Ingegneria e Scienze Informatiche LM
Cesena - Applicazione e servizi web

Progetto
Ship tracker

Studente: Andrea Brighi - Matteo Lazzari
Data: September 4, 2023

Contents

1 Requisiti **2**
- 1.1 Requisiti funzionali . . . 2
  - 1.1.1 Studio iniziale del progetto . . . 2
- 1.2 Requisiti non funzionali . . . 2
  - 1.2.1 Usabilità . . . 2
  - 1.2.2 Efficienza . . . 2
  - 1.2.3 Implementazione . . . 3
  - 1.2.4 Security . . . 3

2 Design **4**
- 2.1 Mockup . . . 4
  - 2.1.1 Bozza . . . 4
  - 2.1.2 Raffinamento . . . 7
- 2.2 Design finale . . . 9
- 2.3 Architettura del sistema . . . 10
- 2.4 Backend . . . 11
- 2.5 Frontend . . . 13

3 Tecnologie **15**
- 3.1 Stack di riferimento . . . 15
- 3.2 Docker . . . 16
- 3.3 Server . . . 16
- 3.4 Client . . . 16
- 3.5 SCSS (Sassy - CSS) . . . 17
  - 3.5.1 Geoapify . . . 17

4 Codice **18**
- 4.1 Frontend . . . 18
  - 4.1.1 Backend service . . . 18
  - 4.1.2 Logger service . . . 19
- 4.2 Backend . . . 19
  - 4.2.1 Mongoose . . . 20
  - 4.2.2 Bcrypt . . . 21

5 Test **22**
- 5.1 Test con utenti . . . 22
  - 5.1.1 Risultati . . . 22

6 Conclusioni **24**
- 6.1 Sviluppi futuri . . . 24

Introduzione

Il progetto in questione consiste nel realizzare un'applicazione web che si occupa del tracciare le navi durante il loro viaggio.

L'utente, per visualizzare le informazioni, deve registrarsi al sistema e può registrare una nave per il quale lui risulta il capitano.

Ogni utente registrato ha la possibilità di vedere la posizione in tempo reale delle navi registrate al sistema e, nel caso in cui anche l'utente abbia una nave a proprio carico, è possibile stabilire una comunicazione tra le navi e/o una torre di controllo.

La torre di controllo è composta da un utente che può visualizzare le navi in transito e visualizzare quali navi hanno bisogno di soccorso

1 Requisiti

In questo capitolo verranno discussi tutti i requisiti funzionali e non che ci si è posti durante lo svolgimento del progetto.

In particolare, si parlerà di requisiti per i 3 tipi di utenti: Il visitatore (l'utente solo registrato), il capitano (l'utente che usufruisce dei servizi della pagina web) e controllore (utente del controllo a terra).

1.1 Requisiti funzionali

Per garantire una miglior esperienza di utilizzo, si è utilizzata una struttura semplice e con poche pagine, con una grafica semplice.

1.1.1 Studio iniziale del progetto

È stato discusso quali potessero essere le varie funzionalità dei vari utenti nell'applicazione. Tra questi, sono stati identificati vari punti:

- La mappa è accessibile a qualsiasi persona che visita la pagina web;
- Un utente può registrarsi per poter accedere al menù di creazione della nave;
- Un utente registrato può creare una nave. Ad ogni utente è associata una nave (si predispone l'utilizzo dell'applicazione al capitano);
- I controllori (utenti speciali) hanno la capacità di visualizzare tutte le informazioni delle navi;
- Il capitano ha la possibilità di cambiare lo stato della nave da "normale" a "in allarme" (e viceversa) in caso di problemi;
- Il capitano ha la possibilità di aprire una comunicazione con altre navi/controllori.

1.2 Requisiti non funzionali

In questa sezione si parlerà dei requisiti non funzionali su cui ci si è soffermati durante l'implementazione.

1.2.1 Usabilità

Goal principale dell'applicazione, si sono seguite linee guida e metodologie HCI. Per seguire questo requisito, le varie componenti sono state studiate per essere più semplici possibili e favorire l'interazione di un utente medio.

1.2.2 Efficienza

Necessitando delle stesse informazioni in punti e scopi diversi del codice, si è optato per la non ridondanza dei dati all'interno del database, portando alcune query ad essere

leggermente più complesse di altre ed ad avere un numero maggiore di iterazioni con il db per svolgere alcune funzionalità. Nonostante tutto, questo va ad influire considerevolmente sull'efficienza dell'applicazione.

1.2.3 Implementazione

Si è cercato di seguire il più possibile l'uso di best practice orientate ad Angular, come il riuso di componenti, che influisce anche sulla leggibilità del codice. Ogni funzionalità dell'applicazione è stata implementata come una componente indipendente.

1.2.4 Security

In un'applicazione che possiede informazioni personali è molto importante occuparsi della sicurezza di questi dati. Perciò si sono seguite pratiche per rendere il tutto più sicuro possibile.

JWT

Utilizzato per implementare il controllo degli accessi, autorizzazioni e gestione delle sessioni. In seguito al login di un utente, vengono restituiti access token e refresh token, con i dati dell'utente utili all'interno dell'applicazione. L'access token è necessario per ogni richiesta effettuata al server, se non è presente esso restituisce un errore 401, per evitare che malintenzionati inseriscano dati all'interno del database o ottengano informazioni delicate presenti al suo interno.

bcrypt

Si è utilizzata la libreria bcrypt per quanto riguarda l'hashing delle password, in modo tale da poterle memorizzare sul database in modo sicuro.

Inoltre, all'interno del server è presente la funzionalità di regolare la grandezza dell'hash usato e il numero di iterazioni di hashing da fare, per ridurre la velocità di crackare password (a discapito di performance).

2 Design

Per quanto riguarda il design dell'applicazione, si sono seguiti diversi step fondamentali. Ad inizio progetto sono stati fatti dei mockup per avere una idea di come gestire il codice e l'interfaccia utente.

Durante la prima fase di design è sempre stato tenuto il considerazione il principio KISS (Keep It Simple, Stupid), in quanto il target di utilizzo non è competente in ambito tecnologico e una interfaccia semplice garantisce un miglior utilizzo e una miglior esperienza generale con l'applicazione.

A seconda delle funzionalità che si sono ritenute più importanti, si è utilizzato un approccio incrementale per lo sviluppo dell'applicazione, partendo dai componenti essenziali. Grazie a ciò, è stato possibile testare l'app a blocchi, velocizzando l'individuazione di problemi e rendendolo il tutto più stabile man mano che si procedeva con lo sviluppo.

2.1 Mockup

I mockup sono stati una parte essenziale per velocizzare lo sviluppo, soprattutto nella fase iniziale del progetto. Infatti, grazie ad essi, si sono potute delineare le funzionalità principali e quali sarebbero stati i punti critici durante lo sviluppo.

2.1.1 Bozza

Inizialmente si era pensato ad un approccio monopagina, ovvero dove tutte le informazioni sono visualizzate direttamente su una sola pagina e l'utente, interagendo con i vari bottoni, poteva visualizzare su mappa tutte le informazioni e accedere alla chat.

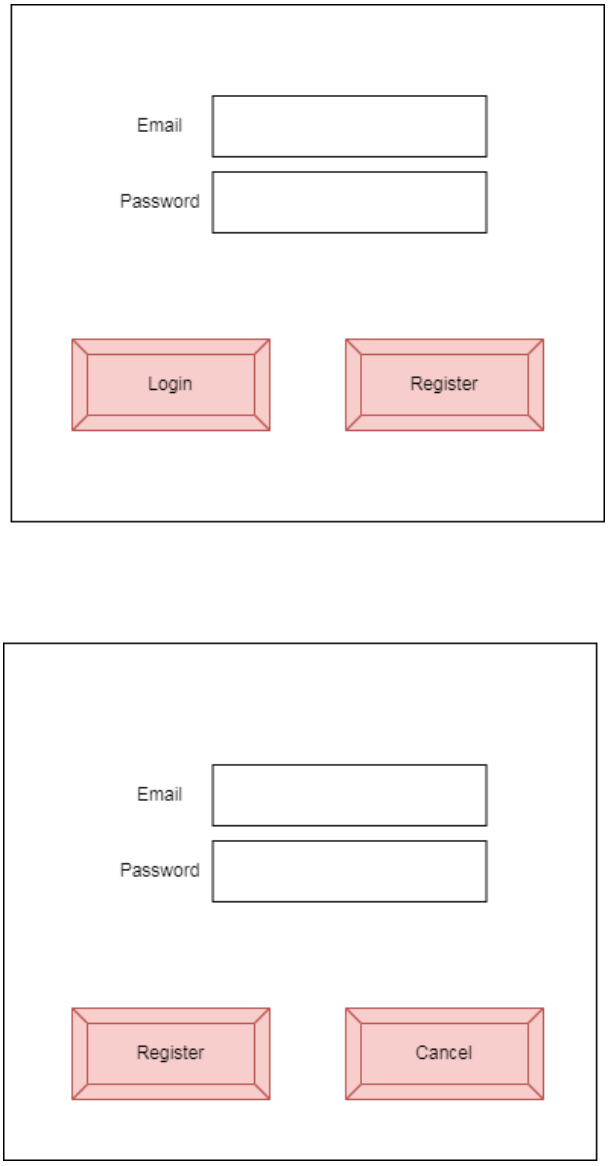

Figure 2.1: Mockup iniziale per la pagina di login e registrazione.

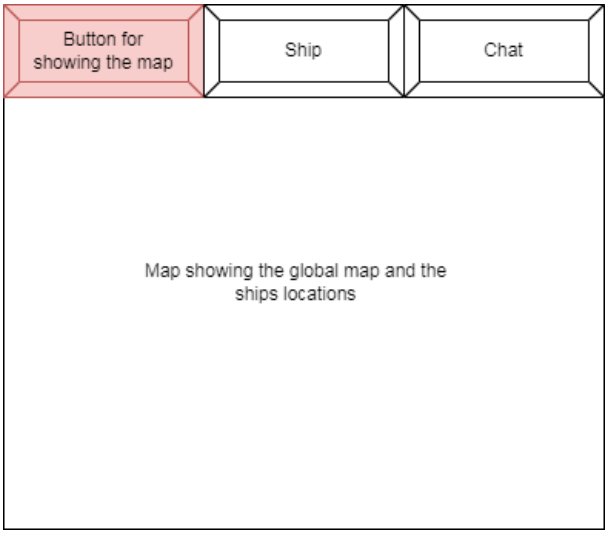

Figure 2.2: Mockup iniziale per la pagina riguardate la visualizzazione della mappa.

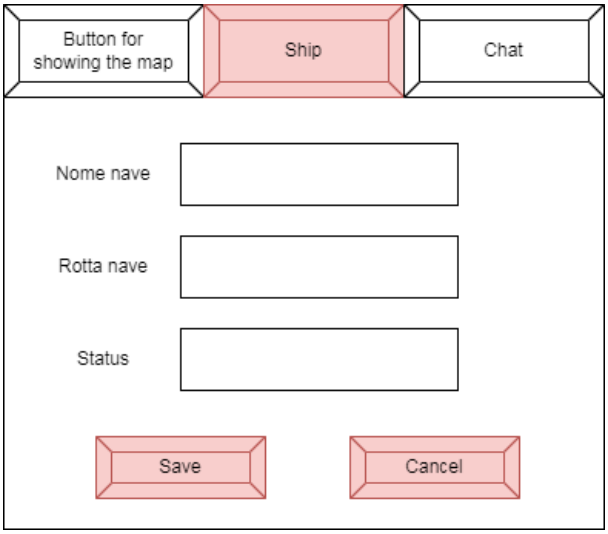

Figure 2.3: Mockup iniziale per la pagina di informazioni sulla nave.

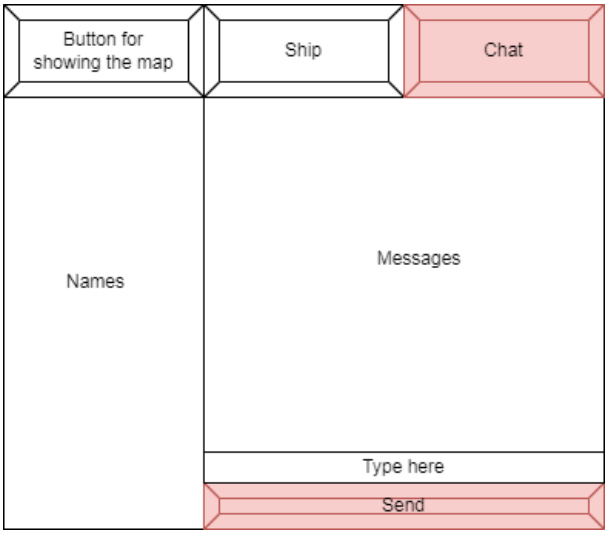

Figure 2.4: Mockup iniziale per la pagina per la chat con altre navi.

2.1.2 Raffinamento

Dopo varie discussioni e analisi (anche da parte di utenti esterni al progetto), si è deciso che una grande interfaccia con numerosi bottoni e sezioni con varie funzionalà, risulterebbe essere complessa e non intuibile.

Per questo si è optato per una suddivisione in tab delle funzionalità, in modo da avere maggior chiarezza e differenziare le varie sezioni.

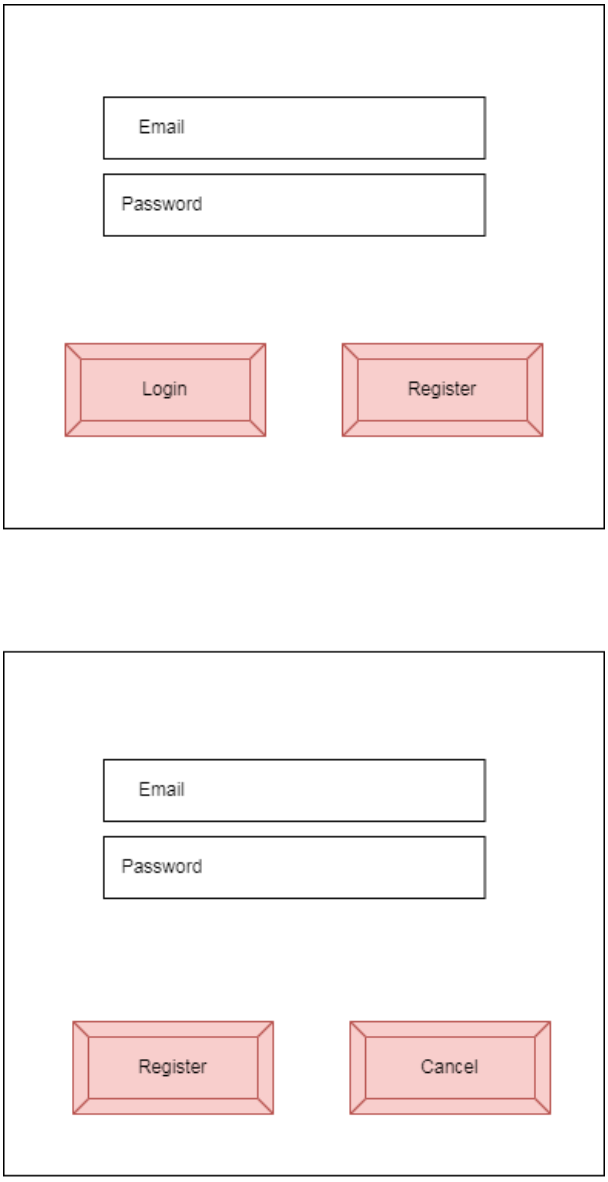

Figure 2.5: Mockup raffinato per la pagina di login.

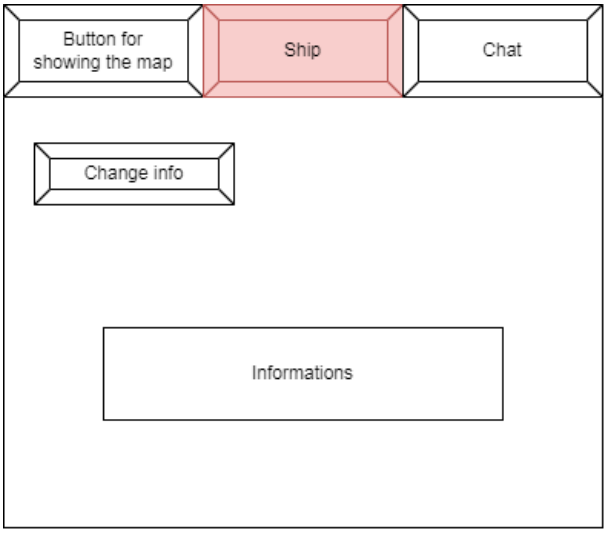

Figure 2.6: Mockup ragginato per la pagina con le informazioni sulle navi.

In particolare, il design è stato fatto per distaccare la parte che riguarda la visualizzazione delle informazioni della nave, con la parte riguardante la modifica di esse. Inoltre, è stato modificato leggermente il design della parte di login/registrazione.

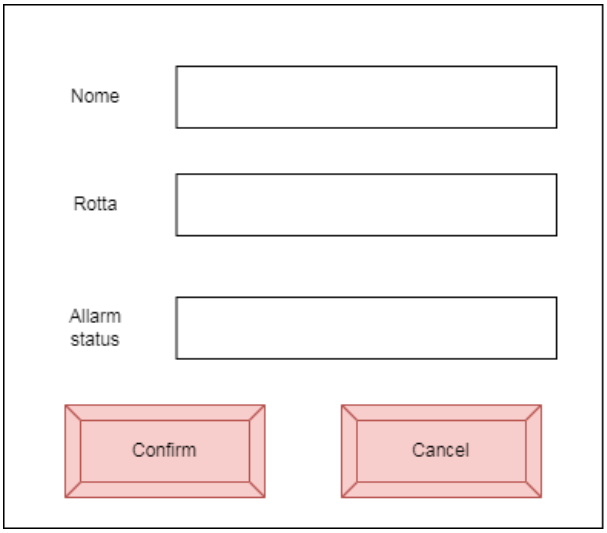

Figure 2.7: Mockup iniziale per la pagina per la chat con altre navi.

2.2 Design finale

Al fine di garantire una buona esperienza all'utente, si è utilizzato uno stile coerente in tutte le varie funzionalità

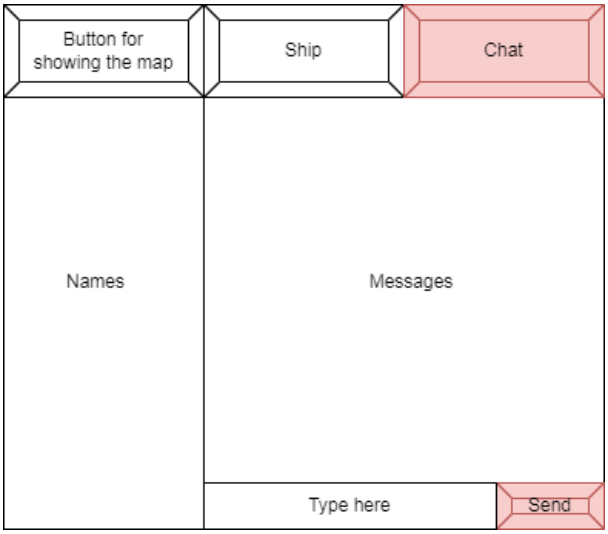

Figure 2.8: Mockup finale per la pagina delle chat.

Tutte le altre UI non sono state modificate in quanto gli utenti esterni al progetto hanno confermato la facilità nell'utilizzo.

2.3 Architettura del sistema

L'architettura del sistema aderisce al modello REST. Ciò permette di avere una struttura modulare, che porta anche ad una miglior scalabilità del sistema. Inoltre, il sistema è stato sviluppato seguendo i metodi/funzioni messi a disposizione dalle librerie, senza legarsi a nessuna funzionalità di sistemi in cloud. Ciò permette una grande dinamicità nel cambiamento del provider, facendo si di non aver bisogno di una riscrittura della logica.

All'interno del sistema sono state modellate diverse entità, ognuna con la relativa funzione:

- Capitano: ha la possibilità di effettuare la registrazione di un veicolo. Una volta verificato, ha la possibilità di visualizzare la mappa interattiva e di aprire comunicazioni dirette con altre navi;

- Torre: Ha la possiblità di effettuare il login presso il sito e di visualizzare tutte le informazioni delle navi e di aprire chat dirette per la comunicazione.

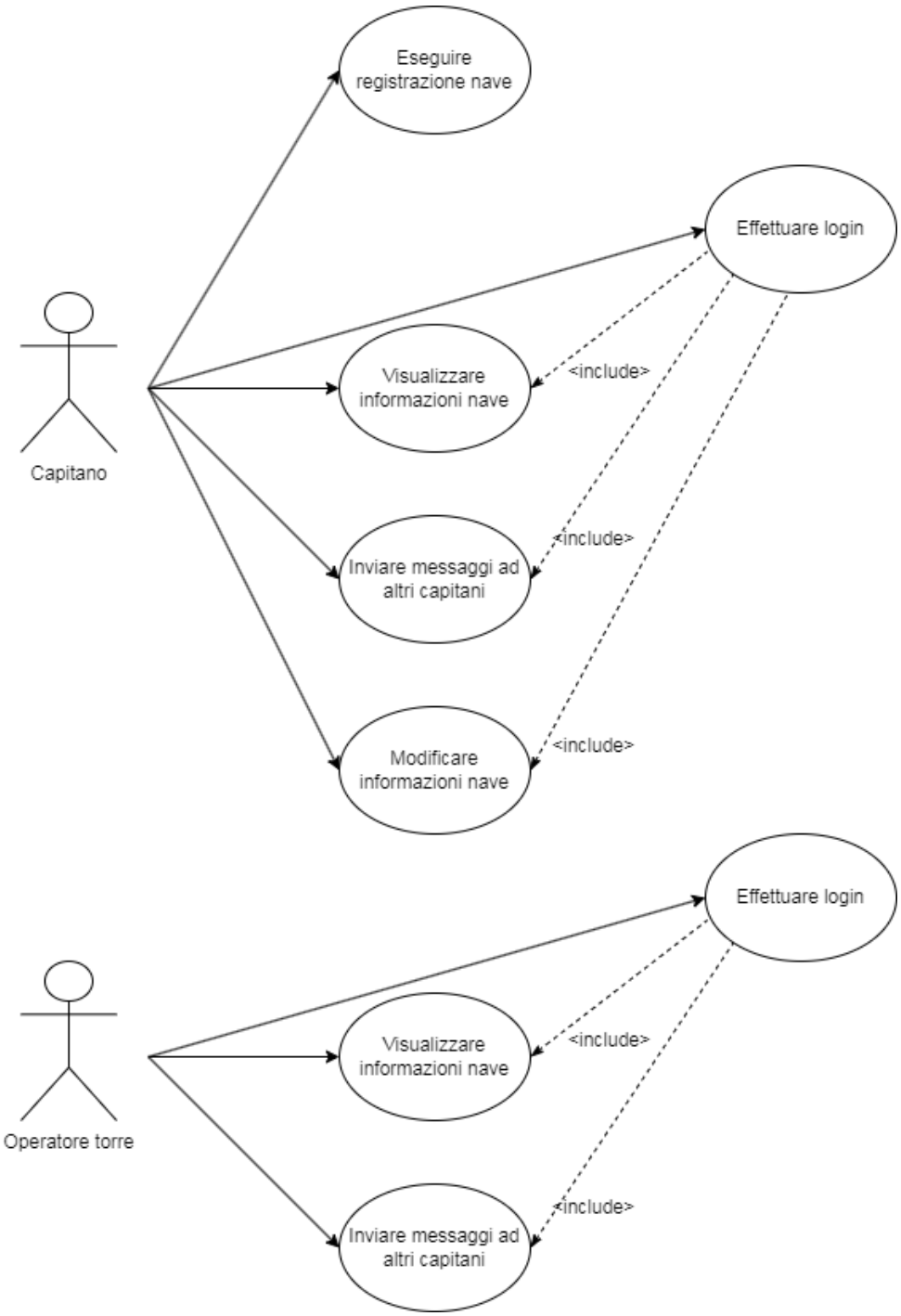

Figure 2.9: Attori e funzionalità del sistema.

2.4 Backend

Il sistema di backend è stato progettando sviluppando il sistema in blocchi indipendenti di codice. Infatti, tutte le funzionalità sono state divise in base al tipo di azione da svolgere (creazione, modifica, rimozione...). Tramite questo sistema, il sistema è possibile suddividerlo in moduli specifici da effettuare il deploy su macchine diverse, andando a soddisfare il requisito di scalabilità orizzontale e verticale specificati.

Per rendere il sistema il più modulare possibile, è stato utilizzato un sistema di routing interno, in grado di indirizzare la chiamata richiesta, direttamente al modulo interessato. Questo ha un grande impatto sia per le performance, sia per la modularità (il server non è un monolite di chiamate HTTP, quindi è possibile scalare e separare i componenti facilmente).

Il sistema comunica con il database tramite un apposito modulo, che ha il solo compito di fare da tramite per le azioni che possono essere eseguite con il database, andando a disaccoppiare la logica del sistema con l'ottenimento dei dati.

La scelta di utilizzare un sistema NoSQL rispetto ad un database relazionale si è basato su alcune scelte effettuate nella parte di progettazione:

- Scalabilità: NoSQL è progettato per poter gestire grandi quantità di dati su più macchine, consentendo un'ampia scalabilità orizzontale. SQL, d'altra parte, è più adatto per database di dimensioni relativamente più piccole o per carichi di lavoro meno intensi;

- Prestazioni: NoSQL è spesso più veloce di SQL in quanto i motori di database NoSQL sono progettati per eseguire query parallele su grandi quantità di dati;

- Facilità di sviluppo: NoSQL spesso semplifica lo sviluppo di applicazioni in quanto non richiede uno schema rigido e consente di modificare rapidamente il modello dei dati.

Tutto ciò ha contribuito ad uno sviluppo più veloce e una migliore comunciazione (a livello di velocità) tra server e database.

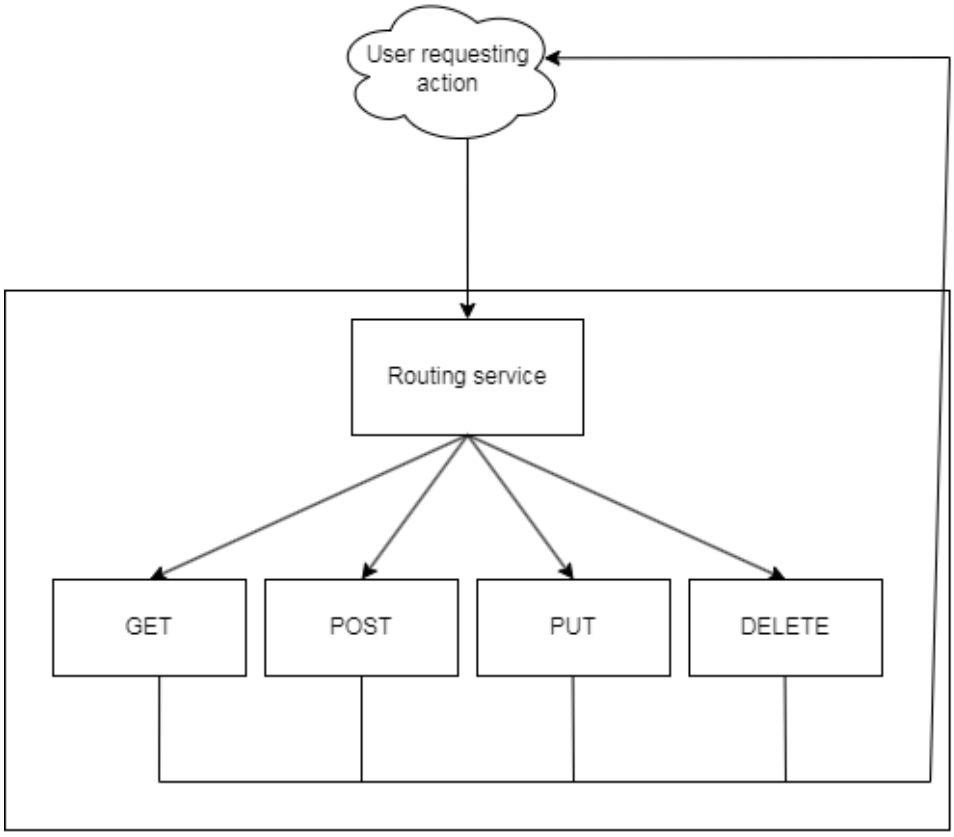

Figure 2.10: Struttura interna del server.

2.5 Frontend

La parte di frontend è stata progettata tenendo in considerazione il paradigma a componenti. Infatti ogni parte del sistema viene devisa in parti più piccole in modo da rendere riutilizzabile il codice.
In particolare viene creata una pagina principale che ha come scopo principale solo il login. Una volta effettuato il login, in base al tipo di utente, si può navigate in uno dei seguenti componenti:

- controllare le proprie navi che sono raggruppate in una lista (ogni elemento è un ulteriore componente)
- controllare la chat composta dalla lista di utenti e un componente che contiene la chat vera e propria.
- visuallizare la mappa
- vedere i dati dell'utente

Vengono anche creati 3 service:

- logger service: service per gestire il login dell'utete, non solo contiene i dati dell'utente, ma salva anche il suo token in modo da evitare login ripetuti in periodi brevi.

- backend service: service che contiene tutte le chiamate al backend
- chat service: service che gestisce la parte dedicata alla gestione dei messaggi per le chat e incapsula la gestione di socket.io.

3 Tecnologie

In questa sezione verranno descritte brevemente le tecnologie che sono state utilizzate per lo sviluppo del client e del server.

3.1 Stack di riferimento

Come stack di riferimento per lo sviluppo del progetto si è scelto MEAN, composto da MongoDB, Express, Angular e NodeJs.

MongoDB

MongoDB è un DBMS non relazionale, nel quale i record non sono salvati in tabelle strutturate, ma sono inseriti in delle aggregazioni, chiamate collezioni.

MongoDB rappresenta i dati salvati come oggetti JSON, caratteristica che garantisce una buona interoperabilità e facilità con server NodeJs.

Express

Express.js è un modulo utilizzato da server NodeJs per implementare un servizio di API RESTful.

AngularJs

AngularJs è un framework open source utilizzato per lo sviluppo di applicazioni web e interfacce utente.

NodeJs

NodeJs è una piattaforma software open source che permette di creare il proprio web server. È capace di garantire un grande numero di connessioni simultanee, con grande scalabilità ed efficienza, grazie alla sua architettura event-loop

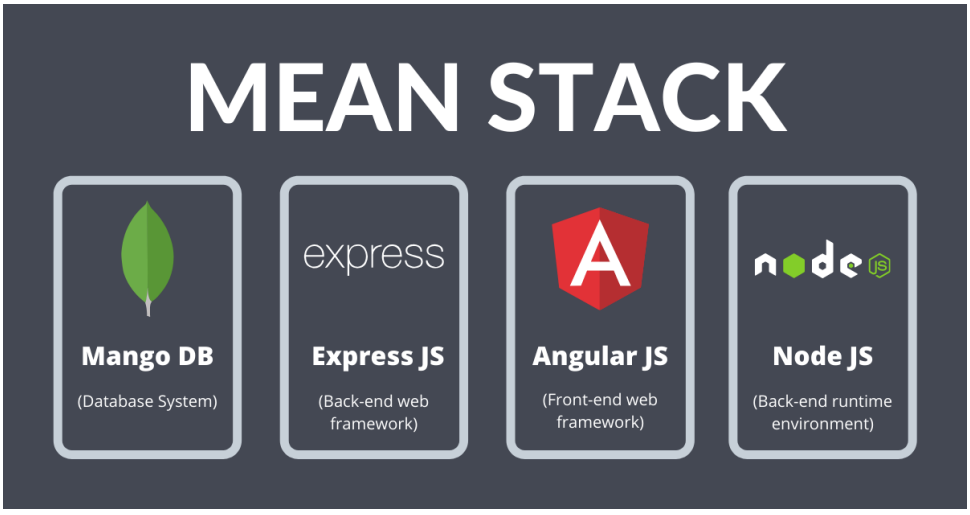

Figure 3.1: Stack di riferimento.

3.2 Docker

Per semplicare la distribuzione del sistema ogni sua parte (DB, backend e frontend) vengono incupsulati in un immagine docker al fine di avere problemi legati a versioni o installazioni.

3.3 Server

In seguito verranno elencate le tecnologie che sono state utilizzate per l'implementazione del server.

Jsonwebtoken

JSON WebToken è uno standard per la creazione di token di autenticazione e autorizzazione, che vengono inclusi nel payload delle richieste HTTP al server.

Bcrypt

Bcrypt è un modulo per la gestione delle password. È in grado di gestire tutte le funzionalità di encrypt e decrypt con una moltitudine di algoritmi. È in grado di generare hash sicuri per salvare le password nel database.

Mongoose

Mongoose è un modulo utilizzato per la modellazione degli oggetti e iterazione con MongoDB.

Socket.io

Socket.io è un modulo event-driven in grado di fornire comunicazione real-time tra client e server, a patto di mantenere una connessione persistente per tutta la durata della comunicazione.

La comunicazione avviene su un canale bidirezionale e basata su scambio di messaggi, tramite il protocollo Web Socket.

3.4 Client

In seguito verranno elencate le tecnologie che sono state utilizzate per l'implementazione del frontend.

Angular Material

Infrastruttura a camponenti basa su Material Design per Angular, material è uno delle stili più usati del frotend e nella creazione di applizaioni android e iOS.

Typescript

TypeScript è un linguaggio di programmazione fortemente tipizzata basato su JavaScript. Il codice Typescript deve essere validato, compilato e tradotto in Javascript.

3.5 SCSS (Sassy - CSS)

SCSS è una delle sintassi usate dal preprocessore CSS Sass ed è un superset di CSS: contiene tutte le funzionalità di CSS, ma è stato ampliato per includere anche le funzionalità di Sass, quindi ogni file CSS valido è allo stesso tempo un file SCSS valido.

Socket.io

Socket.io è un modulo event-driven in grado di fornire comunicazione real-time tra client e server, a patto di mantenere una connessione persistente per tutta la durata della comunicazione, in questa parte del progetto viene utilizzata la parte client del modulo.

3.5.1 Geoapify

Geoapify è una piattaforma per localizzaione e interazione con mappe, al pari di GoogleMaps. Viene preferita a quest'ultima poiché il piano gratuito è più ampio.

Leaflet

Leaflet è la pricipale libreria open-source JavaScript per mappe interattive mobile-friendly. Viene usata per connetersi a fare rendering dei dati ottenuti da Geoapify.

4 Codice

In seguito verranno mostati e argomentati alcuni dei frammenti di codice più importanti

4.1 Frontend

Lato frontend, come già detto, si creca di ridurre la quantità di logica di un singolo componente e di gestire il tutto tramite service e componenet.

4.1.1 Backend service

Questo service server per raggruppare le chiamate http al backend in modo da ridurre le operazioni in caso di cambio di dominio del backend.

```
        @Injectable({
  providedIn: 'root'
})
export class BackendService {

  private baseUrl: String = 'http://backend:3000';

  constructor(
    private http: HttpClient,
  ) { }

    public login(username: String, password: String) {
      let credential = {username: username, password: password};
      return this.http.post<user | undefined>(this.baseUrl +
      '/login/credentials/', credential)
        .pipe(
          catchError(this.handleError)
        );
    }

    ...

    private handleError(error: HttpErrorResponse) {
      if (error.status === 0) {
        console.error('An error occurred:', error.error);
      } else {
        console.error(
          `Backend returned code ${error.status}, body was: `,
          error.error);
      }
      return throwError(() => new Error('Something bad happened;
      please try again later.'));
    }
    }
```

Il pezzo di codice sopra riportato mostra il BackendService con una chiamata (quella per il login) ed anche la funzione che permette di gestire gli errori (che vengono mastrati usando i codice http).

4.1.2 Logger service

Questo service server gestire l'utente loggato ed per salvare il token nella memoria del browser.

```
@Injectable({
  providedIn: 'root'
})
export class LoggerService {

  constructor(private _router: Router) { }

  user: user | undefined = undefined;

  login(user: user) {
    this.user = user;
    localStorage.setItem('token', user.data.token);
    if(user.data.userType === 'user'){
      this._router.navigate(['/user/ship']);
    }else if(user.data.userType === 'controller'){
      this._router.navigate(['/watcher']);
    }
    console.log('logged in');
  }

  getToken() {
    return localStorage.getItem('token');
  }

  logout() {
    this.user = undefined;
    localStorage.removeItem('token');
    this._router.navigate(['/login']);
    console.log('logged out');
  }
}
```

Come mostrato il logger gestisce sia il salavataggio del token che il routing in base all'utente.

4.2 Backend

Per quanto riguarda il lato backend, si sono utilizzati il più possibile le funzionalità offerte dai singoli moduli e favorire le iterazioni con il DB tramite mongoose, rispetto a tenere dati in locale.

4.2.1 Mongoose

Mongoose fornisce soluzioni basate su schemi per la modellazione dei dati dell'applicazione da utilizzare su Mongodb, inoltre fornisce anche metodi con cui effettuare le richieste al database, dopo la sua connessione. Per ogni collection da memorizzare sul database, è stato creato un relativo schema a cui ci si attiene per la memorizzazione dei dati. In seguito si può vedere un esempio di schema utilizzato, in questo caso si riferisce allo schema per la memorizzazione dei documenti riguardanti le lezioni.

```
        exports.createUser = async function(credentials, res) {
        const pass = await passVerifier.encryptPassword(credentials.pas
        var userCreated = ""

        if(credentials.hasOwnProperty("userType")) {
                if(['user', 'controller'].includes(
            credentials.userType)) {
            userCreated = await User.create({username : credentials.use
            password: pass.hash_pass,
            salt: pass.salt,
            token : pass.token,
            userType: credentials.userType});
                }
                else {
                        res.status(400).send({
                message: "User type can only be user or controller"
                        });
                        return;
                }
        }
        else {
                userCreated = await User.create({username : credentials
                password: pass.hash_pass,
                salt: pass.salt,
                token : pass.token,
                userType: credentials.userType});
        }

        const user = credentials.username
        const token = jwt.sign({ user }, jwtKey, {
        algorithm: "HS256",
        expiresIn: jwtExpirySeconds})

        res.cookie("token", token, { maxAge: jwtExpirySeconds * 1000 })

        return {status: "success", message: "User successfully created"
}
```

Come mostrato dalla query per la creazione dell'utente, viene prima verificato che l'utente abbia le proprietà essenziali e, una volta che viene inserito l'utente con le credenziali,

esegue la query.

Inoltre, prima di procede con l'esecuzione delle query, il server esegue un check se i dati passati dall'utente sono conformi con i dati richiesti dalla query. Per questo, i dati passati sono verificati tramite l'utilizzo di JSON schema predefiniti che ne verificano i requisiti per eseguire la query.

```
    router.post("/credentials", async function(req, res) {
    const credentials = req.body;
    if (!utils.matches(credentials, models.loginCredentials())) {
        res.send('Request body is invalid. Provide an username and pass
        return;
    }
    //if no user has been found, return error message
    const response = await db.loginViaCredentials(credentials);
    if(!response.found) {
        res.status(401).send('Wrong credentials')
        return;
    }
    res.json(response.payload)
});
```

4.2.2 Bcrypt

Bcrypt fornisce implementazioni per tutti i principali tipi di cifrari. Nel progetto è stato utilizzato quando l'utente ha la necessità di registrarsi/fare il login.

```
    exports.encryptPassword = async function(password) {
    const salt = crypto.randomBytes(saltLenght).toString('hex')
    //divide the password in two parts
    var middle = Math.floor(password.length / 2);

    //password is mixed with the salt
    const newPass = salt + password.substr(0, middle) + salt +
        password.substr(middle) + salt

    var hash = crypto.createHash('sha512')
                .update(newPass).digest('hex');
    for(var i = 0; i < hashIteration-1; i++) {
        hash = crypto.createHash('sha512')
                    .update(hash).digest('hex');
    }
    return {hash_pass: hash, salt: salt,
            token: await exports.generateToken()}
};
```

Come si può vedere, nell'implementazione adottata sono stati utilizzati anche parametri che permettono di modificare la lunghezza dell'hash o il numero di iterazioni per ottenere l'hash. Questo per aumentare la sicurezza delle password.

5 Test

Per verificare il corretto funzionamento del server, le chiamate al server sono state fatte e testate più volte tramite il tool Postman.

Questo tool ha permesso di verificare, in modo agile, tutte le possibili casistiche di chiamate ai vari servizi del server. Ciò ha garantito che i metodi forniti dal server fossero coerenti e funzionanti in tutte le casistiche possibili e ha facilitato lo sviluppo del frontend.

5.1 Test con utenti

Oltre ad aver effettuato test tramite tool per verificarne il corretto funzionamento, sono stati effettuati test dell'applicazione anche con utenti finali, per verificare che l'effettivo design applicato sia conforme con i requisiti stabiliti ad inizio progetto.

Al termine dello sviluppo, è stato richiesto di rispondere ad un questionario basato sulle direttive consigliate dall'User Experience Questionnaire. Esso è un quesitonario affidabile e veloce per far valutare l'esperienza dell'utente all'intero di un sito.

Il testing finale è stato sottoposto a 5 utenti diversi.

5.1.1 Risultati

I risultati sono riportati su una scala da 0 a 10:

- 0: Per niente;
- 10: Semplice.

Collegarsi al sito

1. È stato facile collegarsi al sito?
   1 persona su 5 ha votato 7;
   4 persone su 5 hanno votato 8.

2. Una volta collegati al sito, è stato facile navigarci dentro?
   1 persona su 5 ha votato 7;
   3 persone su 5 hanno votato 8;
   1 persona su 5 ha votato 9;

Registrazione

1. È stato facile registrarsi al sito?
   2 persone su 5 ha votato 8;
   3 persone su 5 hanno votato 9.

Utilizzo

1. Usufruirai del sito come comandante o visitatore?
   1 persona su 5 ha votato comandante;
   4 persone su 5 hanno votato visitatore.

Utilizzo

In questa sezione si parlerà di utilizzo dopo che si è effettuata la registrazione dell'utente.

1. È risultato semplice navigare nella pagina e trovare le varie informazioni?
   1 persona su 5 ha votato 7;
   4 persone su 5 hanno votato 8.

2. È stato semplice registrare una nave?
   5 persone su 5 hanno votato 8.

3. È stato semplice utilizzare la chat?
   1 persona su 5 ha votato 7;
   1 persona su 5 ha votato 8;
   3 persone su 5 hanno votato 9.

Risultati finali

1. Come si valuta l'utilizzabilità complessiva del sito? (0 difficile / 10 facile)
   5 persone su 5 hanno votato 8.

6 Conclusioni

I test eseguiti hanno mostrato un corretto funzionamento di tutte le principali attivit‘a offerte dall'applicazione.

Lo stack framework utilizzato ha permesso di scrivere codice conciso e in maniera piuttosto veloce, senza troppe ripetizioni.

6.1 Sviluppi futuri

Di seguito sono riportati dei possibili sviluppi futuri dell'applicazione per migliorarne il contenuto/funzionalità:

- Verifica della nave post registrazione, per evitare registrazioni inutili;
- Login tramite token;
- Aggiunta di nuovi stati/descrizione in caso di allarme della nave;
- Aggiungere date / tempistiche di inizio e fine viaggio nella rotta selezionata.