Relazione progetto taw 2020

Alessio Narder 868071

1. Architettura del sistema

L'applicazione si compone di due elementi fondamentali, il backend (parte dell'applicazione lato server) e il frontend (parte dell'applicazione lato client). Il backend dispone di un'interfaccia per poter interagire con client costituita due canali di comunicazione, uno full-duplex realtime (implementato con Socket.io) e uno stateless half-duplex (implementato con express). Entrambi i canali permettono la trasmissione dei dati utilizzando il protocollo http, ma differiscono per la persistenza della connessione. Poichè la scalabilità di un socket è verticale, esso viene utilizzato per l'invio frequente di pacchetti con poche informazioni. Al contrario, le chiamate http (che scalano meglio orizzontalmente) vengono utilizzate per le operazioni CRUD più complesse. Per questi motivi, l'aggiornamento delle chat e delle offerte su ogni inserzione, vengono svolte utilizzando Socket.io. Le restanti operazioni su utenti, inserzioni e autenticazione vengono svolte con chiamate http che rispettano l'interfaccia REST offerta dal backend. Gli utenti sono notificati degli eventi principali (asta invenduta, asta conclusa con successo, nuovo invito di registrazione) attraverso email, che vengono inviate dal backend sfuttando l'API messa a disposizione dalla libreria nodemailer. L'autenticazione degli utenti con email e password permette di ottenere il token JWT necessario per poter svolgere le operazioni dell'area riservata o che richiedono un'identificazione. Il flusso di autenticazione viene gestito dal backend utilizzando le api fornite dalla liberia passport.js. Lo stato dell'applicazione visibile all'utente viene esposto per mezzo di componenti Angular.

Collezione e struttura dei documenti

All'interno del database sussiste un'unica collection, Users, all'interno del quale sono contenuti i documenti relativi agli utenti. Le inserzioni associate ad un'utente sono salvate in prejoin sullo stesso documento e le chat associate ad un'inserzione sono anch'esse in prejoin. Il server successivamente determina quali query eseguire a seconda della richiesta.

Gli schemi dei documenti (anche in prejoin) sono di seguito riportati:

Inserzione

```
const auctionSchema = new mongoose.Schema({
  expires: {
    type: Number,
    required: true
  },
  threshold: {
    type: Number,
    required: false,
    default: 0
  },
  currentPrice: {
    type: Number,
    required: false,
    default: 0
  },
```

```
  isActive: {
    type: Boolean,
    required: true
  },
  book: [bookSchema],
  offers: [OfferSchema],
  winner: {
    type: mongoose.Types.ObjectId,
    required: false,
    default: null,
  },
  chats: {
    required: true,
    type: [ChatSchema]
  }
})
```

Libro di un'inserzione

```
const bookSchema = new mongoose.Schema({
  title: {
    type: String,
    required: true
  },
  author: {
    type: String,
    required: true
  },
  course: {
    type: String,
    required: true
  },
  university: {
    type: String,
    required: true
  }
})
```

Chat

```
const ChatSchema = new mongoose.Schema({
  scope: {
    required: true,
    type: String
  },
  partnerId: {
    required: false,
    type: mongoose.Types.ObjectId
  },
  partnerUs: {
    required: false,
    type: String
  },
  messages: {
    type: [MessageSchema],
    required: false
```

```
  }
})
```

Locazione di uno studente

```
const locationSchema = new mongoose.Schema({
  Comune: {
    type: String,
    required: true
  },
  Provincia: {
    type: String,
    required: true
  },
  Regione: {
    type: String,
    required: true
  },
  Indirizzo: {
    type: String,
    required: true
  },
  CAP: {
    type: String,
    required: true
  }
})
```

Messaggio di una chat

```
const MessageSchema = new mongoose.Schema({
  senderId: {
    type: mongoose.Types.ObjectId,
    required: true
  },
  senderUs: {
    type: String,
    required: true
  },
  message: {
    type: String,
    requie: true
  },
  timestamp: {
    type: Number,
    required: true
  }
})
```

Offerta di un'inserzione

```
const OfferSchema = new mongoose.Schema({
  user: {
    type: mongoose.Types.ObjectId,
    required: true
  },
```

```
  username: {
    type: String,
    required: true
  },
  amount: {
    type: Number,
    required: true
  },
  delta: {
    type: Number,
    required: true
  },
  timestamp: {
    type: Date,
    required: true
  }
})
```

Utente

```
const userSchema = new mongoose.Schema({

  firstname: {
    type: String,
    required: true
  },

  lastname: {
    type: String,
    required: true
  },

  username: {
    type: String,
    required: true
  },

  password: {
    type: String,
    required: true,
    minlength: 8
  },
  email: {
    type: String,
    required: true
  },
  moderator: {
    type: Boolean,
    required: true
  },

  confirmed: {
    type: Boolean,
    required: false,
    default: false
  },
```

```
  location: [locationSchema],
  auctions: [auctionSchema]
})
```

3. Descrizione dell'API REST

L'API REST fornita dal backend è documentata secondo lo standard openapi al seguente link

4. Workflow di autenticazione

L'autenticazione e la persistenza della sessione vengono gestite in 5 casi:

1. L'utente apre una nuova sessione
All'apertura di una nuova sessione, il client controlla la presenza o meno di una chiave `tkn` in `localStorage`. Se essa esiste, ne decripta il contenuto e
verifica che il valore della chiave `exp` sia inferiore al timestamp della data corrente. Se lo è, la navigazione prosegue normalmente, se non lo è vengono eseguiti logout e redirect in home.

2. L'utente è in sessione
Periodicamente, il client verifica la validità del token JWT associato all'utente. Se questo è scaduto, viene eseguito un logout e l'utente viene reindirizzato in home.

2. L'utente richiede il login
L'utente inserisce nome utente e password all'interno di un form. I dati vengono successivamente inoltrati al backend, il quale verifica l'esistenza e lo stato dell'utente (confermato o non confermato). Se l'utente esiste ed è confermato, il server ritorna 200 e un payload con i dati utente e il token JWT. Il server ritorna 400 ed un messaggio d'errore altrimenti.

3. L'utente si vuole registrare
L'utente compila un form di registrazione e invia al backend i dati richiesti. Se l'utente risulta inesistente, egli riceverà una mail con link temporaneo, che gli permetterà di confermare la sua registrazione e di eseguire successivamente il login.

4. L'utente viene invitato come moderatore
Se l'utente risulta inesistente, egli riceverà una mail con un link temporaneo che gli permetterà di registrarsi compilando il form di registrazione. Una volta salvati i dati, il nuovo moderatore potrà autenticarsi o proseguire con la navigazione.

5. L'utente non ricorda la password
L'utente compila un form inserendo il suo indirizzo email. Se l'indirizzo email esiste, il backend provvederà ad inviare una email con link temporaneo che gli permetterà di resettare la password, compilando un apposito form.

5. Descrizione del client web

La descrizione della struttura e dei componenti dell'applicazione Angular è disponibile al seguente link

6. Screenshot

All'interno della cartella relazione sono presenti alcuni video rappresentativi del workflow dell'applicazione