

# Relazione progetto “Battaglia navale” materia “Tecnologie e applicazioni web”

<b>1 - Tema</b>	<b>3</b>
<b>2 - Architettura del sistema</b>	<b>3</b>
<b>3 - Modello dati utilizzato</b>	<b>4</b>
<b>4 - API server web</b>	<b>5</b>
<b>5 - Autenticazione degli utenti</b>	<b>11</b>
<b>6 - Client web Angular</b>	<b>12</b>
6.1 Component	12
6.2 Servizi	13
6.3 Routes	13
<b>7 - Esempi workflow applicazione</b>	<b>14</b>
7.1 Login	14
7.2 Registrazione	15
7.3 Modifica utente e cancellazione	16
7.4 Admin assegna/toglie ruolo amministratore	17
7.5 Creazione chat e invio messaggi tra utenti	18
7.6 Visualizzazione scoreboard	20
7.7 Creazione/unione a una partita	21
7.8 Posizionamento delle navi	22
7.9 Sparo	23

# 1 - Tema

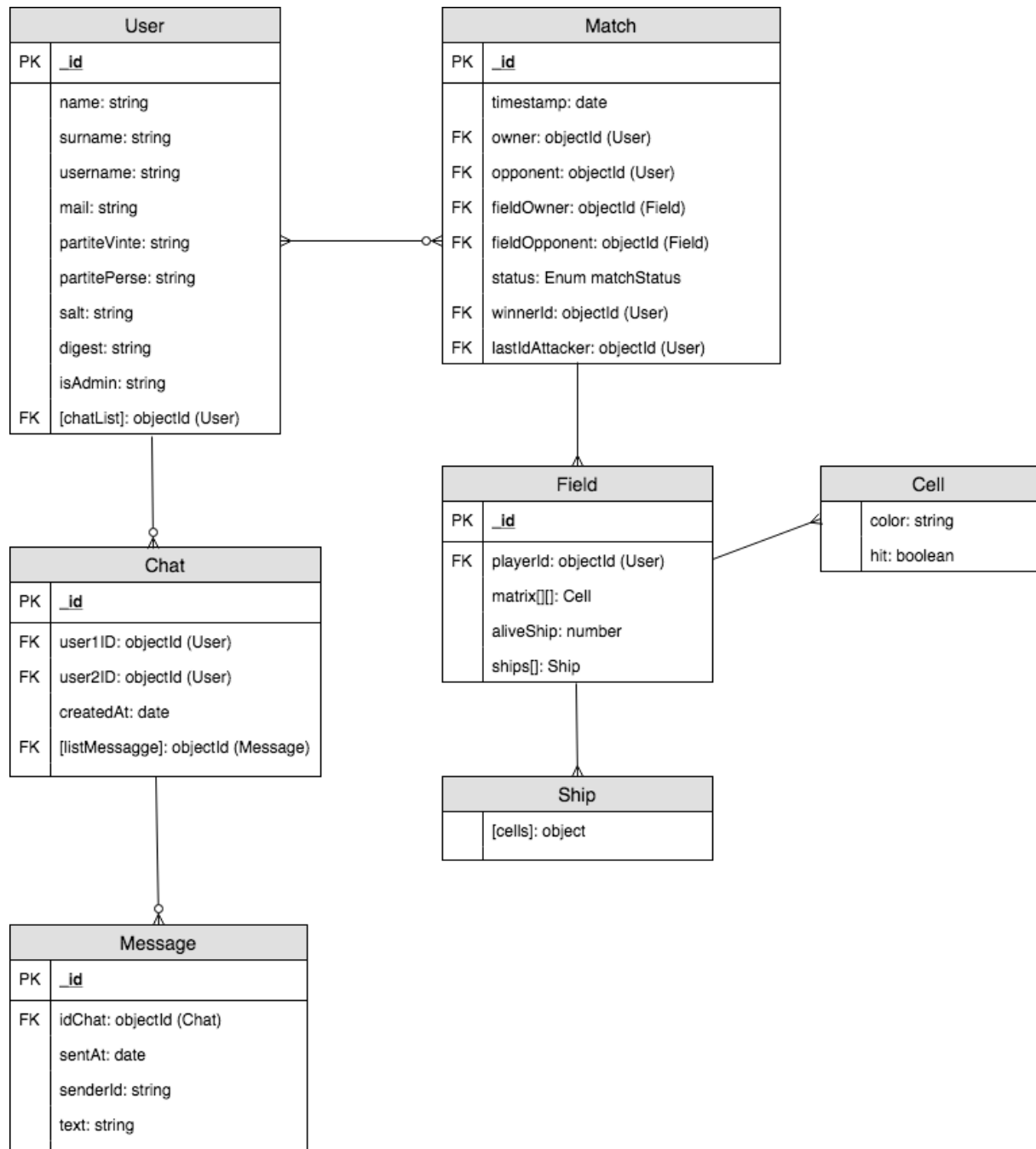
Si realizzi un'applicazione web, comprensiva di server con API stile REST e front-end di tipo SPA, che permetta ad una community di utenti di giocare al gioco della battaglia navale. Il sistema deve permettere la gestione degli utenti (registrazione, login ed eventuale cancellazione), la ricerca di giocatori disponibili ad iniziare una partita, la logica di gioco e una classifica generale degli utenti in base al numero di partite vinte, perse e giocate.

## 2 - Architettura del sistema

L'architettura del sistema segue il "paradigma" MEAN: MongoDB, Express, Angular e Nodejs. Il backend è interamente realizzato utilizzando Nodejs insieme ad Express e MongoDB, grazie alla combinazione di questi framework è possibile realizzare delle API di tipo REST, nel dettaglio le funzionalità richieste sono state soddisfatte in questo modo:

- Nodejs offre un framework per gestire il server web utilizzando un modello a I/O non bloccante e ad eventi, ciò significa che quando verrà interrogato in determinati endpoint, chiederà le risorse al db e le restituirà quando saranno pronte, nel frattempo potrà ricevere e gestire altre richieste.
- Express offre la possibilità di gestire gli endpoint e i metodi con i quali vengono chiamati in modo semplice e chiaro. Di conseguenza express sarà la componente che riceverà le chiamate negli endpoint da parte dei client e farà il routing agli endpoint corretti nel server.
- MongoDB offre un database NoSQL basato sul concetto di collezioni e documenti. Esso può contenere più collezioni, ogni collezione può contenere altre collezioni e documenti. I documenti rappresentano il dato vero e proprio, possono essere disomogenei in una collezione e non sono legati a tabelle come i DB SQL. MongoDB verrà interrogato attraverso il framework mongoose che tramite la definizione degli schema che rappresentano i documenti nel database, permette una gestione delle query semplice e potente basata su eventi. La scelta di utilizzare MongoDB con mongoose calza a pennello, visto che il meccanismo con il quale inserisce i dati e ritorna le risposte delle query è basato tutto su dati in formato JSON.
- Angular è un framework per creare pagine frontend di tipo Single Page Application, ciò significa che tutte le pagine con il quale l'utente interagisce e tutto il codice javascript è 'incorporato' in una singola pagina, questo permette una navigazione tra pagine senza bisogno di caricamenti perchè tutti gli scambi dati tra il client e il server avvengono in modo asincrono e angular gestirà in modo appropriato cosa mandare, come interpretare ciò che ha ricevuto e cosa domandare al server quando riceve l'input dall'utente.

### 3 - Modello dati utilizzato



## 4 - API server web

Nota: i formati di dati scambiati tra il client e il server sono di tipo JSON

Endpoint	Metodo	Attributi
/	GET	-
<b>Descrizione</b>	Ritorna un json contenente la lista degli endpoint principali: { api_version: "1.0", endpoints: ["/users", "/chats", "/scoreboard", "/matches"] }	

Endpoint	Metodo	Attributi
/renew	GET	-
<b>Descrizione</b>	Rinnova il JWT dell'utente autenticato, se tutto procede senza errori viene ritornato anche il token: { error: false, errormessage: "", token: token_renew } altrimenti un messaggio di errore: { error: true, errormessage: "Remindme not setted, please do the login again" }	

Endpoint	Metodo	Attributi
/login	GET	-
<b>Descrizione</b>	Il server riceve una richiesta di autenticazione di tipo Basic Auth contenente le informazioni dell'utente che prova ad autenticarsi e se sono corrette ritorna un token JWT { error: false, errormessage: "", token: tokensigned }	

Endpoint	Metodo	Attributi
/users/:username	GET	-
<b>Descrizione</b>	Il server ritorna un JSON contenente le informazioni dell'utente :username, altrimenti un messaggio di errore.	

Endpoint	Metodo	Attributi
/users/:username	PUT	-
<b>Descrizione</b>	<p>Il server modifica le informazioni dell'utente :username in base al JSON passato nel body della request. Un utente admin può mettere o togliere lo status admin a altri utenti. Altrimenti un utente normale può modificare tutte le sue informazioni.</p> <p>Se un utente non admin o un utente diverso da :username prova a modificare l'utente :username viene ritornato un errore:  { statusCode: 404, error: true, errorMessage: "Unauthorized" }</p> <p>altrimenti se tutto va a buon fine viene ritornato un JSON contenente:  { error: false, errorMessage: "" }</p>	

Endpoint	Metodo	Attributi
/users/:username	DELETE	-
<b>Descrizione</b>	<p>Il server elimina l'utente :username, questo è possibile solamente se l'utente che effettua l'azione è un admin o l'utente stesso. Se non è uno dei 2 viene ritornato un errore:  { statusCode: 404, error: true, errorMessage: "Unauthorized" },</p> <p>altrimenti se tutto va a buon fine viene ritornato un JSON contenente:  { error: false, errorMessage: "" }</p>	

Endpoint	Metodo	Attributi
/users/:username/matches	GET	-
<b>Descrizione</b>	Viene ritornata la lista dei match dove :username ha giocato	

Endpoint	Metodo	Attributi
/users	POST	-
<b>Descrizione</b>	<p>Registra un nuovo utente in base ai dati che il client passa nel body della request:  { mail: "", name: "", surname: "", username: "", password: "" }</p> <p>se vi sono errori viene ritornato un JSON:  { statusCode: 404, error: true, errorMessage: "MongoDB error: " + error },</p> <p>altrimenti se tutto va a buon fine viene ritornato un JSON contenente:  { error: false, errorMessage: "" }</p>	

Endpoint	Metodo	Attributi
/users	GET	keysearched
<b>Descrizione</b>	<p>Il server interroga la lista degli utenti trovando gli utenti che hanno in uno dei campi: username, email, nome, cognome la parola passata nella query keysearched.</p> <p>Se la query keysearched è vuota, ritorna tutta la lista degli utenti.</p>	

Endpoint	Metodo	Attributi
/chats	GET	-
<b>Descrizione</b>	<p>Ritorna un JSON contenente la lista di tutte le chat dell'utente autenticato.</p>	

Endpoint	Metodo	Attributi
/chats	POST	-
<b>Descrizione</b>	<p>Il client invia nel body della richiesta un JSON con il formato:  <pre>{ 'destinatario': username }</pre> e il server crea la chat tra l'utente autenticato e il destinatario.  Se non vi sono errori ritorna un dato così formato:  <pre>{ error: false, errorMessage: "", id: new_chat._id }</pre> contenente l'id della chat appena creata. Se la chat esiste già tra i due utenti ritorna l'errore:  <pre>{ statusCode: 404, error: true, errorMessage: "Chat already exists" }</pre> </p>	

Endpoint	Metodo	Attributi
/chats/:id	GET	-
<b>Descrizione</b>	<p>Ritorna all'utente la lista dei messaggi contenuti nella chat in formato JSON. Se un utente fa la richiesta e non è uno dei 2 partecipanti della chat viene ritornato l'errore:  <pre>{ statusCode: 500, error: true, errorMessage: "User not allowed to see this chat" }</pre> </p>	

Endpoint	Metodo	Attributi
/chats/:id	POST	-
<b>Descrizione</b>	<p>Il client invia nel body della richiesta un JSON con il formato:  <pre>{ 'sentAt': date, 'text': text }</pre> per aggiungere un messaggio alla chat con id :id </p>	

Endpoint	Metodo	Attributi
/chats/:id	DELETE	-
<b>Descrizione</b>	<p>Uno dei 2 partecipanti della chat richiede al server di eliminare la chat :id, se questa viene eliminata con successo viene ritornato un messaggio:</p> <pre>{ error: false, errormessage: "" }</pre> <p>altrimenti:</p> <pre>{ statusCode: 404, error: true, errormessage: "MongoDB error: " + error }</pre>	

Endpoint	Metodo	Attributi
/scoreboard	GET	limit & type
<b>Descrizione</b>	<p>Il server ritorna una lista degli utenti in formato JSON per visualizzare la scoreboard, accetta una limitazione in formato Number (limit) e 3 tipologie di scoreboard in ordine decrescente:</p> <ul style="list-style-type: none"> <li>- total: totale delle partite</li> <li>- partiteVinte: in base al numero di partite vinte</li> <li>- partitePerse: in base al numero di partite perse</li> </ul>	

Endpoint	Metodo	Attributi
/matches	GET	-
<b>Descrizione</b>	Ritorna un JSON contenente la lista dei match nello stato di attesa	

Endpoint	Metodo	Attributi
/matches	POST	-
<b>Descrizione</b>	<p>Crea un nuovo match per l'utente autenticato, prima effettua un controllo per vedere se ha già un match in attesa o sta già giocando, se è così ritorna un errore:</p> <pre>{ error: true, errormessage: "User already got waiting match" }.</pre> <p>Alternativamente ritorna un messaggio di successo contenente l'id del match appena creato:</p> <pre>{ error: false, errormessage: "", id: data._id }</pre>	



Endpoint	Metodo	Attributi
/matches/:id/board	PUT	-
<b>Descrizione</b>	<p>Inserisce nel campo di gioco dell'utente autenticato la lista di navi che vengono passate nel body della richiesta con il formato (esempio):</p> <pre>{   "ships": [     [{"x": 9, "y": 0}, {"x": 9, "y": 1}],     [{"x": 6, "y": 3}, {"x": 6, "y": 4}],     [{"x": 9, "y": 4}, {"x": 9, "y": 5}],     [{"x": 8, "y": 7}, {"x": 8, "y": 8}],     [{"x": 2, "y": 0}, {"x": 2, "y": 1}, {"x": 2, "y": 2}],     [{"x": 0, "y": 0}, {"x": 0, "y": 1}, {"x": 0, "y": 2}],     [{"x": 1, "y": 6}, {"x": 1, "y": 7}, {"x": 1, "y": 8}, {"x": 1, "y": 9}],     [{"x": 4, "y": 6}, {"x": 4, "y": 7}, {"x": 4, "y": 8}, {"x": 4, "y": 9}],     [{"x": 4, "y": 0}, {"x": 4, "y": 1}, {"x": 4, "y": 2}, {"x": 4, "y": 3}, {"x": 4, "y": 4}]   ] }</pre> <p>Se tutto va bene ritorna un messaggio:  <pre>{ error: false, errorMessage: "" }</pre> e notifica i client in ascolto nel socket 'broadcast ' + data._id dove _id è l'id del match.</p> <p>Se il match non è nello stato di building ritorna un errore:  <pre>{ error: true, errorMessage: "Cannot add ships now!" }</pre></p> <p>Se le navi non hanno un formato o un posizionamento corretto ritorna un errore:  <pre>{ error: true, errorMessage: "Invalid ship positioning " + e }</pre></p>	

Endpoint	Metodo	Attributi
/matches/:id_match	GET	type -> se viene valorizzato, vengono popolate le collections che sono riferite dentro al match
<b>Descrizione</b>	Restituisce i dati del :id_match in formato JSON	

Endpoint	Metodo	Attributi
/matches/:id_match/join	PUT	-
<b>Descrizione</b>	<p>L'utente esegue il join alla partita :id_match, se il match non è in stato di attesa viene ritornato l'errore:  <pre>{ error: true, errorMessage: "Match already setted up" }</pre>, se l'utente sta già giocando in un altro match viene ritornato l'errore:  <pre>{ error: true, errorMessage: "User already fighting in a different match." }</pre>.</p> <p>Altrimenti se tutto va a buon fine viene ritornato il messaggio:  <pre>{ error: false, errorMessage: "" }</pre></p>	

Endpoint	Metodo	Attributi
/matches/:id_match	PUT	-
<b>Descrizione</b>	<p>Endpoint utilizzato per sparare in una partita, il client invia nel body delle coordinate nel seguente formato:          { "position" : { "x" : Number, "y" : Number } }</p> <p>Se ha già attaccato e prova a attaccare di nuovo viene ritornato l'errore:          { error: true, errorMessage: "This user already attacked!" }.</p> <p>Se non vi sono stati errori vi possono essere 2 possibilità:</p> <ul style="list-style-type: none"> <li>- L'attacco non ha concluso la partita, allora viene ritornato un messaggio così formato: { error: false, errorMessage: "", message: "Cella colpita correttamente", attacker: req.user.id }</li> <li>- Se l'attacco ha concluso la partita allora viene ritornato un messaggio così formato: { error: false, errorMessage: "", message: "ha vinto il player " + userret.username, winner: userret.username }</li> </ul>	

Tutti gli endpoint a parte il login e la registrazione chiamano una funzione che fa da middleware per controllare l'autenticità del JWT passato dal client.

Gli endpoint se riscontrano degli errori chiamano la funzione per gestire gli errori, che ritorna al client un messaggio:

```
app.use(function (err, req, res, next) {
  console.log("Error middleware endpoint: " + err + JSON.stringify(err));
  res.status(err.statusCode || 500).json(err);
})
```

Infine se l'utente prova a chiamare un endpoint non valido, viene chiamato in causa l'endpoint per restituire un errore 404, pagina non trovata:

```
app.use((req, res, next) => {
  res.status(404).json({ error: true, errorMessage: "Invalid endpoint" });
})
```

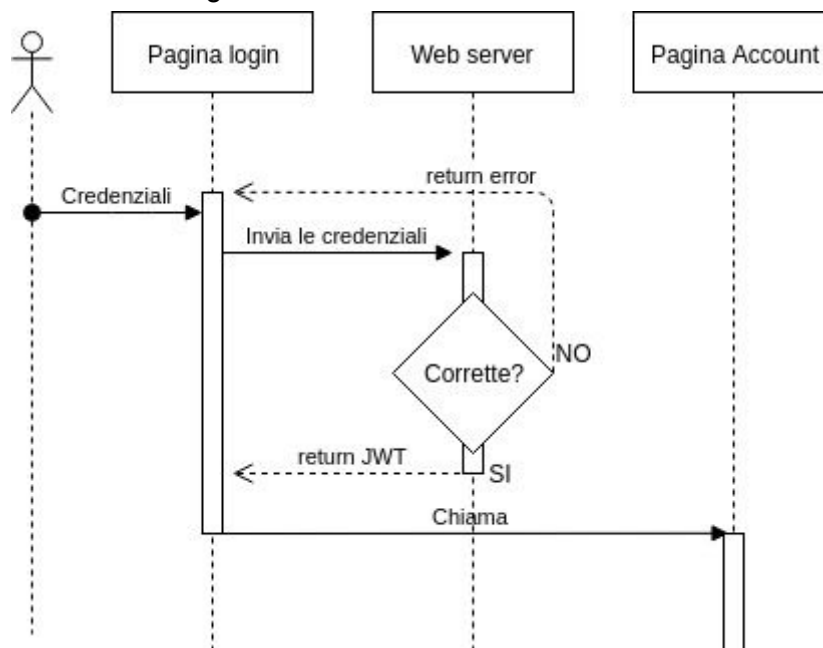
## 5 - Autenticazione degli utenti

Una volta che l'utente crea il suo account nel database nel server non viene salvata la password intera e cifrata, ma solamente un suo digest e un salt cifrati. Quando un utente prova a effettuare un login il server web controlla se il digest e il salt della password inviata corrispondono al digest e al salt salvati nel database corrispondenti al nome utente inserito, possono succedere 2 scenari:

- I dati sono errati, il server web ritorna un messaggio di errore nella risposta al client
- I dati sono corretti, il server web costruisce un JWT contenente le informazioni dell'utente e viene cifrato attraverso un segreto che solamente il server web conosce, infine viene ritornato uno statuscode 200 e il JWT. Questo JWT verrà inviato ogni volta dal client quando effettuerà le richieste al server, visto che nel server web per ogni endpoint, ad eccezione del login e register, è presente un middleware che controlla che l'utente sia autenticato

Per motivi di semplicità nel costruire l'applicazione non è stato utilizzato il protocollo HTTPS, ma in uno scenario 'reale' dove l'applicazione gira nel web, l'utilizzo di questo protocollo è d'obbligo per poter cifrare i dati scambiati dal client e dal server.

Workflow del login



## 6 - Client web Angular

### 6.1 Component

Il server web angular è composto da diversi component, ogni component ha lo scopo di visualizzare i dati e ricevere gli input dall'utente.

- **Index.html:** componente che ha lo scopo di racchiudere la struttura principale di una pagina HTML, contiene tutte le informazioni riguardanti i tag <head> e <body>, contiene inoltre i collegamenti ai file javascript esterni, implementa l'AppComponent.
- **App-Component:** componente principale, questo componente ha lo scopo di "linkare" la struttura della pagina web NavBar e contenuto principale. Inoltre contiene anche un modal che viene triggerato quando si riceve un messaggio.
- **NavBar-Component:** visualizza la navbar del sito, è direttamente collegato al servizio che gestisce l'utente, in base al fatto se l'utente ha effettuato o meno l'accesso, visualizza un menù differente.
- **UserLogin-Component:** componente il cui scopo è visualizzare un form per eseguire il login e un pulsante per rimandare alla pagina di registrazione.
- **UserSignup-Component:** componente il cui scopo è visualizzare un form per registrare un nuovo utente.
- **UserInfo-Component:** componente il cui scopo è visualizzare e poter modificare i dati dell'utente, se un utente accede al proprio profilo, verrà reindirizzato in '/user', altrimenti se visualizza un profilo di un altro utente verrà reindirizzato in '/user/:username'.
- **Scoreboard-Component:** componente il cui scopo è visualizzare la scoreboard in base a dei parametri che l'utente decide di usare applicare.
- **Players-Component:** componente il cui scopo è visualizzare la lista di tutti i giocatori presenti nella piattaforma, dispone anche di un form di ricerca che permette all'utente autenticato di scrivere dei dati per cercare i giocatori.
- **NotFound-Component:** componente il cui scopo è visualizzare una pagina non trovata se l'utente inserisce un url errato.
- **MatchBuilder-Component:** componente il cui scopo è visualizzare la schermata per costruire il match, visualizza una griglia e diverse navi e l'utente può spostarle grazie al "drag & drop"
- **Match-Component:** componente il cui scopo è visualizzare la schermata della partita in corso e permettere di giocare. Visualizza il campo dell'avversario e del giocatore e se l'utente clicca una cella del campo dell'avversario viene sparato il colpo.
- **List-Matches-Component:** componente il cui scopo è visualizzare la lista dei match in attesa per permettere a un utente di entrare in una partita, inoltre permette la creazione di una nuova partita se il giocatore che fa richiesta non è già in gioco in un'altra partita o ha una partita in attesa.
- **List-Chat-Components:** componente il cui scopo è visualizzare la lista delle chat del giocatore, permette inoltre l'eliminazione delle chat.

- **Chat-Component:** componente il cui scopo è visualizzare la lista dei messaggi di una chat e poterne scrivere di nuovi.

## 6.2 Servizi

Sono stati implementati diversi servizi nel client angular:

- **Utilities-Service:** servizio il cui scopo è fornire dei metodi di utilità per altri servizi, quali la possibilità di creare le opzioni di connessione (header e parametri) in modo rapido, oppure controllare se un utente è autenticato o meno in base a un token che viene passato.
- **User-Service:** servizio il cui scopo è interfacciarsi col server per la completa gestione degli utenti. Permette il login, la modifica, la registrazione, la ricerca, la cancellazione e tutta la gestione delle chat dell'utente loggato. Inoltre espone dei metodi per ritornare i dati contenuti nel JWT.
- **SocketIO-Service:** servizio il cui scopo è fornire il metodo 'connect' per connettersi tramite un socket.io-client al server e poter ricevere informazioni in tempo reale in base ai canali dove rimane in ascolto senza dover fare richieste asincrone.
- **Match-Service:** servizio il cui scopo è gestire tutte le informazioni del match, si interfaccia col server per richiedere informazioni riguardo un match specifico, la lista dei match in attesa, di creare e fare join nei match. Inoltre comunica al server anche la logica del gioco mandando la posizione delle navi e le coordinate dei colpi sparati.

## 6.3 Routes

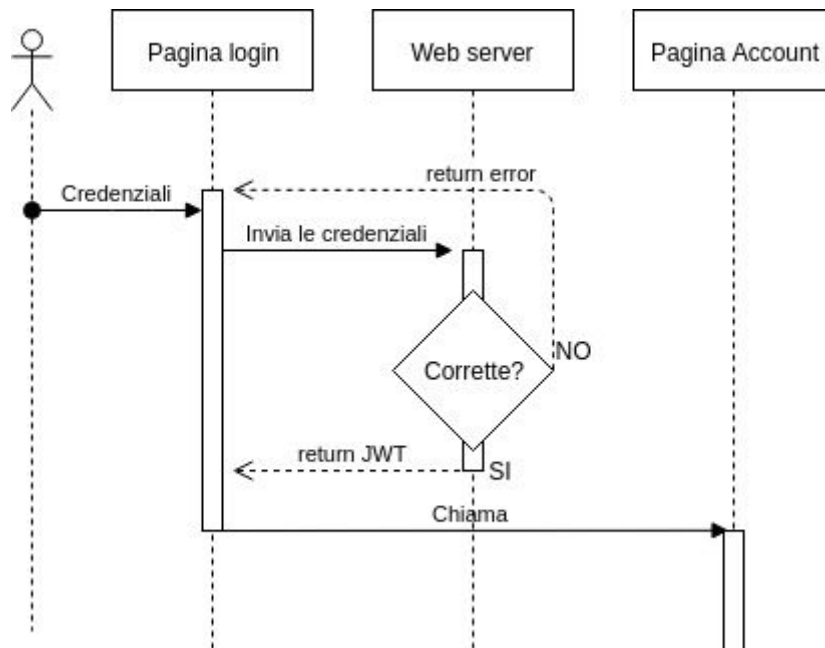
Nel client angular è stato implementato il modulo per gestire il routing delle pagine per poter richiamare i corretti components.

Nel dettaglio, sono state implementate queste routes:

```
const routes: Routes = [
  { path: '', redirectTo: '/login', pathMatch: 'full' },
  { path: 'login', component: UserLoginComponent },
  { path: 'user', component: UserInfoComponent },
  { path: 'user/:username', component: UserInfoComponent },
  { path: 'user/:username/matches', component: ListMatchesComponent },
  { path: 'signup', component: UserSignupComponent },
  { path: 'scoreboard', component: ScoreboardComponent },
  { path: 'players', component: PlayersComponent },
  { path: 'chats', component: ListChatsComponent },
  { path: 'chats/:id', component: ChatComponent },
  { path: 'match', component: ListMatchesComponent },
  { path: 'match/:id/board', component: MatchBuilderComponent },
  { path: 'match/:id', component: MatchComponent },
  { path: '**', component: NotfoundComponent }
];
```

## 7 - Esempi workflow applicazione

### 7.1 Login



L'utente apre la homepage, se non è autenticato si apre la pagina di login, immette le credenziali e, se corrette, il web server ritorna un JWT contenente le informazioni dell'utente, altrimenti ritorna l'errore riscontrato. Una volta che l'utente esegue il login in modo corretto viene automaticamente reindirizzato alla pagina del proprio account.

#### Immissione delle credenziali

Battleship

Log-in Registrati

## Log-in

Inserisci le tue credenziali per accedere

Username

username

Password

password

☒ Ricordami

Login

Registrati

## Pagina del profilo

Battleship Account Partite Giocatori Chats Scoreboard Logout

### Il tuo profilo

Username	<input type="text" value="admin"/>
Nome	<input type="text" value="admin"/>
Cognome	<input type="text" value="admin"/>
Email	<input type="text" value="admin@battleship.it"/>
<input type="text" value="Nuova password"/>	<input type="text" value="Ripeti nuova password"/>

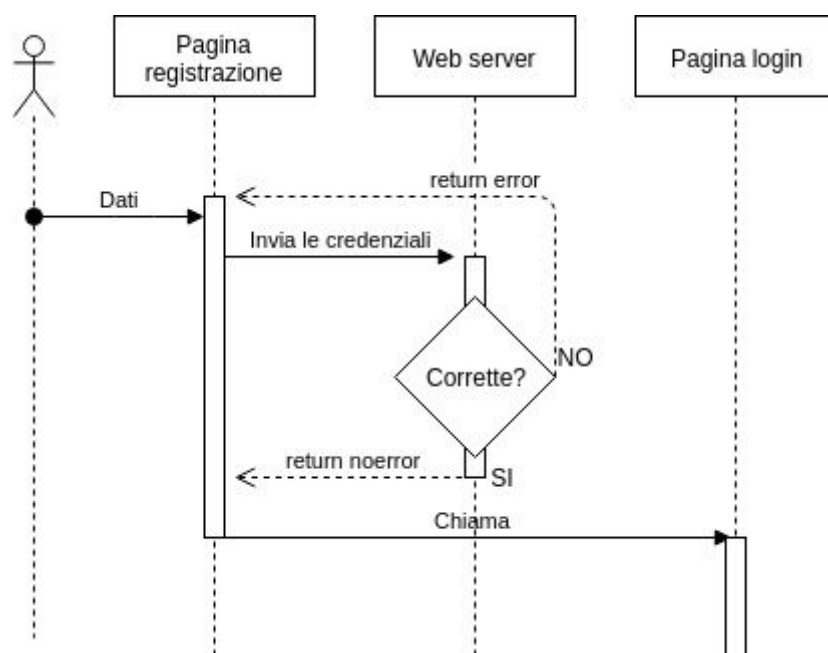
☒ Amministratore

[Partite giocate](#)

[Aggiorna le informazioni](#)

[Elimina l'utente](#)

## 7.2 Registrazione



L'utente non autenticato entra nella pagina di registrazione e immette i propri dati, questi vengono inviati al web server che controlla che non esista già un utente con la email o il nome utente immesso in fase di registrazione. Se accadono conflitti l'utente viene avvisato, altrimenti viene reindirizzato alla pagina di login.

Immissione dei dati per creare un account.

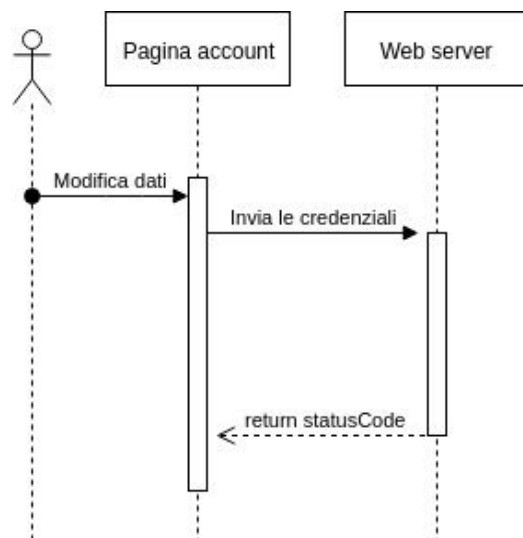
Battleship Log-in Registrati

## Registrazione

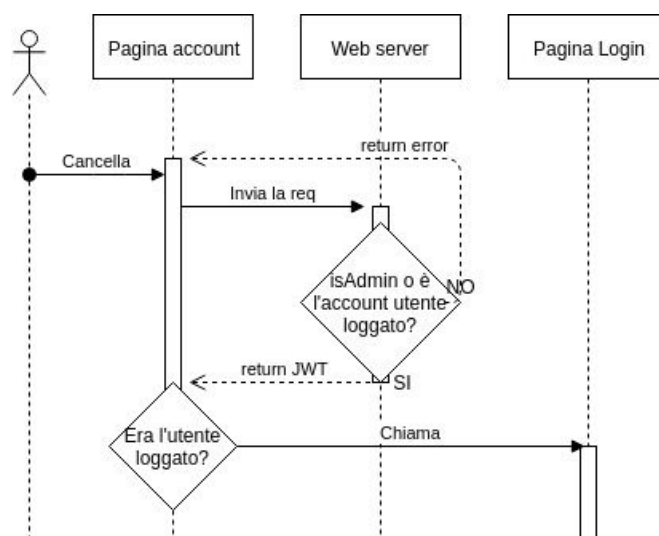
Nome	Cognome
<input type="text" value="enrico"/>	<input type="text" value="vettori"/>
Username	Indirizzo Email
<input type="text" value="enricov"/>	<input type="text" value="a@a.it"/>
Password	Password
<input type="password" value="****"/>	<input type="password" value="****"/>
<input type="button" value="Crea l'account"/>	

## 7.3 Modifica utente e cancellazione

### Modifica



### Cancellazione





Un utente autenticato tramite la propria pagina persona può decidere in qualsiasi momento di cambiare tutti i suoi dati oppure di eliminare il proprio account. Invece un utente amministratore può decidere di eliminare qualsiasi account.

### Modifica o cancellazione dell'account

Battleship

AccountPartiteGiocatoriChatsScoreboardLogout

## Il tuo profilo

Username

Nome

Cognome

Email

Nuova password

☒ Amministratore

Partite giocate

Aggiorna le informazioni

Elimina l'utente


admin

admin

admin

admin@battleship.it

Ripeti nuova password



## 7.4 Admin assegna/toglie ruolo amministratore

Un utente amministratore può decidere di rendere amministratore un altro utente o revocare il ruolo.

### Rimozione/assegnamento del ruolo di amministratore

Battleship

AccountPartiteGiocatoriChatsScoreboardLogout

## Il tuo profilo

Username

Nome

Cognome

Email

Nuova password

☒ Amministratore

Partite giocate

Aggiorna le informazioni

Elimina l'utente


admin

admin

admin

admin@battleship.it

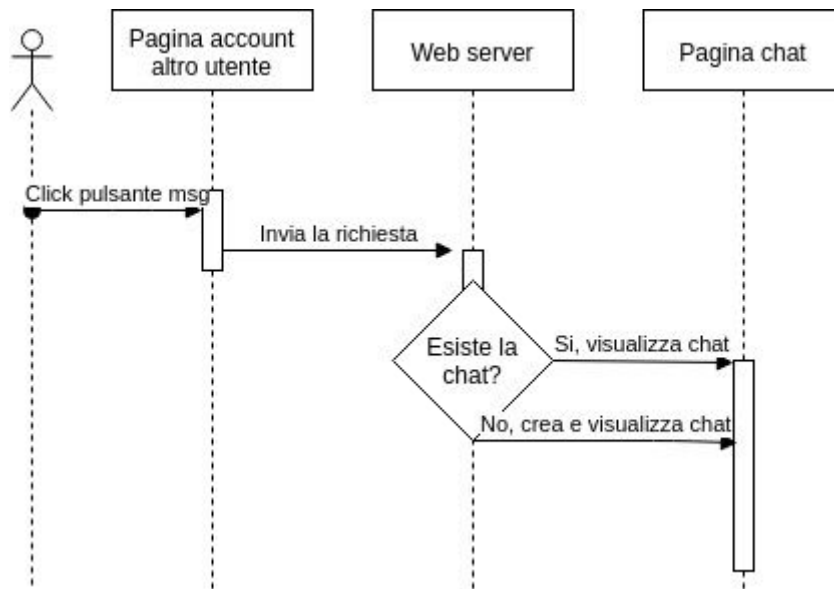
Ripeti nuova password



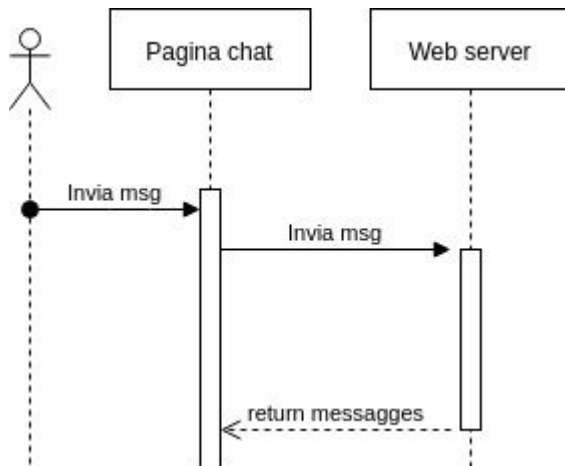
## 7.5 Creazione chat e invio messaggi tra utenti

All'interno dell'applicazione è stata implementata la possibilità di scambiare messaggi tra utenti. Per iniziare è necessario che un utente entri nella scheda di un altro utente e clicchi il pulsante 'invia messaggio', successivamente verrà creata la chat tra i due utenti. E' possibile visualizzare la lista di chat attive nell'apposita schermata.

### Creazione chat



### Invio messaggio



## Creazione di una chat

Battleship Account Partite Giocatori Chats Scoreboard Logout

### Stai visualizzando il profilo di prova3

Username	prova3
Nome	prova3
Cognome	prova3
Email	prova3@battleship.it
Partite vinte	0
Partite perse	0
Partite giocate	0

☐ Amministratore

Partite giocate

Aggiorna le informazioni

Elimina l'utente

Invia un messaggio



## Visualizzazione lista delle chat

Battleship Account Partite Giocatori Chats Scoreboard Logout

### Le tue chat

[Clicca per visualizzare la tra l'utente admin e l'utente prova3](#) [Cancella la chat](#)

[Clicca per visualizzare la tra l'utente prova2 e l'utente admin](#) [Cancella la chat](#)

## Visualizzazione e invio dei messaggi in una chat

Battleship Account Partite Giocatori Chats Scoreboard Logout

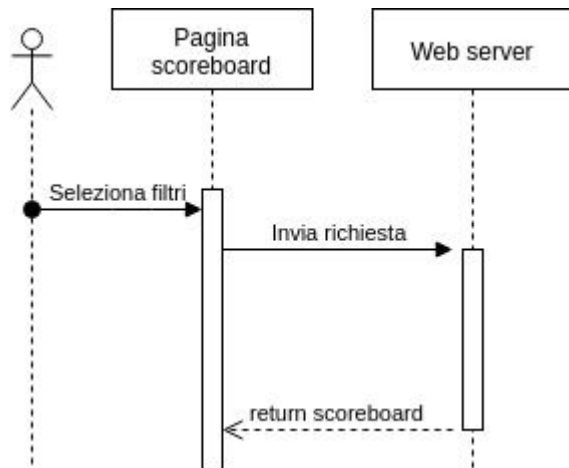
### Messaggi

weilà!  
2018-06-01T20:01:11.270Z

ehilà!  
2018-06-07T19:42:14.239Z

Invia il messaggio

## 7.6 Visualizzazione scoreboard



Gli utenti hanno la possibilità di vedere una scoreboard dove possono applicare dei filtri: visualizzare rispettivamente 5, 10, 20 o 100 utenti per volta e in ordine crescente per partite vinte, perse o totali.

### Visualizzazione scoreboard

Battleship

AccountPartiteGiocatoriChatsScoreboardLogout

## Scoreboard

Username	Partite vinte
<a href="#">admin</a>	0
<a href="#">prova2</a>	0
<a href="#">prova3</a>	0
<a href="#">prova4</a>	0

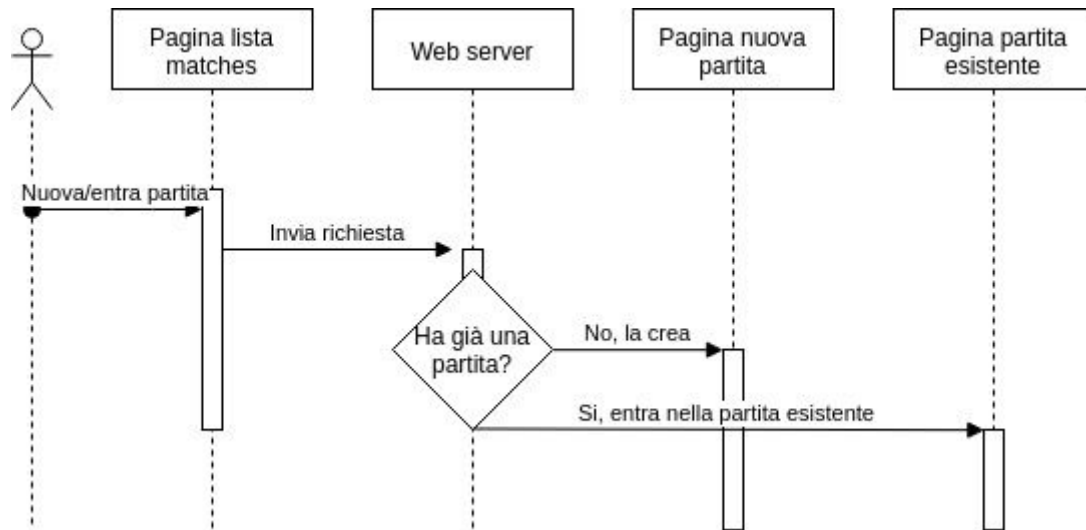
Mostra le prime posizioni nella classifica:

10

Ordina la classifica in base a:

Partite vinte

## 7.7 Creazione/unione a una partita



Gli utenti possono creare una partita alla volta oppure unirsi a una partita per volta.

Battleship

AccountPartiteGiocatoriChatsScoreboardLogout

### Lista dei match in attesa

Data di creazione 2018-05-23T14:49:09.125Z

Entra nella partita di: [prova3](#)

Entra nella partita!

Data di creazione 2018-05-30T19:20:30.088Z

Entra nella partita di: [admin](#)

Entra nella partita!

Data di creazione 2018-05-31T19:13:37.068Z

Entra nella partita di: [prova2](#)

Entra nella partita!

Crea una nuova partita

```
sequenceDiagram
    actor User
    participant P as Pagina posizionamento
    participant W as Web server
    participant PP as Pagina partita

    User->>P: Posizione navi
    activate P
    P->>W: Invia i dati
    activate W
    W-->>P: return error
    deactivate W
    P->>W: Corrette?
    activate W
    W-->>P: NO
    deactivate W
    P->>PP: Chiama
    activate PP
    deactivate PP
    deactivate P
```

The diagram illustrates the sequence of interactions for the 'Pagina posizionamento' (Positioning Page). It involves three lifelines: 'Pagina posizionamento', 'Web server', and 'Pagina partita'. The process begins with a user providing 'Posizione navi' (Ship position) to the 'Pagina posizionamento'. This page then sends 'Invia i dati' (Send data) to the 'Web server'. The 'Web server' returns a 'return error' to the 'Pagina posizionamento'. A decision diamond labeled 'Corrette?' (Correct?) is then reached. If the answer is 'NO', the 'Web server' returns to the 'Pagina posizionamento'. If the answer is 'SI' (Yes), the 'Pagina posizionamento' sends a 'Chiama' (Call) message to the 'Pagina partita'.

## Posizionamento

Account

Partite

Gioatori

Chats

Scoreboard

Logout

Schermata del match

Match in fase di costruzione

Posiziona le tue navi

Cacciatorpediniere - grandezza: 2

C

C

C

C

Sottomarino - grandezza: 3

C

C

Corazzata - grandezza: 4

C

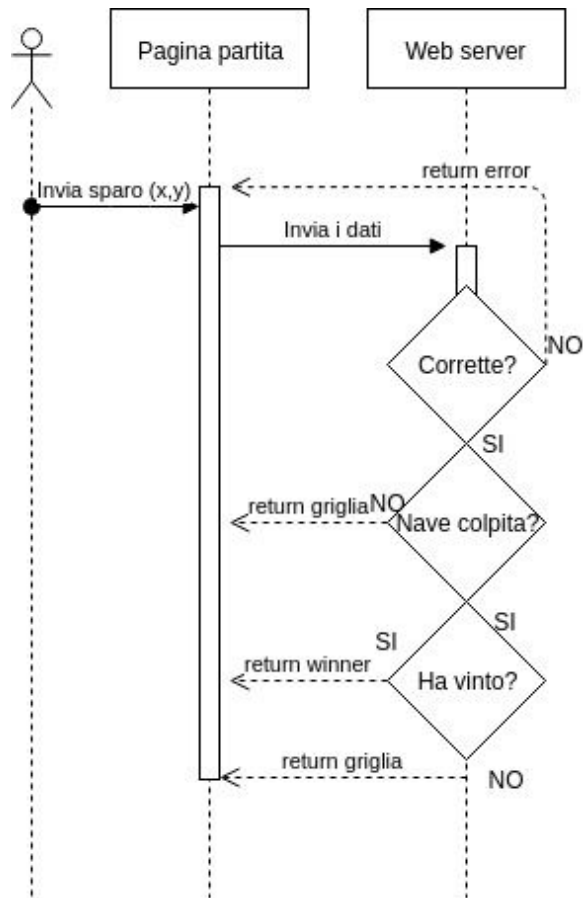
C

Portaerei - grandezza: 5

C

Inizia la partita

## 7.9 Sparo



L'utente clicca su una cella per sparare, se non ha ancora sparato su quella cella il server controlla se ha colpito una nave, se non l'ha colpita ritorna la griglia aggiornata, altrimenti controlla se ha vinto, in caso affermativo ritorna il vincitore, altrimenti la griglia aggiornata.

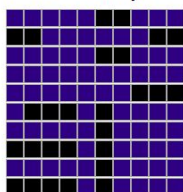
Battleship

[Account](#) [Partite](#) [Giocatori](#) [Chats](#) [Scoreboard](#) [Logout](#)

### Match vs: prova2

Attendi, è il turno dell'avversario

Il tuo campo



Campo avversario

