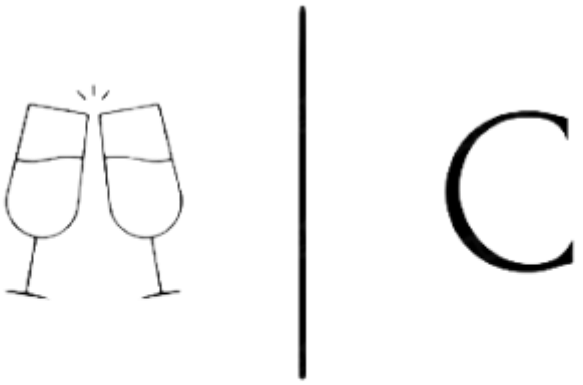


CHEERS

DOVE OGNI DETTAGLIO CONTA

PROGETTO AMORETTI

By

Andrea Abretti

Alessio Panizzieri

Indice

INDICE

Introduzione

Per lo sviluppo del progetto la scelta era ampia senza linee guida troppo stringenti, sono state suggerite alcune idee come: e-commerce su libri, musica, abbigliamento film ecc.., giochi multiplayer come scacchi o battaglia navale sfruttando architettura client-server o p2p e infine come ultima traccia consigliata c'è una sperimentazione legata alla blockchain come ethereum e algorand oppure lo sviluppo di app basate su smart contract. Dopo aver vagliato diverse opzioni e cercandone una differenziata dalle altre abbiamo optato per la seguente:

"Applicazione web per la gestione di eventi (concerti, matrimoni ecc..), in cui un cliente può visionare una lista di location cercando quella adeguata alla sua esigenza, mandare una richiesta per la prenotazione della stessa per un evento che sarà confermato in caso di location libera nella data selezionata, si potrà eventualmente lasciare anche una recensione della location per garantire un feedback terzo per la selezione di una location adeguata. Il gestore ha la possibilità di inserire le proprie location, informazioni ad esse collegate e gestire le richieste in arrivo."

Abbiamo affrontato una prima fase di progettazione e grazie ai pochi vincoli della traccia abbiamo scelto di sfruttare diverse librerie per automatizzare vari processi.

Analisi e specifica dei requisiti

In questa sezione, vengono delineate tutte le entità identificate durante l'analisi, insieme ai rispettivi attributi e alle funzionalità che il programma incorpora. Durante il processo di analisi, sono stati individuati gli attori, i casi d'uso e le relazioni tra di essi, contribuendo alla definizione complessiva del programma.

È importante sottolineare che ogni attore può sfruttare una o più funzionalità tramite i casi d'uso, e, di conseguenza, un singolo caso d'uso può coinvolgere più attori. Tuttavia, è fondamentale notare che ciascun caso d'uso può avere un solo attore principale.

Il sistema è progettato per coinvolgere due tipi di utenti, i quali interagiscono con il servizio in modi unici. Gli utenti identificati sono il Gestore di location e il Cliente, ciascuno dei quali ha accesso a specifiche funzionalità del sistema. Una panoramica dettagliata di ciascun tipo di utente è la seguente:

Gestore:

- **Descrizione:** Rappresenta un ente o una persona responsabile di una location. Può gestire le proprie location aggiungendone diverse, fornendo dettagli, come fotografie, sulle stesse e può controllare tutti gli eventi prenotati nella location. Potrà modificare la password e i vari campi delle proprie location.

- **Casi d'Uso Principali:**
    - **Gestione Location:** Possibilità di aggiornare le informazioni sulle proprie location e aggiunta fotografie.
    - **Inserire le proprie location:** Possibilità di inserire nuove location.
    - **Eliminazione Location:** Possibilità di eliminare una location dal sistema .

Cliente:

- **Descrizione:** Una volta effettuato il login o la registrazione per un nuovo account potrà vedere tutte le location disponibili, manda le richieste di aggiunta per gli eventi e potrà lasciare e/o recensioni per le location e potrà modificare la password del proprio account, per entrambi i tipi di utenti per modificare gli altri campi servirà mandare una richiesta con una motivazione valida.
- **Casi d'Uso Principali:**
    - **Richiesta per un evento:** Possibilità di mandare una richiesta per un determinato evento.
    - **Eliminazione evento:** Eliminazione eventi di propria creazione.
    - **Inserire/eliminare recensioni:** Possibilità di inserire e cancellare recensioni sulle location.

(Non abbiamo implementato un utente admin, in caso di problemi si potrà effettuare una richiesta direttamente a chi offre il servizio)

Funzionalità del sistema

Requisiti funzionali

Specifica dei casi d'uso per ogni singola entità:

Utenti (Gestore, Cliente)

- Registrazione.
- Login.
- Logout.
- Dati Profilo.
- Modifica password.
- Aggiunta Recensione.
- Eliminazione propria recensione.

Gestore

- Tutte le funzioni di Utenti.
- Gestione Location (Aggiunta e Eliminazione).
- Aggiunta e eliminazione di fotografie per le proprie location.

Cliente

- Tutte le funzioni di Utenti.
- Richiesta per un evento.
- Eliminazione evento.

Progettazione: Casi d'uso

Utente generico

Registrazione	
Descrizione	Permette ad un utente di registrarsi all'interno del DataBase, specificando anche il ruolo.
Precondizioni	Compilare tutti i campi, rispettando le condizioni.

Frequenza d'uso	Ogni qual volta un utente si voglia registrare al sistema.
Esecuzione	Controllo dell'effettiva validità dei dati inseriti e successivamente inserisce l'utente nel sistema.

Utente regstrato

Login	
Descrizione	Permette agli utenti di autenticarsi nel sistema, inserendo username e password.
Precondizioni	L'utente che tenta di autenticarsi deve essersi registrato precedentemente e di conseguenza presente nel DB.
Frequenza d'uso	Ogni qual volta un utente vuole accedere al sistema.
Esecuzione	Inserire Username e Password e di conseguenza il sistema controllerà se coincidono con quelli di qualche utente presente nel DB.

Logout	
Descrizione	Permette all'utente autenticato di disconnettersi dal sistema, senza lasciare il proprio profilo aperto.
Precondizioni	L'utente deve avere svolto l'accesso al sistema precedentemente.
Frequenza d'uso	Ogni qual volta ci si voglia disconnettere.
Esecuzione	Premere il bottone di "Logout" e di conseguenza disconnettersi dal sistema

Dati Profilo	
Descrizione	Permette di visualizzare i propri dati come (Nome, Cognome, Indirizzo, E-mail, Telefono ecc...)

Precondizioni	L'utente deve essere collegato al sistema e deve aver svolto l'accesso.
Frequenza d'uso	Ogni qual volta si vogliano visualizzare i propri dati.
Esecuzione	Premere il bottone con il proprio nome visualizzato nella navbar per andare nella pagina dedicata alla visualizzazione dei i dati.

Cambio Password	
Descrizione	Permette il cambio password per un utente registrato.
Precondizioni	Un utente una volta effettuato l'accesso correttamente e dopo essersi spostato nella sezione del profilo.
Frequenza d'uso	Ogni qual volta un utente desideri cambiare password.
Esecuzione	L'utente clicca sul bottone sotto ai propri dati sopra scritto "CambiaPassword".

Gestione recensione	
Descrizione	Sono compresi tutti le funzionalità di gestione recensione: Aggiunta ed Elimina. Permette di andare a lavorare direttamente sulle recensioni in modo da aggiungerle o eliminarle.
Precondizioni	L'utente che vuole aggiungere una recensione può farlo andando nella pagina della location desiderata tramite la home una volta effettuato il login. Per cancellare la recensione bisogna essere l'utente che l'ha scritta.
Frequenza d'uso	Ogni qual volta si voglia lasciare o eliminare una recensione su di una location.
Esecuzione	Premere sul bottone Aggiungi recensione dopo aver scritto nel form un'opportuna descrizione

	della recensione e dopo aver scelto un voto da 1 a 5 per la stessa. Per cancellarla basterà cliccare sul bottone "Elimina" della stessa recensione.

Gestore

Aggiungi location	
Descrizione	La funzionalità di aggiunta di una location da parte del gestore di esso.
Precondizioni	Un utente di tipo gestore deve aver effettuato il login e deve aver cliccato sul tasto "Aggiungi location" dalla navbar.
Frequenza d'uso	Ogni qual volta si desidera aggiungere una location.
Esecuzione	Una volta aver inserito tutti i dati negli appositi form premere sul bottone "Aggiungi Location".

Eliminazione location	
Descrizione	Il gestore che ha aggiunto una location al sito la eliminerà.
Precondizioni	Il gestore per eliminare una location deve aver effettuato il login e deve aver aggiunto la location che vuole eliminare. Una volta cliccato sulla location desiderata dalla home.
Frequenza d'uso	Ogni qual volta si vuole eliminare una location che si ha inserito.
Esecuzione	Cliccare il bottone "Cancella Location"

Gestione Location	
Descrizione	Permette di modificare la location desiderata e aggiungere o eliminare una fotografia per la stessa.

Precondizioni	Bisogna aver effettuato l'accesso con un account "gestore", aver creato una location ed essere passato alla pagina di gestione della location, passando per la pagina del profilo cliccando sulla location scelta
Frequenza d'uso	Ogni qual volta si voglia modificare i dati di una location e/o aggiungere/eliminare una fotografia collegata a quella location.
Esecuzione	Nella pagina della location selezionata cliccare sul campo che si vuole modificare, apparirà un form in cui si scrive il nuovo campo e si clicca Ok. Per aggiungere una fotografia si clicca il bottone "Seleziona file" e si sceglierà il file jpg o png desiderato ed infine si cliccherà il bottone "Aggiungi Immagine". Per eliminarla basterà cliccare sul bottone collegato alla foto "Elimina"

Cliente

Gestione eventi	
Descrizione	Permette di mandare una richiesta per la creazione di un evento per una determinata location oppure di eliminare un evento.
Precondizioni	Bisogna aver effettuato il login e aver cliccato sulla pagina della location scelta per la richiesta di eliminazione/creazione dell'evento.
Frequenza d'uso	Ogni qual volta si voglia aggiungere un evento ad una location.
Esecuzione	Una volta compilato il form per l'aggiunta dell'evento bisogna cliccare il bottone "Manda Richiesta" verrà aggiunto l'evento se nella data selezionata non ce ne sono in programma altri. Per l'eliminazione basterà cliccare il bottone

	"Elimina" collegato all'evento che si vuole eliminare.

Progettazione

Diagramma casi d'uso

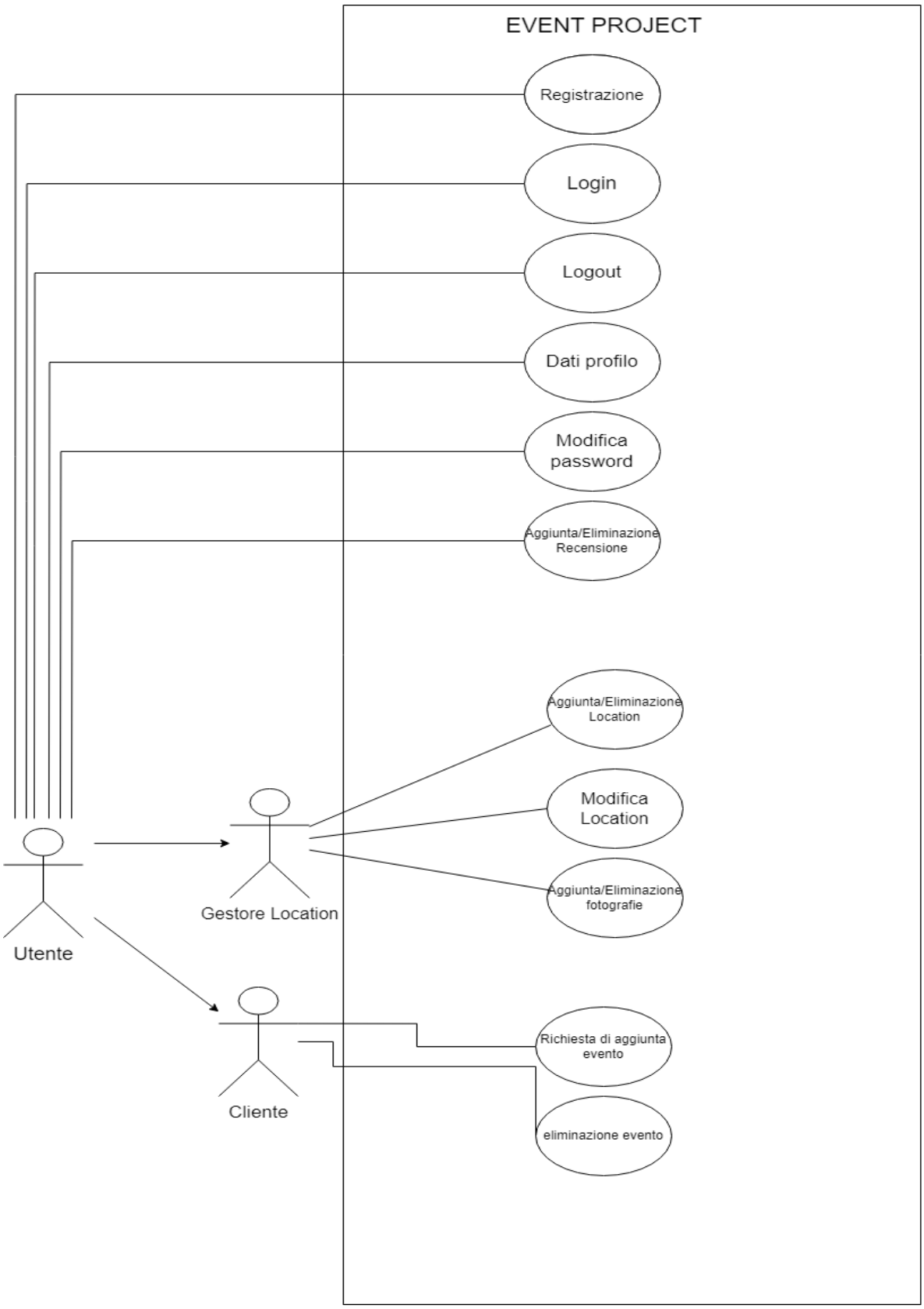
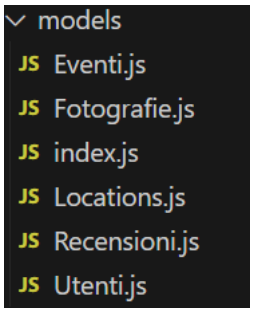


Diagramma classi

Server → models

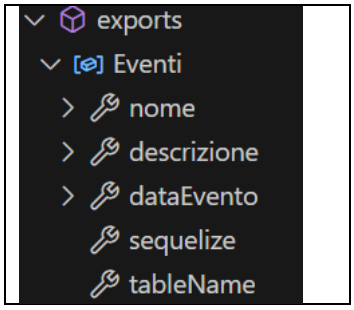
(Tramite la libreria sequelize interagisco con il db e il dbms(MySQL e MariaDB) tramite jvascript senza scrivere direttamente query SQL. Lo abbiamo sfruttato anche per creare le tabelle del db)



Per esempio, per Eventi e Utenti ho diversi metodi:

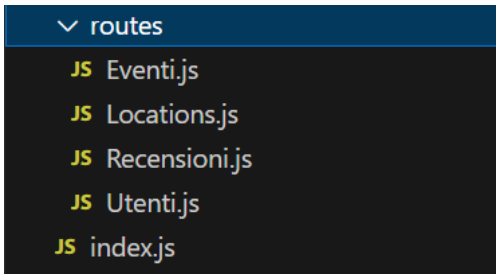
Utenti

Eventi

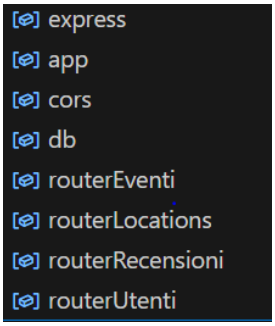


Server → Routes:

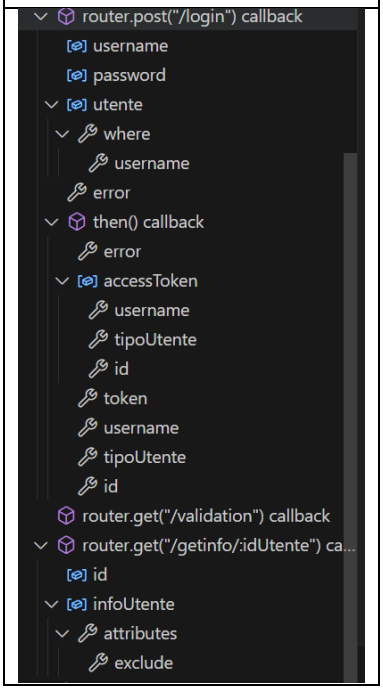
Per gestire le richieste http come GET, DELETE, POST, PUT e il mapping degli url:



Partendo dall'index che ha all'interno le definizioni delle diverse routes:

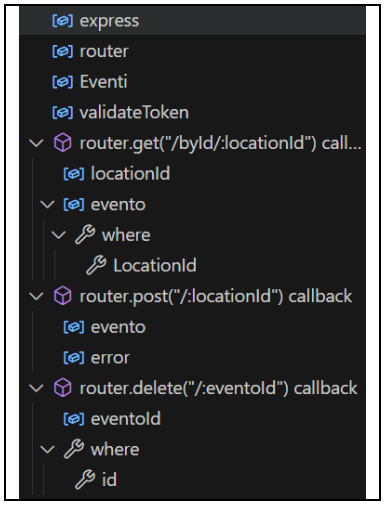


Utenti



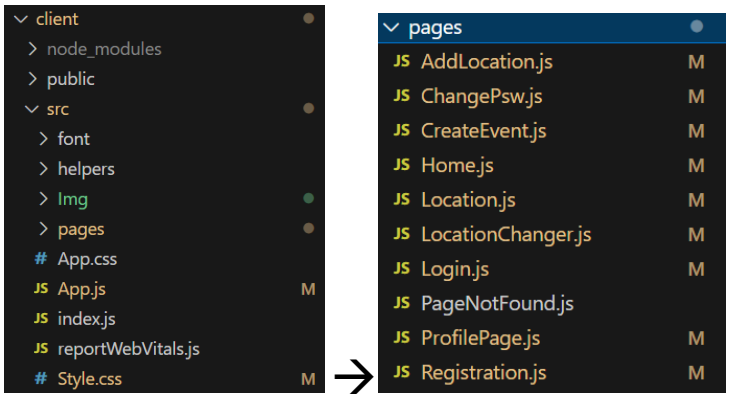
Per esempio Eventi e utenti:

Eventi

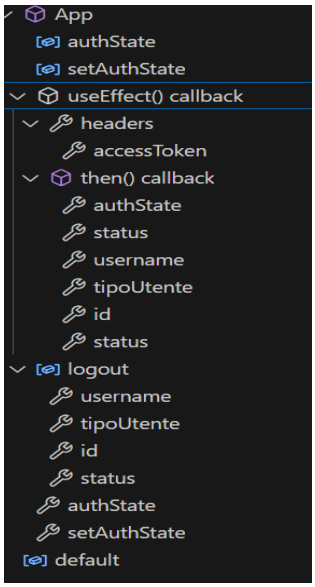


Client→ Pages

Tutte le singole pagine con il loro javascript integrato:

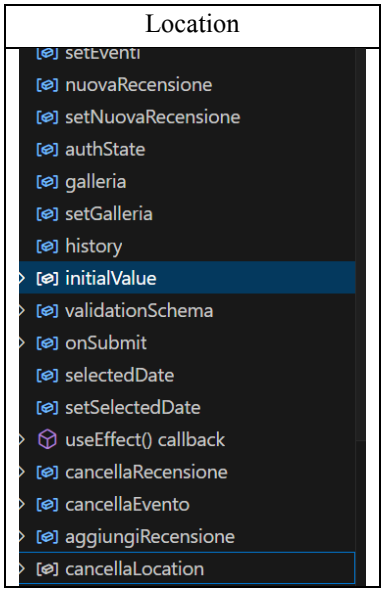


Il file App si comporta come un involucro per l'intera applicazione, infatti, vi saranno le definizioni delle routes parte client e quelle degli elementi visualizzabili in tutte le pagine come la navbar:



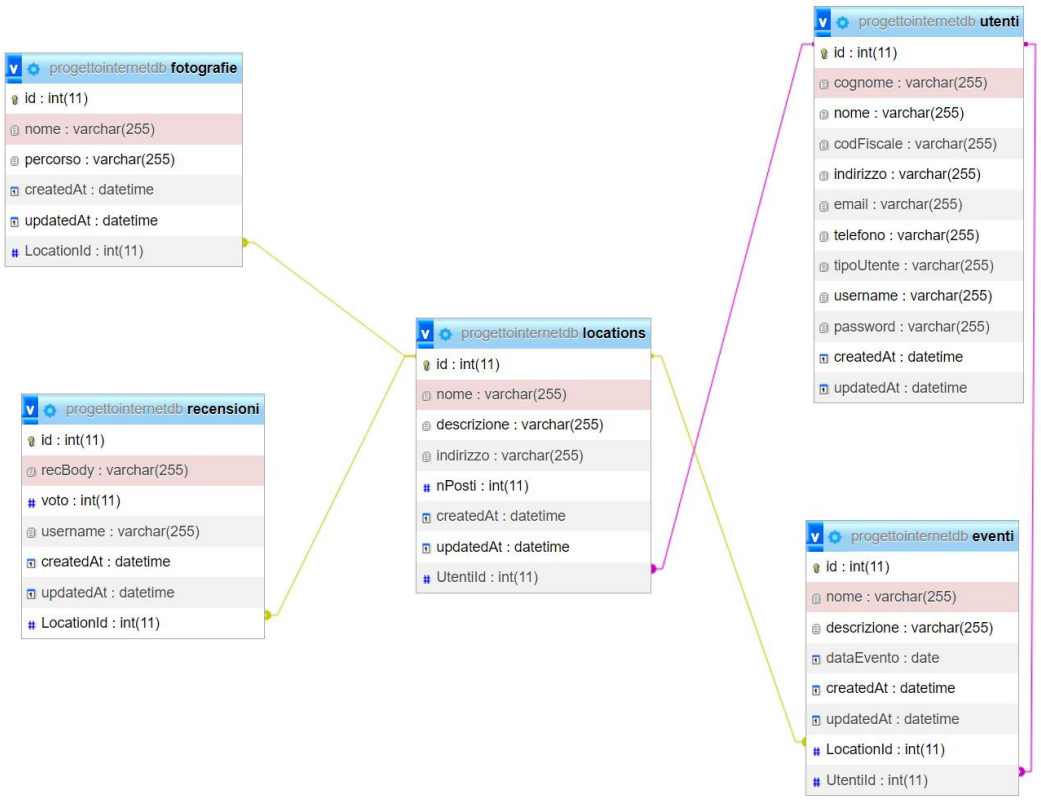
ProfilePage

Mentre per esempio le pagine Location e ProfilePage:



Base Dati

Schema E/R



Realizzazione DataBase in SQL

```sql
--
-- Struttura della tabella `eventi`
--

CREATE TABLE `eventi` (
  `id` int(11) NOT NULL,
  `nome` varchar(255) NOT NULL,
  `descrizione` varchar(255) NOT NULL,
  `dataEvento` date NOT NULL,
  `createdAt` datetime NOT NULL,
  `updatedAt` datetime NOT NULL,
  `LocationId` int(11) DEFAULT NULL,
  `UtentiId` int(11) DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

```sql
--
-- Struttura della tabella `fotografie`
--

CREATE TABLE `fotografie` (
  `id` int(11) NOT NULL,
  `nome` varchar(255) NOT NULL,
  `percorso` varchar(255) NOT NULL,
  `createdAt` datetime NOT NULL,
  `updatedAt` datetime NOT NULL,
  `LocationId` int(11) DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

```sql
--
-- Struttura della tabella `locations`
--

CREATE TABLE `locations` (
  `id` int(11) NOT NULL,
  `nome` varchar(255) NOT NULL,
  `descrizione` varchar(255) NOT NULL,
  `indirizzo` varchar(255) NOT NULL,
  `nPosti` int(11) NOT NULL,
  `createdAt` datetime NOT NULL,
  `updatedAt` datetime NOT NULL,
  `UtentiId` int(11) DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

--
-- Struttura della tabella `recensioni`
--

CREATE TABLE `recensioni` (
  `id` int(11) NOT NULL,
  `recBody` varchar(255) NOT NULL,
  `voto` int(11) NOT NULL,
  `username` varchar(255) NOT NULL,
  `createdAt` datetime NOT NULL,
  `updatedAt` datetime NOT NULL,
  `LocationId` int(11) DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

```sql
--
-- Struttura della tabella `utenti`
--

CREATE TABLE `utenti` (
  `id` int(11) NOT NULL,
  `cognome` varchar(255) NOT NULL,
  `nome` varchar(255) NOT NULL,
  `codFiscale` varchar(255) NOT NULL,
  `indirizzo` varchar(255) NOT NULL,
  `email` varchar(255) NOT NULL,
  `telefono` varchar(255) NOT NULL,
  `tipoUtente` varchar(255) NOT NULL,
  `username` varchar(255) NOT NULL,
  `password` varchar(255) NOT NULL,
  `createdAt` datetime NOT NULL,
  `updatedAt` datetime NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

Testing:

Tramite visual studio code è stata effettuata un'analisi metodica di tutto il codice, per riuscire ad individuare la maggior parte delle problematiche e degli errori del codice stesso. Sono stati effettuati due tipi di test:

White Box Testing, realizzato dai sottoscritti autori del codice, analizzando il flusso di dati durante il funzionamento, il codice in maniera approfondita per andare a rivelare eventuali bug che sono stati prontamente risolti. In particolare, sono testate tutte le richieste GET, PUT, DELETE e POST tramite il software **Insomnia** molto utile perché permette di fare richieste http alle API e di vedere le risposte ed eventuali errori per garantire un ottimale funzionamento delle funzionalità base.

Black Box Testing, È stato invece realizzato da tre persone con, praticamente, zero esperienza nello sviluppo di applicazioni web/software, in questo siamo venuti a conoscenza di come rendere il design maggiormente "User friendly", e per andare a risolvere eventuali errori che un utente senza alcuna conoscenza del software può generare in modo casuale.

Al termine di questi ci siamo sentiti soddisfatti del risultato ottenuto.

Strategia per la realizzazione del progetto e tecnologie utilizzate

Inizialmente, abbiamo discusso della natura di un eventuale tipo di progetto, una volta definita la linea guida per procedere avendo scelto una web app ci siamo confrontati nuovamente per decidere come strutturare la base del progetto, definendo entità, attributi e le varie funzionalità coinvolte. Una volta definiti questi aspetti, abbiamo suddiviso i compiti tra di noi. Durante lo sviluppo del programma, ci tenevamo costantemente aggiornati reciprocamente, fornendoci feedback sulle nuove implementazioni.

L'approccio collaborativo ha coinvolto una comunicazione continua: quando uno di noi apportava modifiche al codice, l'altro veniva prontamente informato, mantenendo un flusso costante di aggiornamenti. Affrontavamo i problemi nel codice in maniera collaborativa, chiedendo aiuto quando necessario per risolvere le sfide incontrate. Questo approccio non solo favoriva una soluzione più rapida, ma contribuiva anche a mantenere una coerenza nel progresso del progetto.

È importante sottolineare che la maggior parte della collaborazione è avvenuta a distanza, utilizzando GitHub come piattaforma principale. La creazione di un repository ci ha consentito di caricare e condividere le modifiche in modo efficiente, garantendo che entrambi gli sviluppatori fossero sempre informati sullo stato corrente del progetto. Questa modalità di lavoro remota ha dimostrato di essere produttivo nel mantenere un'efficace sincronia tra i membri del team durante lo sviluppo del progetto.

Per lo sviluppo del progetto, abbiamo adottato l'IDE (Integrated Development Environment) Visual Studio Code perché grazie alle sue diverse estensioni e alla libertà che concede lo abbiamo preferito ad altri. Per quanto riguarda la gestione del database, abbiamo impiegato MySQL come tecnologia. Per l'intero progetto si è scelto di sfruttare NodeJs per la comunicazione tra il database e il client, per lo sviluppo di quest'ultimo si è optato per l'utilizzo di React che ha permesso la creazione di pagine dinamiche e il riutilizzo di alcune parti all'interno dell'intera applicazione.

Per l'intero sviluppo abbiamo sfruttato diverse librerie come **Sequelize** che ci ha permesso di gestire le tabelle del db direttamente dal codice javascript. Per la validazione dei form e per la creazione di strutture dati complesse direttamente dal client abbiamo implementato e utilizzato la libreria **Yup** insieme a **Formik** che ci ha semplificato la gestione degli errori, l'invio del form, in generale la scrittura del codice in maniera significativa. Per garantire una maggior sicurezza durante il login e la registrazione si sfrutta **Bcrypt**, una libreria che ci permette, lato server, l'hashing delle password in maniera semplice e sicura, utilizzato insieme alla libreria **JsonWebToken(JWT)**, che garantisce di trasmettere i dati tra client e server in modo compatto e sicuro.