UNILIBRARY
Applicazioni e Servizi Web

Medjouwo Diagong Christ Valtes - 0000935441
christ.medjoumo@studio.unibo.it

Magadjou Tchendjou Idene Prisca - 000097859
ideneprisca.magadjou@studio.unibo.it

September 30, 2020

Contents

1 Introduction

Nel ambito delle applicazioni e dei servizi web sempre più si fa largo l'esigenza di sviluppare applicazioni più dinamiche e interattive. Il progetto nasce dall'esigenza di realizzare una piattaforma online per la gestione di una biblioteca universitaria che possiede materiale cartaceo e digitale.
L'obiettivo del sistema è quello di consentire ad un utente di visualizzare e prenotare a scelta sua dei libri cartacei e/o download libri digitale.

- In primo piano, L'applicazione si rivolge ad un'unica categoria d'utente **il cliente** alla ricerca di un Libro da caricare o da prenotare.
- In secondo piano l'applicazione dispone una interfaccia amministratore che gli consente di eseguire operazioni **CRUD** .

La piattaforma interattiva mette a disposizione dell'utente:

- una **Chat** in tempo reale sulla quale l'utente può chiedere informazioni ad un bibliotecario.
- Un sistema di notifiche che avviserà un'utente in lista di attesa quando un libro è pronto per essere ritirato.

Nel quadro della progettazione, l'architettura del nostro sistema di gestione, siamo orientati verso un sito dinamico con un modelle a quattro livelli cioe Browser, Presentation, Application Logic e storage.

2 Requirements

2.1 functional requirements

La nostra soluzione andrà di fatto a comporre:

- Una interfaccia Utente sulla quale può
  1. Registrarsi(le informazioni relative alla sua identità: nome, numero di telefono e email).
  2. Login (email e password) l'email verrà usato da sistema di notifiche.
  3. Cercare un libro e conoscere il suo stato attuale cioè se è disponibile, prenotato o solo consultazione interna. L'utente ha la possibilità di scaricare direttamente i libri di tipo digitale.
  4. Chiedere informazioni nella chat.
  5. Eventualmente registrarsi in **lista di attesa** nel caso in cui un libro sia indisponibile.
- Interfaccia amministratori è molto importante saper che per l'implementazione della nostro sistema, l'amministratore e il bibliotecario rappresentano lo stesso utente. Attraverso quel interfaccia si può eseguire operazioni CRUD cioè **C**reate(aggiungere libri nella base di dati) **R** Read(elencare libri) **U** Update(modificare libri) Delete(cancellare libri sulla piattforma).
- Un sistema di notifiche deve avvisare quando il libro che era prenotato è pronto per essere ritirato da un'utente che era in lista di attesa.
- una chat studente/bibliotecario deve consentire chiedere informazioni.

2.2 non functional requirements

Lo sviluppo del sistema pone alcuni requisiti non funzionali che devono essere presi in considerazione per avere una migliore comprensione dello scopo del progetto. Poiché la maggior parte di essi è correlata ad aspetti relativi alle prestazione del sistema di prenotazione e alla reattività del sito web ci'o'e reattività della chat studente/bibliotecario e del sistema di notifiche. Le tecnologie usate e il modo di usarli hanno un impatto maggiore per promuovere questi requisiti durante la progettazione dell'applicazione.

2.3 scenari

Attraverso un use case diagram modelizziamo i diversi casi d'uso del sistema dai diversi utenti tanti quelli attivi che quelli reattivi.

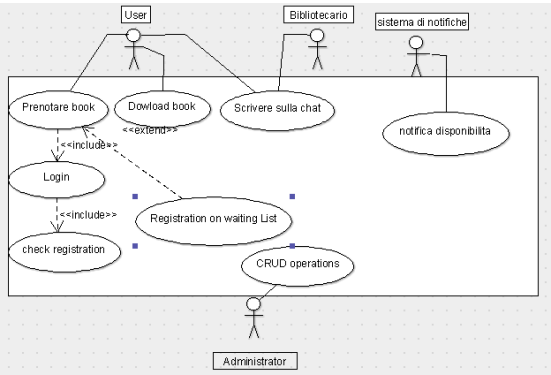

Figure 1: use case diagram

3 Design

la nostra applicazione è stata realizzata seguendo un modello iterativo basato su UCD (User Centered Design) con utenti "reali".Durante le fase di design e di test, due(02) utenti sono stati coinvolti nella di analisi e design dei diversi modi di implementazione delle varie diverse funzionalità esposti dall'applicazione. Per prima cosa, sono stati individuati i target, a partire dai quali sono state raccolte i requisiti di ogni utenti coinvolto poi questi requisiti sono stati utilizzati per capire come le funzionalità dell'applicazione dovessero rispondere alle caratteristiche degli utenti. Questa metodologia centralizzata su utenti è stata integrata con metodologia Agile ciòè subdivisione del lavoro tra i membri della team, lavoro iterativo e modulare. Basandosi sui requisiti di base del progetto, e su quali dei target user, abbiamo provato a t capire in che senso le funzionalità del sistema dovessero rispondere alle caratteristiche degli diversi utenti.

TARGET USER ANALISIS
Sono stati individuati i seguenti target:

- **studente**
- **amministratore**

Personas: Elsi Elsi è uno studente registrata che vuole prenotare un libro

- **Contesto A:** Elsi è una nuova utente che si registra ,login e prenota il libro.
- **Contesto b:** Elsi è gia regitrata , effetua il login per prenotare ma non trova il libro che vuole prenotare.

Personas: Admin Admin è un utente che indossa il ruolo di bibliotecario e amministratore del sistema

- **Contesto A:** Admin vuole gestire tuttte le operazione .
- **Contesto D :** Admin risponde nella chat come bibliotecario.

Scenario A	
1	Elsi inserisce I suoi dati
2	Elsi si registra
3	Elsi accede al suo profilo
4	Elsi ricerca e seleziona un libro
5	Elsi prenota il libro se il libo è available
6	Scrive nella chat per chiedere diverse informazioine

Scenario B	
1	Elsi inserisce I suoi dati
3	Elsi accede al suo profilo
4	Elsi ricerca e seleziona un libro
5	Elsi va nella lista di attesa se il libro già prenotato o in Maintenance
6	Scrive nella chat per chiedere diverse informazione

Scenario C	
1	Admin va nella sua pagina
2	Consulta la lista dei book
3	Effettua tute le operazioni di CRUD

Scenario d	
1	Risponde ad un messagio nella chat

3.1 Structure

L'architettura della nostra piattaforma è basata su un modello a quattro livelli, Browser, Presentation, Application Logic e storage in cui, il livello di application logic genera un output completamente privo di aspetti presentazionali (ad esempio, un file XML) e lo dà in input al livello presentation logic).

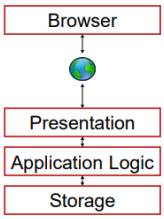

Figure 2: Modello a quattro livelli

3.2 Behaviour

Come visto precedentemente, il progetto si articola in tre principali entità alcune attive ed altre reattive, ognuna con il proprio flusso di esecuzione interno.

- Lo Studente e l'amministratore costituiscono entità attive del sistema.
- Il sistema di notifiche un'entità reattiva con un comportamento periodico ma di alto livello: reagisce come detto prima in risposta ad un'aggiornamento dello stato di un libro specifico mandando una notifica allo studento registrato sulla lista di attesa per quel libro.

4 Tecnologie

In seguito dell'analisi teorica è stata posta in atto una fase più operativa, andando a confrontare varie tecnologie e framework, selezionando quelle più adatte secondo noi e le nostre aspettative per lo svolgimento di quel progetto. Sono state descritte le principali tecnologie utilizzate, e il processo che ha portato alla scelta di queste. Essendo lo scopo del progetto usare le tecnologie attuale del web dinamico viste a lezione, si è subito scatata l'ipotesi di una soluzione Stack basata sul paradigma "Javascript Everywhere".

4.1 Stack MERN

In base alla familiarità tecnologica ed alla documentazione la scelta finale è ricaduta sull'utilizzo dello stack MERN che è costituito di quattro tecnologie e che consentono la progettazione della nostra applicazione in corrispondenza ai quattro livelli della nostra architettura.

- **MongoDB** è un data Base No SQL, reale e prezioso che ci consente la gestione della persistenza. In modo particolare si è fatto uso della liberia Mongoose **mongoose** per interfacciarsi al database MongoDB. Si è scelta questa libreria perché risulta essere una delle pi'u utilizzate nel suo campo perche l'unica presentata a lezione.
- **Express jsexpress js** come framework di sviluppo JavaScript lato server della nostra applicazione.
- **React JS** (3)è il framework di sviluppo modulare JavaScript lato client molto flessibile. React consente la creazione di componenti riutilizzabili, con input di dati, che possono cambiare nel tempo
- **Node js** (4) è l'ambiente di esecuzione per l'applicazione server-side.

4.2 Autenticazione e Autorizzazione

Allineandosi alle prassi di sicurezza previste per la memorizzazione delle password, si è deciso di fare uso del modulo bcrypt(**?**) che ha consentito di salvare le password su database in maniera sicura (sotto forma di hash). Nell'elaborare una richiesta HTTP proveniente da un client, il server fa uso di un JWT (Jason Web Token)(8) per garantire allo stesso tempo autenticazione e autorizzazione.

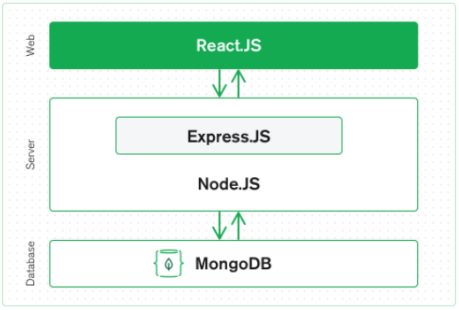

Figure 3: Stack Mern

Il JWT viene generato dal server al momento della registrazione o del login dell'utente e restituito al client. Quest'ultimo provvederà a presentarlo al server allegandolo ad ogni richiesta futura.

4.3 Supporto allo stile

Per quanto riguarda lo stile, ci si è appoggiati al toolkit UIkit(9) è un framework Frontend per la progettazione veloce di interfacce web responsive. UIkit utilizza Flexbox per realizzare il proprio Grid Stystem e permettere la realizzazione di siti caratterizzati da un layout fluido.si è fatto direttamente uso del supporto di Flexbox per l'implementazione dei componenti Footer, Header e Navbar delle users pages che ci ha consentito di ottenere un comportamento più predicibile e stabile degli elementi della pagina quando questa doveva adattarsi a display di diversi device con diverse dimensioni.

4.4 Communicazione Client-Server

Per la gestione delle richiesteasincrone si usa della libreria axios. Per l'implementazione delle funzionalità real-time quali l'invio dei messaggi in chat da parte di studenti e bibliotecari, si è deciso di fare uso della libreria socket.io (**?**) presentata a lezione.

5 Codice

5.1 Login e Registration

L'applicazione web come detta prima riserva funzionalità specifiche per loggati con ruolo studenti e per l'utente loggato amministratore. Per questo motivo lato server, si

è fatto un meccanismo di identificazione e autorizzazione che permetta di individuare il ruolo dell'utente loggato, di conseguenza, valutare il suo diritto di accedere ad una certa funzionalità. Lato cliente, è possibile capire se l'utente è loggato e a quale tipologia appartiene (studenti o amministratore).

```
const login = (req, res, next) => {
  var username = req.body.username;
  var password = req.body.password;

  User.findOne({ $or: [{ email: username }] }).then((user) => {
    if (user) {
      bcrypt.compare(password, user.password, function (err, result) {
        if (err) {
          res.json({
            error: err,
          });
        }
        if (result) {
          let token = jwt.sign({ name: user.name }, "AZQ,PI)0(", {
            expiresIn: "4h",
          });
          res.json({
            message: "Login succesfully!",
            token,
            user,
          });
        } else {
        }
      });
    } else {
      res.json({
        message: "No user found!",
      });
    }
```

Figure 4: Utlizzo di Bcrypt e JWT lato Server

Si è usato **Localstorage e Sessionstorage** per salvare i dati di un'utente specfico utilizzando il browser.il Localstorage consente una registrazione persistente come usando il **cookie session** ma con una migliore capacità senza aver bisogno di aggiungere i date nel header della richiesta HTTP.

Per motivo di sicurezza, abbiamo utilizzato le tecnologie **JWT e BCRYPT** durante la registrazione e il Login di un utente.

- **Jason Web Token** , ci consente di generare un Token ad ogni connessione di un utente.
- **bcrypt**, consente di criptare le password utenti prima di memorizzarle sul database. Durante la registrazione, viene criptata la password lato cliente prima di inviarla attraverso una richiesta http, poi nuovamente criptata lato server prima di memorizzarla sul database. In fase di login, la password viene criptata lato cliente prima dell'invio, poi nuovamente criptata lato server e confrontata con quella memorizzata sul database.

```
const login = (req, res, next) => {
  var username = req.body.username;
  var password = req.body.password;

  User.findOne({ $or: [{ email: username }] }).then((user) => {
    if (user) {
      bcrypt.compare(password, user.password, function (err, result) {
        if (err) {
          res.json({
            error: err,
          });
        }
        if (result) {
          let token = jwt.sign({ name: user.name }, "AZQ,PI)0(", {
            expiresIn: "4h",
          });
          res.json({
            message: "Login succesfully!",
            token,
            user,
          });
        } else {
        }
      });
    } else {
      res.json({
        message: "No user found!",
      });
    }
  ``
```

Figure 5: Utlizzo di Bcrypt e JWT lato Server

```
useEffect(() => {}, []);
const history = useHistory();
const sumit = () => {
  let data = qs.stringify({
    username: username,
    password: password,
  });

  const config = {
    headers: {
      "Content-Type": "application/x-www-form-urlencoded",
    },
  };

  axios.post("http://localhost:5000/AuthRoute/login", data, config).then(
    (response) => {
      localStorage.setItem("login", JSON.stringify(response.data));
      history.push("/");
      window.location.reload();
    },
    (error) => {
      console.log(error);
    }
  );
};
```

Figure 6: Utlizzo di Bcrypt e JWT lato Client

5.2 Gestione della prenotazione di un libro

- La funzione BorrowBook contenuta nel file *"Borrow-Controller"* della cartella *"Backend"* viene chiamata Quando il ritiro di un libro viene prenotato tramite il pulsante **"Borrow"** dell'interfaccia utente.
  Fa decrementare la quatità de quel libro quindi ha un impatto anche sul cambiamento dello stato del libro.
- La funzione "ReturnedBook" implementa la modifica dello stato di un libro una volta che viene ritornato in biblioteca da un'utente.

```
28  const borrowBook=(req,res,next)=>{
29    console.log(req.body)
30    console.log("d")
31      var foundBook = new borrow({
32          returned: false,
33          userId: req.body.userId,
34          bookId: req.body.bookId,
35          boorowingdate:new Date(req.body.boorowingdate),
36          returningdate:
37           new Date(Date.now() + (3 * 24 * 60 * 60 * 1000)),
38              actualReturnDate: Date.now(),
39      });
40      foundBook
41      .save()
42      .then(updatedBorrowedBook => {
43         // console.log(req.body.bookId)
44          Book.findOneAndUpdate(
45              { _id: req.body.bookId},
46              {
47                  $inc: {
48                    quantity: -1
49                  }
50                },
51              function (err, book) {
52                if (err) res.send(err);
53                else {
54                      res.end("borrowin effectuated");
55                }
56              }).then((borrowingRecord)=>{
```

Figure 7: Funzione di prenotazione da uno studente

5.3 Gestione delle routes lato server

Le API esposte dal nostro server sono organizzate sulla base del ambiente su cui agiscono, quindi sono stati suddivisi in diversi file.

```
const returnBook=(req,res)=>{

    const borrowId = req.body
    if(!borrowId ){
        res.status(400).send('Ensure borrowId is present')
    }
    //console.log(borrowId)
    console.log(req.body);

      const { returningdate } = req.body;
      const surcharge = moment(Date.now()) > moment(returningdate);
      const bookToReturn = req.foundBook;

      Boorow.update(
            {_id: req.body.borrowId },

            {   returned: true,
                actualReturnDate: Date.now(),
            },
            function (err,book) {
                if (err) res.send(err);
                else {
                    res.end("Super emprunt");
                }
            }
        )
        .then((borrowUpdated)=>{

            if (borrowUpdated[0] === 0) {
                 res.status(404).send({
                  message: 'You have not borrowed this book'
```

Figure 8: Funzione di modifica dello stato del libro

Authroute				
Method	route	parameter	response	description
POST	/register	Name: String Email: String Password: String		Registrare nel Sistema un nuovo utente se (creare il suo profilo) creando un nuovo web token
POST	/login	Email: String Password: String	Profile_type: String _id : Object Id	Aprire il profile di un utente particolare

Figure 9: Authentication route

bookroute				
Method	route	parameter	response	description

GET	/book/:id	_id: String	lista di oggetti contenenti informazioni sui libri
			{
			Status: String,
			id: BookId
			pageCount: Number,
			publishedDate: Date,
			authors: String,
			coverImageName: String,
			coverImageType: String,
			categoryId: String,
			}
				L'utente cerca un libro specifico e display it. (List_book)

GET	/book/:id	_id: String	Un unico oggetto contenendo informazioni sui libri	Consente all'amministratore di aggiungere nuovi libri nel database(read_book)
PUT	/book/:id	_id: String		Consente all'amministratore di aggiornare/modificare dati sui libri nel database (update_book)
DELETE	/book/:id	_id: String		Consente all'amministratore di cancellare libri nel database (delete_book)
POST	/books/			Create_book
GET	/books/		Un unico oggetto contenendo informazioni sui libri	Consente all'utente di Cercare libri (find_book)

Figure 10: Gestione admin route

bookroute				
Method	route	parameter	response	description

GET	/book/:id	_id: String	lista di oggetti contenenti informazioni sui libri
			{
			Status: String,
			id: BookId
			pageCount: Number,
			publishedDate: Date,
			authors: String,
			coverImageName: String,
			coverImageType: String,
			categoryId: String,
			}
				L'utente cerca un libro specifico e display it. (List_book)

GET	/book/:id	_id: String	Un unico oggetto contenendo informazioni sui libri	Consente all'amministratore di aggiungere nuovi libri nel database(read_book)
PUT	/book/:id	_id: String		Consente all'amministratore di aggiornare/modificare dati sui libri nel database (update_book)
DELETE	/book/:id	_id: String		Consente all'amministratore di cancellare libri nel database (delete_book)
POST	/books/			Create_book
GET	/books/		Un unico oggetto contenendo informazioni sui libri	Consente all'utente di Cercare libri (find_book)

Figure 11: Gestione admin route

Categoryroute				
Method	route	parameter	response	description
POST	/Category			Dare una categoria ad un
GET	/categories		CategoryType: String	Ricuperare la categoria di un libro

Uploadroute				
Method	route	parameter	response	description
POST	/fichier	file		Ritorna informazioni specifiche su un'utente registrato

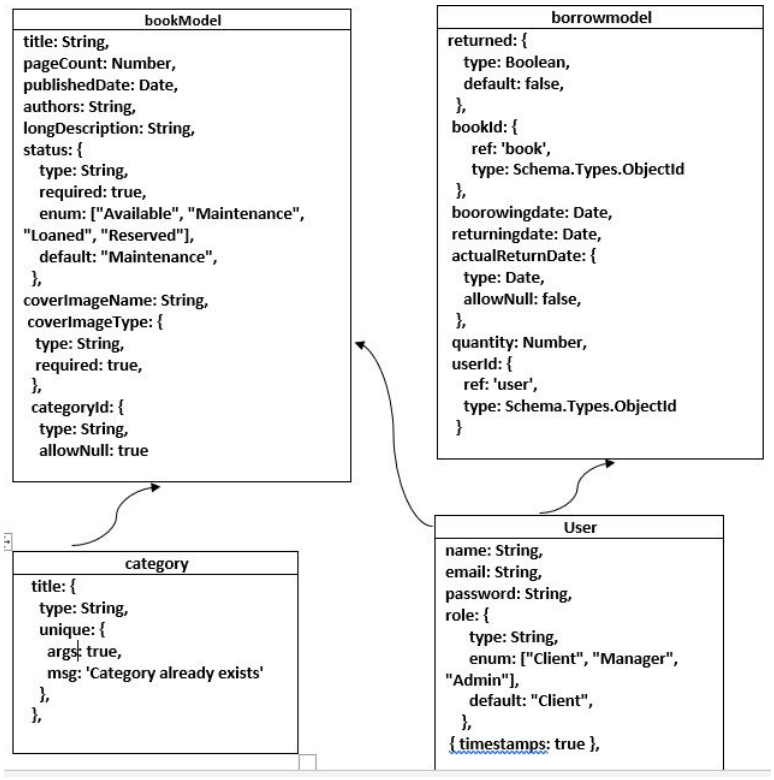

Figure 12: application models

5.4 gestione della persistenza

6 Test

Le funzionalità frontend implementate sono state testate dal team sia su browser **Chrome** che su **Firefox** con l'obiettivo di garantire portabilità. Le API sviluppate lato server sono state testate utilizzando lo strumento **Postman** che ha permesso di verificare che tali API. avessero il comportamento desiderato prima di procedere con l'integrazione con il front-end e quindi con il test della funzionalità completa.

6.1 Test con utente

6.2 Euristica di Nielson

Dal punto di vista dell'usabilità dell'interfaccia, il sistema è stato sottoposto preventivamente ad euristica di Nielsen(6). Di seguito si riportano alcune considerazioni emerse.

- **Visibilità dello stato del sistema**: le eventuali latenze presenti danno un feedback visivo tramite messaggi o dialog boxes all'utente che lo rendono consapevole e rassicurato del fatto che il sistema sta elaborando la sua richiesta.
- **Corrispondenza tra sistema e mondo reale**: termini e icone utilizzate nel sistema corrispondono a quelli usati in maggior parte nelle applicazioni web.
- **Controllo e libertà**:si è cercato di ridurre il più possibile il numero di operazioni necessarie all'utente limitandosi ad un'unico login dopo la registrazione per poter portare a buon fine un task. Non esistono diversi modi per svolgere lo stesso task.
- **Consistenza e standard**: i pulsanti che abilitano l'avvio di un task presentano le stesse caratteristiche visive stutte le diverse interfacce. Definita inizialmente la gamma dei colori da utilizzare, si è fatto uso uniforme di questi all'interno del sistema.
- **Prevenzione dell'errore**: si è cercato il più possibile di evitare situazioni ambigue per l'utente che lo potessero portare a commettere errori. L'utente è guidato nelle procedure che abilitano l'esecuzione di un task attraverso label, suggerimenti e messaggi di successo o di errore.

- **Riconoscimento anziché ricordo**: Usando React, Il layout dinamico invece che statico è comune a tutte le immagini utilizzate e su tutte le interfacce, lo stile di ogni pagina è uniforme per tutto il sistema.
- **Flessibilità ed efficienza d'uso**: le funzionalità offerte dal sistema sono già molto semplici e limitati ad un semplice clic per cui non si 'e ritenuto necessario introdurre modalità di utilizzo alternative.
- **Design e estetica minimalista** : KISS e Less Is More sono stati i principi chiavi che si è deciso di seguire. Già dalla schermata di "Home": essa rivolge l'attenzione dell'utente verso una prima lista di libri.
- **Aiuto all'utente**: i messaggi di errore che vengono visualizzati sono espressi in modo comprensibile all'utente.
- **Documentazione**: non si è ancora ritenuto necessario produrre una guida per l'utente.

6.3 Usability test

Il sistema, una volta realizzato, è stato sottoposto all'attenzione di entrambe le tipologie di utente coinvolte in modo da misurare la semplicità di utilizzo delle diverse interfacce utenti, l'obiettivo di quel test è stato di comprendere cosa dovrebbe essere migliorato.
Eseguendo il test, ad entrambe le categorie di utenti sono stati assegnati un elenco di task da portare a buon fine. intervenendo il meno possibile, la team di sviluppo ha ricoperto un ruolo di osservatore . Di seguito viene riportato un riassunto dei risultati ottenuti.
1- Studente
0intervistato in fase di stesura dei requisiti 'e stato poi

coinvolto nuovamente durante la fase di testing. I task che sono stati individuati per quest'ultimo sono:

- **Task 1** lo studente deve eseguire una registrazione completa;
- **Task 2**: lo studente una volta registrato, deve effettuare un login con il suo profilo;
- **Task 3**: lo studente deve effettuare la ricerca di un libro a scelta sua;
- **Task 4**: se quel libro viene trovato sul sito lo studente deve guardare il suo stato ;
- **Task 5**: se il libro allo stato **"available"** deve prenotare un ritiro in biblioteca;
- **Task 6**: lo studente deve chiedere informazioni su un specifico libro sulla chat.
  Lo studente è riuscito ad eseguire la maggior parte delle task senza aver bisogno di un aiuto da parte degli osservatori, però al momento della prenotazione, cliccando sul pulsante "Borrow" succede un'errore che purtroppo ci sembra essere un'errore di back-end invece di front end.
  Quell'errore si nota guardando lista dei libri prenotati si accorge che non viene registrata la prenotazione effetuatta.

2- Amministratore

- **Task 1** l'amministratore deve accedere all ìa sua schermata "Localhost:3000/admin";
- **Task 2**: l'amministratore deve aggiunngere nuovi libri prenotabili sulla piattaforma;
- **Task 3**: l'amministratore deve cancellare libri a scelta sua sulla piattaforma:

- **Task 4**: l'amministratore deve aggiornare informazioni relativi ad un libro disponibile sull'applicazione.
- **Task 4**: l'amministratore/bibliotecario rispondere a domande fatte da un'utente sulla chat.

7 Deployment

7.1 Installazione

- Clonare il repository git clone https://gitlab.com/medjouwo/blas.git
- Dalla cartella principale del progetto,spostarsi nella cartella src poi, inizializzare il front-end con il commando npm install
- Essendo nella cartella src, spostarsi di nuova nella cartella backend del progetto, inizializzare il server generale dell'app con il commando npm install
- Nella cartella backend spostsarsi di nuovo nella cartella chatserver ed effettuare il commando npm install per finire

7.2 Messa in funzione

Dalla cartella principale del progetto, avviare il front-end npm start

Nella cartella backend del progetto avviare il server con nodemon index.js

Nella cartella chatserver che si trova dentro la cartella backend del progetto avviare il server della chat con nodemon index.js

- Collegarsi al sito da un browser http://localhost:3000/

8 Conclusioni

I risultati ottenuti dai test effettuati e i consigli espressi dagli utenti coinvolti, hanno portato il team alle seguenti considerazioni. IL sistema di autenticazione utenti funziona per bene e hanno accessi a tutti i diversi libri presente sulla piattaforma. Per quanto riguarda le funzionalità che interessano lo studente ci vorrebbe lavori futuri per poter migliorare:

- la gestione del sistema di notifiche dell'utenza sulla disponibilità di un libro e sulla chat per consentire comunicazione a distanza,
- Si potrebbe implementare anche la possibilità di ricercare un libri per nomi di autore che renderebbero più efficace la ricerca.

Per quanto riguarda le funzionalità che interessano il bibliotecario e l'amministratore le operazioni sui libri vengono tutte eseguite con successo .
Quel progetto ci ha consentito di applicare quai tutti i concetti attuali del dominio del Web visti a lezione e anche di oltrepassare questi concetti collegandosi a concetti legati alla sicurezza in rete ad esempio per poter portare a buon fine il progetto.

References

[1] “Mongoose”,
`https://mongoosejs.com`

[2] ”Express js”,
`https://expressjs.com`

[3] ”React”,
`https://reactjs.org`

[4] ”Node js”,
`https://expressjs.com`

[5] *Metodologie Design.*, Silvia Mirri
Corso Applicazione e Servizi Web Unibo

[6] *metodologie sviluppo e testing.*, Silvia Mirri
Corso Applicazione e Servizi Web Unibo

[7] “Bcrypt”,
`https://github.com/kelektiv/node.bcrypt.js`

[8] “Json Web Token”,
`https://github.com/auth0/node-jsonwebtoken`

[9] ”UIkit”
`https://getuikit.com/`