



Commandline Team - Progetto "TuTourSelf"

Manuale Manutentore

Versione	2.0.0
Approvazione	Nicola Agostini
Redazione	Daniele Penazzo Alberto Battistini Giulia Corò
Verifica	Marco Giollo
Stato	Approvato
Uso	Esterno
Destinato a	Commandline Team Prof. Tullio Vardanega Prof. Riccardo Cardin TuTourSelf S.r.l.

Descrizione

Questo documento contiene il Manuale Manutentore per il progetto TuTourSelf del gruppo CommandLine Team

Contenuto

1	Changelog	4
2	Introduzione	5
2.1	Scopo del documento	5
2.2	Scopo del prodotto	5
2.3	Ambiguità e glossario	5
2.4	Assunzioni di conoscenza	5
3	Principali tecnologie utilizzate	5
3.1	Tecnologie per client e server	5
3.1.1	Google Maps API	5
3.1.2	Socket.io	5
3.1.3	Axios	6
3.2	Tecnologie per il client	6
3.2.1	React.js	6
3.2.2	HTML 5	6
3.2.3	CSS 3	6
3.2.4	babeljs	6
3.2.5	Bootstrap	6
3.3	Tecnologie per il server	6
3.3.1	Amazon Web Services (AWS)	6
3.3.2	Node.js	6
3.3.3	MongoDB e Mongoose	6
3.3.4	validate-vat	7
3.3.5	Passport.js	7
3.3.6	nodemailer	7
3.3.7	Express.js	7
3.3.8	BCrypt-NodeJS	7
3.3.9	i18next	7
3.3.10	React-i18next	7
3.4	Tecnologie per il Testing	7
3.4.1	Mocha	7
3.4.2	Istanbul/nyc	7
3.4.3	ESlint	7
3.4.4	UnitJS	7
3.4.5	Mockgoose	7
3.4.6	enzyme	8
4	Requisiti di Sistema	8
4.1	Dispositivi supportati	8
4.1.1	Browser Supportati	8
5	Configurazione dell'ambiente di lavoro	8
5.1	Installazione delle dipendenze	8
5.2	Installazione di Node.js	8
5.3	Installazione di NPM	9
5.4	Configurazione dell'applicazione	9
5.4.1	Clonazione della repository	9
5.4.2	Installazione/Aggiornamento dei moduli	9
5.4.3	Configurazione iniziale	9
5.4.4	Avvio dei test	9
5.4.5	Avvio del server	10
5.4.6	Facoltativo: Indice geospaziale	10
5.4.7	Facoltativo: Rigenerazione della Guida Dettagliata Alle Classi	10

6	Architettura	10
6.1	Formalismo	10
6.2	Visione generale	10
6.2.1	Frontend	11
6.2.2	Backend	12
6.2.3	Package contenuti	12
6.2.4	Principali librerie e framework usati	12
7	TuTourSelf::Frontend	13
7.1	TuTourSelf::Frontend:admin	14
7.2	TuTourSelf::Frontend:chat	14
7.3	TuTourSelf::Frontend::eventManagement	14
7.4	TuTourSelf::Frontend::feedback	15
7.5	TuTourSelf::Frontend::imageForm	15
7.6	TuTourSelf::Frontend::localManagement	15
7.7	TuTourSelf::Frontend::map	16
7.8	TuTourSelf::Frontend::proposal	16
7.9	TuTourSelf::Frontend::search	16
7.10	TuTourSelf::Frontend::signupForms	16
7.11	TuTourSelf::Frontend::usersAppBar	17
7.12	TuTourSelf::Frontend::usersInterface	17
7.13	TuTourSelf::Frontend::usersProfileEditor	17
7.14	TuTourSelf::Frontend::usersProfilePage	17
7.15	TuTourSelf::Frontend::viewablePage	18
8	TuTourSelf::Backend	18
8.1	TuTourSelf::Backend::models	18
8.2	TuTourSelf::Backend::signup	19
8.3	TuTourSelf::Backend::local_management	19
8.4	TuTourSelf::Backend::login	19
8.5	TuTourSelf::Backend::chat	19
8.6	TuTourSelf::Backend::search	20
8.7	TuTourSelf::Backend::routes	20
8.8	TuTourSelf::Backend::config	20
8.9	TuTourSelf::Backend::event_management	21
9	Interfacce esposte	21
9.1	Routes	21
9.1.1	Admin	21
9.1.2	Agreement	22
9.1.3	Artist	23
9.1.4	Auth	23
9.1.5	Bans	24
9.1.6	Event	25
9.1.7	Feedback	26
9.1.8	Local	27
9.1.9	Map	27
9.1.10	Place	27
9.2	Profile	28
9.2.1	Reports	29
9.2.2	Search	30
9.3	Segnali WebSocket	30
9.3.1	Backend \Rightarrow Frontend	30
9.3.1.1	history	30
9.3.1.2	backToLogin	30
9.3.1.3	notification	30
9.3.1.4	youdead	30

9.3.1.5	roomList	30
9.3.1.6	newProposal	30
9.3.1.7	proposalRevoked	30
9.3.1.8	proposalConfirmed	30
9.3.1.9	agreementRevokeConfirmation	30
9.3.1.10	message	30
9.3.2	Frontend \Rightarrow Backend	30
9.3.2.1	roomMessage	30
9.3.2.2	roomConnect	30
9.3.2.3	messageAck	30
9.3.2.4	getRooms	30
9.3.2.5	leave	30
10	Descrizione dettagliata delle classi	31
11	Punti di estensione	31
11.1	Internazionalizzazione	31
11.1.1	Inserimento di una nuova lingua	31
11.2	Nuovi modelli nel backend	32
11.3	Nuove routes nel backend	33
11.4	Nuove pagine di frontend	34
A	Glossario	36

1 Changelog

Versione	Data	Nominativo	Ruolo	Descrizione
2.0.0	2018-06-13	Nicola Agostini	Responsabile	Approvazione del documento.
1.1.0	2018-06-13	Maro Giollo	Verificatore	Verifica del documento.
1.0.6	2018-06-07	Daniele Penazzo	Progettista	Riferimento Documentazione Classi
1.0.5	2018-06-06	Daniele Penazzo	Progettista	Aggiunta interfaccia backend
1.0.4	2018-06-05	Giulia Corò	Progettista	Inserita descrizione delle nuove classi del frontend
1.0.3	2018-06-05	Giulia Corò	Progettista	Inserito punto di estensione frontend
1.0.2	2018-06-05	Daniele Penazzo	Progettista	Aggiornamento interfacce backend
1.0.1	2018-06-03	Daniele Penazzo	Progettista	Incremento installazione e tecnologie usate
1.0.0	2018-05-07	Alberto Battistini	Responsabile	Approvazione del documento.
0.1.0	2018-05-06	Marco Giollo	Verificatore	Prima verifica dell'intero documento.
0.0.7	2018-05-05	Giovanni Motterle, Giulia Corò	Progettisti	Inserite descrizioni tecnologie per il client
0.0.6	2018-05-05	Giulia Corò	Progettista	Inserita descrizione delle componenti frontend
0.0.5	2018-05-04	Daniele Penazzo	Progettista	Inserite classi frontend
0.0.4	2018-05-04	Daniele Penazzo	Progettista	Aggiunta sezioni visione generale e riferimento classi
0.0.3	2018-05-03	Alberto Battistini	Progettista	Descrizione tecnologie Google Maps API e SuperAgent
0.0.2	2018-05-03	Daniele Penazzo	Progettista	Aggiunta tecnologie, intro e requisiti
0.0.1	2018-05-02	Daniele Penazzo	Progettista	Prima redazione

Tabella 1: Changelog di questo documento

2 Introduzione

2.1 Scopo del documento

Questo documento rappresenta il *Manuale del Manutentore* per l'applicazione web *TuTourSelf* sviluppata da CommandLine Team.

La finalità di questo documento è fornire al manutentore le modalità di installazione e di utilizzo del prodotto, i requisiti necessari per l'uso dello stesso, oltre ai moduli ed i *framework_G* utilizzati per lo sviluppo.

Questo documento inoltre contiene la struttura dei *package_G* e delle *classi_G* contenuti nell'applicazione, così da rendere più semplice la ricerca e l'eventuale aggiunta di funzionalità.

2.2 Scopo del prodotto

Il capitolato si prefigge di creare una piattaforma per facilitare comunicazione ed eventuale successivo accordo_G tra artisti indipendenti e locali.

In particolare, il prodotto si prefigge di:

- Creare un database centralizzato di artisti/gruppi, che possa inglobare collegamenti con eventuali altri profili di ciascun artista/gruppo (es. sito web personale, Youtube, Bandcamp, Pinterest, varie pagine social);
- Creare un database centralizzato di locali convenzionati, che puntualizzi la tipologia di artista richiesta e la strumentazione messa a disposizione dal locale stesso;
- Creare un'applicazione web (che poi in futuro possa essere facilmente trasformata in applicazione mobile) che permetta **facilmente** di mettere in contatto artisti e locali (tenendo conto della compatibilità tra le caratteristiche dell'artista e quelle del locale), consentendo di giungere ad un accordo tra le due parti. L'accordo, tra le altre cose, deve definire la data dell'esibizione, la durata dello show e il compenso concordato;
- Creare un sistema che permetta ad artisti, locali e pubblico di rilasciare feedback in base a parametri prestabiliti (comportamento, professionalità, quantità di pubblico, incasso ecc.);
- Creare un modulo che permetta al gestore del locale di pagare facilmente il cachet all'artista/gruppo.

2.3 Ambiguità e glossario

Al fine di evitare ambiguità dovute all'uso di linguaggio tecnico specifico, in appendice A questo documento è inserito un glossario che riporta le definizioni dei termini riportati con una "G" a pedice.

2.4 Assunzioni di conoscenza

Nella redazione di questo manuale, si assume che il manutentore abbia conoscenze nel settore della programmazione ad eventi e JavaScript, oltre che una conoscenza almeno basilare del linguaggio UML2.

3 Principali tecnologie utilizzate

3.1 Tecnologie per client e server

3.1.1 Google Maps API

Google Maps è un servizio che consente la ricerca e visualizzazione di carte geografiche di buona parte della terra. Attraverso le *Google Maps API* è possibile realizzare mappe personalizzate, applicazioni mobile che le integrino, oppure creare applicazioni web che forniscono servizi di ricerca personalizzati.

3.1.2 Socket.io

Socket.io è un modulo che consente ad un'applicazione node.js di mettere a disposizione dell'utente una comunicazione bidirezionale basata su eventi in tempo reale.

3.1.3 Axios

Axios è un client HTTP per node.js e browsers, che permette di fare richieste HTTP frontend \implies backend o backend \implies server esterno.

3.2 Tecnologie per il client

3.2.1 React.js

React è una libreria JavaScript utilizzata per costruire interfacce grafiche di single-page applications e applicazioni mobile. È open source ed è mantenuta da Facebook e da una comunità di sviluppatori. Attraverso React è possibile definire gli elementi fondamentali di una UI, ovvero i componenti, e assemblarli in modo da ottenere la pagina desiderata. I cambiamenti nello stato dei componenti sono automaticamente riflessi nell'interfaccia grafica, rendendo più agevole lo sviluppo di progetti medio/grandi.

3.2.2 HTML 5

L'HTML è un linguaggio di markup di pubblico dominio, la cui sintassi è stabilita dal World Wide Web Consortium (*W3C*). La versione attuale, la quinta, è stata rilasciata dal W3C nell'ottobre 2014.

3.2.3 CSS 3

Il CSS è un linguaggio usato per definire la formattazione di documenti HTML, XHTML e XML. Le regole per comporre il CSS sono contenute in un insieme di direttive emanate a partire dal 1996 dal *W3C*. Le specifiche CSS3 sono costituite da sezioni separate dette "moduli". A causa di questa modularizzazione, le specifiche CSS3 hanno differenti stati di avanzamento e stabilità.

3.2.4 babel.js

Babel è un transpiler che permette di convertire codice sorgente scritto secondo lo standard ECMAScript 6_G in codice ECMAScript 5_G.

3.2.5 Bootstrap

Bootstrap è definito un "HTML, CSS, and JS toolkit from Twitter", ovvero una raccolta di strumenti grafici, stilistici ed impaginazione che permettono di avere a disposizione una gran quantità di funzionalità e di stili modificabili e adattabili a seconda delle nostre esigenze. Bootstrap è compatibile con tutte le ultime versioni dei principali browser e la sua principale funzione è il responsive web design, ovvero il fatto che il design delle pagine web sia in grado di adattarsi dinamicamente a seconda della grandezza e delle caratteristiche del dispositivo utilizzato. Esso si pone, perciò, come una libreria multidispositivo e multiplatforma.

3.3 Tecnologie per il server

3.3.1 Amazon Web Services (AWS)

Amazon Web Services è un servizio di *cloud computing*_G che permette la creazione di istanze di server virtuali con possibilità di scalabilità automatizzata.

3.3.2 Node.js

Node.js è un ambiente dedicato all'esecuzione di codice *JavaScript*_G lato server. Questo ambiente è dotato di un'architettura ad eventi che permette input/output asincroni, allo scopo di ottimizzare *throughput*_G e *scalabilità*_G di un'applicazione web.

3.3.3 MongoDB e Mongoose

MongoDB è un database di tipo *NoSQL*_G orientato ai documenti in stile *JSON*_G, mentre Mongoose è uno strumento di modellazione per oggetti in MongoDB.

3.3.4 validate-vat

Validate-vat è uno script scritto in CoffeeScript atto a verificare e validare partite IVA emesse da un qualunque stato europeo.

3.3.5 Passport.js

Passport è un modulo di autenticazione per Node.js, con supporto per login interne o tramite Facebook, Twitter e Google.

3.3.6 nodemailer

Nodemailer si propone come un modulo molto semplice per l'invio di email tramite node.js.

3.3.7 Express.js

Express.js è un framework per applicazioni web minimale, estensibile tramite plugins e con un set di caratteristiche atte alla creazione rapida di applicazioni web sia multi-page che single-page, oltre ad applicazioni di tipo ibrido.

3.3.8 BCrypt-NodeJS

Questo è un'implementazione in JavaScript puro dell'algoritmo di hashing BCrypt di dimensioni molto ridotte.

3.3.9 i18next

i18next è un framework di internazionalizzazione utilizzabile su nodeJS o altri ambienti basati su javascript e non solo.

3.3.10 React-i18next

React-i18next fornisce delle componenti React specifiche per velocizzare e facilitare l'internazionalizzazione di applicazioni basate su React.

3.4 Tecnologie per il Testing

3.4.1 Mocha

Mocha è un framework di testing per java che funziona in maniera asincrona. Consente l'inserimento di moduli e contiene degli hook per moduli di reporting.

3.4.2 Istanbul/nyc

Istanbul è un modulo di reporting che consente di analizzare la code coverage e le righe non coperte da test, nyc è la sua interfaccia da linea di comando.

3.4.3 ESLint

ESLint è uno strumento di analisi statica per Javascript e JSX, con possibilità di impostare le proprie regole stilistiche e possibilità di integrazione con la maggior parte degli IDE ed editor esistenti.

3.4.4 UnitJS

Unit.js è un framework dedicato prettamente allo unit testing e mette a disposizione strumenti per il mocking, lo spying e lo stubbing di moduli oltre alla creazione di assert più leggibili tramite "should" e "must".

3.4.5 Mockgoose

Mockgoose è uno strumento specificamente dedicato al mocking di database mongoose, così da poter effettuare test senza sporcare il database di produzione.

3.4.6 enzyme

Enzyme è un set di strumenti dedicati al testing di applicazioni React.

4 Requisiti di Sistema

TuTourSelf è un'applicazione web funzionante tramite browser, sia mobile che desktop.

4.1 Dispositivi supportati

L'applicazione è stata testata e risulta compatibile con i seguenti sistemi desktop:

- Microsoft Windows 10;
- Apple OSX Sierra;
- Gentoo Linux (Profile v17).

4.1.1 Browser Supportati

Di seguito viene fornito un breve elenco con le versioni minime di tutti i browser sui quali il funzionamento del nostro prodotto è garantito:

- Google Chrome_G: il sistema viene supportato dalla versione **49**;
- Internet Explorer_G: il sistema viene supportato dalla versione **9**;
- Microsoft Edge_G: il sistema viene supportato dalla versione **15**;
- Firefox_G: il sistema viene supportato dalla versione **5**;
- Safari_G: il sistema viene supportato dalla versione **11**.

ATTENZIONE: Per rendere effettivo il funzionamento su tali browser, deve necessariamente essere abilitato JavaScript_G.

5 Configurazione dell'ambiente di lavoro

5.1 Installazione delle dipendenze

Per funzionare, è necessario installare alcune dipendenze prima di procedere alla configurazione dell'applicazione *TuTourSelf*.

5.2 Installazione di Node.js

Per prima cosa è necessario installare Node.js, è possibile installarlo tramite il proprio gestore di pacchetti (da root):

Gentoo Linux e derivate: emerge nodejs.

Debian e derivate/Ubuntu: apt-get install nodejs.

Fedora \geq 22: dnf install nodejs.

Fedora $<$ 22 e derivate: yum install nodejs.

ArchLinux e derivate: pacman -S nodejs.

Alternativamente è possibile fare riferimento al sito ufficiale <https://nodejs.org>.

5.3 Installazione di NPM

Per poter installare le altre dipendenze di TuTourSelf è necessario installare *Node Package Manager* (NPM), solitamente questo applicativo è installato durante l'installazione di Node.js, è possibile controllarne la presenza tramite il comando `npm -v`, in caso positivo tale comando dovrebbe restituire la versione di NPM installata. In caso non fosse disponibile è possibile fare riferimento al sito ufficiale <https://www.npmjs.com/get-npm>.

5.4 Configurazione dell'applicazione

5.4.1 Clonazione della repository

Per prima cosa è necessario clonare la repository privata fornita. Questo può essere fatto con il comando `git clone [URL]`.

5.4.2 Installazione/Aggiornamento dei moduli

Successivamente è necessario spostarsi all'interno della cartella del repository ed assicurarsi che sia presente il file `package.json`.

Se il file è presente è possibile installare tutte le dipendenze tramite il comando `npm run rebuild` che si occuperà dell'installazione di tutti i moduli necessari ed ad avviare la build del frontend.

5.4.3 Configurazione iniziale

Prima di avviare il server è necessario completare alcune configurazioni. Per prima cosa è necessario completare la configurazione del modulo di invio email, inserendo email e password della casella email all'interno del file `/src/config/mailer.js`, modificando i seguenti campi:

```
1 auth: {
2   user: '<email qui>',
3   pass: '<password casella email>'
4 }
```

Successivamente è necessario configurare i token ed i secret per l'autenticazione tramite Facebook e Google SignIn. È necessario modificare la seguente sezione del file `/src/config/auth.js`:

```
1 module.exports = {
2   'facebookAuth' : {
3     'clientId'    : 'App ID',
4     'clientSecret' : 'Secret Applicazione',
5     'callbackURL' : 'https://www.dominio.com/auth/facebook/callback',
6     'profileURL'  : 'https://graph.facebook.com/v2.5/
7                   me?fields=first_name,last_name,email',
8   },
9   'googleAuth' : {
10    'clientId'     : 'ID Client Google',
11    'clientSecret' : 'Secret Client',
12    'callbackURL'  : 'https://www.dominio.org/auth/google/callback'
13  }
14 }
```

Per avere i dati necessari, è sufficiente creare una nuova applicazione sul sito di Facebook Developers (<https://developers.facebook.com/>) e Google Developers Console (<https://console.developers.google.com/>).

5.4.4 Avvio dei test

TuTourSelf dispone di due script per l'avvio dei test, il primo è eseguibile tramite `npm test` ed esegue tutti i test, creando un rapporto di copertura del codice sintetico, su file, utile per essere interpretato ed inserito automaticamente in un cruscotto informativo che supporti serie temporali.

Il secondo script è eseguibile tramite il comando `npm run test2` che esegue tutti i test crea un rapporto completo

su schermo, mostrando la copertura di statements, branch, funzioni e linee, oltre che le linee di codice non coperte da test; questo script è utile per incrementare la copertura dei test, andando ad agire miratamente sulle porzioni di codice non coperte.

5.4.5 Avvio del server

Al termine dell'installazione, della build e della configurazione è possibile avviare il server dalla root del progetto tramite il comando `npm start`.

5.4.6 Facoltativo: Indice geospaziale

Per migliorare le prestazioni della ricerca su mappa, è possibile creare un indice geospaziale di MongoDB.

Dopo aver avviato il server ed aver lasciato che avvengano alcune registrazioni, avviare la console di mongo da linea di comando sulla macchina dove l'applicazione è installata tramite il comando `mongo`.

Successivamente selezionare il database dell'applicazione tramite il comando `use tutourself`.

Usando il comando `show collections`, assicurarsi che esista la collezione "local"; se esiste è possibile creare un indice geospaziale con il seguente comando:

```
db.local.createIndex(location: "2dsphere")
```

Se MongoDB risponde con una risposta simile a questa:

```
1 {
2     "createdCollectionAutomatically" : false,
3     "numIndexesBefore" : 1,
4     "numIndexesAfter" : 2,
5     "ok" : 1
6 }
```

allora saprete che l'indice è stato creato correttamente (e sarà automaticamente mantenuto aggiornato da MongoDB). L'uso di un indice geospaziale non è obbligatorio per il funzionamento dell'applicazione, ma permette un sensibile incremento prestazionale nell'uso della pianificazione dei tour su mappa.

5.4.7 Facoltativo: Rigenerazione della Guida Dettagliata Alle Classi

TuTourSelf contiene nel proprio codice un generatore di documentazione in stile JavaDoc/Doxygen. Per poter generare la versione più recente di tale documentazione è sufficiente eseguire il comando

```
jsdoc src -r -c config/jsdoc.config.json -d docs
```

dalla root del progetto. Per visualizzare la guida è sufficiente aprire il file `/docs/index.html`.

6 Architettura

6.1 Formalismo

L'architettura dell'applicativo sarà descritta dal generale al particolare; quindi si descriveranno prima i package e le componenti e successivamente si vedranno in dettaglio le classi contenute, fornendone una descrizione, specificandone l'uso e le relazioni con altre classi.

6.2 Visione generale

TuTourSelf è un'applicazione web suddivisa in due parti principali che comunicano tra loro trasmettendosi oggetti JSON tramite richieste HTTP.

6.2.1 Frontend

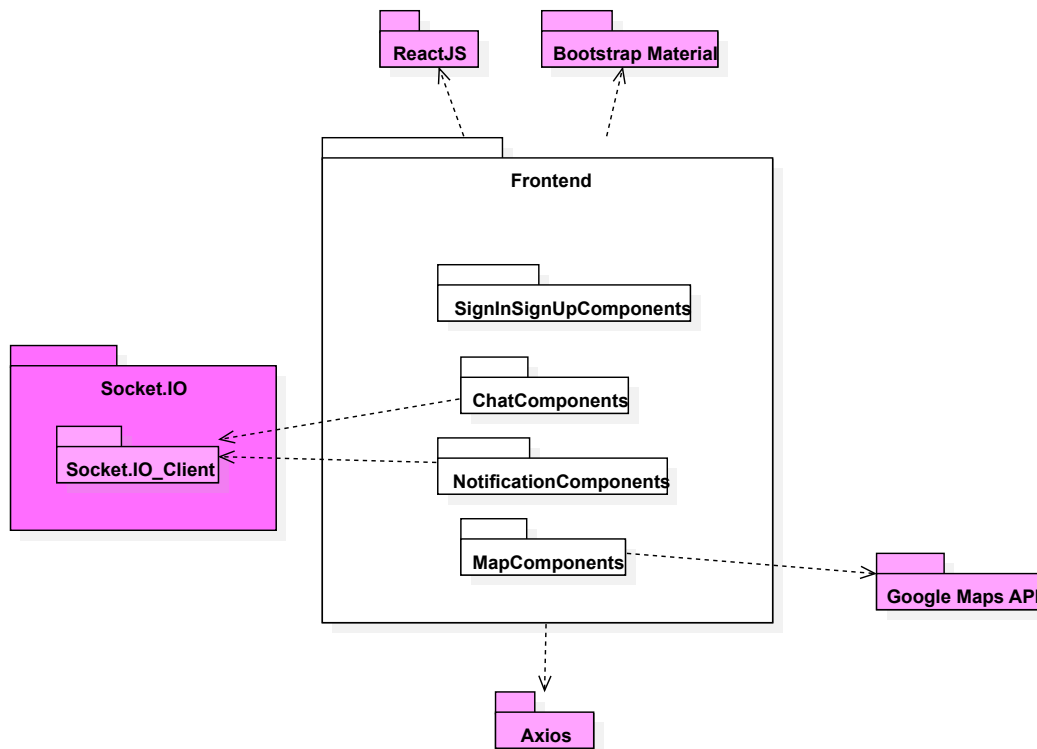


Immagine 1: Visione generale del frontend

Il frontend è una *Single Page Application* (SPA) costruita facendo uso di HTML5, CSS3, JavaScript e React. Questa parte dell'applicazione è intesa per essere semplice e leggera, scaricando la maggior parte del lavoro sul backend ed è costruita secondo un modello a componenti, incoraggiato da React. Il frontend fa uso di Axios per comunicare con il backend tramite richieste HTTP a specifiche path.

6.2.2 Backend

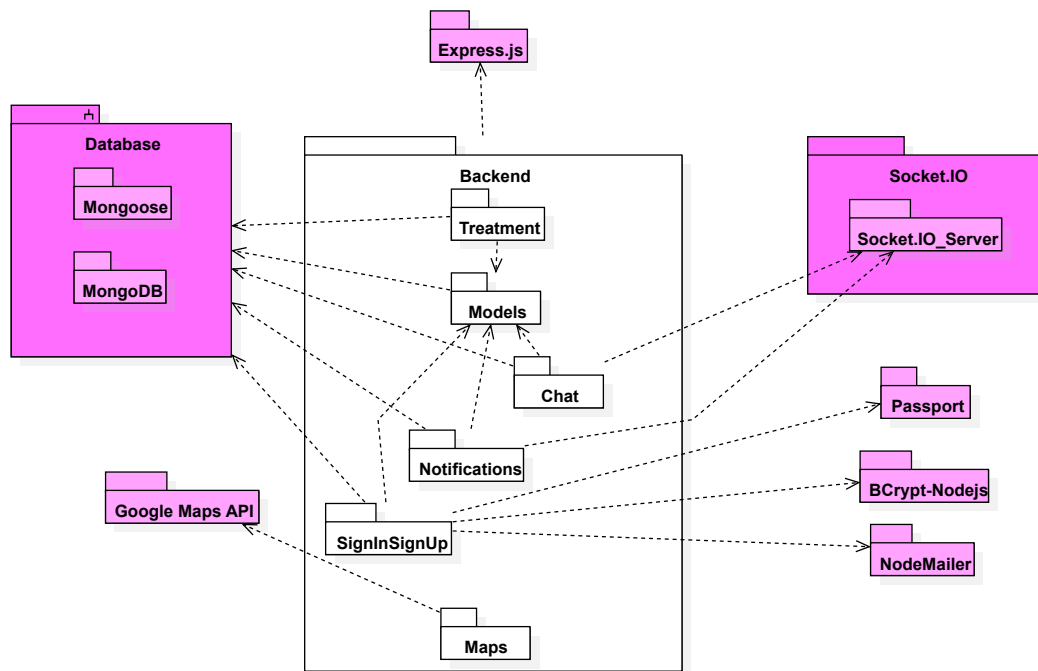


Immagine 2: Visione generale del backend

Il backend è stato sviluppato in modo modulare, facendo uso del paradigma ad eventi incoraggiato da Node.js. Il backend si occupa della conservazione dei dati e della loro elaborazione, della gestione delle sessioni, delle ricerche e dell'invio dei dati richiesti al frontend.

6.2.3 Package contenuti

TuTourSelf::Frontend Contenente le classi ed i package che vanno a comporre il frontend di TuTourSelf;

TuTourSelf::Backend Contenente le classi ed i package che vanno a comporre il backend di TuTourSelf.

6.2.4 Principali librerie e framework usati

Express: un framework minimale, ed estensibile tramite plugin, per la creazione di applicazioni web tramite Node.js, implementa un più pulito ed efficace sistema di routing;

React: un framework JavaScript che permette di creare Single Page Applications facendo uso di un modello a componenti, incoraggiando il riuso;

Bootstrap Material: un framework HTML/CSS/JavaScript che permette di costruire facilmente pagine in stile Material Design;

Mongoose: driver e tool di modellazione per MongoDB, mongoose permette di definire facilmente schemi e modelli per il database, imponendo una struttura ben definita ai documenti;

Socket.io: libreria per la comunicazione in tempo reale tramite websockets, socket.io permette di implementare in maniera semplice e pulita sistemi di chat e di notifica in tempo reale;

Axios: una libreria di comunicazione asincrona tramite richieste HTTP, permette di comunicare facilmente e velocemente con il backend tramite richieste poste su determinate routes, oltre ad essere usato per la comunicazione con le API di Google Maps;

Passport: un framework di autenticazione estensibile tramite l'implementazione di uno strategy pattern interno, permette l'autorizzazione e l'autenticazione all'applicazione;

BCrypt-Nodejs: una libreria che fornisce un'implementazione nativa dell'algoritmo di salted hashing BCrypt, usata per confrontare e fare salted hashing di password;

NodeMailer: una libreria che consente l'invio automatizzato di email, usato per inviare le email di verifica delle registrazioni effettuate.

7 TuTourSelf::Frontend

Il pacchetto frontend contiene alcune componenti generali che non si ritiene conveniente inserire in pacchetti separati.

TuTourSelf::Frontend::Appbar La barra delle applicazioni che contiene i link necessari per la navigazione sul sito dell'utente non autenticato, ossia:

- L'homepage contenente una barra di ricerca;
- La pagina di autenticazione;
- La pagina di registrazione.

TuTourSelf::Frontend::BasicConfig Questo modulo esporta un oggetto che definisce il dominio al quale effettuare le richieste asincrone utilizzando il package *axios*.

TuTourSelf::Frontend::ConfirmDialog Questa componente rappresenta un *dialog* per poter confermare o annullare le proprie azioni.

TuTourSelf::Frontend::Dashboard La bacheca dell'utente admin. In questa pagina l'utente potrà visualizzare le segnalazioni eseguite e gli utenti esclusi dalla piattaforma.

TuTourSelf::Frontend::LoggedSearch In questa pagina l'utente potrà effettuare una ricerca delle pagine di artisti, locali ed eventi registrati nel sito.

TuTourSelf::Frontend::NotificationHelper Questa componente è una classe d'appoggio che serve a mostrare le notifiche in maniera più semplice.

TuTourSelf::Frontend::SecondStepSignup Questa pagina fa parte della procedura di registrazione al sito. Rappresenta appunto il "secondo step" della registrazione, che consiste nella scelta del proprio tipo utente (artista, gestore di locali oppure spettatore), che servirà per il reindirizzamento dell'utente ad una form specifica per il proprio tipo utente.

TuTourSelf::Frontend::Signin La pagina di autenticazione. Un utente che è già registrato al sito potrà accedervi da questa pagina in ogni momento, grazie all'inserimento della propria e-mail e della password scelta in precedenza.

TuTourSelf::Frontend::SigninSignupSearch Questa è la componente radice che tramite le route di React reindirizza le varie pagine visibili all'utente non autenticato, ossia:

- *TuTourSelf::Frontend::UnloggedSearch*;
- *TuTourSelf::Frontend::Signin*;
- *TuTourSelf::Frontend::Signup* .

TuTourSelf::Frontend::Signup La pagina di registrazione. Qui è visibile il "primo step" della registrazione. L'utente dovrà inserire e-mail, password e nuovamente la password per un'ulteriore conferma, dopodiché riceverà un'email per la conferma della registrazione.

TuTourSelf::Frontend::ThirdStepSignup Il "terzo step" della registrazione di un utente. Questa componente ha il compito di instradare le ruote alle tre diverse form per la registrazione degli utenti, che sono:

- *[TuTourSelf::Frontend::signupForms::ArtistSignup]*;
- *[TuTourSelf::Frontend::signupForms::LocalManagerSignup]*;
- *[TuTourSelf::Frontend::signupForms::SpectatorSignup]*.

TuTourSelf::Frontend::UnloggedSearch Pagina che contiene l'homepage del sito, che comprende il logo TuTourSelf, una breve descrizione, e la barra di ricerca.

TuTourSelf::Frontend::UserInterface Questa è la componente radice che tramite le route di React reindirizza le varie interfacce utente, a seconda del tipo di utente che è autenticato in quel momento. Queste sono:

- `[TuTourSelf::Frontend::usersInterface::ArtistInterface];`
- `[TuTourSelf::Frontend::usersInterface::LocalManagerInterface];`
- `[TuTourSelf::Frontend::usersInterface::SpectatorInterface].`

7.1 TuTourSelf::Frontend:admin

Questo pacchetto contiene tutte le componenti che riguarda il sistema di amministrazione interna.

TuTourSelf::Frontend::admin::banListViewer Questa componente permette all'amministratore di vedere e gestire i ban degli utenti.

TuTourSelf::Frontend::admin::messageViewer Questa componente permette all'amministratore di vedere i messaggi che gli utenti si sono scambiati tramite chat.

TuTourSelf::Frontend::admin::reportListViewer Questa componente permette all'amministratore di vedere e gestire la lista degli utenti segnalati.

7.2 TuTourSelf::Frontend:chat

Questo pacchetto contiene tutte le componenti che riguardano il sistema di chat interno.

TuTourSelf::Frontend::chat::ActiveChat Componente che rappresenta la finestra della chat attiva.

TuTourSelf::Frontend::chat::ChatHistory Componente che rappresenta l'anteprima di una chat avuta in precedenza, sulla quale si potrà cliccare per aprire la rispettiva chatroom. L'anteprima della chat consiste nella visualizzazione del solo nome dell'interlocutore.

TuTourSelf::Frontend::chat::ChatHistoryItem Componente che rappresenta lo spazio della chat nel quale verranno mostrate tutte le conversazioni precedenti.

TuTourSelf::Frontend::chat::ChatInterlocutorMessage Componente che rappresenta un singolo messaggio di testo che viene ricevuto.

TuTourSelf::Frontend::chat::Chat Pagina della chat. Tutte le componenti che riguardano la chat sono figlie di questa componente.

TuTourSelf::Frontend::chat::ChatMyMessage Componente che rappresenta un singolo messaggio di testo che viene inviato.

TuTourSelf::Frontend::chat::ChatProposal Componente che rappresenta lo spazio all'interno della pagina chat nel quale verrà posizionata la proposta.

TuTourSelf::Frontend::chat::SocketWrapper Componente che incapsula tutte le chiamate che avvengono tra il backend e il frontend attraverso il package *socket.io*, il quale viene utilizzato nel sistema soltanto per funzioni relative alla chat.

7.3 TuTourSelf::Frontend::eventManagement

Questo pacchetto contiene le componenti riguardanti il sistema di gestione degli eventi.

TuTourSelf::Frontend::eventManagement::EventCreationForm Questa componente renderizza un dialog con una form per la creazione dell'evento, in cui si dovranno inserire Titolo e Descrizione, dato che gli altri campi vengono compilati in automatico dall'accordo.

TuTourSelf::Frontend::eventManagement::EventItem Questa componente renderizza l'anteprima di un oggetto evento, visualizzabile e modificabile dal gestore di locale.

TuTourSelf::Frontend::eventManagement::EventItemArtist Questa componente renderizza l'anteprima di un oggetto evento, visualizzabile dall'artista, e non modificabile.

TuTourSelf::Frontend::eventManagement::EventManagementPage Questa componente rappresenta la pagina del gestore di locali per la gestione degli eventi. In essa può visualizzare e modificare i propri eventi.

TuTourSelf::Frontend::eventManagement::EventShowcase Questa componente rappresenta la pagina dell'artista per la gestione degli eventi. In essa può visualizzare gli eventi ai quali partecipa.

TuTourSelf::Frontend::eventManagement::ModifyEvent Questa componente renderizza un dialog con una form per la modifica dell'evento.

TuTourSelf::Frontend::eventManagement::SmallEventItem Questa componente renderizza l'anteprima ridotta di un oggetto evento, che verrà mostrato nelle *viewablePage* di artisti e locali.

7.4 TuTourSelf::Frontend::feedback

Questo pacchetto contiene le componenti riguardanti il sistema di invio e gestione dei feedback.

TuTourSelf::Frontend::feedback::ArtistFeedbackFormLocal Questa componente renderizza una finestra modale con la form che permette al gestore di locale di rilasciare un feedback ad un artista.

TuTourSelf::Frontend::feedback::ArtistFeedbackFormSpectator Questa componente renderizza una finestra modale con la form che permette allo spettatore di rilasciare un feedback ad un artista.

TuTourSelf::Frontend::feedback::ArtistFeedbackViewer Questa componente renderizza la media dei feedback ottenuti da un artista, che sarà visualizzabile all'interno della sua pagina pubblica.

TuTourSelf::Frontend::feedback::LocalFeedbackFormArtist Questa componente renderizza una finestra modale con la form che permette all'artista di rilasciare un feedback ad un locale.

TuTourSelf::Frontend::feedback::LocalFeedbackFormSpectator Questa componente renderizza una finestra modale con la form che permette allo spettatore di rilasciare un feedback ad un locale.

TuTourSelf::Frontend::feedback::LocalFeedbackViewer Questa componente renderizza la media dei feedback ottenuti da un locale, che sarà visualizzabile all'interno della sua pagina pubblica.

7.5 TuTourSelf::Frontend::imageForm

Questo pacchetto contiene le componenti riguardanti il sistema di gestione delle immagini (upload e rimozione).

TuTourSelf::Frontend::imageForm::ImageInput Questa componente contiene i bottoni per l'upload e la rimozione di un immagine (profilo, di eventi o di locali).

7.6 TuTourSelf::Frontend::localManagement

Questo pacchetto contiene le componenti riguardanti il sistema di gestione dei locali da parte dell'utente registrato come gestore di locali.

TuTourSelf::Frontend::localManagement::LocalCreationForm Form per l'aggiunta di un nuovo locale alla lista dei propri locali.

TuTourSelf::Frontend::localManagement::LocalPreview Componente che rappresenta la miniatura di un locale, sulla quale il gestore può cliccare per accedere alla rispettiva pagina locale.

TuTourSelf::Frontend::localManagement::ManageLocals Pagina dove il gestore può visualizzare e gestire tutti i propri locali.

7.7 TuTourSelf::Frontend::map

Questo pacchetto contiene le componenti riguardanti le ricerche su mappa e la pianificazione del tour.

TuTourSelf::Frontend::map::MapWithADirectionsRenderer Componente che rappresenta la mappa in cui viene renderizzato il percorso intrapreso dall'utente . Sono visualizzati su mappa e su lista i locali che si trovano lungo il percorso e che distano entro un raggio massimo, impostato opzionalmente, dal percorso scelto.

TuTourSelf::Frontend::map::Map Componente che rappresenta una mappa Google base in cui collocare dei punti attraverso coordinate geografiche.

TuTourSelf::Frontend::map::Locals Form per la scelta di origine e destinazione del percorso intrapreso dall'utente. Opzionalmente si può impostare anche un raggio che di default è 10 km; saranno visualizzati su mappa solamente i locali che rientrano nell'area del cerchio creato da tale raggio.

TuTourSelf::Frontend::map::Events Componente che rappresenta la mappa in cui vengono visualizzati gli eventi ricercati dall'utente.

7.8 TuTourSelf::Frontend::proposal

Questo pacchetto contiene le componenti riguardanti proposta, controproposta ed accordo.

TuTourSelf::Frontend::proposal::ProposalItem Questa componente rappresenta la sezione della chat nella quale appariranno le proposte.

TuTourSelf::Frontend::proposal::ReplyProposal Questa componente rappresenta il dialog per rispondere alla proposta, ed inviare quindi una controproposta.

TuTourSelf::Frontend::proposal::RevokeProposalModal Questa componente renderizza una finestra modale per la revoca della proposta.

TuTourSelf::Frontend::proposal::ViewReplyProposal Questa componente rappresenta una singola proposta, con visualizzati tutti i dati ad essa relativi.

7.9 TuTourSelf::Frontend::search

Questo pacchetto contiene le componenti riguardanti la ricerca tramite keywords di locali, artisti ed eventi.

TuTourSelf::Frontend::search::SearchResult Questa componente rappresenta la pagina che mostra tutti i risultati di una ricerca per keywords.

TuTourSelf::Frontend::search::SingleResult Questa componente rappresenta un singolo risultato della ricerca, che sarà cliccabile per poter accedere alla pagina del risultato in questione.

7.10 TuTourSelf::Frontend::signupForms

Questo pacchetto contiene le componenti riguardanti il "terzo step" della registrazione degli utenti, ossia la parte che riguarda i dati specifici per ogni tipo di utente.

TuTourSelf::Frontend::signupForms::ArtistSignup Form per la registrazione di un utente come artista.

TuTourSelf::Frontend::signupForms::LocalManagerSignup Form per la registrazione di un utente come gestore di locali.

TuTourSelf::Frontend::signupForms::SpectatorSignup Form per la registrazione di un utente come spettatore.

7.11 TuTourSelf::Frontend::usersAppbar

Questo pacchetto contiene le appbar specifiche per ogni categoria di utente, oltre ad una aggiuntiva mostrata durante le fasi intermedie della registrazione.

TuTourSelf::Frontend::usersAppbar::ArtistAppbar Barra delle applicazioni visibile ad un utente autenticato come artista.

TuTourSelf::Frontend::usersAppbar::LocalManagerAppbar Barra delle applicazioni visibile ad un utente autenticato come gestore di locali.

TuTourSelf::Frontend::usersAppbar::SecondStepSignup Barra delle applicazioni visibile ad un utente che sta completando la sua registrazione (ossia che sta effettuando il "secondo step" oppure il "terzo step" della registrazione).

TuTourSelf::Frontend::usersAppbar::SpectatorAppbar Barra delle applicazioni visibile ad un utente autenticato come spettatore.

7.12 TuTourSelf::Frontend::usersInterface

Questo pacchetto contiene le interfacce specifiche per ogni tipo di utente.

TuTourSelf::Frontend::usersInterface::ArtistInterface Interfaccia specifica per un utente autenticato come artista. Le varie pagine sono raggiunte grazie ai link presenti nell'*ArtistAppbar*.

TuTourSelf::Frontend::usersInterface::LocalManagerInterface Interfaccia specifica per un utente autenticato come gestore di locali. Le varie pagine sono raggiunte grazie ai link presenti nella *LocalManagerAppbar*.

TuTourSelf::Frontend::usersInterface::SpectatorInterface Interfaccia specifica per un utente autenticato come spettatore. Le varie pagine sono raggiunte grazie ai link presenti nella *SpectatorAppbar*.

7.13 TuTourSelf::Frontend::usersProfileEditor

Questo pacchetto contiene le sezioni di modifica profilo dei vari tipi d'utenza.

TuTourSelf::Frontend::usersProfileEditor::EditArtistProfile Form per la modifica delle informazioni contenute nella pagina di un artista.

TuTourSelf::Frontend::usersProfileEditor::EditManagerProfile Form per la modifica delle informazioni contenute nella pagina di un gestore di locali.

TuTourSelf::Frontend::usersProfileEditor::EditPlacePage Form per la modifica delle informazioni contenute nella pagina di un locale.

TuTourSelf::Frontend::usersProfileEditor::EditSpectatorPage Form per la modifica delle informazioni contenute nella pagina di uno spettatore.

7.14 TuTourSelf::Frontend::usersProfilePage

Questo pacchetto contiene le componenti atte alla visualizzazione delle pagine di profilo.

TuTourSelf::Frontend::usersProfilePage::ArtistPage Pagina pubblica di un utente autenticato come artista.

TuTourSelf::Frontend::usersProfilePage::LocalPage Pagina pubblica di un locale.

TuTourSelf::Frontend::usersProfilePage::ManagerPrivateProfile Pagina profilo di un utente autenticato come gestore di locali.

TuTourSelf::Frontend::usersProfilePage::SpectatorPrivateProfile Pagina profilo di un utente autenticato come spettatore.

7.15 TuTourSelf::Frontend::viewablePage

TuTourSelf::Frontend::viewablePage::ArtistToLocalDialogProposal Finestra modale che contiene la form per l'invio di una proposta ad un locale da parte di un artista.

TuTourSelf::Frontend::viewablePage::ArtistViewablePage Pagina di un artista, che può essere visualizzata dagli utenti attraverso i risultati della ricerca.

TuTourSelf::Frontend::viewablePage::BanForm Finestra modale che contiene la form per il ban di un utente. Questa funzionalità è accessibile solo all'amministratore dell'applicazione.

TuTourSelf::Frontend::viewablePage::EventsDisplay Componente posizionata all'interno delle pagine di artista e locale, che contiene gli eventi correlati ad essi.

TuTourSelf::Frontend::viewablePage::LocalViewablePage Pagina di un locale, che può essere visualizzata dagli utenti attraverso i risultati della ricerca.

TuTourSelf::Frontend::viewablePage::MgrToArtistDialogProposal Finestra modale che contiene la form per l'invio di una proposta ad un artista da parte di un gestore di locali.

TuTourSelf::Frontend::viewablePage::ReprtUser Finestra modale per la segnalazione di un utente da parte di un altro utente.

TuTourSelf::Frontend::viewablePage::ViewablePage Contenitore delle pagine che possono essere visualizzate dagli utenti attraverso i risultati della ricerca.

8 TuTourSelf::Backend

Nel pacchetto backend sono presenti alcune classi e moduli che non si ritiene opportuno raggruppare in specifici pacchetti, per motivi di funzionalità.

TuTourSelf::Backend::Server contiene la configurazione e le istruzioni di avvio del server;

TuTourSelf::Backend::Routes contiene la dichiarazione e la configurazione delle routes accettate dal server;

TuTourSelf::Backend::chatHelper questo modulo si occupa della configurazione e l'inizializzazione della chat e dei socket di notifica.

TuTourSelf::Backend::userSocketArray questa classe copre una lacuna esposta da Passport-Socket.io, tale lacuna impedisce di collegare un certo utente alla lista di socket aperti. Questa classe risolve il problema offrendo funzionalità di inserimento, recupero ed eliminazione di socket, in maniera simile ad una multimappa;

TuTourSelf::Backend::notificationManager questa classe si occupa della gestione e dell'invio tramite socket.io di notifiche in tempo reale;

TuTourSelf::Backend::pendingNotifications questa classe rappresenta un'estensione del modello di mongoose dedicato alla memorizzazione delle notifiche in sospenso, offrendo funzionalità quali recupero delle notifiche in sospenso dato un utente.

8.1 TuTourSelf::Backend::models

Questo pacchetto contiene tutti i modelli mongoose (e le classi che li estendono), usati per la conservazione dei dati usati dall'applicazione.

TuTourSelf::Backend::bannedEmails questo modello contiene i dati riguardanti gli utenti esclusi dalla piattaforma;

TuTourSelf::Backend::models::emailConfirmation questo modello rappresenta le conferme di registrazione in attesa;

TuTourSelf::Backend::event questo modello contiene le informazioni riguardanti gli eventi;

TuTourSelf::Backend::feedback questo modello contiene i feedback lasciati dagli utenti;

TuTourSelf::Backend::models::local questo modello rappresenta un locale all'interno del sistema;

TuTourSelf::Backend::models::message rappresenta un messaggio di chat, memorizzato all'interno del database;

TuTourSelf::Backend::models::pendingNotifDB rappresenta una notifica destinata ad un utente al momento offline;

TuTourSelf::Backend::models::proposal rappresenta una proposta memorizzata in database;

TuTourSelf::Backend::reports questo modello contiene le segnalazioni degli utenti;

TuTourSelf::Backend::revokeAgreement questo modello contiene gli accordi in via di revoca che sono in attesa di conferma;

TuTourSelf::Backend::models::room rappresenta una stanza di chat, memorizzata nel database;

TuTourSelf::Backend::models::treatment rappresenta un'istanza di accordo, memorizzata nel database;

TuTourSelf::Backend::models::user rappresenta un utente registrato presso l'applicazione.

8.2 TuTourSelf::Backend::signup

Questo pacchetto contiene la business logic per la registrazione di nuovi utenti nel sistema.

TuTourSelf::Backend::signup::register questo modulo definisce la business logic e la configurazione di Passport per la registrazione di nuovi utenti tramite sistema interno, Facebook e Google SignIn.

TuTourSelf::Backend::signup::secondStepSignup definisce le business logic riguardante il completamento della registrazione, con scelta della categoria di utenza di cui entrare a far parte e l'inserimento dei dati personali.

8.3 TuTourSelf::Backend::local_management

TuTourSelf::Backend::signup::localCreation definisce la business logic riguardante la creazione di un nuovo locale;

TuTourSelf::Backend::signup::localRemove definisce la business logic riguardante la cancellazione di un locale;

TuTourSelf::Backend::signup::localUpdate definisce la business logic riguardante l'aggiornamento di un locale esistente.

8.4 TuTourSelf::Backend::login

Questo pacchetto contiene la business logic e le configurazioni necessarie per l'autenticazione di un utente al sistema.

TuTourSelf::Backend::signIn::loginMethods definisce la configurazione di Passport per la login tramite autenticazione interna, Facebook o Google SignIn.

8.5 TuTourSelf::Backend::chat

Questo pacchetto contiene la business logic della chat.

TuTourSelf::Backend::chat::chat_HistoryManager questa classe è utilizzata per la gestione dello storico di una stanza di chat, comprende funzionalità per la memorizzazione di messaggi e per il recupero dello storico;

TuTourSelf::Backend::chat::chat_MessageManager questa classe serve alla gestione dei messaggi in tempo reale, richiamando il gestore dello storico per la memorizzazione e facendo uso di Socket.io per la trasmissione in tempo reale e la notifica.

TuTourSelf::Backend::chat::chat_RoomManager questa classe è usata per la creazione, il recupero e la gestione delle stanze di chat;

8.6 TuTourSelf::Backend::search

Questo pacchetto contiene la business logic e le componenti atte al funzionamento del sistema di ricerca.

TuTourSelf::Backend::search::search_Keywords questo modulo si occupa della ricerca tramite keywords e restituzione dei risultati ottenuti dalla ricerca nel database;

TuTourSelf::Backend::search::search_event questo modulo si occupa della ricerca tramite keywords e restituzione degli eventi in programma;

TuTourSelf::Backend::search::search_managerid questo modulo si occupa della ricerca di locali, dato l'id del gestore.

8.7 TuTourSelf::Backend::routes

Questo pacchetto contiene le routes di Express.js e le annesse configurazioni.

TuTourSelf::Backend::routes::admin questo modulo contiene la configurazione delle route di amministrazione;

TuTourSelf::Backend::routes::agreement questo modulo contiene la configurazione delle route riguardanti l'accordo e le proposte;

TuTourSelf::Backend::routes::artist questo modulo contiene la configurazione delle route per la gestione dei dati specifici per artisti;

TuTourSelf::Backend::routes::auth questo modulo contiene la configurazione di tutte le routes che fanno capo al sistema di registrazione ed autenticazione, sia di utenti che di locali;

TuTourSelf::Backend::routes::bans questo modulo contiene la configurazione delle routes concernenti le esclusioni dalla piattaforma;

TuTourSelf::Backend::routes::event questo modulo contiene la configurazione delle routes riguardanti gli eventi;

TuTourSelf::Backend::routes::feedback questo modulo contiene la configurazione delle routes riguardanti l'inserimento e la richiesta di valori dei feedback;

TuTourSelf::Backend::routes::local questo modulo contiene la configurazione delle routes riguardanti i locali;

TuTourSelf::Backend::routes::map questo modulo contiene la configurazione delle routes per la raccolta dati per le mappe e la pianificazione dei tour tramite mappa;

TuTourSelf::Backend::routes::place questo modulo contiene la configurazione delle routes riguardanti i locali, soprattutto le immagini;

TuTourSelf::Backend::routes::profile questo modulo contiene la configurazione delle routes riguardanti i dati di profilo di un utente generico;

TuTourSelf::Backend::routes::report questo modulo contiene la configurazione delle routes riguardanti le segnalazioni degli utenti;

TuTourSelf::Backend::routes::search questo modulo contiene la configurazione delle routes di ricerca.

8.8 TuTourSelf::Backend::config

Questo pacchetto contiene la configurazione dei moduli di database, invio mail ed autenticazione.

TuTourSelf::Backend::config::auth questo modulo contiene la configurazione dei moduli di registrazione ed autenticazione dell'applicazione;

TuTourSelf::Backend::database questo modulo contiene la configurazione del database;

TuTourSelf::Backend::mailer questo modulo contiene la configurazione del sistema di invio di notifiche tramite email.

8.9 TuTourSelf::Backend::event_management

Questo pacchetto contiene i moduli per la gestione degli eventi derivanti da un accordo tra gestori ed artisti.

TuTourSelf::Backend::event_management::EventCreation questo modulo si occupa della creazione di nuovi eventi;

TuTourSelf::Backend::event_management::EventRemove questo modulo si occupa dell'eliminazione di eventi esistenti;

TuTourSelf::Backend::event_management::EventUpdate questo modulo si occupa dell'aggiornamento di eventi esistenti.

9 Interfacce esposte

Il backend, allo scopo di esporre le proprie funzionalità al frontend, fornisce un'interfaccia in stile REST, oltre che un sistema di notifica in tempo reale tramite websocket. In questa sezione si vedranno in dettaglio le routes ed le notifiche websocket usate.

9.1 Routes

9.1.1 Admin

POST /admin/remove-user/ Provvede alla rimozione forzata di un utente dal database, da parte di un amministratore. Richiede come corpo della richiesta il seguente JSON:

```
1 {  
2     user_id: id_utente_in_database  
3 }
```

POST /admin/remove-local/ Provvede alla rimozione forzata di un locale dal database, da parte di un gestore di locale. Richiede come corpo della richiesta il seguente JSON:

```
1 {  
2     local_id: id_locale_in_database  
3 }
```

POST /admin/remove-event/ Provvede alla rimozione forzata di un evento dal database, da parte di un gestore di locale. Richiede come corpo della richiesta il seguente JSON:

```
1 {  
2     event_id: id_evento_in_database  
3 }
```

POST /admin/ban-user/ Provvede all'esclusione di un dato indirizzo email dalla piattaforma. Richiede come corpo della richiesta il seguente JSON:

```
1 {  
2     email: email_da_escludere,  
3     reason: motivo_esclusione  
4 }
```

GET /admin/solve-report/:id Provvede a contrassegnare come "risolta" una data segnalazione, il parametro ":id" contiene l'id in database della segnalazione.

GET /admin/getMessages/:roomid Provvede a ritornare lo storico messaggi di una stanza di chat, il parametro ":roomid" contiene l'id in database della stanza di chat. Lo storico dei messaggi è tornato all'interno del corpo della risposta, all'interno della chiave "messages".

POST /admin/getRooms/ Provvede a ritornare le stanze di chat che contengono gli utenti passati nel corpo della richiesta. Il corpo della richiesta deve essere il seguente JSON:

```
1 {
2     users: [id_utente_1, id_utente_2]
3 }
```

Le stanze sono ritornate nel corpo della risposta, all'interno della chiave "rooms".

9.1.2 Agreement

GET /agreement/:id Provvede a ritornare il corpo di una proposta, dopo aver controllato i permessi necessari per potervi accedere. Il parametro ":id" è l'id in database della proposta, il corpo della proposta viene ritornato nella risposta, all'interno della chiave "proposal".

POST /agreement/proposal Provvede all'inserimento di una nuova proposta all'interno della trattativa. Richiede come corpo della richiesta il seguente JSON:

```
1 {
2     from: id_inviante_della_proposta,
3     to: id_ricevente_la_proposta,
4     place: id_locale_interessato_dalla_proposta,
5     treatment: id_accordo,
6     comment: commento_proposta,
7     duration: durata_evento_proposta,
8     amount: ammontare_compenso,
9     eventDate: data_evento_proposta
10 }
```

POST /agreement/treatment/users Provvede a ritornare tutte le trattative, in corso e chiuse, tra due utenti, riguardanti un certo locale. Richiede come corpo della richiesta il seguente JSON:

```
1 {
2     userArray: [id_utente_1, id_utente_2],
3     local: id_locale
4 }
```

Le trattative sono ritornate all'interno della risposta, nella chiave "treatments".

GET /agreement/treatment/:id Ritorna la trattativa e tutte le proposte collegate, dato l'id della trattativa cercata, inserito nell'URI al posto del parametro ":id". Il risultato è ritornato all'interno della chiave "proposals" della risposta.

POST /agreement/confirm Conferma una proposta, trasformando la trattativa in un accordo. Richiede il seguente JSON come corpo della richiesta:

```
1 {
2     proposal: id_proposta
3 }
```

POST /agreement/revoke Provvede alla revoca, o conferma di revoca, di una certa proposta o accordo. Richiede il seguente JSON come corpo della richiesta:

```
1 {  
2     proposal: id_proposta  
3 }
```

9.1.3 Artist

POST /artist/name Provvede alla ricerca e restituzione del nome d'arte di un artista, dato l'id dell'utente. Richiede il seguente JSON come corpo della richiesta:

```
1 {  
2     id: id_utente  
3 }
```

9.1.4 Auth

POST /auth/signup Provvede a processare la registrazione di un nuovo utente.

```
1 {  
2     email: email_da_registrare ,  
3     password: password_da_registrare  
4 }
```

GET /auth/confirm-email/:ID Provvede a confermare l'account collegato ad una conferma in attesa. Il parametro ":ID" deve essere l'id in database della conferma in attesa.

POST /auth/login/ Provvede ad effettuare la login di un dato utente. Richiede come corpo della richiesta il seguente JSON:

```
1 {  
2     email: email_da_registrare ,  
3     password: password_da_registrare  
4 }
```

POST /auth/whoami/ Provvede a restituire l'utente che fa la richiesta, così da mantenere l'utente loggato presso la piattaforma.

POST /auth/spectator3rd Provvede a compilare i dati in database di un utente che ha deciso di registrarsi come spettatore. Richiede un payload JSON in questo formato:

```
1 {  
2     firstName: nome_spettatore ,  
3     lastName: cognome_spettatore ,  
4     birthDate: data_nascita_spettatore ,  
5     address: indirizzo_spettatore ,  
6     city: citta'_spettatore ,  
7     state: nazione_spettatore ,  
8     zipCode: CAP_spettatore ,  
9     phone: numero_telefono_spettatore  
10 }
```


POST /auth/artist3rd Provvede a compilare i dati in database di un utente che ha deciso di registrarsi come artista. Richiede un payload JSON in questo formato:

```
1 {
2     artName: nome_arte_artista ,
3     firstName: nome_artista ,
4     lastName: cognome_artista ,
5     description: descrizione_artista ,
6     birthDate: data_nascita_artista ,
7     address: indirizzo_artista ,
8     city: citta'_artista ,
9     state: nazione_artista ,
10    zipCode: CAP_artista ,
11    phone: numero_telefono_artista ,
12    link: link_facebook_artista
13 }
```

POST /auth/manager3rd Provvede a compilare i dati in database di un utente che ha deciso di registrarsi come gestore di locali. Richiede un payload JSON in questo formato:

```
1 {
2     firstName: nome_gestore ,
3     lastName: cognome_gestore ,
4     birthDate: data_nascita_gestore ,
5     address: indirizzo_gestore ,
6     VATNumber: partita_IVA_gestore ,
7     businessName: ragione_sociale_gestore ,
8     phone: numero_telefono_gestore ,
9     city: citta'_gestore ,
10    state: nazione_gestore ,
11    zipCode: CAP_gestore
12 }
```

GET /auth/logout/ Provvede al logout dell'utente che fa la richiesta.

GET /auth/fork/ Provvede alla scelta della route più adatta a seconda dello stato di completamento della registrazione di un utente, usata per le registrazioni tramite google e facebook.

GET /auth/google/ Provvede all'avvio della registrazione tramite Google Signup, usando Passport.

GET /auth/google/callback Mette a disposizione una callback per Passport da usare quando Google ritorna i tokens di registrazione.

GET /auth/facebook/ Provvede all'avvio della registrazione tramite Facebook, usando Passport.

GET /auth/facebook/callback Mette a disposizione una callback per Passport da usare quando Facebook ritorna i tokens di registrazione.

9.1.5 Bans

GET /bans/getAll Ritorna un elenco completo degli indirizzi email escluse dalla piattaforma e delle motivazioni collegate a tali esclusioni. I dati richiesti sono inseriti nella risposta alla chiamata, nella chiave "bans".

GET /bans/lift/:id Permette di rimuovere un'esclusione dal database, dato l'id dell'esclusione stessa, all'interno del parametro ":id".

GET /bans/get/:email Permette di verificare se un dato indirizzo email è stato escluso dalla piattaforma. Richiede un parametro ":email" contenente l'indirizzo email da verificare. Il risultato viene ritornato all'interno della risposta, nella chiave "banned", che conterrà un boolean.

POST /bans/new/ Permette di inserire una nuova esclusione all'interno del database. Richiede un payload JSON nel seguente formato:

```
1 {  
2     email: indirizzo_email_da_escludere ,  
3     reason: motivazione_esclusione  
4 }
```

ATTENZIONE: Un utente bandito dalla piattaforma perderà il proprio profilo ed i propri locali.

9.1.6 Event

POST /event/create/ Permette la creazione e salvataggio di un nuovo evento in database. Il corpo della richiesta deve avere il seguente formato JSON:

```
1 {  
2     acceptedProposalId: id_proposta_accettata ,  
3     title: titolo_evento ,  
4     date: data_evento ,  
5     duration: durata_evento_in_minuti ,  
6     QR_url: url_del_qr_collegato ,  
7     comment: commento_evento ,  
8     artist_id: id_artista_partecipante ,  
9     local_id: id_locale_ospitante ,  
10    paybill_url: url_locandina  
11 }
```

POST /event/delete/ Consente la cancellazione di un evento dal database. Il corpo della richiesta deve avere il seguente formato JSON:

```
1 {  
2     event_id: id_evento  
3 }
```

POST /event/update/ Consente l'aggiornamento di un evento esistente. Il payload della richiesta deve essere un oggetto in JSON con il seguente formato:

```
1 {  
2     _id: id_evento ,  
3     title: titolo_evento ,  
4     imgUrl: url_immagine_evento ,  
5     dateTime: data_ora_evento ,  
6     duration: durata_evento ,  
7     QR_URL: URL_QR_collegato ,  
8     comment: commento_evento ,  
9     playbill_URL: URL_locandina  
10 }
```

GET /event/find/:id Consente di sapere se esiste un evento collegato ad una proposta data esiste, il parametro ":id" contiene l'identificativo della proposta richiesta. Il risultato è un boolean, restituito all'interno della risposta alla chiamata, nella chiave "result".

GET /event/get/:id Restituisce il corpo di un evento, dato il proprio identificativo in database. Il parametro ":id" contiene l'id dell'evento da cercare in database ed il risultato è restituito nella chiave "result" della risposta alla chiamata.

GET /event/search/:keywords Effettua una ricerca per parole chiave di un evento. Il parametro ":keywords" contiene le parole chiave con cui effettuare la ricerca. Il risultato è restituito all'interno della risposta alla chiamata, nella chiave "result".

GET /event/local/search/:id Effettua una ricerca degli eventi collegati ad un dato locale. Il parametro ":id" contiene l'identificativo in database del locale. Il risultato è restituito all'interno della risposta alla chiamata, nella chiave "result".

GET /event/artist/search/:id Effettua una ricerca degli eventi collegati ad un dato artista. Il parametro ":id" contiene l'identificativo in database del artista. Il risultato è restituito all'interno della risposta alla chiamata, nella chiave "result".

POST /event/upload/:id Provvede all'upload di un'immagine collegata ad un evento. Il parametro ":id" contiene l'identificativo dell'evento in database.

GET /event/picture/get Provvede alla restituzione in base64, dell'immagine collegata all'evento salvata. La route richiede un payload nel seguente formato JSON:

```
1 {
2     id: id_evento
3 }
```

Il base64 dell'immagine è restituito nella risposta alla chiamata, nella chiave "img".

GET /event/picture/remove/:id Provvede alla rimozione dell'immagine riguardante un evento. Il parametro ":id" contiene l'identificatore di tale evento nel database.

9.1.7 Feedback

POST /feedback/new Provvede all'inserimento di un nuovo feedback in database, richiede il seguente JSON come corpo della richiesta:

```
1 {
2     feedback: modello_feedback_da_inserire
3 }
```

GET /feedback/get/artist/:artistid Restituisce la media dei feedback memorizzati nel database per l'artista con identificativo inserito nel parametro ":artistid". Il risultato è ritornato nella risposta alla chiamata, all'interno della chiave "results".

GET /feedback/get/local/:localid Restituisce la media dei feedback memorizzati nel database per il locale con identificativo inserito nel parametro ":localid". Il risultato è ritornato nella risposta alla chiamata, all'interno della chiave "results".

POST /feedback/getAlreadySent/ Restituisce un boolean che dice se un dato utente ha già rilasciato un feedback ad un locale/artista per un dato evento. Il corpo della richiesta deve essere conforme al seguente JSON:

```
1 {
2     reviewer: id_utente_che_rilascia_il_feedback ,
3     eventid: id_evento_da_votare
4 }
```

9.1.8 Local

POST /local/create Permette la creazione di un nuovo locale, il corpo della richiesta deve essere conforme al seguente JSON:

```
1 {
2     name: nome_locale,
3     manager_id: id_manager_locale,
4     city: citta'_locale,
5     address: indirizzo_locale,
6     state: nazione_locale,
7     zip: CAP_locale,
8     phone: numero_telefono_locale,
9     facebook: URL_facebook_locale,
10    tripadvisor: URL_tripadvisor_locale,
11    description: descrizione_locale
12 }
```

POST /local/delete Permette l'eliminazione di un locale, dato l'identificativo del locale stesso in database. Richiede il seguente JSON come corpo della richiesta:

```
1 {
2     local_id: id_locale_da_rimuovere
3 }
```

POST /local/update Permette di aggiornare le informazioni di un locale. Richiede il seguente JSON come corpo della richiesta:

```
1 {
2     _id: id_locale,
3     name: nome_locale,
4     imgUrl: url_immagine_locale,
5     city: citta'_locale,
6     address: indirizzo_locale,
7     state: nazione_locale,
8     zip: CAP_locale,
9     phone: numero_telefono_locale,
10    facebook: URL_facebook_locale,
11    tripadvisor: URL_tripadvisor_locale,
12    description: descrizione_locale
13 }
```

9.1.9 Map

POST /map/get-locals/ Permette di ritornare i locali in un raggio di 3km da un percorso. Richiede in input gli steps che compongono il percorso, all'interno della seguente struttura JSON:

```
1 {
2     steps: [array_di_steps_del_percorso]
3 }
```

9.1.10 Place

POST /place/upload Permette di caricare un'immagine relativa ad un locale.

GET /place/picture/get Permette di recuperare un'immagine relativa ad un dato locale. Richiede come payload il seguente JSON:

```
1 {  
2     id: id_locale  
3 }
```

Il risultato è tornato in formato base64, all'interno della risposta alla chiamata, nella chiave "img".

GET /place/picture/get/normal Permette di recuperare un'immagine in formato nativo (.png o .jpg). Richiede come payload il seguente JSON:

```
1 {  
2     id: id_locale  
3 }
```

Il risultato è inviato come file nella risposta alla chiamata.

GET /place/picture/remove/:id Permette la rimozione di un'immagine di profilo di un dato locale. Il parametro ":id" contiene l'identificatore del locale in database.

9.2 Profile

POST /profile/upload/ Consente il caricamento di un'immagine relativa ad un profilo utente.

GET /profile/picture/get/ Consente il recupero di un'immagine relativa ad un profilo. Richiede come payload un JSON nel seguente formato:

```
1 {  
2     id: id_profilo  
3 }
```

Il risultato è restituito in formato base64 all'interno della risposta alla chiamata, nella chiave "img".

GET /profile/picture/remove Permette la rimozione di un'immagine di profilo, rimpiazzandola con l'immagine di default.

POST /profile/update/manager/ Permette di aggiornare le informazioni di un gestore di locale, richiede il seguente JSON come payload:

```
1 {  
2     "firstName": nome_gestore_locale ,  
3     "lastName": cognome_gestore_locale ,  
4     "birthDate": data_nascita_gestore ,  
5     "address" : indirizzo_gestore ,  
6     "phone": numero_telefono_gestore ,  
7     "state": nazione_gestore ,  
8     "city": citta'_gestore ,  
9     "zip": CAP_gestore ,  
10 }
```

POST /profile/update/spectator/ Permette di aggiornare le informazioni di uno spettatore, richiede il seguente JSON come payload:

```
1 {  
2     "firstName": nome_spettatore ,  
3     "lastName": cognome_spettatore ,  
4     "birthDate": data_nascita_spettatore ,
```

```
5      "address": indirizzo_spettatore ,
6      "city":  città'_spettatore ,
7      "state":  nazione_spettatore ,
8      "zipData":  CAP_spettatore ,
9      "phone":  numero_telefono_spettatore
10 }
```

POST /profile/update/artist/ Permette di aggiornare le informazioni di un' artista, richiede il seguente JSON come payload:

```
1 {
2     "artName"   : nome_arte_artista ,
3     "artScope"  : ambito_artistico ,
4     "description" : descrizione_artista ,
5     "firstName": nome_artista ,
6     "lastName":  cognome_artista ,
7     "birthDate": data_nascita_artista ,
8     "city":     città'_artista ,
9     "address":  indirizzo_artista ,
10    "state":    nazione_artista ,
11    "zipCode":  CAP_artista ,
12    "phone":    numero_telefono_artista ,
13    "link" : {
14        facebook: URL_facebook_artista
15    }
16 }
```

POST /profile/delete/ Consente l'eliminazione di un profilo, richiede il seguente JSON come corpo della richiesta:

```
1 {
2     user_id: id_utente_da_eliminare
3 }
```

9.2.1 Reports

GET /reports/get/:id Permette il recupero di una segnalazione, dato l'identificatore univoco, contenuto nel parametro ":id". Il risultato è restituito nella risposta alla chiamata, all'interno della chiave "report".

GET /reports/getAll Permette il recupero di tutte le segnalazioni aperte, che sono ritornate nella risposta alla chiamata, all'interno della chiave "reports".

POST /reports/new/ Consente la creazione di una nuova segnalazione, richiede il seguente JSON come payload:

```
1 {
2     reported: id_utente_segnalato ,
3     reason:  motivazione_segnalazione ,
4     comment: commento_segnalazione ,
5     status:  stato_segnalazione
6 }
```

GET /reports/remove/:id Permette la rimozione di una segnalazione, dato l'identificatore univoco, inserito nel parametro ":id".

GET /reports/solve/:id Permette di contrassegnare come risolta una data segnalazione, dato l'identificatore univoco all'interno del parametro ":id".

9.2.2 Search

GET /search/:keywords Permette di cercare artisti o locali tramite parole chiave inserite nel parametro ":keywords". I risultati sono tornati nella risposta alla chiamata, nelle chiavi "artists" e "locals".

GET /search/locals/:manager_id Permette di trovare i locali gestiti da un gestore, dato l'identificatore univoco del gestore stesso, all'interno del parametro ":manager_id". Il risultato è tornato all'interno della risposta alla chiamata, nella chiave "locals".

9.3 Segnali WebSocket

9.3.1 Backend \Rightarrow Frontend

9.3.1.1 history Restituisce al frontend lo storico completo dei messaggi di una stanza.

9.3.1.2 backToLogin Segnale per il frontend che dovrebbe ridirezionare incondizionatamente l'utente alla login (solitamente a causa di un riavvio del server).

9.3.1.3 notification Rappresenta una notifica a schermo che il frontend deve mostrare.

9.3.1.4 youdead Segnale da parte del backend di esclusione di un utente dalla piattaforma, la ricezione di questo segnale da parte di un'istanza del frontend comporta l'immediata disconnessione da parte dell'utente collegato.

9.3.1.5 roomList Restituisce al frontend l'elenco delle stanze di chat aperte.

9.3.1.6 newProposal Notifica il frontend dell'arrivo di una nuova proposta, per l'aggiornamento dell'elenco proposte nella chat.

9.3.1.7 proposalRevoked Notifica il frontend della revoca di una proposta, per l'aggiornamento dell'elenco proposte nella chat.

9.3.1.8 proposalConfirmed Notifica il frontend dell'avvenuta conferma di una proposta, per l'aggiornamento dell'elenco proposte nella chat.

9.3.1.9 agreementRevokeConfirmation Notifica il frontend dell'apertura di una richiesta di revoca di un accordo, per l'aggiornamento dello stato dell'elenco proposte nella chat.

9.3.1.10 message Restituisce al frontend un messaggio di chat appena inviato.

9.3.2 Frontend \Rightarrow Backend

9.3.2.1 roomMessage Usato per inviare un nuovo messaggio di chat al backend.

9.3.2.2 roomConnect Usato per richiedere la connessione ad una data stanza.

9.3.2.3 messageAck Usato per inviare la conferma di visualizzazione di un messaggio.

9.3.2.4 getRooms Richiesta al backend di invio delle stanze di chat aperte.

9.3.2.5 leave Richiesta di uscita da una determinata stanza di chat.

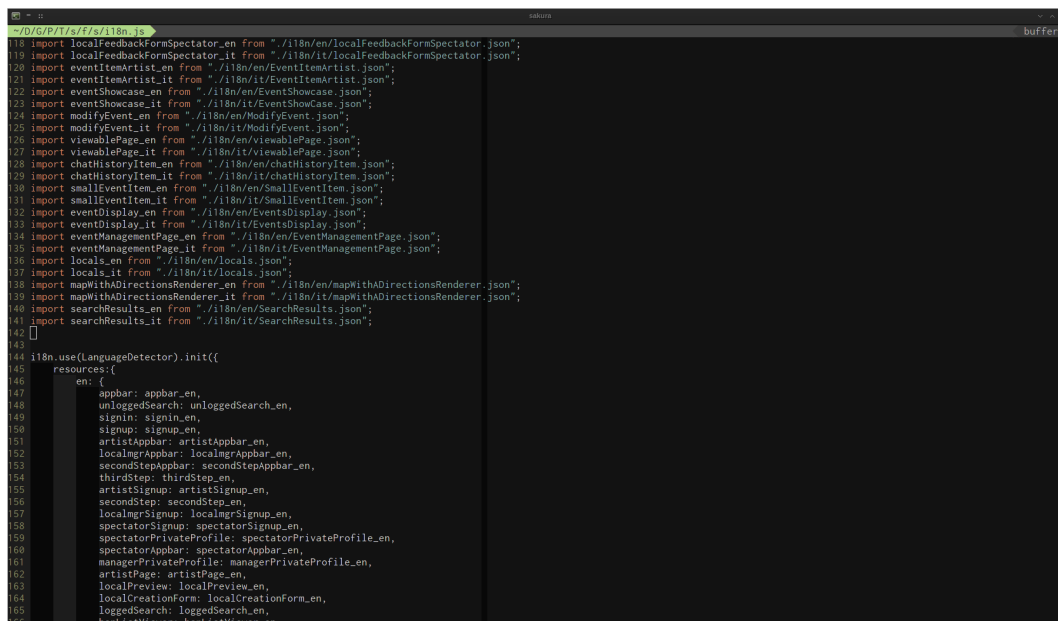
10 Descrizione dettagliata delle classi

Per una descrizione dettagliata delle singole classi, callback e metodi, si prega di far riferimento alla Guida Dettagliata alle Classi, disponibile in forma ipertestuale all'interno della cartella `/docs/`.

11 Punti di estensione

11.1 Internazionalizzazione

È possibile aggiungere nuove lingue all'applicazione, grazie all'uso di `i18next` e `React-i18next`. Il fulcro di questo punto di estensione è il file `/src/frontend/src/i18n.js`, il quale contiene la mappa delle lingue attualmente disponibili:



```

18 import localFeedbackFormSpectator_en from "../i18n/en/localFeedbackFormSpectator.json";
19 import localFeedbackFormSpectator_it from "../i18n/it/localFeedbackFormSpectator.json";
20 import eventItemArtist_en from "../i18n/en/EventItemArtist.json";
21 import eventItemArtist_it from "../i18n/it/EventItemArtist.json";
22 import eventShowcase_en from "../i18n/en/EventShowcase.json";
23 import eventShowcase_it from "../i18n/it/EventShowcase.json";
24 import modifyEvent_en from "../i18n/en/ModifyEvent.json";
25 import modifyEvent_it from "../i18n/it/ModifyEvent.json";
26 import viewablePage_en from "../i18n/en/viewablePage.json";
27 import viewablePage_it from "../i18n/it/viewablePage.json";
28 import chatHistoryItem_en from "../i18n/en/chatHistoryItem.json";
29 import chatHistoryItem_it from "../i18n/it/chatHistoryItem.json";
30 import smallEventItem_en from "../i18n/en/SmallEventItem.json";
31 import smallEventItem_it from "../i18n/it/SmallEventItem.json";
32 import eventDisplay_en from "../i18n/en/EventsDisplay.json";
33 import eventDisplay_it from "../i18n/it/EventsDisplay.json";
34 import eventManagementPage_en from "../i18n/en/EventManagementPage.json";
35 import eventManagementPage_it from "../i18n/it/EventManagementPage.json";
36 import locals_en from "../i18n/en/locals.json";
37 import locals_it from "../i18n/it/locals.json";
38 import mapWithADirectionsRenderer_en from "../i18n/en/mapWithADirectionsRenderer.json";
39 import mapWithADirectionsRenderer_it from "../i18n/it/mapWithADirectionsRenderer.json";
40 import searchResults_en from "../i18n/en/SearchResults.json";
41 import searchResults_it from "../i18n/it/SearchResults.json";
42
43
44 i18n.use(LanguageDetector).init({
45   resources: {
46     en: {
47       appBar: appBar_en,
48       unloggedSearch: unloggedSearch_en,
49       signin: signin_en,
50       signup: signup_en,
51       artistAppBar: artistAppBar_en,
52       localmgrAppBar: localmgrAppBar_en,
53       secondStepAppBar: secondStepAppBar_en,
54       thirdStep: thirdStep_en,
55       artistSignup: artistSignup_en,
56       secondStep: secondStep_en,
57       localmgrSignup: localmgrSignup_en,
58       spectatorSignup: spectatorSignup_en,
59       spectatorPrivateProfile: spectatorPrivateProfile_en,
60       spectatorAppBar: spectatorAppBar_en,
61       managerPrivateProfile: managerPrivateProfile_en,
62       artistPage: artistPage_en,
63       localPreview: localPreview_en,
64       localCreationForm: localCreationForm_en,
65       loggedSearch: loggedSearch_en,
66       banListViewer: banListViewer_en,

```

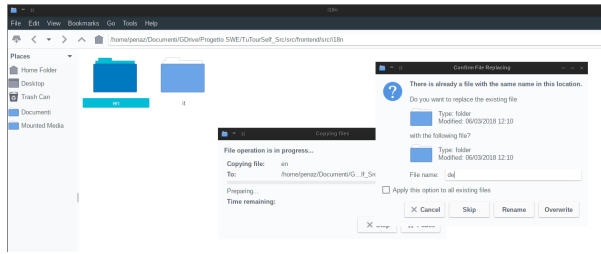
Immagine 3: Un pezzo del file `i18n.js`

L'altro punto focale è il contenuto della cartella `/src/frontend/src/i18n/`, che contiene tutti i file JSON contenenti i termini e le rispettive versioni tradotte.

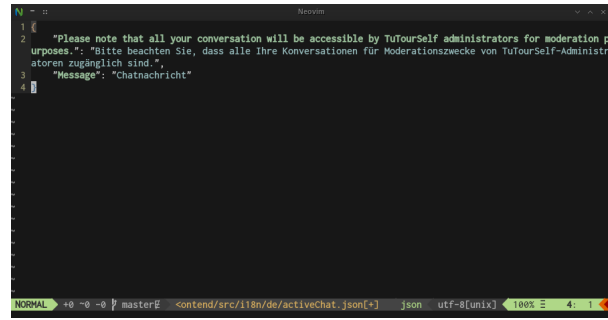
L'applicazione si occupa automaticamente del rilevamento della lingua dalle impostazioni del browser, offrendo automaticamente l'interfaccia nella lingua di sistema o, se questa non è disponibile, in inglese.

11.1.1 Inserimento di una nuova lingua

Per inserire una nuova lingua è sufficiente creare una nuova copia della cartella `"en"`, utilizzando il codice lingua prescelto (in questo caso `"de"` per Tedesco) e compilare le traduzioni:



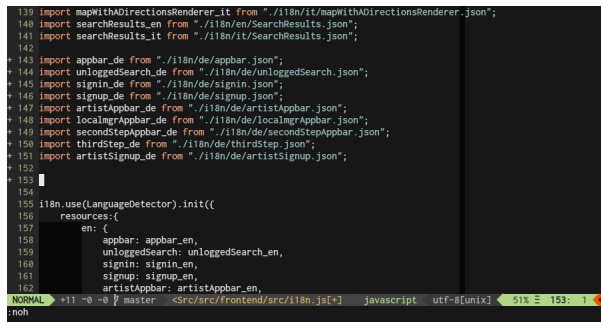
(a) Passo 1: Copiare la cartella "en"



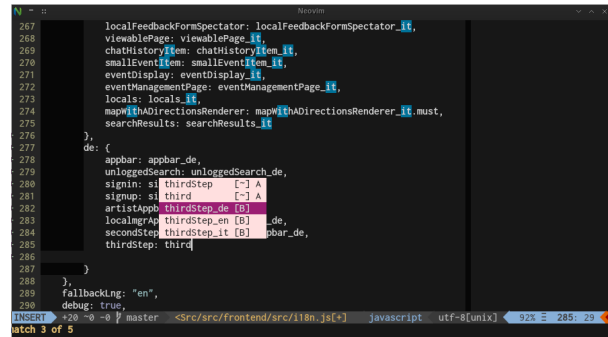
(b) Passo 2: Compilare le traduzioni

Immagine 4: Inserimento di una nuova lingua - Parte 1

Successivamente, per rendere nota la presenza della nuova lingua, è necessario modificare il file `/src/frontend/src/i18n.js`, importando i nuovi file JSON creati ed aggiungendo la sezione con il codice lingua adatto (in questo caso "de") all'interno delle resources:



(a) Passo 3: Importazione dei nuovi JSON



(b) Passo 4: Creazione delle nuove resources

Immagine 5: Inserimento di una nuova lingua - Parte 2

Dopo aver ricompilato il frontend tramite il comando `npm run build` eseguito sulla root del progetto, la nuova lingua sarà automaticamente selezionata appena un utente con la configurazione adatta si collegherà all'applicazione.

11.2 Nuovi modelli nel backend

È possibile estendere il numero di modelli gestiti dal backend aggiungendo nuovi files all'interno della cartella `/src/models/`, questo permetterà a nuovi moduli di poter utilizzare i nuovi modelli e di mantenere ordine all'interno del repository di codice.

Ecco un template di un nuovo modello mongoose:

```
1 /*
2  * FileName: newModel.js
3  * Date: 2018-06-05
4  * Authors:
5  * - Nome Cognome
6  * E-mail: commandlineteam@gmail.com
7  *
8  * License: Copyright 2018
9  *
10 * Description:
11 * This is a new model to integrate into the TuTourSelf Application
12 *
13 * Changelog:
```

```

14  * Date          | Author          | Description
15  * -----|-----|-----
16  * 2018-06-05 | Nome Cognome   | First version
17  */
18
19  var mongoose = require("mongoose"),
20      schema = mongoose.Schema;
21
22  const schemaTypes = mongoose.Schema.Types;
23
24  const newSchema = new schema({
25      //Schema data here
26  })
27
28  class newModel extends mongoose.Model{
29      //Model methods here
30  }
31
32  module.exports = mongoose.model(newModel, newSchema, 'newModelName');

```

Successivamente il nuovo modello potrà essere usato tramite `require("newModel.js")` oppure tramite lo statement `import` di ES6.

11.3 Nuove routes nel backend

Per esporre nuove funzionalità, è necessario mettere a disposizione delle nuove routes a cui il frontend possa collegarsi. Prima di tutto è necessario creare un nuovo file all'interno della cartella `/src/routes/`, seguendo questo template:

```

1  /*
2  * FileName: newRoute.js
3  * Date: 2018-06-05
4  * Authors:
5  * - Nome Cognome
6  * E-mail: commandlineteam@gmail.com
7  *
8  * License: Copyright 2018
9  *
10 * Description:
11 * This is a new route to expose new functionality to the TuTourSelf app
12 *
13 * Changelog:
14 * Date          | Author          | Description
15 * -----|-----|-----
16 * 2018-06-05 | Nome Cognome   | First version
17 */
18
19 var express = require("express"),
20     app = express.Router();
21
22
23 app.post("/create",function(req,res) {
24     //A new "post" route
25 });
26
27 app.get("/getInfo",function(req,res) {
28     //A new "get" route
29 });

```

```
30  
31 module.exports = app;
```

Ora è possibile esporre la nuova route ad express, modificando il file `/src/routes.js` ed aggiungendo un nuovo statement di importazione:

```
var newRoute = require("./routes/newRoute.js");
```

e dichiarando su quale path si vogliano esporre le nuove routes, tramite:

```
app.use("/newRouteRoot", newRoute);
```

Ecco uno screenshot del risultato:



```
27 var authRoute = require("./routes/auth.js"),  
28     searchRoute = require("./routes/search.js"),  
29     profileRoute = require("./routes/profile.js"),  
30     agreementRoute = require("./routes/agreement.js"),  
31     eventRoute = require("./routes/event.js"),  
32     localRoute = require("./routes/local.js"),  
33     bansRoute = require("./routes/bans.js"),  
34     feedbackRoute = require("./routes/feedback.js"),  
35     reportsRoute = require("./routes/report.js"),  
36     adminRoute = require("./routes/admin"),  
37     placeRoute = require("./routes/place.js"),  
38     artistRoute = require("./routes/artist.js"),  
39     mapRoute = require("./routes/map.js"),  
+ 40     newRoute = require("./routes/newRoute.js");  
41  
42 module.exports = function(app){  
43     app.use("/auth", authRoute);  
44     app.use("/search", searchRoute);  
45     app.use("/profile", profileRoute);  
46     app.use("/agreement", agreementRoute);  
47     app.use("/event", eventRoute);  
48     app.use("/local", localRoute);  
49     app.use("/feedback", feedbackRoute);  
50     app.use("/bans", bansRoute);  
51     app.use("/reports", reportsRoute);  
52     app.use("/admin", adminRoute);  
53     app.use("/place", placeRoute);  
54     app.use("/artist", artistRoute);  
55     app.use("/artist", artistRoute);  
56     app.use("/map", mapRoute);  
+ 57     app.use("/newRouteRoot", newRoute);  
58 };
```

Immagine 6: Risultato della modifica delle route

11.4 Nuove pagine di frontend

E' possibile estendere il frontend con nuove pagine, innanzitutto creando una nuova componente React , come nel seguente esempio. Questa nuova componente avrà una struttura base come da seguente screenshot, e dovrà inoltre includere tutte le componenti grafiche ed import + riferimento ad eventuali componenti figli.

```
import React, { Component } from 'react';

class NomeComponente extends Component {
  render() {
    return (
      <div>
        <p>Qui puoi strutturare il tuo componente</p>
      </div>
    );
  }
}

export default NomeComponente;
```

Immagine 7: Esempio di nuova componente React

Il passaggio seguente da fare sarà creare una nuova route all'interno della componente più ad alto livello che instrada le route per la parte di sito desiderata (nello specifico *ArtistInterface*, *SpectatorInterface*, *LocalManagerInterface* per aggiungere la nuova pagine all'interfaccia specifica di uno dei tre tipi di utenti, e *SigninSignupSearch* per le pagine relative all'utente non autenticato), da cui poter visualizzare la nuova pagina all'interno della single-page-application_G. Un esempio di creazione di una route si può vedere nel seguente screenshot.

```
<div>
  <Route path="/user/nuovopath" render={
    (routeProps) => <NuovoComponente {...routeProps}/>}/>
</div>
```

Immagine 8: Esempio di creazione di una nuova route e di collegamento della nuova componente a questa route

Dopodiché, per poter accedere alla nuova route, questa deve essere linkata nell'appbar, aggiungendo un nuovo link cliccabile, che porterà alla nuova pagine, come da screenshot seguenti:

```
<Link to="/user/nuovopath"
  onClick={() => this.handleClick()}
  style={{color:this.state.links.linkNewComp}}
  className="nav-link" >
  Link al nuovo componente
</Link>
```

Immagine 9: Esempio di creazione di un nuovo link all'interno dell'appbar

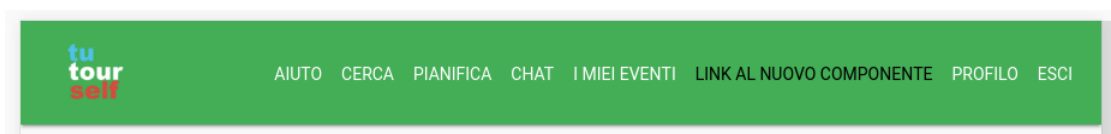


Immagine 10: Esempio di come viene renderizzato il nuovo link all'interno dell'appbar

A Glossario

A

Accordo: intesa di due o più parti dopo che è avvenuta una contrattazione attraverso proposte e controproposte di esibizione.

Apple Safari: *Browser_G* creato dalla compagnia informatica Apple Corp.

B

Browser: Applicativo usato per la navigazione sul web. **AJAX:** Acronimo di Asynchronous JavaScript and XML, è una tecnica di sviluppo software per la realizzazione di applicazioni web interattive

C

Cloud Computing: Paradigma di erogazione di risorse informatiche caratterizzato dalla disponibilità su richiesta di tali risorse tramite internet.

Closed-Source: *Vedi Software Proprietario.*

Classi: Un costrutto di un linguaggio di programmazione usato come modello per creare oggetti.

E

ECMAScript5: Linguaggio di programmazione standardizzato e mantenuto da Ecma International. Aggiunge lo "strict mode" inteso a provvedere un più completo controllo di errori ed a scoraggiare gli sviluppatori ad incappare in errori.

ECMAScript6: Apporta modifiche sintattiche significative che aprono la porta ad applicazioni più complesse, includendo classi e moduli definendoli semanticamente come lo "strict mode" in ECMAScript 5.

Edge: *Vedi Microsoft Edge.*

F

Framework: Insieme integrato di componenti software prefabbricate, usato in congiunzione a nuovo codice scritto per soddisfare dei requisiti specifici.

Firefox: *Vedi Mozilla Firefox.*

G

Google Chrome: Browser sviluppato da un team di Google LLC.

I

Internet Explorer: Browser sviluppato da Microsoft Corp., l'ultima versione rilasciata è stata Internet Explorer 11. Ora è stato sostituito dal più moderno *Microsoft Edge_G*.

J

JavaScript: Linguaggio di programmazione interpretato, solitamente usato per lo sviluppo frontend ma di recente introdotto anche per lo sviluppo e la programmazione di Backend.

JSON: JavaScript Object Notation, è una notazione testuale che permette di descrivere degli oggetti in JavaScript.

M

Malware: Contrazione di "Malicious Software" (letteralmente "Software Malevolo"), fa riferimento a tutti quei software che hanno come obiettivo (primario o collaterale), il danneggiamento di un sistema informatico, il furto di dati o qualunque attività atta a danneggiare un'eventuale vittima.

Microsoft Edge: Browser sviluppato da Microsoft Corp. ed attualmente supportato.

MapReduce: Funzione in due passaggi, composta da:

- Map: Trasformazione di un input in un certo output;
- Reduce: Combinazione di una tupla in input in una certa tupla di output di cardinalità inferiore.

Mozilla Firefox: Browser web sviluppato da Mozilla.

N

Node.js: Ambiente di esecuzione JavaScript *server-side* basato sul motore V8 di Google Chrome.

NoSQL: è un movimento che promuove sistemi software dove la persistenza dei dati è caratterizzata dal fatto di non utilizzare il modello relazionale, di solito usato dai database tradizionali.

NPM: Acronimo di Node Package Manager, è il gestore pacchetti predefinito di Node.js.

O

Opportunistic TLS: Estensione di protocolli per la comunicazione testuale, Opportunistic TLS offre la possibilità di passare da una connessione in chiaro ad una criptata usando una porta separata per la comunicazione crittografata. Per far partire tale connessione criptata si fa solitamente uso del comando STARTTLS.

P

Package: è una collezione di classi e interfacce correlate.

R

REST: Abbreviativo di REpresentational State Transfer, è un'architettura per sistemi software distribuiti basata sull'uso di una struttura di URL ben precisa e l'uso di verbi HTTP specifici (GET, POST, DELETE, ...).

Ransomware: Contrazione di Ransom Software (letteralmente "Software che chiede il riscatto"), un particolare tipo di Malware il cui comportamento più diffuso è la crittografia di dati sensibili in modo che il proprietario non possa più accedervi. A crittografia completata, il software chiede un riscatto affinché il proprietario possa accedere nuovamente ai propri dati.

S

Safari: Vedi *Apple Safari*.

Scalabilità: una caratteristica del software facilmente modificabile nel caso di variazioni notevoli nella mole o della tipologia di dati utilizzati.

Server-side: Lato Server, solitamente inteso come "esecuzione di codice da parte del server".

Sharding: Sistema di partizione orizzontale dei dati in un database, in cui ciascuna "shard" (pezzo di database) è contenuto all'interno di una diversa macchina, così da alleggerire il carico sulle singole macchine.

Single-page-application: intende un'applicazione web o un sito web che può essere usato o consultato su una singola pagina web con l'obiettivo di fornire una esperienza utente più fluida e simile alle applicazioni desktop dei sistemi operativi.

Software proprietario: Il software proprietario, chiamato anche closed source, è un software la cui licenza consente al beneficiario il suo utilizzo sotto particolari condizioni ed impedendone altre come la modifica, la condivisione, lo studio, la ridistribuzione o l'ingegneria inversa.

STARTTLS: Vedi *Opportunistic TLS*.

T

Threads: Sottoprocessi, adatti all'esecuzione concorrente su sistemi di elaborazione multiprocessore o multicore.

TLS: Acronimo di Transport Layer Security, TLS è una famiglia di protocolli crittografici di presentazione.

Throughput: capacità di trasmissione "effettivamente utilizzata" di un canale di comunicazione.

U

URL: Acronimo di Uniform Resource Locator, un URL è una stringa che identifica univocamente l'indirizzo di una risorsa internet.

V

Variabile d'ambiente: Variabile presente nell'ambiente globale di un interprete dei comandi, accessibile dai processi tramite meccanismi indipendenti dal sistema operativo in uso.