

# Sviluppo

Progetto:

*4SPOT*

Titolo del documento:

*Documento di architettura*

Document info:

Doc. name	Sviluppo
Doc. number	D4
Description	<i>Il documento include user flows, api doc, codice e testing.</i>

# Scopo del documento

Il presente documento ha lo scopo di analizzare da un punto di vista implementativo il Progetto 4Spot, in modo da garantire un'adatta comprensione del suo funzionamento.

Le parti trattate sono:

- User Flow
- Application documentation
- API documentation
- FrontEnd implementation
- GitHub repository and Deployment info
- Testing

# 1. User Flows

In questa sezione vengono analizzate da un punto di vista astratto le interazioni Utente - Sistema. In particolare, viene descritto il previsto flusso di utenza dell'applicazione completa.

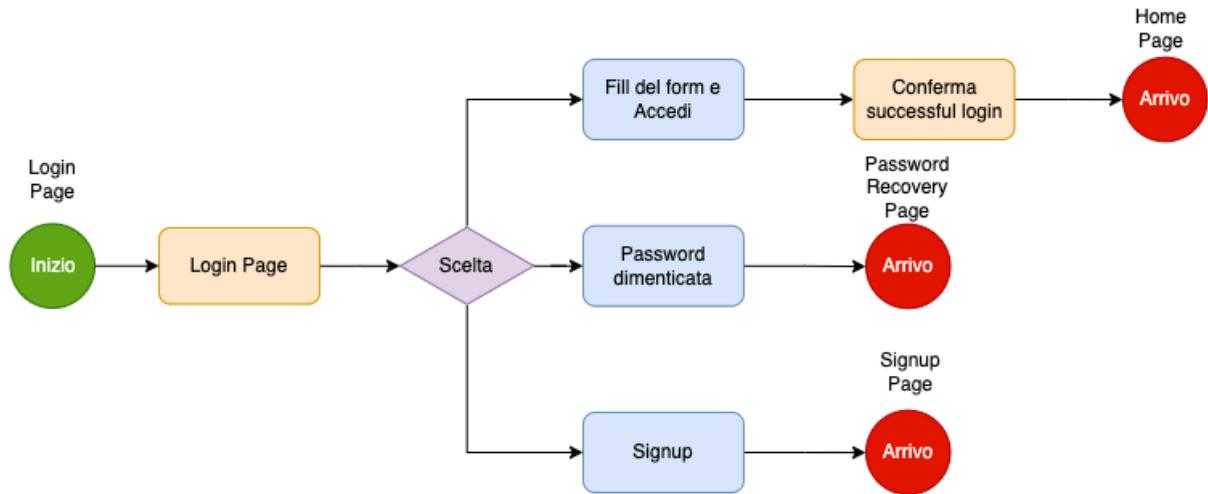
Non vengono pertanto trattate tematiche tecniche, in modo da garantire un approccio legato al Design e alla User Experience.

## Legenda

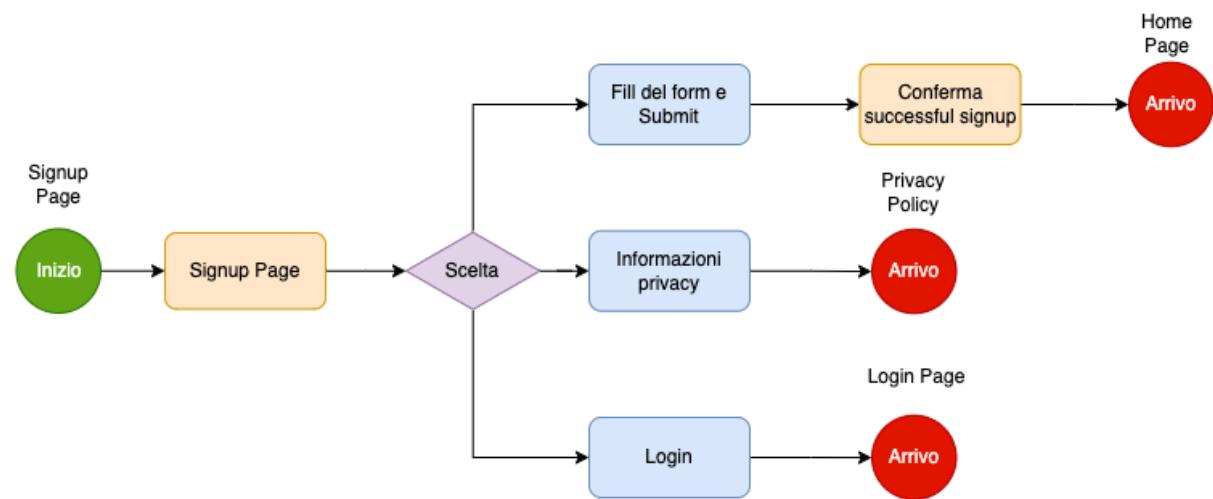


## Pagine di Accesso

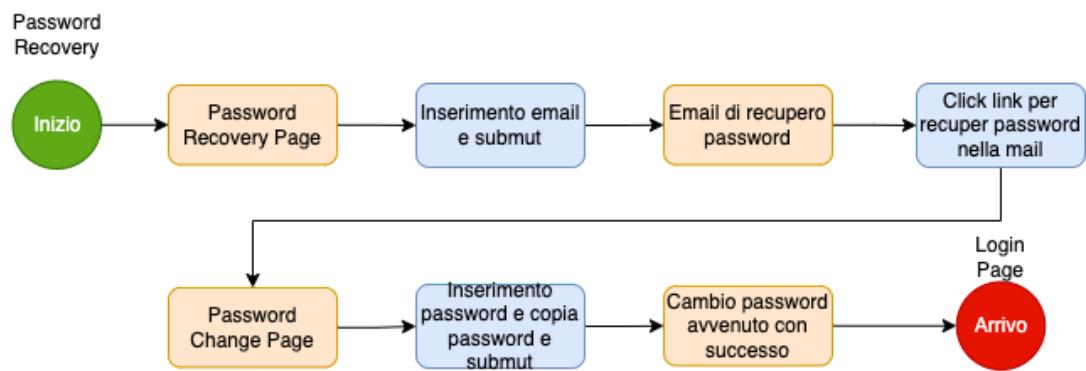
### Pagina di Login



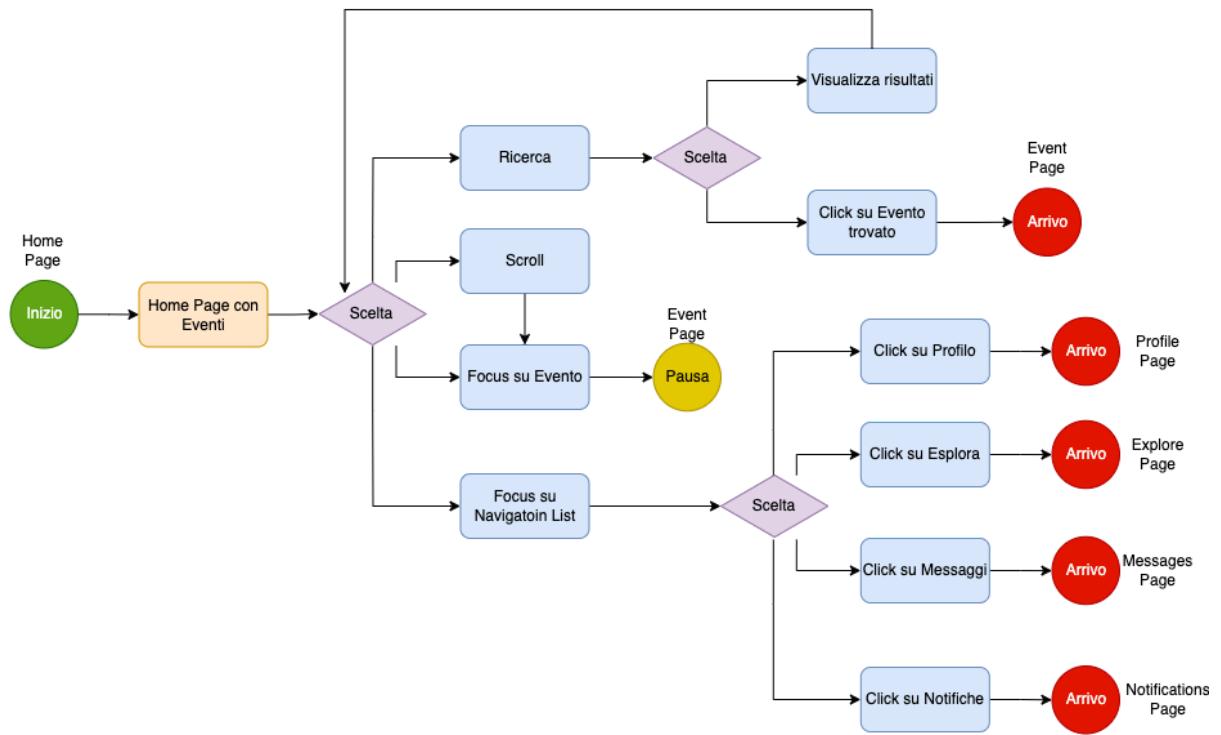
## Pagina di Signup



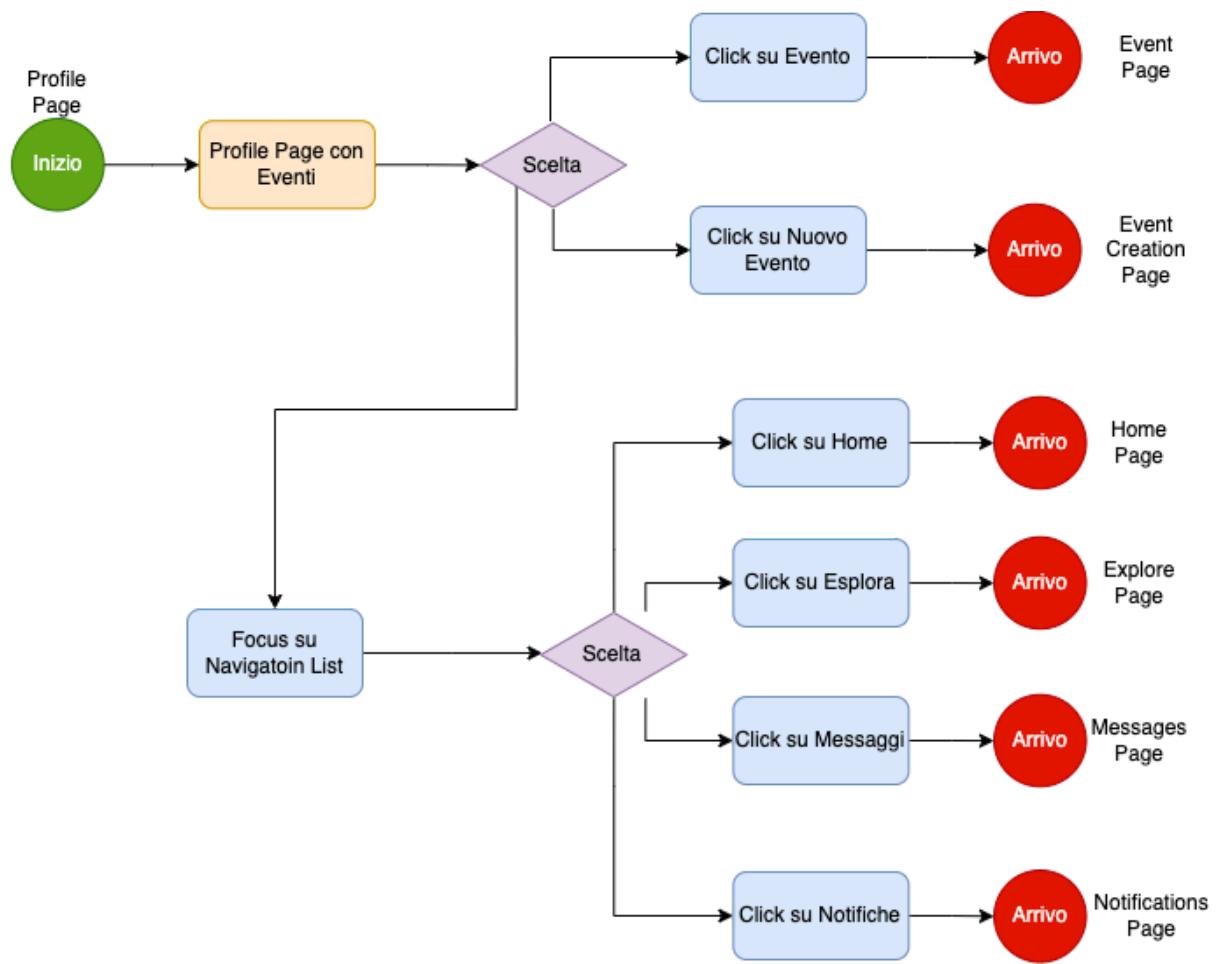
## Pagina di Password Recovery



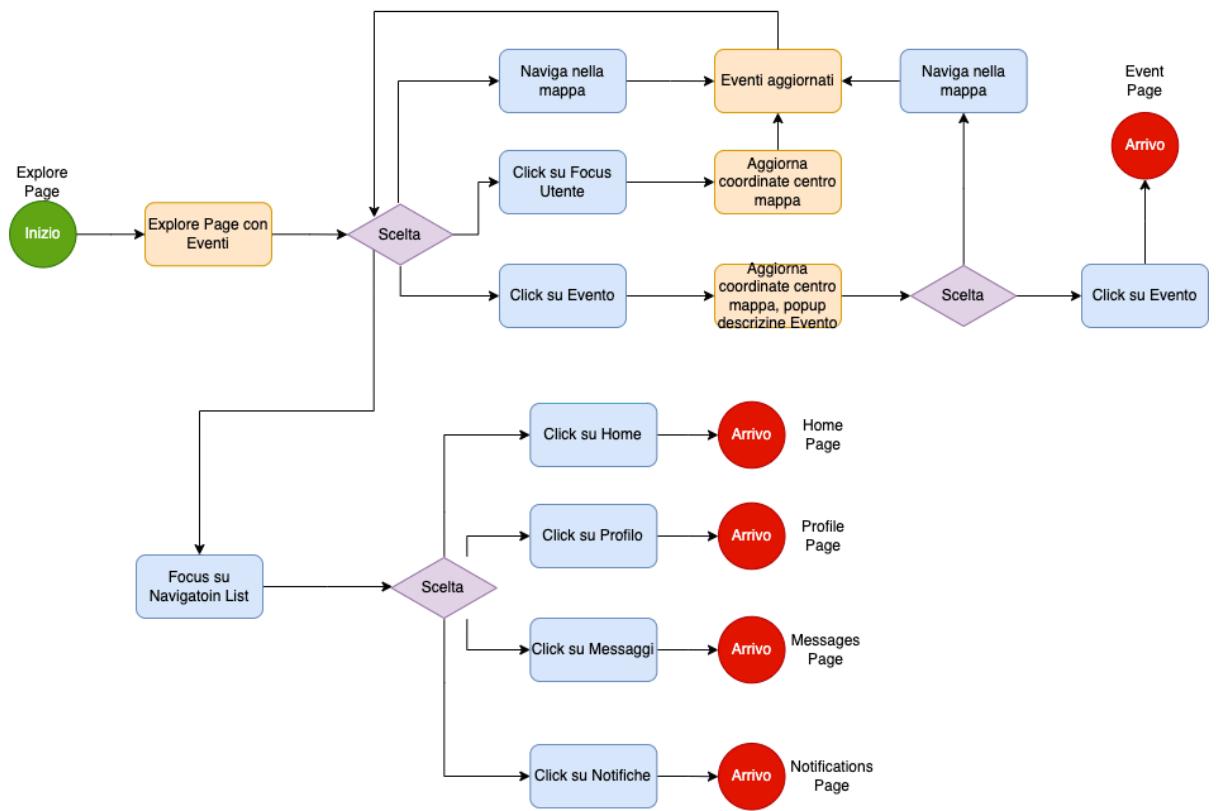
## Home



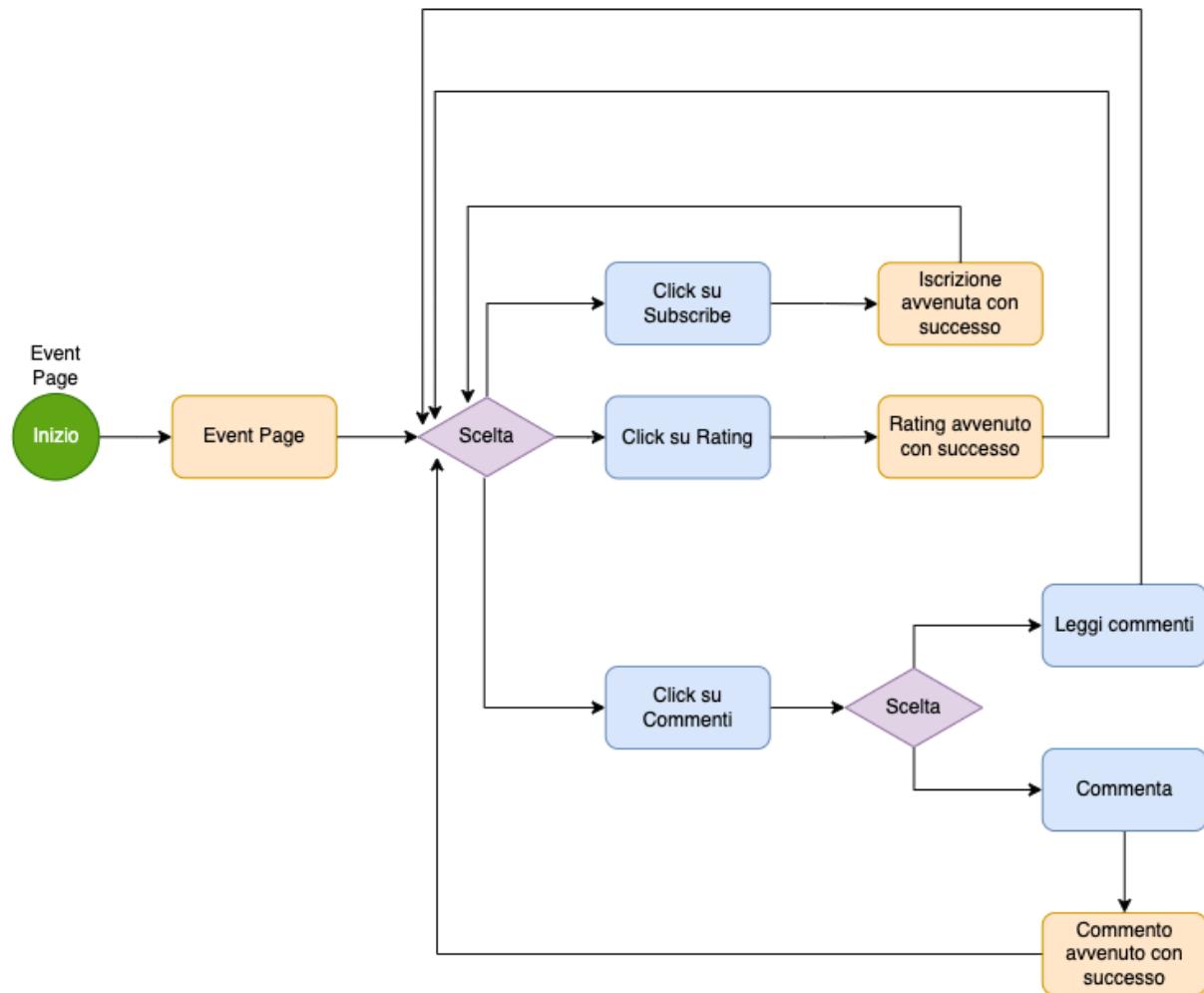
## Profilo



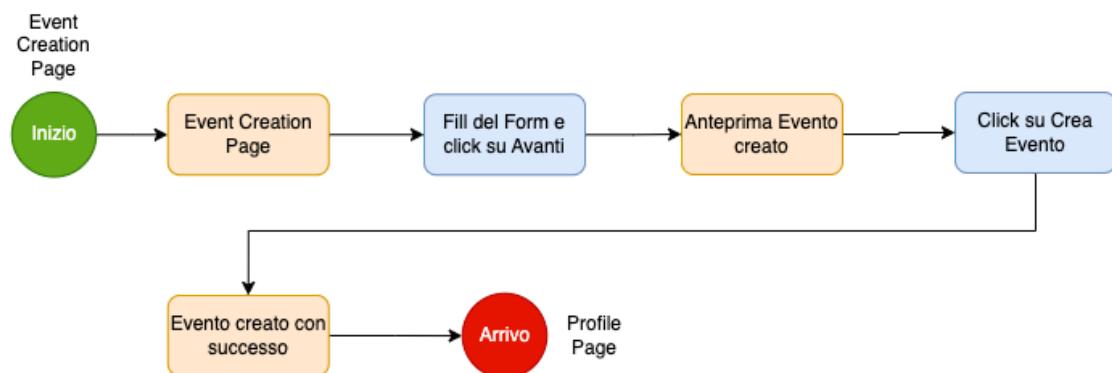
## Esplora



## Evento



## Creazione Evento

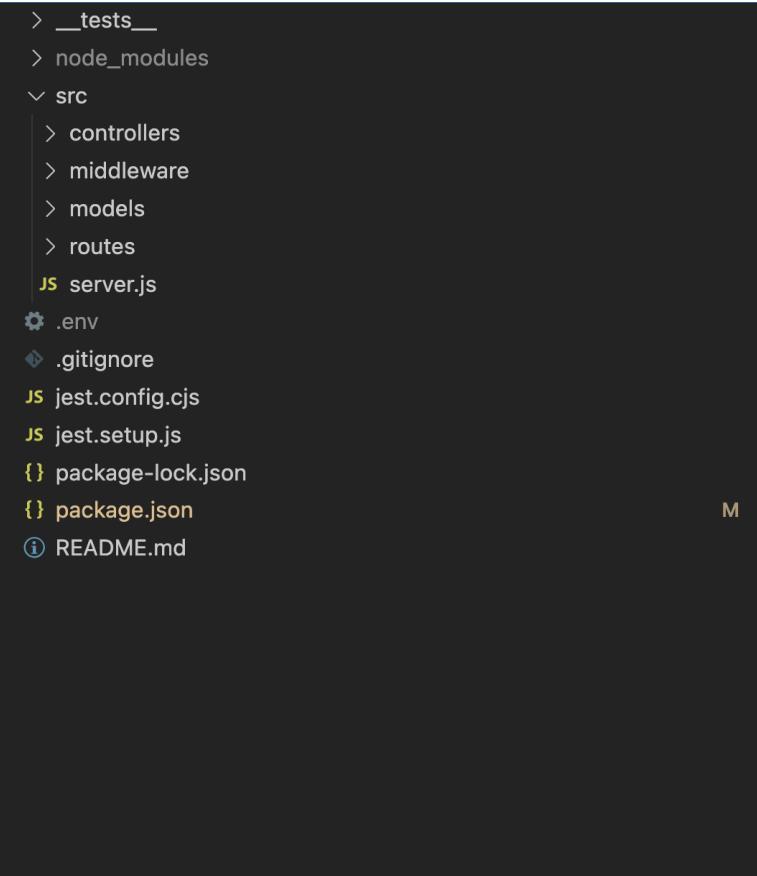


## 2. Application Documentation

In questa sezione vengono descritte varie caratteristiche implementative del prodotto. In particolare, si fornisce una descrizione di struttura e definizione delle API.

### Struttura del Progetto

BackEnd



```
> __tests__  
> node_modules  
✓ src  
| > controllers  
| > middleware  
| > models  
| > routes  
| JS server.js  
⚙ .env  
❖ .gitignore  
JS jest.config.cjs  
JS jest.setup.js  
{ } package-lock.json  
{ } package.json  
ⓘ README.md
```

Tralasciando i vari file e cartelle necessari per il corretto funzionamento di npm, il progetto ha una directory principale, `src`, in cui risiede il codice, e `__test__`, per le Unit Test.

In `src` ci sono quattro sottocartelle:

- `controllers`, per la logica delle API
- `middleware`, in cui sono definite funzioni specifiche per determinate task, spesso in comune fra varie routes
- `models`, in cui risiedono i vari modelli del Database
- `routes`, per le route delle API

e il file `server.js` che inizia l'ascolto.

## FrontEnd

The screenshot shows a file explorer window with the following structure:

```
~/Uni/IngegneriaSoftware/Project/frontend/app/node_modules
  ↘ app
    ↘ public
      > assets
      ★ favicon.ico
      <> index.html
    ↘ src
      > arcades
      > assets
      > components
      > middlewares
      > states
      > styles
      ▼ App.vue
      JS main.js
    .env
    .gitignore
    B babel.config.js
    {} jsonconfig.json
    {} package-lock.json
    {} package.json
    ⓘ README.md
    JS vue.config.js
```

Come si può notare la struttura ricalca quella di default di Vue.js. Le parti interessanti sono in `public`, dove sono inseriti i file raggiungibili a runtime dal client, tra cui le immagini (in `assets`), e `src`.

I due file in `src` sono `App.vue`, come di default (ovviamente adattato all'applicazione), e il `main.js`, l'entry point.

Le cartelle in `src` sono:

- `arcades`: in cui sono inseriti alcuni json per il test della mappa
- `assets`: come in `public`, per le immagini, caricate staticamente
- `components`: tutti gli altri file `.vue`
- `middlewares`: per funzioni specifiche
- `states`: in cui viene definito codice utilizzato per la gestione e il mantenimento di dati in locale
- `styles`: file `.css` utilizzati dalle componenti Vue

## Dipendenze

### BackEnd

- Sendgrid mail
- Bcrypt
- Body-parser
- Cors
- Dotenv
- Express
- Express-validator
- Json Web Token
- Mongoose
- Node-fetch
- Jest
- Supertest
- swagger-ui-express

### FrontEnd

- Vue
- @vueuse/core
- core-js
- Pinia
- vue-router
- @babel/core
- @babel/eslint-parser
- @vue/cli-plugin-babel
- @vue/cli-plugin-eslint
- @vue/cli-service
- Eslint
- eslint-plugin-vue
- Leaflet
- Vue Leaflet
- @types/leaflet

## Environment Variables

### Variabili di ambiente BackEnd

- **DB\_HOST**: Per Debug
- **DB\_USER**: Per Debug
- **DB\_PASS**: Per Debug
- **PORT**: Porta del BackEnd in ascolto

- **MONGODB\_URI**
- **JWT\_SECRET**: Secret per hashing JWT
- **FRONTEND\_HOST**: Nome Host FrontEnd
- **FRONTEND\_PORT**: Port FrontEnd
- **SENDER\_MAIL**: Necessita account Sendgrid
- **SENDGRID\_API\_KEY**: Necessita account Sendgrid

Variabili di ambiente FrontEnd

- **BACKEND\_HOST**: Nome dell'host del BackEnd
- **BACKEND\_PORT**: Numero porta BackEnd
- **DEFAULT\_EVENT\_IMAGE**: Per Evento fantoccio

## Database

In questo paragrafo viene esposta la struttura del DB, per come è stato utilizzato in fase di sviluppo. Quasi tutti i modelli qui sotto elencati sono utilizzati o utilizzabili attraverso richieste con Postman. Sotto sono riportati tre esempi: Stakeholder, Evento e RecoveryToken.

Struttura

```
comments  
currencies  
events  
eventtypes  
locations  
ratings  
recoverytokens  
settings  
stakeholders  
supereventtypes  
tickets
```

## Stakeholder

Questo modello segue la classe esposta nella sezione “Progetto API”. Si noti come l’hash della password venga salvato in base64, in modo da poterlo manipolare come stringa.

```
_id: ObjectId('65ad55e0dc147f557665d3c6')
username: "gio"
email: "gpettinacci5@gmail.com"
hashPass: "$2b$10$0YYTexzd3zPsPGdGzl3R0.AOSwT6cl8J30gX1s9wmwgYUccCPgNFi"
hasAcceptedTerms: true
▶ followers: Array (empty)
▶ following: Array (empty)
__v: 0
```

## Event

Questo evento segue la dichiarazione del Diagramma delle Classi del Delivery 3. L’immagine viene salvata come stringa in base64, con un header “image...” in modo da poterla passare come campo dell’attributo src in <img>.

```
_id: ObjectId('65c16282b5b50f036fba2ccb')
title: "Allora"
organiser: ObjectId('65ad55e0dc147f557665d3c6')
code: "Oam5VRhJGXlhxWaNrJeOCBlXGv4NYRZCDuHT6UsdME8="
▶ tickets: Array (empty)
▶ types: Array (empty)
▶ location: Object
  date: 2025-12-12T10:11:00.000+00:00
  noUnderage: false
  hasQR: false
  hashtags: Array (1)
    image: "data:image/jpeg;base64,/9j/4AAQSkZJRgABAQAAAQABAAAD/2wCEAAYEBQYFBAYGBQY..."
  ▶ ratings: Array (empty)
  ▶ comments: Array (empty)
  ▶ price: Object
    description: "Desc"
  occupiedSpots: 0
__v: 0
```

## RecoveryToken

Un Recovery token viene salvato nel DB quando un Client esegue un recovery della password. Periodicamente il backend elimina dal DB i token scaduti.

```
_id: ObjectId('65c2eeecf718311af8af92e1')
user: ObjectId('65ad55e0dc147f557665d3c6')
token: "0c6388299bc2a9721c38bb10779e23be21ec2b6d43355629e10ee4ec398468d5"
createdAt: 2024-02-07T02:46:04.382+00:00
expiresAt: 2024-02-07T03:46:04.382+00:00
__v: 0
```

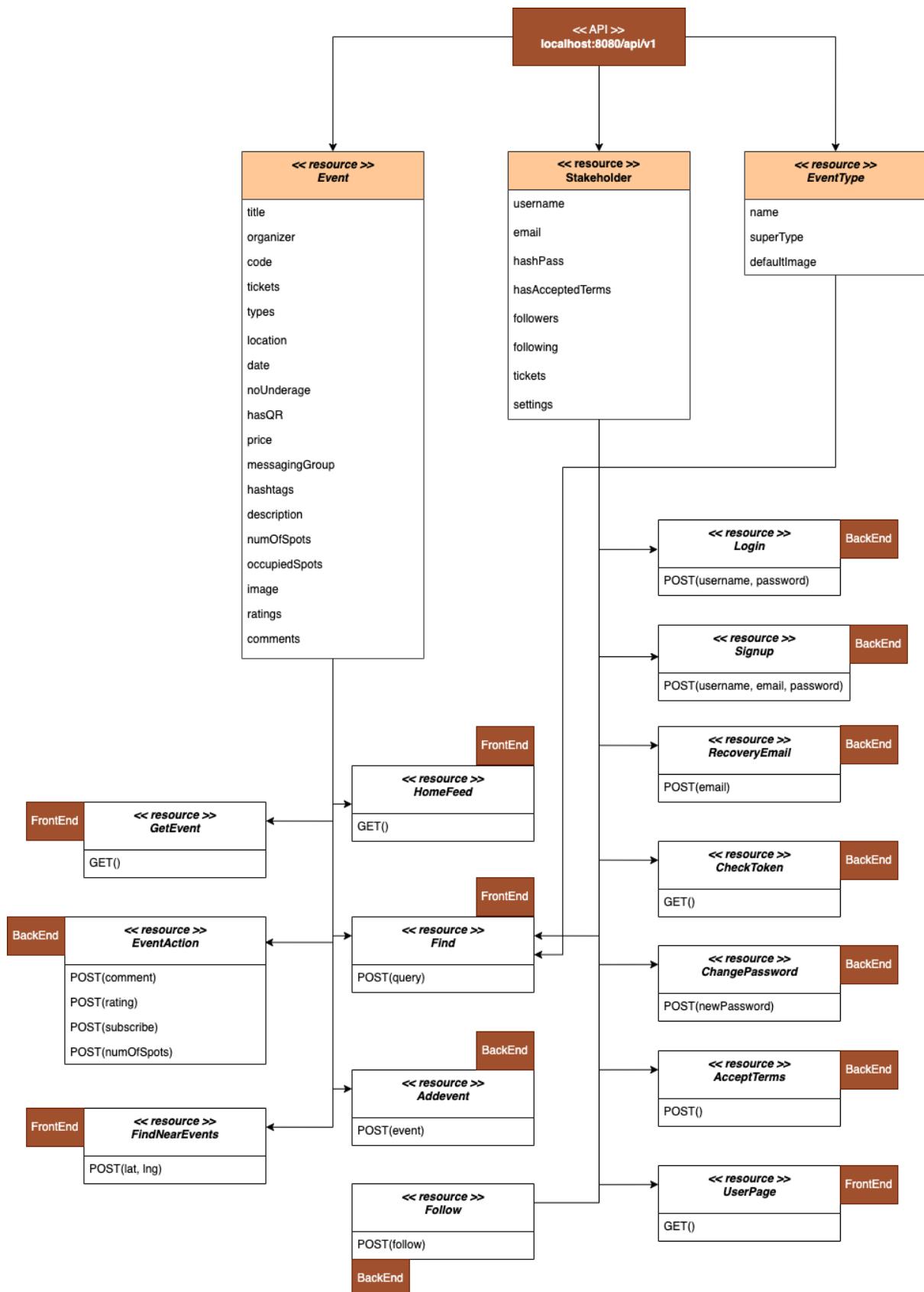
## Progetto API

### Resource Extraction dal Class Diagram

Dal Class Diagram sono in effetti state estratte solo tre classi, Event, Stakeholder ed Event Type (da notare che Stakeholder non coincide con l'originale per semplicità nello sviluppo).

In effetti è facile notare come siano solo questi effettivamente i diretti interessati dalle API, direttamente (magari essendo ritornati in una risposta), o indirettamente, contenendo un oggetto, a sua volta contenuto in una delle tre classi elencate, i cui attributi sono direttamente inseriti nella corrispondenza con il backend.

Come si può notare, ogni risorsa interessa generalmente una sola classe; l'unica eccezione è "Find", che, come sarà descritto nella sezione Front End, ritorna sia Eventi, che Utenti, che tipologie di Eventi nella barra di ricerca.

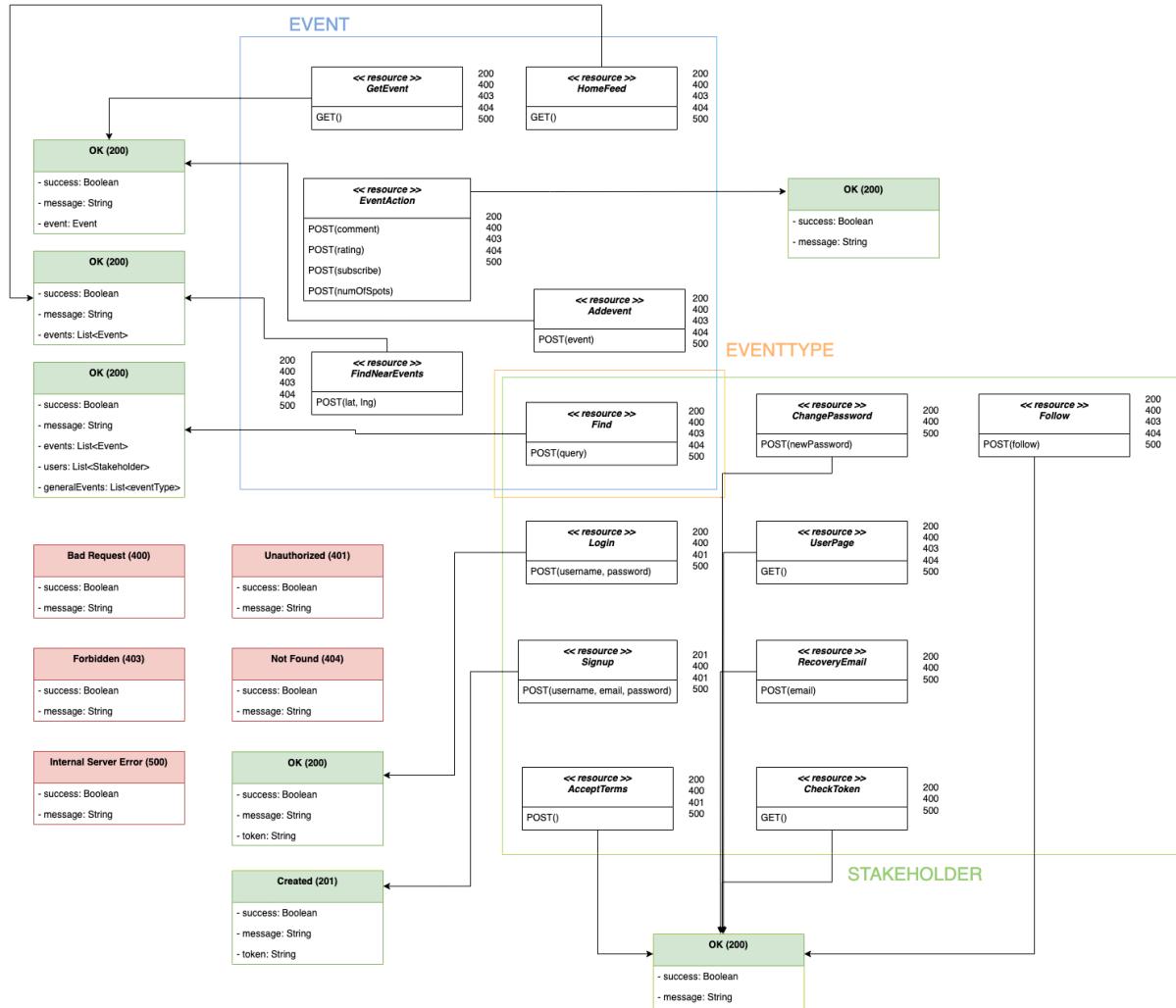


## Resource Models

In questo diagramma vengono prese le risorse del Resource Extraction diagram e vengono associate a tutte le possibili risposte del server che possono provocare.

Da notare come, per pulizia di rappresentazione, ogni classe (Event, Stakeholder e EventType) è ora rappresentata come una cornice che racchiude tutte le risorse alla quale è collegata nel diagramma precedente.

Ogni risorsa ha affiancata una lista di numeri, che rappresentano i vari 'response status' che possono provocare. Non vengono utilizzate frecce per collegare gli stati di errore (in rosso) perché sono gli stessi per tutte le risorse. Ci limitiamo quindi a collegare solo gli stati 200 e 201 in base agli attributi ritornati dal server in risposta al client.



## Sviluppo API

In questo paragrafo vengono descritte le varie risorse del Resource Extraction Diagram nel loro funzionamento, mostrando le immagini delle route. Per ovvi motivi non vengono riportati i codici dei controllori, ma eventualmente è riportata una descrizione delle task che svolgono.

(Nel codice sono state inserite anche delle OPTION, in modo che non ci siano problemi di Cross-Origin Resource Sharing, CORS, nel FrontEnd; qui non verranno trattate)

### Get Event

Con questa GET vengono richiesti al backend i campi dell'Evento con code :eventCode.  
Ci sono delle funzioni da analizzare:

- `setGlobalHeaders` aggiunge degli header importanti per garantire il corretto funzionamento del frontend
- `checkToken` verifica che il token nell'header `Authorization` sia valido e da esso ricava l'utente che ha inviato la richiesta
- `checkTermsAcceptance` verifica che l'Utente abbia accettato i Termini e Condizioni

```
JavaScript
// GET `/api/${apiVersion}/events/:eventCode`
router.get(
  `/api/${apiVersion}/events/:eventCode`,
  setGlobalHeaders,
  [
    checkToken,
    checkTermsAcceptance
  ],
  eventController.getEventPage
);
```

### Event Action

Con questa POST l'Organizzatore dell'Evento può:

- Commentare
- Variare il numero di posti

mentre un altro Utente può:

- Commentare

- Iscriversi
- Valutare (rating) l'Evento

Il controller dell'API capisce cosa fare in base all'Utente loggato e ai campi dell'header.

```
JavaScript
// POST `/api/${apiVersion}/events/:eventId`
router.post(
  `/api/${apiVersion}/events/:eventId`,
  setGlobalHeaders,
  [
    checkToken,
    checkTermsAcceptance
  ],
  eventController.postEventPage
);
```

## Find Near Events

Questa POST, utilizzata nella mappa, permette di ricavare gli Eventi con coordinate comprese negli intervalli `lat` e `lng`, campi del body della richiesta.

```
JavaScript
// POST `/api/${apiVersion}/map`
router.post(
  `/api/${apiVersion}/map`,
  setGlobalHeaders,
  [
    checkToken,
    checkTermsAcceptance
  ],
  eventsMap
);
```

## AddEvent

Questa POST permette di creare, se gli argomenti passati sono corretti, un evento.  
Ritorna l'evento creato.

```
JavaScript
// POST `/api/${apiVersion}/addevent`
router.post(
  `/api/${apiVersion}/addevent`,
  setGlobalHeaders,
  [
    checkToken,
    checkTermsAcceptance
  ],
  fillEvent
);
```

## Home Feed

Richiede una lista di Eventi da mostrare nel feed della Home dell'applicazione.

```
JavaScript
// GET `/api/${apiVersion}/home`
router.get(
  `/api/${apiVersion}/home`,
  setGlobalHeaders,
  [
    checkToken,
    checkTermsAcceptance
  ],
  getHomeFeed
);
```

## Find

Questa POST viene utilizzata dalla Pagina Home in fase di ricerca. Ritorna tre liste di Eventi, Utenti e EventTypes filtrati dalla query passata nel body della richiesta.

```
JavaScript
// POST `/api/${apiVersion}/home`
router.post(
  `/api/${apiVersion}/home`,
  setGlobalHeaders,
  [
    checkToken,
    checkTermsAcceptance
  ]
```

```
],
filterEventsByQuery
);
```

## Login

Questa POST viene utilizzata in fase di Login. Prende in ingresso dal body username e password.

```
JavaScript
// POST `/api/${apiVersion}/login`
router.post(
  `/api/${apiVersion}/login`,
  setGlobalHeaders,
  validateUserInput,
  login,
);
```

## Signup

Questa POST viene utilizzata in fase di Signup. Prende in ingresso dal body username, email e password.

```
JavaScript
// POST `/api/${apiVersion}/register`
router.post(
  `/api/${apiVersion}/register`,
  setGlobalHeaders,
  validateUserInput,
  async (req, res) => {
    // Check for validation errors from the Express Validator middleware
    const errors = validationResult(req);
    if (!errors.isEmpty()) {
      return res.status(400).json({ errors: errors.array() });
    }
    try {
      // Call the controller function to register the user
      const registrationResult = await register(req, res);
    }
  }
);
```

```
// Return the response provided by the controller
/* return
res.status(registrationResult.status).json(registrationResult); */
return res;
} catch (error) {
  console.error(error);
  return res.status(500).json({ message: 'Registration failed' });
}
);
}
```

## Recovery Email

Questa POST è la prima eseguita dal FrontEnd durante la fase di Password Recovery. Prende dal body della richiesta la `email` in ingresso ed invia a tale indirizzo un link al FrontEnd. Ritorna anche un Token temporaneo che il Client deve utilizzare per concludere il Recovery della Password.

```
JavaScript
// POST `/api/${apiVersion}/recover`
// Password recovery request route
router.post(
  `/api/${apiVersion}/recover`,
  setGlobalHeaders,
  [
    validateUserInput,
  ],
  getRecoveryToken
);
```

## Check Token

Questa GET viene inviata dal FrontEnd dopo che l'Utente ha clickato sul link inviatogli per mail. Serve per verificare che il token temporaneo sia valido.

```
JavaScript
// GET `/api/${apiVersion}/recover/:token`
// Password reset route with token verification
```

```
router.get(
  `/api/${apiVersion}/recover/:token`,
  setGlobalHeaders,
  verifyToken
);
```

## Change Password

Questa POST è l'ultima richiesta del Password Recovery. Il backend legge il campo **password** dal body e il **token** temporaneo dai parametri e scrive la nuova password nel DB.

```
JavaScript
// POST `/api/${apiVersion}/recover/:token`
// Password reset form submission route
router.post(
  `/api/${apiVersion}/recover/:token`,
  setGlobalHeaders,
  [
    validateUserInput,
  ],
  resetPassword
);
```

## Accept Terms

Questa POST si utilizza per accettare i Termini e Condizioni.

```
JavaScript
// POST `/api/${apiVersion}/accept-terms`
router.post(
  `/api/${apiVersion}/accept-terms`,
  setGlobalHeaders,
  checkToken,
  acceptTerms
);
```

## User Page

Questa GET ritorna l'Utente richiesto.

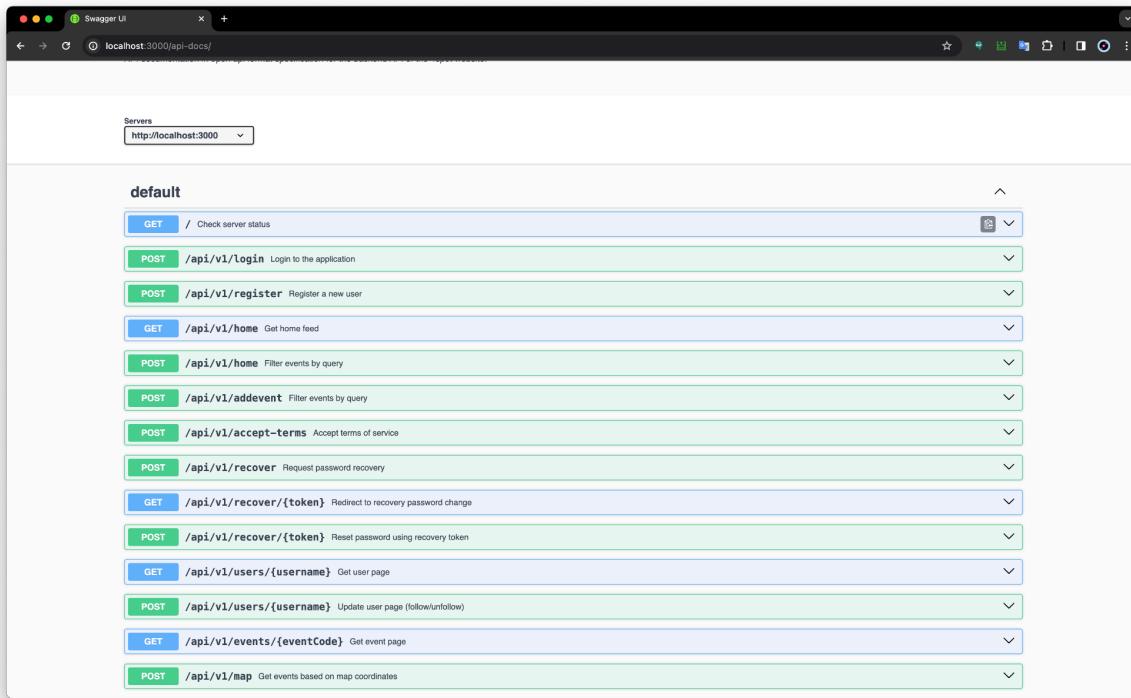
```
JavaScript
// GET `/api/${apiVersion}/users/:username`
router.get(
  `/api/${apiVersion}/users/:username`,
  setGlobalHeaders,
  [
    checkToken,
    checkTermsAcceptance
  ],
  usersController.getUserPage
);
```

## Follow

La POST seguente viene utilizzata per aggiungere alla lista `following` di chi la richiede l'Utente con `username= :username`.

```
JavaScript
// POST `/api/${apiVersion}/users/:username`
router.post(
  `/api/${apiVersion}/users/:username`,
  setGlobalHeaders,
  [
    checkToken,
    checkTermsAcceptance
  ],
  usersController.postUserPage
);
```

### 3. API Documentation



The screenshot shows the Swagger UI Express interface running at `localhost:3000/api-docs`. The top navigation bar has tabs for 'Servers' and 'API'. Under 'Servers', there is a dropdown set to `http://localhost:3000`. The main content area is titled 'default' and lists various API endpoints with their methods and descriptions:

- `GET /` Check server status
- `POST /api/v1/login` Login to the application
- `POST /api/v1/register` Register a new user
- `GET /api/v1/home` Get home feed
- `POST /api/v1/home` Filter events by query
- `POST /api/v1/addevent` Filter events by query
- `POST /api/v1/accept-terms` Accept terms of service
- `POST /api/v1/recover` Request password recovery
- `GET /api/v1/recover/{token}` Redirect to recovery password change
- `POST /api/v1/recover/{token}` Reset password using recovery token
- `GET /api/v1/users/{username}` Get user page
- `POST /api/v1/users/{username}` Update user page (follow/unfollow)
- `GET /api/v1/events/{eventCode}` Get event page
- `POST /api/v1/map` Get events based on map coordinates

Per testare le API del BackEnd è stato utilizzato Swagger UI Express.

Le API implementate sono le seguenti (assumere `/api/v1` per ognuna):

- `POST /login`: Richiedi Login
- `POST /register`: Richiedi Signup
- `GET /home`: Richiedi feed Home
- `POST /home`: Filter Events, Users and Event Types data una query
- `POST /accept-terms`: Accetta termini e condizioni
- `POST /recover`: Richiedi messaggio per recovery
- `GET /recover/{token}`: Check recovery token
- `POST /recover/{token}`: Cambia password
- `GET /users/{username}`: Ottieni Utente
- `POST /users/{username}`: Follow/Unfollow
- `GET /events/{eventCode}`: Get Event
- `POST /map`: Ottieni Eventi in base a coordinate

## 4.FrontEnd Implementation

### Login Page

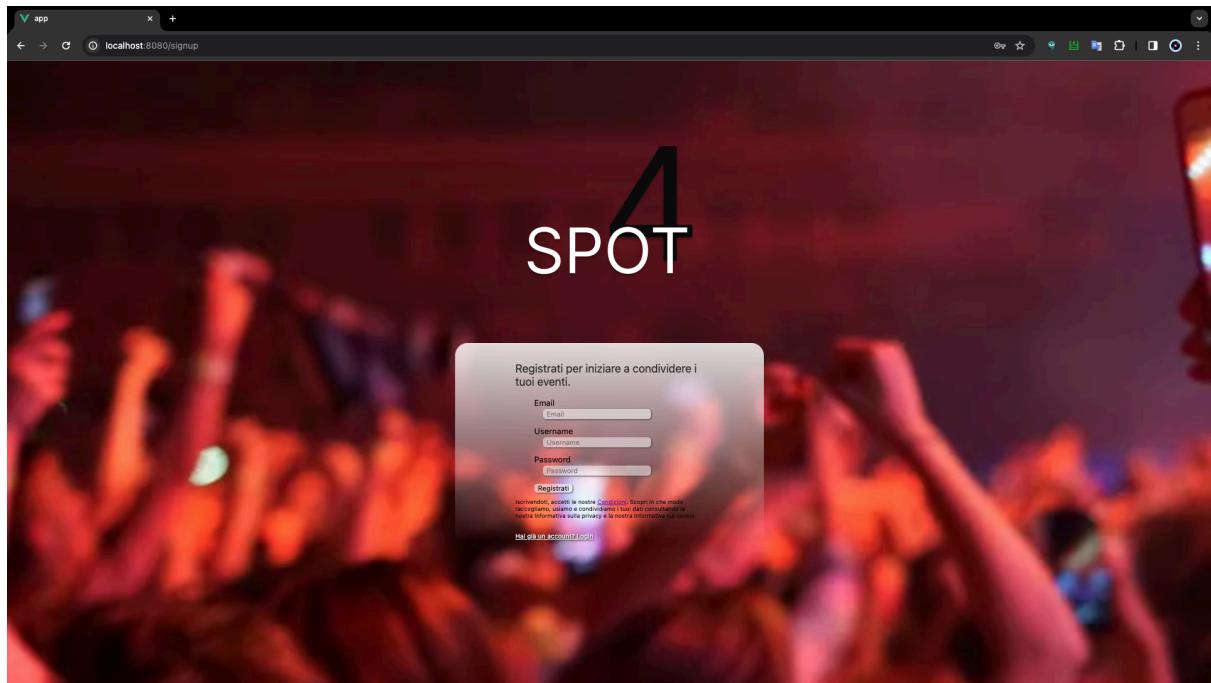


Con la Login Page l'Utente effettua l'accesso. I campi 'Username' e 'Password' sono obbligatori.

Clickando su 'Password dimenticata?' l'Utente viene reindirizzato alla Recovery Password Page per recuperare la password.

Clickando su 'Non hai ancora un account? Iscriviti.' l'Utente viene reindirizzato alla Signup Page.

## Signup Page



L'Utente deve inserire Email, Username e Password per potersi iscrivere. La password deve essere lunga almeno 8 caratteri, avere una lettera minuscola, una maiuscola, un numero e un carattere speciale.

Tutti i campi sono obbligatori.

Clickando su 'Condizioni' l'Utente accede alla Pagina Privacy Policy.

Clickando su 'Hai già un account? Login' l'Utente viene reindirizzato alla Login Page.

## Privacy Policy Page



### Privacy Policy of 4spot

4spot operates the 4spot website, which provides the SERVICE.

This page is used to inform website visitors regarding our policies with the collection, use, and disclosure of Personal Information if anyone decided to use our Service, the 4spot website.

If you choose to use our Service, then you agree to the collection and use of information in relation with this policy. The Personal Information that we collect are used for providing and improving the Service. We will not use or share your information with anyone except as described in this Privacy Policy.

The terms used in this Privacy Policy have the same meanings as in our Terms and Conditions, which is accessible at homepage.html, unless otherwise defined in this Privacy Policy.

#### Information Collection and Use

For a better experience while using our Service, we may require you to provide us with certain personally identifiable information, including but not limited to your name, phone number, and postal address. The information that we collect will be used to contact or identify you.

#### Log Data

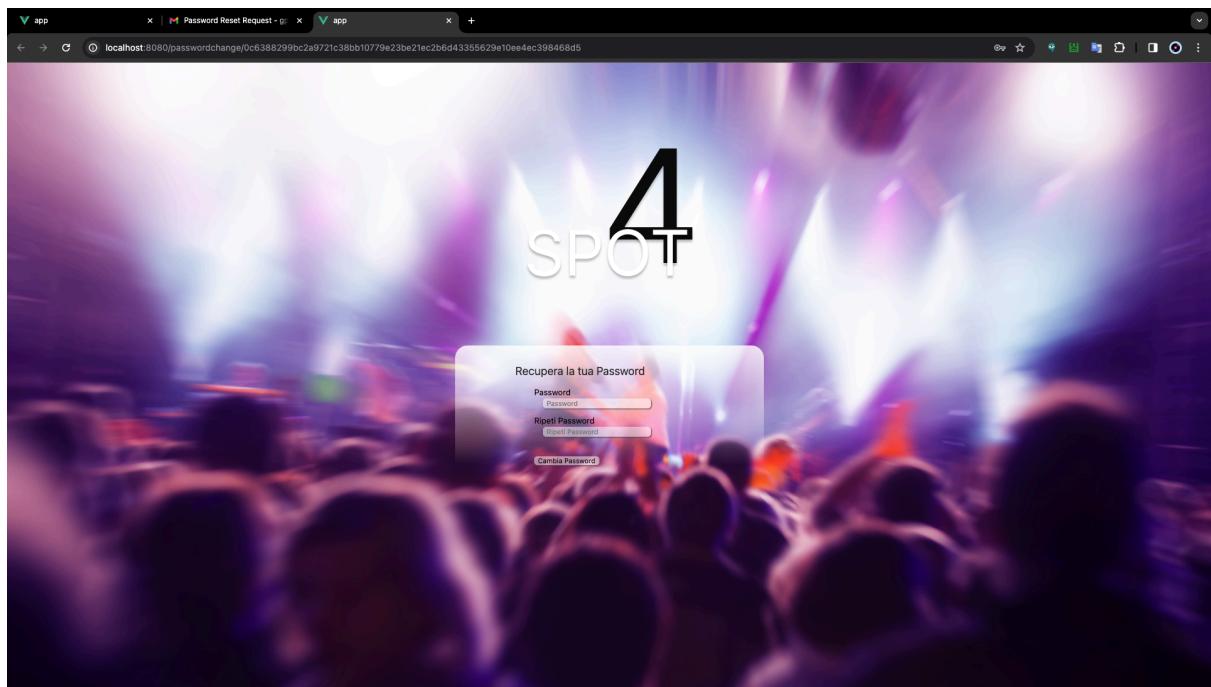
We want to inform you that whenever you visit our Service, we collect information that your browser sends to us that is called Log Data. This Log Data may include information such as your computer's Internet Protocol ("IP") address, browser version, pages of our Service that you visit, the time and date of your visit, the time spent on those pages, and other statistics.

Semplicemente una lista di termini e condizioni.

## Password Recovery Page



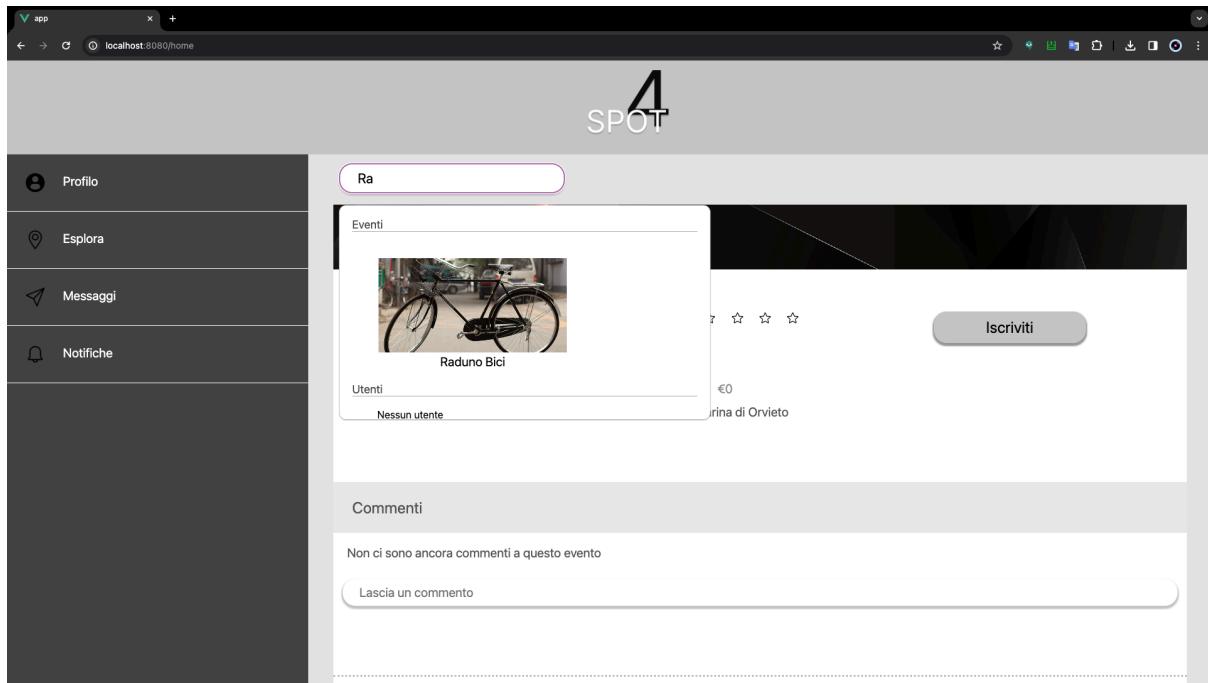
Con questa schermata l'Utente deve inserire la **email** associata al suo account. Ad essa verrà indirizzato un messaggio con un link al FrontEnd per cambiare la password.



L'Utente deve inserire entrambi i campi 'Password' e 'Ripeti Password', i quali devono coincidere.

Anche in questo caso la password deve rispettare le condizioni imposte in Login Page.

## Home Page

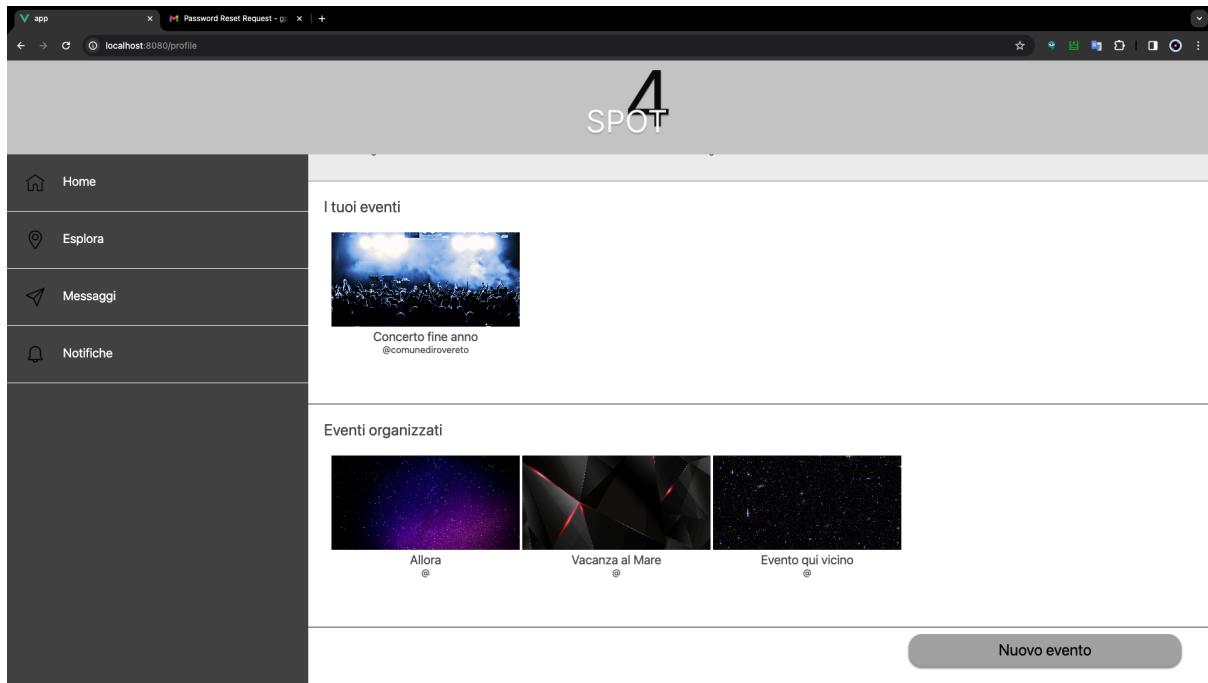


Nella Home Page l'Utente può:

- Guardare gli Eventi in feed 'scrollando' con il mouse
- Interagire con tali eventi
- Clickare sulla striscia 'Commenti' per aprire una finestra popup che mostra tutti i commenti associati all'Evento
- Cercare Eventi, Utenti e Tipologie di Evento nella barra Ricerca. Clickando su un Evento ritornato può accedere alla relativa Event Page
- Passare ad una delle seguenti pagine:
  - Profile Page
  - Explore Page
  - Messaging Page
  - Notifications Page

clickando sugli appositi bottoni.

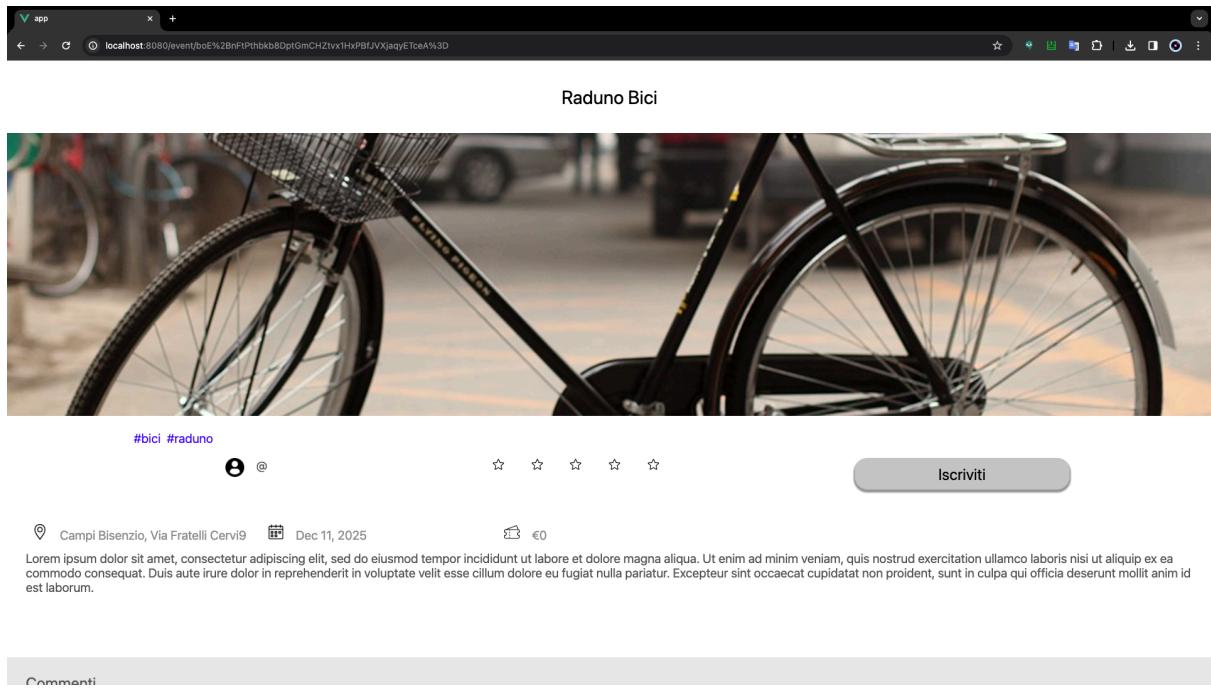
## Profile Page



Nella Profile Page l'Utente può accedere al proprio Profilo e:

- Visualizzare gli eventi da lui organizzati in un formato ridotto
- Visualizzare gli eventi organizzati dai suoi amici in un formato ridotto (*non funzionante*)
- Clickare sul bottone 'Nuovo evento' per essere reindirizzato alla Event Creation Page
- Clickare su un Evento per accedere alla relativa Event Page
- L'Utente può inoltre passare ad una delle seguenti pagine:
  - Home Page
  - Profile Page
  - Messaging Page
  - Notifications Pageclickando sugli appositi bottoni.

## Event Page



In questa pagina l'Utente può visualizzare l'Evento e interagirci nei seguenti modi:

- Clickando su 'Iscriviti' per iscriversi
- Clickando sulle stelle per lasciare un Rating
- Lasciando un Commento dall'apposita barra di input

(Queste interazioni funzionano solo con PostMan)

## Event Creation Page

The screenshot shows a web browser window titled 'app' with the URL 'localhost:8080/create-event'. The page has a light gray background and features a central heading 'Crea il tuo Evento!' in bold black font. Below the heading is a horizontal list of form fields, each consisting of a label and an input field:

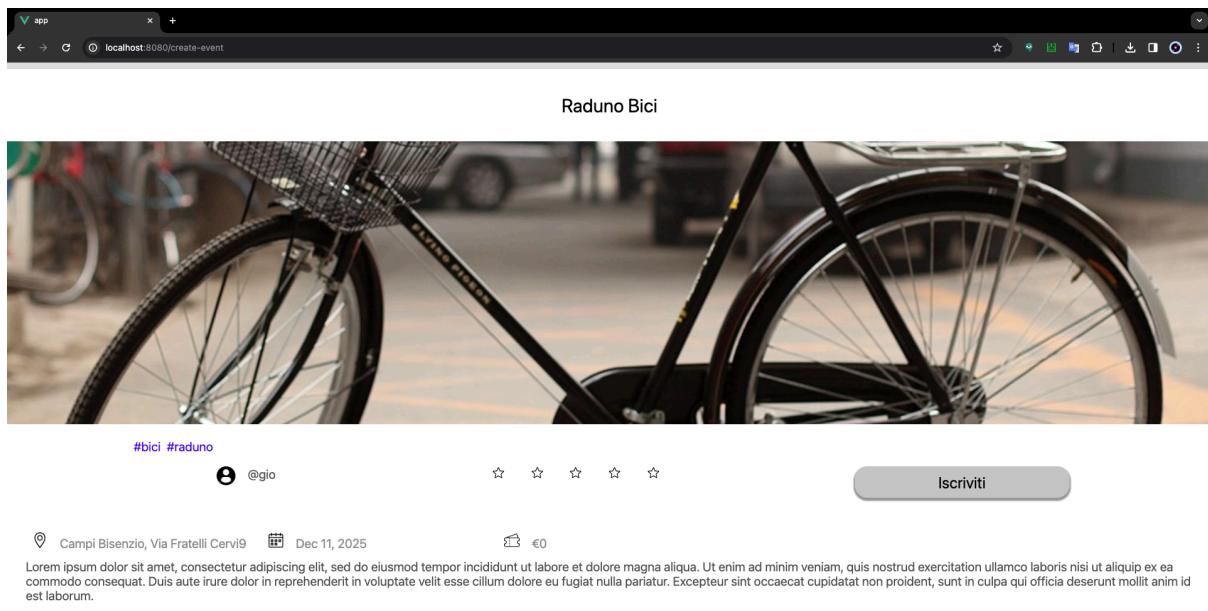
- Titolo:** Raduno Bici
- Descrizione:** (Placeholder text: Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam...)
- Codice postale:** 50013
- Paese:** Italy
- Città:** Campi Bisenzio
- Via:** Via Fratelli Cervi
- Numero civico:** 9
- Data:** (Placeholder text: DD/MM/YYYY)

In questa schermata l'Utente deve compilare il form per poter pubblicare l'Evento.

I campi obbligatori sono:

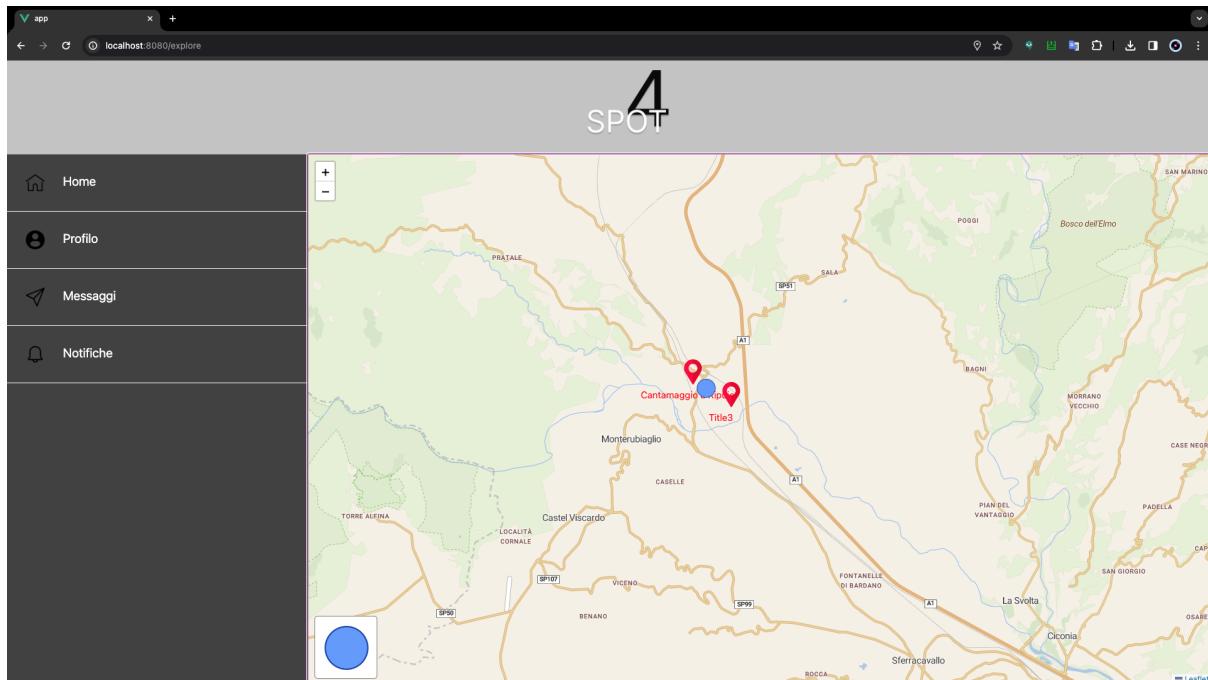
- Titolo
- Codice postale
- Paese
- Città
- Via
- Numero civico
- Data
- Immagine
- Tipologie

Clickando su bottone 'Avanti' si visualizza l'anteprima dell'Evento che si deve creare.



Qui si può visualizzare l'anteprima dell'Evento che si intende creare prima che sia fatta alcuna POST al backend.

## Explore Page



In questa pagina l'Utente può visitare la mappa, ottenendo ogni volta una lista di Eventi locali filtrati dal backend.

Clickando sul quadrato con il locatore circolare blu in basso a sinistra, cetrare la mappa sulla sua posizione attuale.

L'Utente può inoltre passare ad una delle seguenti pagine:

- Home Page
- Profile Page
- Messaging Page
- Notifications Page

clickando sugli appositi bottoni.

## 5. GitHub

### Github Organization

<https://github.com/4spot-team>

L' organizzazione è organizzata in quattro repositories:

- **documentation**: Inseriti tutti i diagrammi utilizzati, nei vari branch
- **frontend**: Frontend sviluppato in Vue. Per eseguire spostarsi in `/app` e utilizzare `npm install` e `npm run serve`
- **backend**: Backend sviluppato con Express. Per eseguire `npm install` e `npm start`
- **deliverables**: File PDF di tutti e 5 i deliverables

## 6. Testing

Nel backend è stato testato il controllore della Mappa. Per il testing abbiamo utilizzato 'jest' e 'supertest'. Jest si connette al server backend (app di express.js) e al database di mongoDB. Inoltre, per poter accedere agli endpoint del backend da testare (coperti dai check dei token del middleware), esegue l'accesso una tantum attraverso un account di prova e utilizza il token prodotto dal backend per poter accedere in seguito alla mappa. Quindi esegue una suite di test che vanno a coprire tutti i possibili casi d'uso di questo endpoint (con una copertura di righe di codice del 92%, dove l'8% rimanente riguarda errori interni dovuti ad eventuali problemi di librerie esterne come express, e non al sorgente dell'applicazione, quindi non testabili direttamente). Supertest viene utilizzato per emulare delle richieste http, senza le quali non sarebbe possibile accedere agli endpoint dell'API.