application Implementation and Documentation	5
2.1 Project Structure	5
2.2 Project Dependencies	6
2.3 Project Data or DB	6
2.4 Project APIs	8
2.4.1 Resources Extraction from the Class Diagram	8
2.4.2 Resources Models	9
2.5 Sviluppo API	11
2.5.1 Create User (Registrazione)	11
2.5.2 Login	11
2.5.3 Inserimento Recensione	12
2.5.4 Inserimento POI	12
2.5.5 Eliminazione POI	13
2.5.6 Modifica POI	
3. API Documentation	
4. FrontEnd Implementation	
5. GitHub Repository and Deployment Info	
6. Testing	

SCOPO DEL DOCUMENTO

Attraverso questo documento si riportano tutte le informazioni che sono necessarie per lo sviluppo di una parte dell' applicazione All About Trento.

In particolare, sono presenti tutti gli strumenti necessari per realizzare i servizi di gestione dei punti d'interesse (POI) dell' applicazione All About Trento.

Partendo dalla descrizione degli *User Flows* legate al ruolo del 'Gestore' (di seguito denominato 'G') dell'applicazione, il documento prosegue con la presentazione delle API necessarie (tramite l'*API Model* e il Modello delle risorse) per poter visualizzare, inserire e modificare sia i punti d'interesse che i vari utenti necessari all'applicazione All About Trento.

Le APIs fornite per interagire con l'applicazione saranno accompagnate da una descrizione in linguaggio naturale per aiutare la comprensione del loro utilizzo.

Nel documento è stata riportata la documentazione, i test effettuati e una descrizione delle funzionalità fornite per ogni API realizzata.

1. USER FLOWS

In questo capitolo del documento di sviluppo vengono mostrati e descritti gli "*user flows*" per quanto riguarda il ruolo del 'Gestore' della nostra applicazione.

La figura sottostante (Figura 1) descrive l'*user flow* relativo alla gestione delle informazioni inerenti ai punti d'interesse (POI) e ai vari utenti della nostra applicazione.

Il Gestore 'G' può, in ogni momento, accedere al sistema e consultare le liste riguardanti i vari punti d'interesse e gli utenti loggati al sito.

Il Gestore può inserire nuovi POI e utenti in caso di necessità dopo aver compilato il form di inserimento.

Per lo stesso principio, il Gestore può anche modificare o addirittura eliminare POI e utenti.

Oltre a queste funzionalità, il Gestore si occupa di gestire anche la parte delle recensioni che vengono effettuate dagli utenti.

All'interno della Figura 1 presentiamo anche la relazione tra le varie azioni che l'utente 'Gestore' può fare e le features descritte in Sezione 2.

Allo scopo di poter apprendere meglio l'immagine e il suo contenuto è stata realizzata una legenda che descrive i simboli usati nello user flow e anche presentata in Figura 1.

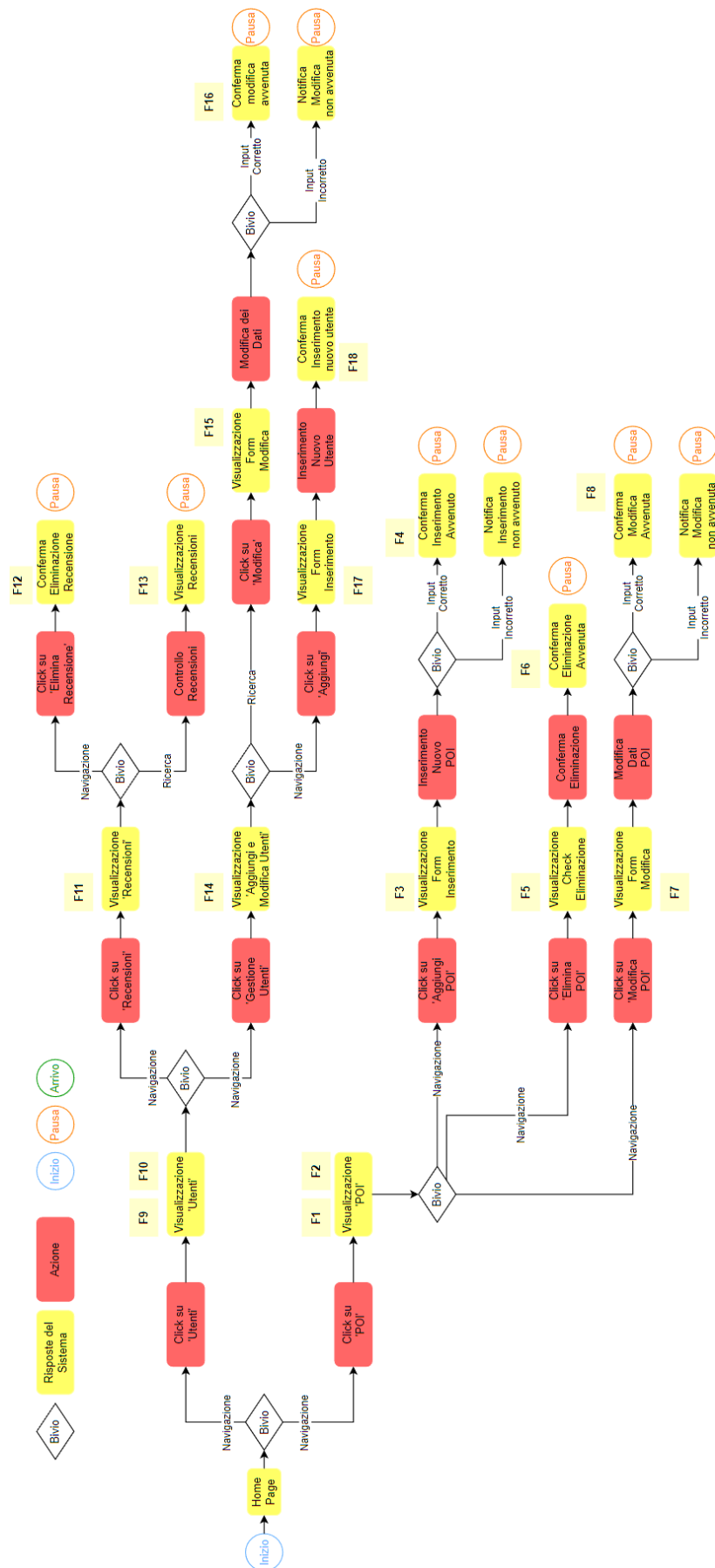


Figura 1. User Flow per il Gestore 'G'

2. APPLICATION IMPLEMENTATION AND DOCUMENTATION

Nei capitoli precedenti abbiamo identificato le varie features che devono essere implementate per la nostra applicazione con un'idea di come l'utente finale può utilizzarle nel suo flusso applicativo.

Queste funzionalità seguono un preciso flusso definito dallo *User Flow* e comporta diversi comportamenti secondo il quale il sistema deve adattarsi a rispondere in base all'azione eseguita

L'applicazione è stata sviluppata utilizzando *NodeJS* e *VueJS*.

Per la gestione dei dati abbiamo utilizzato *MongoDB*.

2.1 PROJECT STRUCTURE

La struttura del progetto è suddivisa in 3 directory principali (vedi Figura 2):

- directory "**APP**"; vengono definite le API del sito All About Trento e tutti i moduli utilizzati per l'implementazione
- directory "**UPI**"; viene definita l'interfaccia lato utente del servizio
- directory "**PHOTOS**"; vengono caricate tutte le immagini relative ad ogni funzionalità.

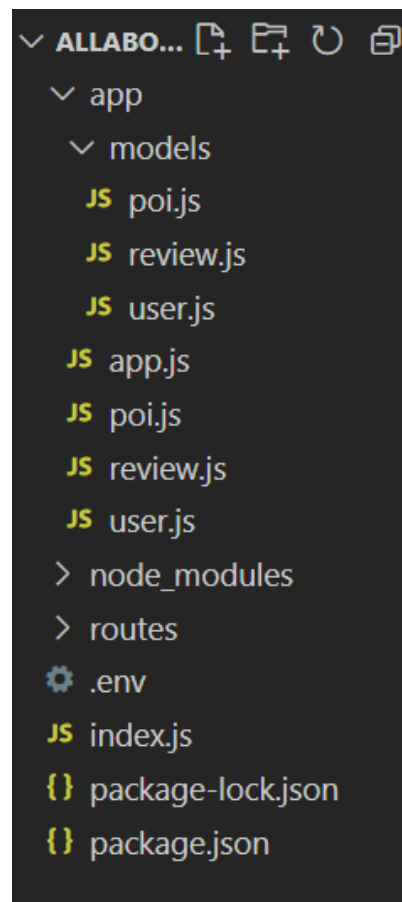


Figura 2. Struttura Codice Sorgente

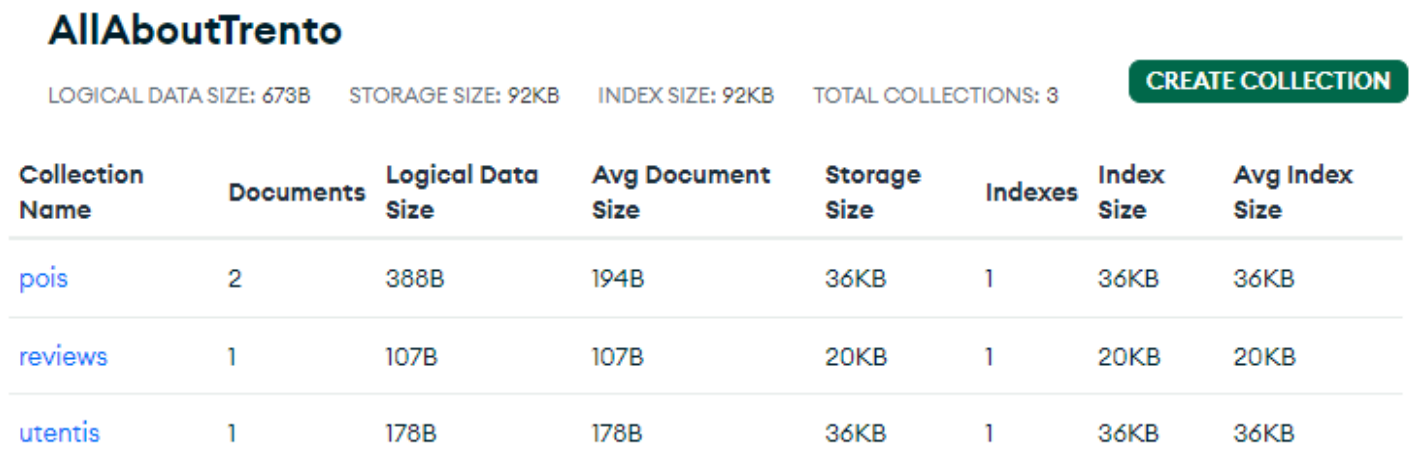
2.2 PROJECT DEPENDENCIES

I seguenti moduli *Node* sono stati utilizzati e aggiunti al file *Package.json*:

- MongoDB
- Cors
- Mongoose
- Dotenv
- Express
- Multer
- Supertest
- JsonWebToken
- Vue
- Vue-Router

2.3 PROJECT DATA OR DB

Per la gestione dei dati del sito sono state definite 3 principali strutture dati su cui operare: una collezione di “POI”, una collezione di “Utenti” e una collezione di “Recensioni” (vedi sotto Figura 3).



Collection Name	Documents	Logical Data Size	Avg Document Size	Storage Size	Indexes	Index Size	Avg Index Size
pois	2	388B	194B	36KB	1	36KB	36KB
reviews	1	107B	107B	20KB	1	20KB	20KB
utentis	1	178B	178B	36KB	1	36KB	36KB

Figura 3. Collezione Dati usati nell'applicazione

Rappresentiamo, dunque, gli utenti, i punti d'interesse e le recensioni definendo i tipi di dato mostrati nella Figura 4, nella Figura 5 e nella Figura 6:

```
_id: ObjectId('6399a105490ca809bc399b20')
nome: "Federico"
cognome: "Verdi"
email: "federico.verdi@studenti.unitn.it"
numTelefono: "3385695622"
password: "Federico.Verdi"
ruolo: "gestore"
__v: 0

_id: ObjectId('6395e28933e65f41c8bda54b')
nome: "Marco"
cognome: "Rossi"
email: "marco.rossi@studenti.unitn.it"
numTelefono: "3253565185"
password: "Marco.Rossi"
ruolo: "utente"
__v: 0
```

Figura 4. Tipo di Dato “Utente”

```
_id: ObjectId('6394ad8a19897b010dabcd67')
nome: "duomo"
tipologia: "monumento"
descrizione: "Duomo di trento"
posizione: "piazza duomo"
stato: "aperto"
orari_apertura: "10-19"
immagini: "https://upload.wikimedia.org/wikipedia/commons/9/98/Trento-Piazza_del_..."
__v: 0
```

Figura 5. Tipo di Dato “POI”

```
_id: ObjectId('63999c44055d491123a244cb')
titolo: "Recensione cattedrale San Vigilio"
descrizione: "Al momento della visita era in restauro l'abside (visto molti anni fa)..."
valutazione: 4
__v: 0
```

Figura 6. Tipo di Dato “Recensione”

2.4 PROJECT APIs

2.4.1 RESOURCES EXTRACTION FROM THE CLASS DIAGRAM

Dal diagramma delle classi, ricaviamo 3 risorse: lo *'user'*, la *'recensione'* e il *'POI'*.

Ognuna di queste risorse ha i propri parametri, che vanno a definire nello specifico quella determinata risorsa e che la rappresenteranno all'interno di una realtà.

Da queste risorse otteniamo, inoltre, le funzionalità collegate ad esse, definite anche loro come risorse.

Si noti che le risorse *'placeReview'* e *'findByTags - Review'* sono collegate da più risorse; entrambe condividono la risorsa *'recensione'*, la prima utilizza anche la risorsa *'user'* mentre la seconda è associata alla risorsa *'POI'*.

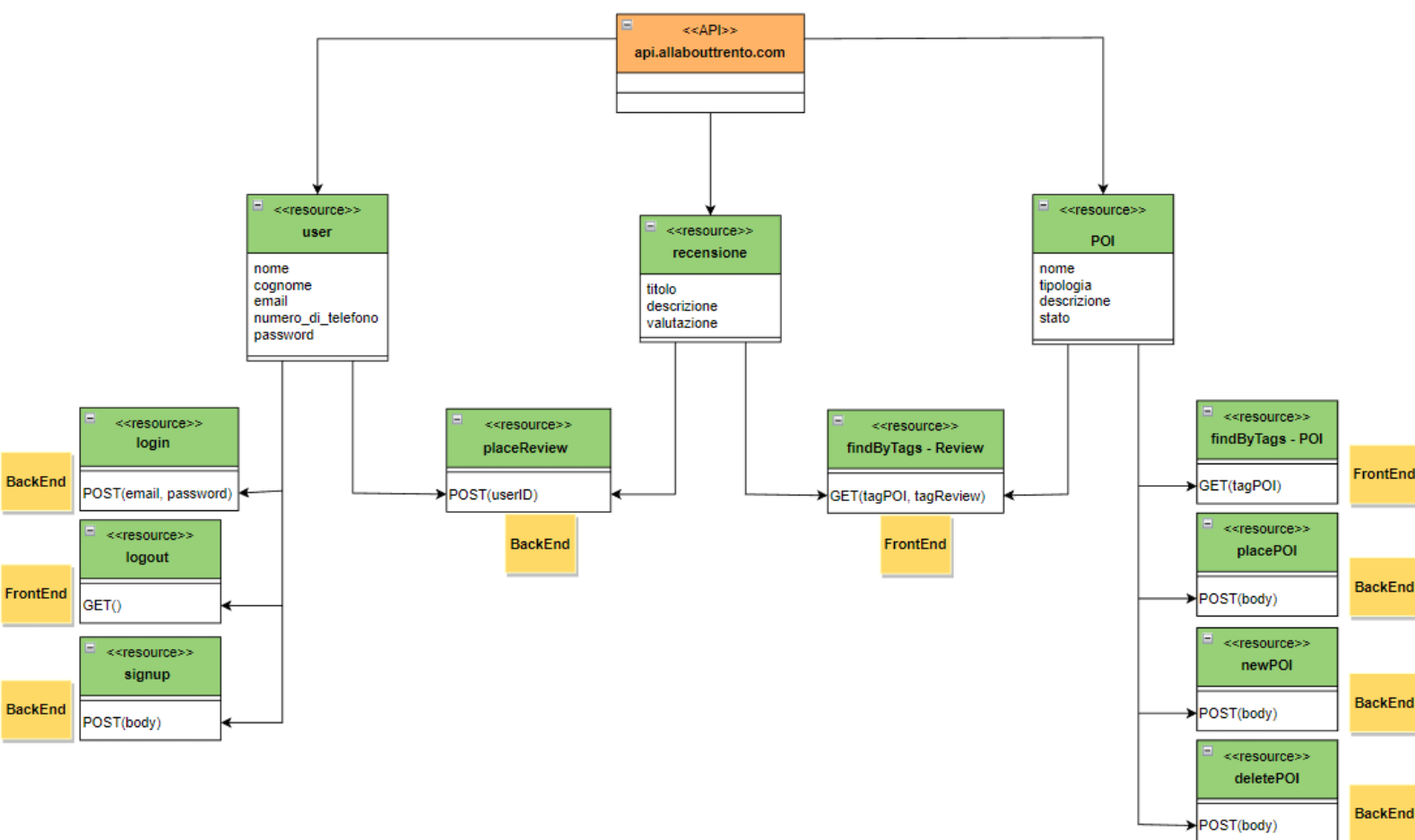


Figura 7. Diagramma delle Risorse

2.4.2 RESOURCE MODELS

Una volta estratte le risorse dal diagramma delle classi, andiamo a costruire il *Resources Model*.

Dalle 3 risorse principali estratte ('*user*', '*recensione*' e '*POI*') creiamo le API collegate ad esse.

Ogni API rappresenta una funzionalità collegata a quella specifica risorsa.

Una volta associate alla loro risorsa, le API creano ulteriori risorse che rappresentano cosa fa nello specifico quella specifica funzione.

Dallo schema si possono notare 2 risorse specifiche: '*ERROR*' e '*UNAUTHORIZED*'.

Queste 2 risorse sono associate a quasi tutte le API in quanto rappresentano delle determinate situazioni (in questo caso, errore o azione non autorizzata) che possono accadere durante l'utilizzo di una funzione.

All'interno del nostro schema, vengono presentate le seguenti API:

- **CREATE USER** (*signup*)
- **LOGIN USER** (*login*)
- **POI LIST** (*findByTags - POI*)
- **ADD POI** (*placePOI*)
- **MODIFY POI** (*newPOI*)
- **ELIMINATE POI** (*deletePOI*)
- **POI REVIEW LIST** (*findByTags - Review*)
- **ADD REVIEW** (*placeReview*)

Come spiegato nel punto precedente (vedi 2.4.1 *RESOURCES EXTRACTION FROM THE CLASS DIAGRAM*), le risorse '*placeReview*' e '*findByTags - Review*' condividevano più risorse.

Questo è stato riportato anche durante la creazione del *Resources Model*; difatti, alle 2 API definite per queste risorse, vengono associate più risorse che servono al corretto funzionamento della funzione.

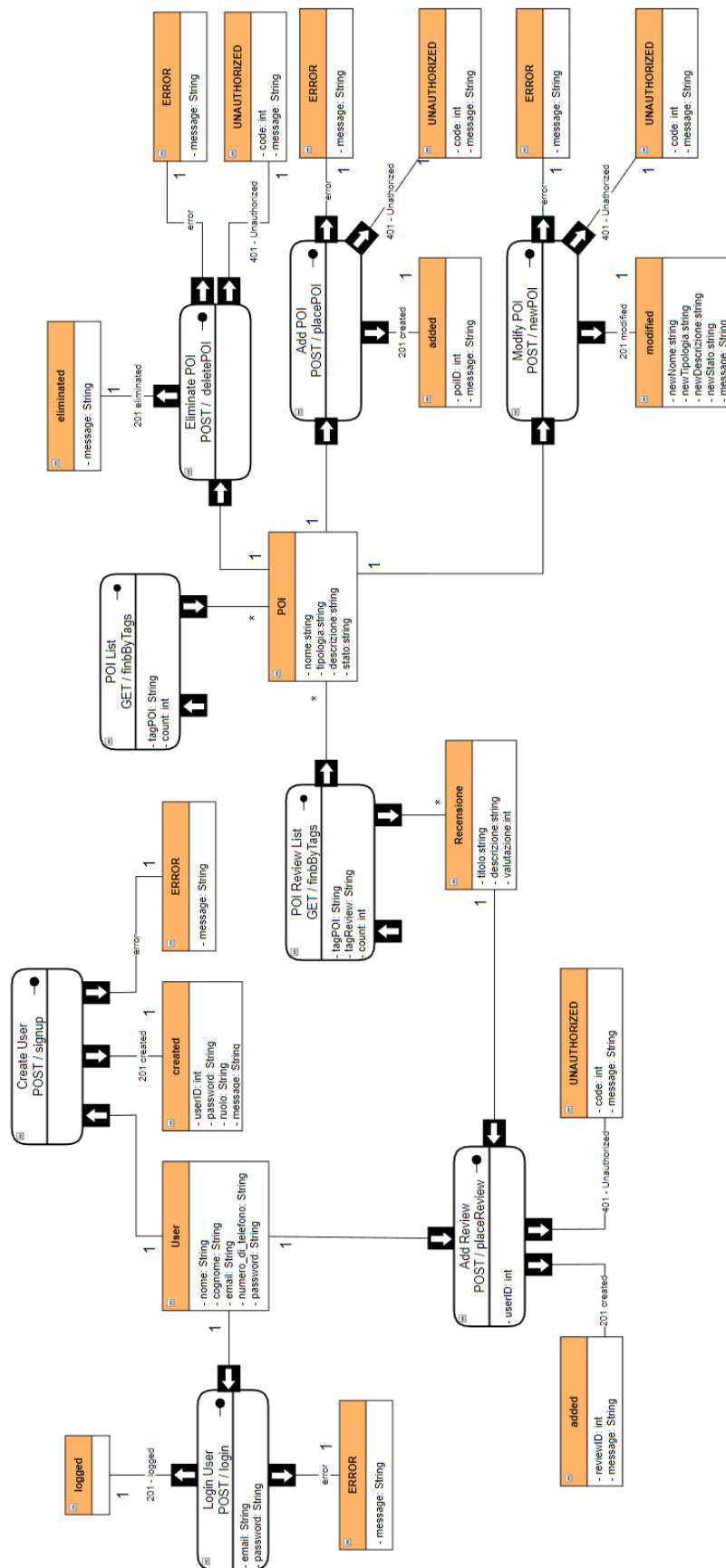


Figura 8. Diagramma delle API (Resources Model)

2.5 SVILUPPO API

Vengono elencate le varie API implementate per soddisfare i vari requisiti del sistema dal punto di vista dell'utente.

2.5.1 CREATE USER (Registrazione)

Questa API permette al sistema di registrare nuovi utenti all'interno del database.

Viene creato un nuovo oggetto di tipo 'Utente' e, tramite il form di inserimento, vengono inserite tutti i dati riguardanti quello specifico utente.

Ritorna un messaggio di avvenuta registrazione altrimenti un messaggio di errore qualora qualcosa fosse andato storto.

```
//post a new utente
router.post('/', async (req, res) => {
  let utenti = new Utenti({
    nome : req.body.nome,
    cognome : req.body.cognome,
    email : req.body.email,
    numTelefono : req.body.numTelefono,
    password : req.body.password,
    ruolo : req.body.ruolo
  });

  utenti = await utenti.save();
  let utentiId = utenti.id;
  console.log('Utente saved successfully');
  res.location("/utentis/" + utentiId).status(201).send();
});
```

Figura 9. Inserimento di un nuovo Utente

2.5.2 LOGIN

Questa API viene utilizzata per gestire il Login degli utenti all'interno del sito.

Viene effettuato il controllo sulle credenziali inserite (email e password), ritorna lo status di 'logged' se le credenziali sono corrette altrimenti un messaggio di errore qualora qualcosa andasse storto.

```
function login() {
  fetch(HOST+"/autenticazione", {
    method: "POST",
    headers: { "Content-Type": "application/json" },
    body: JSON.stringify({ email: email.value, password: password.value }),
  })
  .then((resp) => resp.json())
  .then(function (data) {
    setLoggedUser(data);
    emit("login", loggedUser);
    return;
  })
  .catch((error) => console.error(error));
}

function logout() {
  clearLoggedUser();
}
```

Figura 10. Login (e Logout)

2.5.3 INSERIMENTO RECENSIONE

L'API qui riportata permette l'inserimento di una recensione da parte dell'utente registrato.

Viene creato un nuovo oggetto di tipo 'Review' e vengono salvate all'interno del database tutte le informazioni immesse nel form di inserimento.

Ritorna un messaggio di avvenuto inserimento altrimenti un messaggio di errore qualora qualcosa fosse andato storto.

```
//post a new review
router.post('/', async (req, res) => {
  let review = new Review({
    titolo : req.body.titolo,
    descrizione : req.body.descrizione,
    valutazione : req.body.valutazione
  });

  review = await review.save();
  let reviewId = review.id;
  console.log('Review saved successfully');
  res.location("reviews/" + reviewId).status(201).send();
});
```

Figura 11. Inserimento di una Recensione

2.5.4 INSERIMENTO POI

Questa API permette l'inserimento di un nuovo POI all'interno del sistema.

Viene creato un nuovo oggetto di tipo 'POI' e vengono salvate all'interno del database tutte le informazioni immesse nel form di inserimento.

Ritorna un messaggio di avvenuto inserimento altrimenti un messaggio di errore qualora qualcosa fosse andato storto.

```
//post a new poi
router.post('/', async (req, res) => {
  let poi = new Poi({
    nome : req.body.nome,
    tipologia : req.body.tipologia,
    descrizione : req.body.descrizione,
    posizione: req.body.posizione,
    stato : req.body.stato,
    orari_apertura : req.body.orari_apertura,
    immagine : req.body.immagine
  });

  poi = await poi.save();
  let poiId = poi.id;
  console.log('Poi saved successfully');
  res.location("pois/" + poiId).status(201).send();
});
```

Figura 12. Inserimento di un POI

2.5.5 ELIMINAZIONE POI

L'API riportata rappresenta la funzione di eliminazione di un POI all'interno del sistema.

Viene richiesto quale POI eliminare e, qualora il POI esistesse, viene cancellato e ritorna un messaggio di avvenuta eliminazione.

Se il POI non venisse trovato ritorna un messaggio di errore che enuncia il mancato ritrovamento del POI all'interno del sistema.

```
//delete a poi
router.delete('/:id', async (req, res) => {
  let poi = await Poi.findById(req.params.id).exec();
  if (!poi) {
    res.status(404).send()
    console.log('Poi non trovato')
    return;
  }
  await poi.deleteOne()
  console.log('Poi rimosso')
  res.status(204).send()
});
```

Figura 13. Eliminazione di un POI

2.5.6 MODIFICA POI

Figura 14. Modifica di un POI

3. API DOCUMENTATION

4. FRONTEND IMPLEMENTATION

5. GITHUB REPOSITORY AND DEPLOYMENT INFO

6. TESTING