UniLibrary

Applicazioni e Servizi Web

Ilaria Ciavatti - 0000714136 {ilaria.ciavatti@studio.unibo.it}

Marzo 2021

0.1 Introduzione

Il nome dell'applicazione creata è UniLibrary e si tratta di un'applicazione pensata per una biblioteca universitaria. Gli obiettivi del progetto sono di soddisfare i requisiti presenti nella sezione requisiti. Inoltre l'autrice si è concentrata sul parametro di usabilità dell'applicativo cercando di creare un'interfaccia il più possibile semplice, ed intuitiva. In questo report sono presenti le parti di codice più rilevanti del progetto, in particolare è presente l'intero codice relativo alla **chat** ed ai meccanismi di propagazione dei messaggi fra gli utenti ed i bibliotecari.

0.2 Requisiti

L'applicazione prevede la possibilità di cercare un libro verificandone la disponibilità; inserire una prenotazione nel database qualora il titolo non fosse disponibile; effettuare il login come studente; effettuare il login come personale bibliotecario; poter effettuare operazioni CRUD sui dati relativi ai libri della biblioteca pervio accesso come personale bibliotecario; poter scrivere direttamente al personale tramite una chat, senza bisogno di effettuare il login; possibilità per il personale di chattare con più utenti e fornire informazioni utili relative al servizio.

0.3 Design

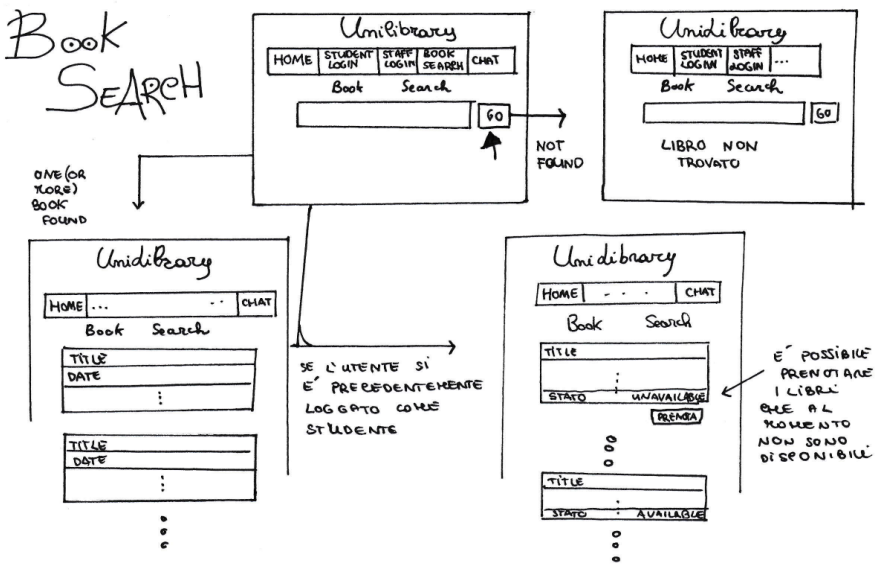
In questa sezione sono presenti brevi descrizioni delle varie componenti strutturali del software e della sua architettura. Il pattern design di questa applicazione è il *model-view-view model*(MVVM). Il linguaggio di frontend e la sua capacità di *binder* hanno determinato tale design pattern. Nel modello MVVM sono disponibili tre componenti principali: il modello, la vista e il modello della vista. La vista è responsabile della definizione della struttura, del layout e dell'aspetto degli elementi visualizzati dall'utente sullo schermo. Il modello della vista implementa le proprietà e i comandi a cui è possibile associare i dati della visualizzazione e notifica la visualizzazione di eventuali modifiche di stato tramite gli eventi di notifica delle modifiche. Il modello della vista è inoltre responsabile del coordinamento delle interazioni della vista con le classi del modello richieste. Ogni modello di visualizzazione fornisce i dati di un modello in un modulo che può essere utilizzato facilmente dalla visualizzazione. A tale scopo, il modello di visualizzazione a volte esegue la conversione dei dati. L'inserimento di questa conversione dei dati nel modello di visualizzazione è una scelta consigliata, perché fornisce proprietà a cui è possibile associare la visualizzazione. Ad esempio, il modello di visualizzazione potrebbe combinare i valori di due proprietà per renderlo più semplice da visualizzare nella visualizzazione. Le classi modello sono classi non visive che incapsulano i dati dell'app. Pertanto, il modello può essere considerato come la rappresentazione del modello

di dominio dell'applicazione, che in genere include un modello di dati insieme alla logica di business e di convalida.

0.3.1 Interfaccia utente

L'interfaccia utente è stata creata cercando di attenersi ad un preciso principio: **less is more**. L'applicativo ha quindi come obiettivo quello di adottare un design minimalista con lo scopo di aumentare l'usabilità del software. Sono stati sviluppati più percorsi, che elenchiamo brevemente. Tramite l'applicativo uno studente, o una studentessa, può effettuare il login. L'interfaccia è molto semplice ed è la stessa del login dei membri dello staff, che possiamo visionare in figura 2. Due spazi per inserire le credenziali di input ed il bottone per procedere con il login. Viene mantenuta fissa la scritta in cima, *"UniLibrary"*, e la barra di navigazione. Vi è inoltre la possobilità di chattare, anche in questo caso l'interfaccia è piuttosto minimale. Uno spazio di input per immettere il messaggio ed il *button* per inviarlo. Infine, i due percorsi principali dell'applicativo sono disegnati in figura 1 e figura 2.

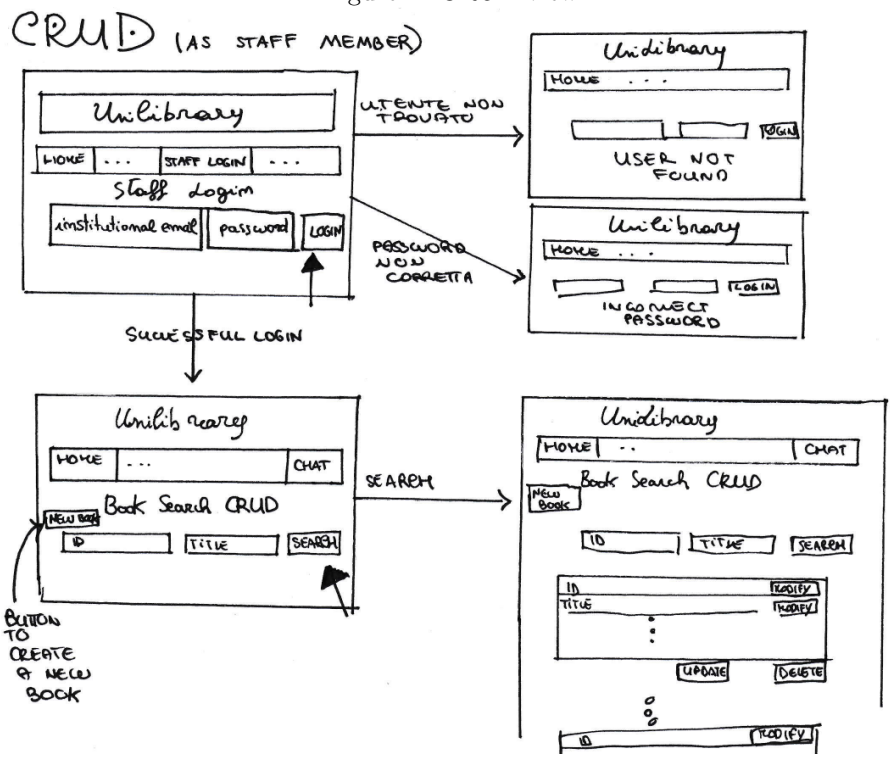
Figure 1: Book search view



0.4 Tecnologie

Lo stack delle tecnologie, dell'applicativo *UniLibrary*, è lo stack MEVN. MongoDB, Express, Vuejs, Nodejs. In questa sezione vi è un breve elenco delle tecnologie, non solo quelle specifiche dello stack sopra citato, utilizzate e le relative motivazioni.

Figure 2: CRUD view



0.4.1 MongoDB

MongoDB è uno dei più noti e diffusi database non relazionali, orientati ai documenti.*"MongoDB si allontana dalla struttura tradizionale basata su tabelle dei database relazionali in favore di documenti in stile JSON con schema dinamico."*[5] MongoDB permette di utilizzare lo *sharding* suddividendo i dati in *shard*. Quando si parla di *"sharding"* o *"sharding pattern"* si parla del seguente concetto: *"dividing data store into a set of horizontal partitions or shards. This can improve scalability when storing and accessing large volumes of data."*[1] Questo meccanismo ha come scopo la scalabilità, nel caso di un'università tale aspetto risulta vantaggioso in quanto la *scalability*, non è più legata alla qualità della macchina, come nel caso del *vertical scaling*.[2] Le università spesso utilizzano macchine non particolarmente potenti e talvolta datate. Tornando a MongoDB, servendosi dello sharding, scala orizzontalmente. *"Horizontal Scaling involves dividing the system dataset and load over multiple servers, adding additional servers to increase capacity as required. While the overall speed or capacity of a single machine may not be high, each machine handles a subset of the overall workload, potentially providing better efficiency than a single high-speed high-capacity server. "*[2] Infine MongoDB è un database **"free to use"**[8], tale

Figure 3: MEVN Stack

caratteristica è vantaggiosa nel caso di un'istituzione pubblica come l'Università che tende a massimizare il risultate con minimo spreco di risorse versate dai contribuenti. Un'altra motivazione che giustifica la scelta è la facilità di utilizzo e la presenza fra gli argomenti trattati durante il corso.

0.4.2 Express

Partendo con la definizione di *nodejs* si ha : "As an asynchronous event-driven JavaScript runtime, Node.js is designed to build scalable network applications."[7] Inoltre *Node.js is an open-source, cross-platform, back-end JavaScript runtime environment that runs on the V8 engine and executes JavaScript code outside a web browser. Node.js lets developers use JavaScript to write command line tools and for server-side scripting—running scripts server-side to produce dynamic web page content before the page is sent to the user's web browser.*[3] *"A framework layered on tope of node.js."* Express è usato per costruire il lato backend dell'applicativo usando strutture e funzioni di nodejs. L'uso di Express è più semplice e sintetico rispetto all'uso diretto di nodejs.

0.4.3 VueJS

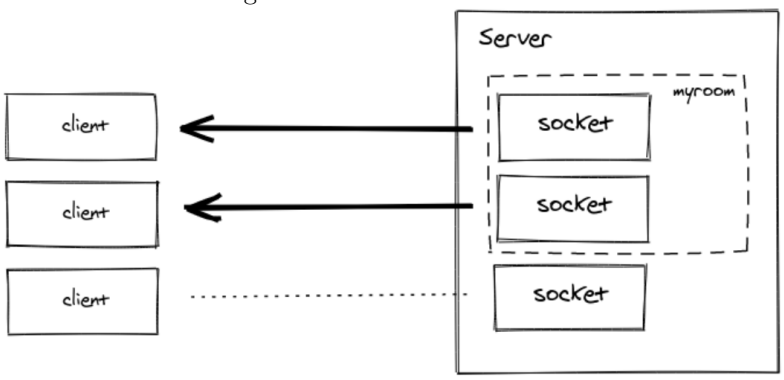
"Vue is a progressive framework for building user interfaces. Unlike other monolithic frameworks, Vue is designed from the ground up to be incrementally

adoptable."[4] Vue.js è stato scelto anche per la ridotta curva d'apprendimento rispetto ai suoi rivali, **react** ed **angular**.

0.4.4 Socket.IO

La chat prevede l'uso della libreria Socket.IO. Tale libreria permette comunicazioni **real-time** ed **event-based**. *"Socket.IO enables real-time, bidirectional and event-based communication. It works on every platform, browser or device, focusing equally on reliability and speed."*[13] Nello specifico della nostra

Figure 4: Socket.IO Rooms



applicazione, i requisiti delineano una specifica tipologia di chat, una "live customer chat". La chat infatti si ispira alle diffuse chat di supporto al cliente presenti in una moltitudine di siti commerciali. Le caratteristiche della chat dell'applicazione, così come le "customer" chat, sono le seguenti : l'assente vincolo all'autenticazione previa interazione tramite chat, la possibilità di mettersi in contatto con il bibliotecario (piuttosto che con l'addetto ai servizi alla clientela dell'impresa commerciale) per poter richiedere le informazioni d'interesse. Vi è quindi la necessità di creare una particolare forma d'interazione fra le parti: tutti i bibliotecari online riceveranno tutti i messaggi di tutti gli utenti che usano la chat, il singolo utente non vedrà i messaggi degli altri utenti, il singolo utente ricevendo un messaggio non sa quale dei bibliotecari gli/le ha risposto perchè il sistema tratta i bibliotecari come una singola entità. Maggiori informazioni sul funzionamento dell'applicativo nella sezione relativa al codice. Per implementare tale logica applicativa si ricorre alle room di Socket.IO. "A room is an arbitrary channel that sockets can join and leave. It can be used to broadcast events to a subset of clients."[13]

0.4.5 JSON Web Token

"JSON Web Token (JWT) is an open standard (RFC 7519) that defines a compact and self-contained way for securely transmitting information between parties

as a JSON object." [6] Alcuni possibili utilizzi dei JSON web token includono l'autenticazione e lo scambio di informazioni. I JSON web token sono un buon modo per scambiare informazioni in modo sicuro fra le parti. In questo progetto vengono utilizzati nelle procedure di login di studenti e bibliotecari, quindi nel processo di autenticazione. In tale frangente, nello specifico immaginiamo il login da parte di un bibliotecario: il bibliotecario inserisce le credenziali, il server verifica la presenza dell'account in base all'email e verifica quindi la correttezza della password, il server, in caso di autenticazione effettuata con successo, restituisce un JSON web token. E' buona pratica che nel token non siano memorizzate informazioni riservate. Nel caso di questo progetto, il JSON web token restituito allo studente/essa o al bibliotecario/a conterrà l'email ma non la password del relativo account. La struttura di un JSON web token è la seguente:

- **Header** *"The header typically consists of two parts: the type of the token, which is JWT, and the signing algorithm being used, such as HMAC SHA256 or RSA."*[6]
- **Payload** *"The second part of the token is the payload, which contains the claims. Claims are statements about an entity (typically, the user) and additional data. Do note that for signed tokens this information, though protected against tampering, is readable by anyone. Do not put secret information in the payload or header elements of a JWT unless it is encrypted. The signature is used to verify the message wasn't changed along the way."* [6]
- **Signature** *"To create the signature part you have to take the encoded header, the encoded payload, a secret, the algorithm specified in the header, and sign that."*[6]

Nel paragrafo **"Codice"**, una sottosezione riguarda il codice di autenticazione da parte di un/a bibliotecario/a. In tale sottosezione, viene esaminata nel dettaglio la modalità di utilizzo dei JSON web token, all'interno dell' applicativo "Unilibrary".

0.5 Codice

In questo paragrafo sono presenti le parti di codice più rilevanti del progetto.

0.5.1 Textual search

Le prime 15 righe di codice si trovano sul file BookTextualSearch.js. In questo file è presente un metodo, dichiarato secondo la sintassi di *vuejs*, di nome *findWithTextualSearch*. Il funzionamento del metodo consiste nel prendere le *keywords* inserite nel campo di ricerca del template (sempre presente in BookTextualSearch.js) e inviarle al server. Per inviarle il client si serve di **axios** e la relativa richiesta **post**, includendo le *keywords* come *content* della richiesta.

Nel momento in cui viene ricevuta una risposta alla richiesta, essa viene memorizzata nella variabile **books**. Inoltre viene posta la variabile **notAnswered** a false. Tale variabile, tramite il *conditional rendering*, controlla la visibilità o la "presenza" del bottone e del campo input relativi alla ricerca. Nel momento in cui si riceve la risposta della ricerca, questi due elementi non vengono inclusi nel DOM. Il risultato è quello di una pagina più "pulita".

```
1
2  findWithTextualSearch(){
3                  if(this.keywords) //not null, not undefined, not empty
4                  {
5                    axios.post("/api/textual_search",{content:this.keywords})
6                         .then(response=> {this.books = response.data
7                           this.notAnswered="false";
8
9                         }).catch(error=>(console.log(error)));
10
11
12                 }
13             }
14
```

Come si vede dalle righe 16-17, l'api che risponde alla richiesta di *findWithTextualSearch* è il metodo *textual_search* del server.

```
15
16    app.route('/api/textual_search')
17    .post(booksController.textual_search);
18
```

Nelle righe 19-34 è riportato il codice di *textual_search*. La callback dell'operazione *find*, si limita ad inviare il risultato della ricerca o un (eventuale) errore.

```
19
20 exports.textual_search=function(req,res){
21    userText=req.body.content;
22
23
24    Book.find({ $text: { $search: userText }} , function(err,book){
25
26       if(err)
27       res.send(err);
28       else {
29          console.log(book);
30          res.json(book);
31       }
32    });
33 };
34
```

0.5.2 Staff/Personnel login

Il codice frontend relativo al login del personale, è costituito dal template in html con eventuali direttive di vuejs, e dal codice eseguito lato client. Da linea

35 a 45 abbiamo il template. Le credenziali richieste ad un bibliotecario sono : password e email istituzionale (ovvero l'email fornitale/gli dall'università). Il metodo di login richiamato con cliccando sul relativo bottone del template[righe 47 - 67] invia una http post request al server. I dati inviati con la richiesta sono le credenziali per l'autenticazione. La callback che gestisce la risposta del server, aspetta un **access token**. Nel caso in cui lo riceva, il campo **isLoggedInAsStaff** verrà settato a true. Viene quindi utilizzato il router per il rendering allo spazio "protetto" a cui si accede nel momento in cui ci si è autenticati come personale della biblioteca. Lo spazio protetto prevede la possibilità di effettuare operazioni CRUD. La modalità di ottenimento del permesso, può essere sufficiente nel caso specifico poichè si suppone che non ci siano, verosimilmente, utenti non autorizzati che desiderino modificare il catalogi di libri della biblioteca. Ad ogni modo, un possibile miglioramento, dell'attuale codice, consiste nel inserire una verifica dell'*access token* ad ogni richiesta di modifica dei dati, e non solo prima di accedere alla pagina. Un'altra alternativa di più veloce implementazione, ma non sicura come la prima possibilità, è quella di controllare che l'*access token* contenga un valore invece di far riferimento al campo *IsLoggedInAsStaff*. E' stata scelta questa modalità per facilità e velocità implementativa, unita al contesto applicativo relativo ad una biblioteca universitaria.

```
<div id="login">
       <h2>Staff Login</h2>
       <input type="text" name="institutional_email" v-model="email"
           placeholder="institutional email" />
       <input type="password" name="password" v-model="password"
           placeholder="Password" />
       <button type="button" v-on:click="login()">Login</button>
       <div class="error_message_line">{{this.errMsg}}</div>
</div>
```

```
login() {

             axios.post("http://localhost:3000/api/personnel-login",{
                 email:this.email , password:this.password })

             .then(response=>{

             if(response.data.accessToken){
               localStorage.setItem('AccessToken',response.data.accessToken);

               data.isLoggedInAsStaff=true;
               router.push({name:"secure-crud"});
             }else{
               this.errMsg=response.data;

             }

```

```
64            }).catch(error => (console.log(error)));
65
66  }
67
```

Lato server, quando viene inviata una richiesta di autenticazione come membro dello staff, viene effettuata la ricerca sul database relativo ai/alle bibliotecari/e usando la e-mail istituzionale ricevuta. Se viene trovato un *document* con tale e-mail si procede al controllo della password. Nel caso in cui le credenziali vengano verificate con successo, il server creerà il token di accesso ed lo invierà al client. Per la creazione del JSON web token, viene utilizzato il modulo *'jsonwebtoken'*. La creazione del token avviene a riga 88 tramite il metodo jwt.sign. All'interno del token vengono inserite la e-mail istitizionale ed il ruolo *"staff"*. Il secondo elemento, per come è strutturato il codice, non è strettamente necessario.

```
68
69  var mongoose = require('mongoose');
70  const jwt=require('jsonwebtoken');
71  Staff = require("../models/LibraryStaffModel.js")(mongoose);
72
73  accessTokenSecret="095034853098";
74
75  exports.validate = function(req, res) {
76    const {email,password}=req.body;
77    console.log(email);
78
79     Staff.findOne({institutional_email:email},function(err,member){
80    console.log("Membro:"+member);
81
82    if(member){
83      if(err)
84        res.send(err);
85      else{
86            if(member.password==password){
87                const accessToken=jwt.sign(
88                    {institutional_email:member.institutional_email,
89                    role:"staff"}, accessTokenSecret);
90                res.json({accessToken});
91            }else{
92              res.send('Password Incorrect');
93            }
94
95        }
96    }else{
97      res.send("Utente non trovato");
98    }
99
100   });
101 };
102
```

0.5.3 Chat

In questa sottosezione viene esaminato il codice della chat dell'applicativo "Unilibrary". La prima parte, righe [103-228], è il codice eseguito dal server, ovvero il codice in backend. La chat si serve della libreria Socket.IO. La struttura dell'interazione prevede che all'arrivo di ogni nuovo utente (in questo caso il termine *utente* indica qualcuno che non si sia autenticato come membro del personale della biblioteca universitaria) si crea una nuova *room*. Su tale room effettuano un *join* tutti i bibliotecari (o bibliotecarie) online e l'utente stesso. Il *join* ad una room implica la ricezione di tutti i messaggi emessi all'interno di essa. In questo modo tutti i bibliotecari online possono leggere e rispondere all'utente, e l'utente può leggere e rispodere ai bibliotecari. L'utente vede i suoi messaggi ma non quelli di altri utenti.

```
103
104 var http=require('http').createServer(app);
105 var io=require('socket.io')(http);
106
107 io.on('connection',(socket)=>{
108
109   console.log('a user connected'+", the user socket.id is: "+socket.id);
110
111   //user disconnected
112   socket.on('disconnect',()=>{
113     console.log('a user disconnected'+", the user socket.id is:
114         "+socket.id);
115   });
116
117   //ADD STAFF MEMBER
118
119   socket.on('add staff member',function( ){
120
121     delete users_hashmap[socket.id];
122     socket.isAdmin=true;
123
124
125     admins_hashmap[socket.id]=socket;
126
127     if(!(Object.keys(users_hashmap).length === 0)){
128       _.each(users_hashmap,function(userSocket){
129
130         let room="room-"+userSocket.roomID;
131         socket.join(room, function(){console.log("socket.join
132             effettuato");});
133
134
135       });
136
137     }
138
```

```
139
140   });
141   //add user
142   socket.on('add user',function(){
143
144     socket.isAdmin=false;
145     socket.roomID=socket.id;
146     let room;
147     room="room-"+socket.id;
148     socket.join(room);
149     _.each(admins_hashmap,function(adminSocket){
150       adminSocket.join(room);
151     })
152     msg="benvenuta";
153     if(Object.keys(admins_hashmap).length === 0)
154     {
155
156       msg=msg+", purtroppo al momento non ci sono bibliotecari online.
157           Orari: lun: 8-12, mar:15-16.
158           E-mail:administration@Unilibrary.it";
159     }
160     io.sockets.in(room).emit('welcome msg',msg);
161     users_hashmap[socket.roomID]=socket;
162
163
164   });
165
166
167   //chat message
168   socket.on('chat message',function(message){
169
170     completeMessage={};
171     definitiveMessage={};
172     if(message.roomID!=null)
173     {
174
175       completeMessage.content=message.content;
176       completeMessage.roomID=message.roomID;
177       definitiveMessage=completeMessage;
178
179       room="room-"+message.roomID;
180       socket.to(room).emit('chat message', definitiveMessage);
181
182
183
184     }else{
185       //inviamo il messaggio nella socket dell'utente
186         completeMessage.content=message.content;
187         completeMessage.roomID=socket.id;
188         definitiveMessage=completeMessage;
```

```
189
190
191
192         var room="room-"+socket.id;
193           io.sockets.in(room).emit('prova prova');
194           //socket.to(room).emit('chat message');
195         socket.to(room).emit('chat message',definitiveMessage);
196
197
198 io.of('/').in(room).clients(function(error,clients){
199       var numClients=clients.length;
200       console.log("numero client in room "+room+":"+numClients);
201       clients.forEach(function(client) {console.log(client);
202
203       });
204
205    });
206     }
207 });
208
209 socket.on('disconnect',function(){
210   if(socket.isAdmin)
211   {
212
213     delete admins_hashmap[socket.id];
214
215   }else{
216     //is a user
217
218     delete users_hashmap[socket.roomID];
219
220     _.each(admins_hashmap,function(adminSocket){
221         adminSocket.leave(socket.roomID)});
222   }
223 });
224
225
226
227 });
228
```

Lato client, viene aperta una connessione via socket tramite la riga 230 *"socket=io()"*. Nel caso in cui non si tratti di un bibliotecario, viene emesso l'evento *"add user"* in modo da poter iniziare la procedura di creazione della relativa *room*. Nel momento in cui la socket riceve l'evento *chat message*, il messaggio viene memorizzato in un campo di nome *local_chat_message*. Nel caso in cui il ricevente sia un bibliotecario, vi è la necessità di sapere quale utente ha inviato il messaggio. Si identifica l'utente tramite la *roomID*, visto che ad ogni utente è associata una specifica stanza e una stanza è associata ad un solo utente. Si memorizza quindi il messaggio e la rispettiva *roomID*. Se il

ricevente è un utente non deve domandarsi chi sia il mittente. Può essere solo uno dei bibliotecari (visti come un'unica entità). In questo caso il messaggio viene memorizzato fra i *messages* ed il mittente settato a *Librarian*.

```
var socket=io();
   if(!data.isLoggedInAsStaff){
      socket.emit('add user');

   }

   socket.on('welcome
       msg',(message)=>{data.messages.push({content:message,from:"server"})});

     socket.on('chat message',(chat_message)=>{

         data.local_chat_message=chat_message;

            if(data.isLoggedInAsStaff){

          data.find=data.roomIDs.includes(data.local_chat_message.roomID);

               if(!data.find)
               {
                  data.roomIDs.push(data.local_chat_message.roomID);
               }

               data.messagesForStaff.push({roomID:data.local_chat_message.roomID,
                   content:data.local_chat_message.content, from:"user"});

            }else{

               data.messages.push({content:chat_message.content,from:"Librarian"});

         }
            });
```

Infine la parte di template va dalla riga 262 alla riga 313. Nella chat utente (USER CHAT), è presente un ciclo che per ogni stanza visualizza i messaggi. Quindi visualizza le varie chat utente. Nella chat per i membri dello staff (STAFF MEMBER CHAT), tramite due direttive *v-for* vengono visualizzate le varie chat, ed i relativi messaggi. Le chat non vengono mostrate entrambe grazie alla direttiva *v-if-else*.

```
<div id="both-chats">
   <!--USER CHAT-->
<template id="#genericUserChat" v-if="!data.isLoggedInAsStaff &&
    data.wantChat">
      <ul>
         <li v-for="message in data.messages">
```

```
269
270           <strong>{{message.from}}:</strong>
271               {{message.content}}
272             </li>
273             </ul>
274             <input type="text" v-model="data.newMsg" placeholder="Enter
275                 message here"/>
276             <div class="chat-buttons">
277                                 <button
278                                     v-on:click="sendFromUser()">Send</button>
279                                 <button v-on:click="data.wantChat=false"
280                                     v-show="data.wantChat==true">Hide
281                                     Chat</button>
282                                 <button v-on:click="data.wantChat=true"
283                                     v-show="data.wantChat==false">Show
284                                     Chat</button>
285           </div>
286 </template>
287
288
289 <!--STAFF MEMBER CHAT-->
290 <template v-else-if="data.isLoggedInAsStaff && data.wantChat">
291
292
293 <div id="chat">
294 <ul v-for="id in data.roomIDs">
295 <li v-for="message in specific_room_messages(id)" ><!--per ogni stanza-->
296
297    <div class="chatmessage"> <strong> {{message.from}}
298        </strong>:{{message.content}}</div>
299
300 </li>
301
302    <input type="text" v-model="data.newStaffMessages[id]"
303        placeholder="Enter message here"/><button
304        v-on:click="sendFromStaff(id)">Send</button>
305 </ul>
306 </div>
307    <div class="visibility-chat-buttons">
308
309                    <button v-on:click="data.wantChat=false"
310                        v-show="data.wantChat==true">Hide Chat</button>
311
312    </div>
313
```

0.6 Accessibility

Nello svolgimento del progetto *UniLibrary*, particolare rilevanza riveste il tema dell'accessibilità. *"The Web is fundamentally designed to work for all people, whatever their hardware, software, language, location, or ability. When the Web meets this goal, it is accessible to people with a diverse range of hearing, movement, sight, and cognitive ability. Thus the impact of disability is radically changed on the Web because the Web removes barriers to communication and interaction that many people face in the physical world. However, when websites, applications, technologies, or tools are badly designed, they can create barriers that exclude people from using the Web."*[9] L'accessibilità è un requisito che si ha la necessità di perseguire nel momento in cui si vogliano creare siti web di qualità. Il termine *Accessibility* o accessibilità, indica la fruibilità delle risorse web, anche da parte di persone con un qualche tipo di disabilità ed anche persone senza disabilità nelle seguenti circostanze:

- persone che usano apparecchi con schermi di ridotte dimensioni e/o differenti modalità di input;
- persone che riscontrano la modifica di alcune abilità dovute all'invecchiamento;
- persone con una connessione internet dall'ampiezza di banda limitata;
- persone con disabilità temporanea, come la perdita degli occhiali da vista o un braccio rotto;
- persone che si trovano in situazioni che prevedono limitazione, come in piena luce solare, o il trovarsi in un ambiente in cui non è, per qualche ragione, consentito ascoltare contenuti audio.

. La *Web Accessibility* dipende da molteplici componenti, incluse le tecnologie web, i browser, i siti web etc. La **WAI**, *W3C Web Accessibility Initiative* ha sviluppato delle specifiche tecniche e delle linee guida per descrivere e supportare l'accessibilità nella pratica. Queste specifiche sono considerate standard internazionali per l'accessibilità del web. Alcuni di questi principi e specifiche vengono utilizzate nel codice di questo progetto.

0.6.1 Accessibility principles

In questa sezione vengono introdotti alcuni dei requisiti della *Web Accessibility* per i siti web, le applicazioni web, i browser ed altri strumenti.

0.6.2 Page Structure

Un contenuto ben strutturato permette una navigazione più efficiente. Per ottenere tale struttura nelle applicazioni, ci si serve dell'HTML e WAI-ARIA. Iniziamo con le **Page Regions**: *"Mark up different regions of web pages and applications, so that they can be identified by web browsers and assistive technologies."*[12]. Elenchiamo ora le varie componenti di una pagina.*Most websites have a region at the top of every page that contains site-wide information, such as the website logo, search function, and navigation options. HTML5 provides the ¡header¿ element, which can be used to define such a region.*[12] Vi sono poi altre componenti: **footer**, **nav**, **main**, **aside**. *"Similar to the page header, most websites also have a region at the bottom of every page that contains site-wide information, such as copyright information, privacy statements, or disclaimers. HTML5 provides the ¡footer¿ element, which can be used to define such a region."*[12]. *"The HTML5 ¡nav¿ element can be used to identify a navigation menu. A web page can have any number of navigation menus. Use labels to identify each navigation menu."*[12]. La regione *main*:*"Use the HTML5 ¡main¿ element to identify the main content region of a web page or application."*[12] Infine, la componente *aside*:*"Use the HTML5 ¡aside¿ element to identify regions that support the main content, yet are separate and meaningful sections on their own; For example, a side note explaining or annotating the main content."*[12] Nello specifico caso del codice di questo progetto, vi sono solo alcune di queste regioni. Ad esempio, l'interfaccia utente non presenta mai la necessità, per come è strutturata,di avere una parte che abbia le caratteristiche di un *footer*, nè una parte che abbia le caratteristiche di un *aside*. Sono però presenti un *header*, un *nav* ed un *main*. Oltre ad individuare le specifiche regioni e ad assegnare loro gli opportuni tag, è buona pratica l'*etichettamento*. *"Provide labels to distinguish two page regions of the same type, such as "main navigation" and "sub-navigation" menus using a ¡nav¿ element on the same page. Labels are also used to change the default identification of page regions, for example, to identify a ¡aside¿ region as "advertisement". Regions that are unique, such as ¡main¿, do not need additional labels."*[12] Nel codice del progetto , però, non vi sono elementi doppi. Quindi non vi è necessità di applicare etichette per discernere fra più elementi con lo stesso tag (fra quelli elencati come regioni della pagina). La terza pratica è l'*headings*. *"Headings communicate the organization of the content on the page. Web browsers, plug-ins, and assistive technologies can use them to provide in-page navigation."*[12] Nel codice le intestazioni sono usate e non vengono omessi numeri intermedi. *"Nest headings by their rank (or level). The most important heading has the rank 1 (¡h1¿), the least important heading rank 6 (¡h6¿). Headings with an equal or higher rank start a new section, headings with a lower rank start new subsections that are part of the higher ranked section. Skipping heading ranks can be confusing and should be avoided where possible"*[12] Come ultimo aspetto, la *context structure*."*Mark up website content semantically, so that the website is extensible. Valid semantics create content that is reusable and more meaningful to assistive technologies."*[12] Il codice usa frequentemente uno degli elementi semantici all'interno del *main*

content: la lista non ordinata,*ul, unordered list.*

0.6.3 Forms

Le *form* sono comunemente usate per l'inserimento di dati da parte dell'utente. Costituiscono uno dei modi più comuni in cui avviene l'interazione fra sistema applicativo ed utenza. Alcuni esempi di contesti in cui vengono usate le *form* sono i seguenti: *login*,*registering*, *purchasing*. In *Unilibrary* sono presenti due *form* di *login*, quella degli studenti e quella del personale della biblioteca, il *form* di ricerca (con un solo campo in cui inserire le *keyword* per la ricerca di un libro), il *form* di inserimento di un nuovo libro nel catalogo (la *create*). In questa sezione elenchiamo le buone pratiche, dal punto di vista dell'*accessibilità* di un applicativo, e come sono state inserite nel codice del progetto. La prima pratica che analizziamo è quella del *labeling controls*. Con *labeling controls* si intende quanto segue: *"Provide labels to identify all form controls, including text fields, checkboxes, radio buttons, and drop-down menus. In most cases, this is done by using the ¡label¿ element."*[11] E' importante che una etichetta, o *label*, descriva lo scopo della componente del form, o *form control*. Una etichetta può essere associata ad un elemento del form, in modo implicito o esplicito. Per associazione *esplicita*, si intende creare un tag *label* che preceda il *form control* che si vuole etichettare. Tale tag avrà il campo *for* uguale al campo *id* dell'elemento da etichettare. L'associazione *implicita* invece, deve essere usata quando non si conosce a priori l'*id* dell'elemento da etichettare, ad esempio nel caso in cui esso venga generato da uno script. In questo caso si crea un'associazione implicita fra l'etichetta e l'elemento da etichettare, includendoli entrambi nel tag *label*. L'associazione *esplicita* è preferibile, quando possibile: *"Generally, explicit labels are better supported by assistive technology."*[11]

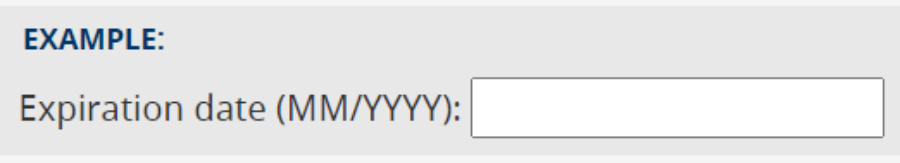
```
<label for="keywords" class="visuallyhidden">Search:</label>
                <input type="text" v-model="keywords"
                    v-if="notAnswered===true" id="keywords"
                    name="keywords" placeholder="Please write the
                    keyword/s to find the book, es title, edition,
                    author/s etc :-)"/>
```

Il codice mostra un ulteriore caso: *"A label for a form control helps everyone better understand its purpose. In some cases, the purpose may be clear enough from the context when the content is rendered visually. The label can be hidden visually, though it still needs to be provided within the code to support other forms of presentation and interaction, such as for screen reader and speech input users."*[11] In questo caso si nasconde l'etichetta, pur includendola. La si nasconde definendo regole di stile appropriate, nello specifico la classe *visuallyhidden*.

```
label.visuallyhidden
  border: 0
  clip: rect(0 0 0 0)
```

```
height: 1px
margin: -1px
overflow: hidden
padding: 0
position: absolute
width: 1px
```

Passiamo ora ad una diversa pratica, le *form instructions*. In generale :*"Provide instructions to help users understand how to complete the form and use individual form controls. Indicate any required and optional input, data formats, and other relevant information. Screen readers often switch to "Forms Mode" when they are processing content within a ¡form¿ element. In this mode they usually only read aloud form elements such as ¡input¿, ¡select¿, ¡textarea¿, ¡legend¿, and ¡label¿. It is critical to include form instructions in ways that can be read aloud."*[10] E' possibile inserire le istruzioni all'interno delle etichette, come nell'immagine che segue: Nel caso in cui non si inseriscano le instruzioni



all'interno dell'etichetta, il design è più flessibile.Ci si può servire due diversi approci, i quali usano due diversi attributi: *aria-labelledby* e *aria-describedby*. Un altro tipo di *form instructions* è il *placeholder text*. *"Placeholder text provides instructions or an example of the required data format inside form fields that have not yet been edited by the user. Placeholder text is usually displayed with lower color contrast than text provided by users, and it disappears from form fields when users start entering text. If the placeholder text contains instructional information or examples that disappear, it makes it more difficult for users to check their responses before submitting the form."* [10] Tale tipologia d'istruzione viene usata all'interno dell'applicativo *Unilibrary*. I *placeholders* guidano l'utente ma non vengono usati dalle tecnologie d'assistenza. **"Assistive technologies, such as screen readers, do not treat placeholder text as labels. Moreover, at the time of writing this tutorial, placeholder text is not broadly supported across assistive technologies and not displayed in older web browsers."**[10] Un'altra criticità del meccanismo dei *placeholder* è il loro stile di default. Il colore ad essi associato è infatti spesso un colore che non contrasta abbastanza con lo sfondo e quindi non soddisfa i requisiti minimi di contrasto della WCAG. Ciò implica che per molte persone è difficile riuscire a vedere i placeholder. Per ovviare a questo problema, è stato inserito il codice in figura 0.6.3 all'interno del foglio di stile. Questo codice cambia il colore di default con il colore *light grey* che ha un contrasto sufficiente a soddisfare i requisiti, assumendo che il colore in *background* sia il bianco.

CODE SNIPPET:

```
::-webkit-input-placeholder {
   color: #767676;
   opacity: 1;
}

:-moz-placeholder { /* Firefox 18- */
   color: #767676;
   opacity: 1;
}

::-moz-placeholder {  /* Firefox 19+ */
   color: #767676;
   opacity: 1;
}

:-ms-input-placeholder {
   color: #767676;
   opacity: 1;
}
```

0.7 Test

Test effettuati sul codice e test con utenti.

0.7.1 Valutazione euristica

Il testing sull'interfaccia è stato effettuato tramite la valutazione delle euristiche che fanno parte del *Decalogo di Nielsen* (Jakob Nielsen, 1994). Tale valutazione non è stata svolta da utenti reali. Per ogni euristica è associato un numero da 1 a 4. Tale numero indica il punteggio ottenuto dall'applicativo relativamente all'euristica, la quale costituisce il criterio di valutazione.

- *Visibilità dello stato del sistema.* Il principio incarnato da questa euristica è che *"il sistema deve sempre tenere informato l'utente su cosa sta facendo"*. Nel caso specifico, considerando i vari momenti in cui l'utente interagisce on il sistema, lo score è di 2 su 4. Durante il login, di studenti e membri dello staff, il sistema comunica l'esito dell'autenticazione. L'esito dell'interazione non viene però comunicato durante le interazioni CRUD.

- *Corrispondenza tra sistema e mondo reale.* Ovvero: *"il sistema deve parlare il linguaggio dell'utente, con parole, frasi e concetti a lui familiari."* Nell'applicativo non ci sono icone, tutti i bottoni presentano etichette, si suppone abbastanza chiare. La valutazione è 3.

- *Controllo e libertà. "L'utente deve avere il controllo del contenuto informativo e muoversi liberamente tra i vari argomenti."* Nell'applicativo sono state evitate le *"procedure costrittive troppo lunghe (iscrizioni)"* e le

"azioni non volute dall'utente". Non c'è però la possibilità di effettuare *undo* e *redo*. La valutazione è 3 su 4.

- *Consistenza e standard."L'utente deve aspettarsi che le convenzioni del sistema siano valide per tutta l'interfaccia."* L'interfaccia del sito è piuttosto consistente. Valutazione pari a 4.
- *Prevenzione dell'errore."Evitare di porre l'utente in situazione ambigue, critiche e che possono portare all'errore."* Si ha se non si creano situazioni ambigue e si *mette sempre a disposizione la possibilità di tornare indietro*. Valutazione 2 su 4.
- *Riconoscimento anzichè ricordo*. Ovvero: *"Le istruzioni per l'uso del sistema devono essere ben visibili e facilmente recuperabili."*Poichè è tutto molto schematico e abbastanza intuitivo, la valutazione è di 3 su 4.
- *Flessibilità ed efficienza d'uso." Offrire all'utente la possibilità di un uso differenziale (a seconda della sua esperienza) dell'interfaccia"*. Aldilà dei contenuti a cui si accede tramite autenticazione, non vi è possibilità di uso differenziale fra utenti con lo stesso permesso di accesso. Valutazione 1 su 4.
- *Design ed estetica minimalista*. Dare maggior importanza al contenuto che all'estetica. Durante il design dell'interfaccia si è tenuto in considerazione il principio "less is more". Si spera che l'interfaccia sia altamente minimalista. Valutazione 4.
- *Aiuto all'utente. "Aiutare l'utente a riconoscere, diagnosticare e recuperare l'errore."* I messaggi di errore sono espressi in linguaggio comprensibile e in modo preciso. Non si chiede conferma per le operazioni importanti (che nel caso specifico potrebbero essere le modifiche che è possibile fare tramite le api che implementano le operazioni CRUD).
- *Documentazione."Anche se il sistema dovrebbe essere usabile senza documentazione è preferibile che essa sia disponibile"*

0.8 Deployment

Rilascio, installazione e messa in funzione. Nella cartella "Collection" del progetto, sono presenti tre file JSON da caricare in robo3T. Si tratta delle collezione degli studenti, del personale della biblioteca, delle prenotazioni, ed infine dei libri. Sono file che contengono pochissime istanze, utilizzati per testare il funzionamento del software. Creare il db test, in caso esista già posizionarvi le collezioni del progetto. Chiamare la collezione relativa agli studenti, **Student**, la collezione relativa ai libri **Books**, quella relativa al personale della biblioteca, **LibraryStaff**, infine quella relativa alle prenotazioni **Reservations**. Creare un *text index* sul campo *description_for_textual_search* della collezione *Books*.

Per avviare l'applicazione, posizionarsi sulla cartella del progetto , *progetto* , e lanciare il comando **node index.js**.

0.9 Conclusioni

In conclusione, il progetto ha permesso di utilizzare le tecnologie più diffuse nel contesto della creazione di siti web. Da questa pratica nascono alcune osservazioni. Rendere una pagina web maggiormente accessibile, richiede frequentemente semplici accorgimenti che "costano" relativamente poco a chi programma , ma hanno l'enorme beneficio di permettere la creazione di siti web fruibili a più persone e, quindi, di più alta qualità.

Bibliography

[1] https://docs.microsoft.com/en us/azure/architecture/patterns/sharding. Sharding pattern.

[2] https://docs.mongodb.com/manual/sharding/. Mongodb documentation.

[3] https://en.wikipedia.org/wiki/Node.js. Node.js.

[4] https://it.vuejs.org/v2/guide/. Introduction, what is vue.js?

[5] https://it.wikipedia.org/wiki/MongoDB. Mongodb.

[6] https://jwt.io/introduction. Jwt.

[7] https://nodejs.org/en/about/. About — node.js.

[8] https://www.mongodb.com/what-is mongodb. What is mongodb?

[9] https://www.w3.org/WAI/fundamentals/accessibility intro/. Introduction to web accessibility.

[10] https://www.w3.org/WAI/tutorials/forms/instructions/. Form instructions.

[11] https://www.w3.org/WAI/tutorials/forms/labels/. Labeling controls.

[12] https://www.w3.org/WAI/tutorials/page structure/regions/. Page structure.

[13] Socket.IO. https://socket.io/.