Relazione Progetto per Programmazione Applicazioni Web e Mobile

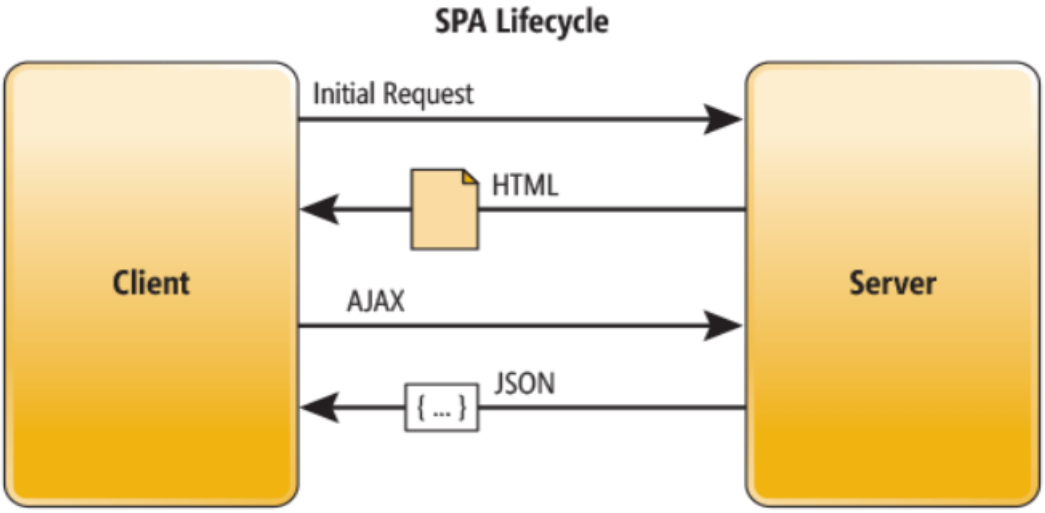
Descrizione progetto:

Ho sviluppato una semplice applicazione web per la gestione e la prenotazione di hotel, simile a Booking.com.
È una Single Page Application, basata su un backend in NodeJs e un frontend in React.
Nella pagina iniziale è possibile visualizzare le principali località dove poter scegliere l'hotel che più si preferisce e una sezione dove si può vedere il numero totale di strutture disponibili, divisi in base al tipo, ad esempio hotel, bed and breakfast, chalet, ecc.
Nella parte superiore della pagina c'è la possibilità di accedere oppure, se si è un utente nuovo, registrarsi. Questo è fondamentale per poter prenotare una camera all'interno delle varie strutture.
Sotto c'è una barra di ricerca dove è possibile inserire la località, il periodo, il numero di persone e il numero di stanze che si desidera. Una volta premuto il pulsante di ricerca, si potrà visualizzare una lista di strutture disponibili in quella località. A sinistra è presente una sezione per personalizzare ancora di più la ricerca, ad esempio scegliendo il range di prezzo. A destra è possibile, scorrendo la lista di strutture, selezionare quella desiderata attraverso il pulsante "Vedi disponibilità". Una volta premuto verrà visualizzata la pagina contenenti tutte le info di quella struttura. All'interno della pagina, se si è già fatto l'accesso, si potrà prenotare una stanza attraverso l'apposito pulsante. Altrimenti si verrà reindirizzati alla sezione di login/registrazione.
Quando si accede, se si è un amministratore di sistema, è possibile accedere alla dashboard riservata all'admin. Qui è possibile inserire nuovi utenti, normali o altri admin, nuovi hotel e visualizzare le liste di tutti gli utenti e hotel memorizzati nel database. Da qui è possibile anche eliminarli.

Tecnologie utilizzate:

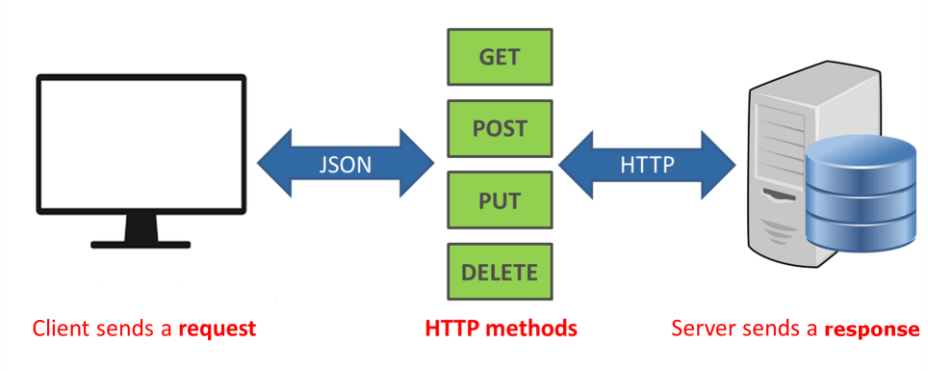
- **Backend:** per il backend ho utilizzato NodeJs con il web framework ExpressJs.
- **Frontend:** per il frontend ho utilizzato la libreria javascript React.

- **Architettura dell'applicazione:** per questo progetto ho scelto di realizzare una single page application: ovvero una wep app interamente costruita su un'unica pagina.
Nella prima chiamata da parte del frontend, il backend restituisce una pagina HTML insieme ad una bella quantità di javascript, che il browser salva, parsa ed esegue ad ogni interazione con l'utente. Queste interazioni vengono fatte, attraverso il javascript, manipolando il DOM; quindi, la maggior parte del lavoro la fa il browser. Il backend ha il compito di mandare il primo HTML ed il javascript e di fornire un API che dovrà elaborare le richieste ricevute restituendo del json da inserire nel DOM.



- **DBMS:** per la persistenza dei dati ho scelto come database MongoDB. L'applicazione, attraverso la libreria mongoose, accede al database con il metodo mongoose.connect() e crea degli Schema degli elementi del mio model. In seguito attraverso la libreria axios potrò raccogliere i dati (fetching data) attraverso i metodi Promise e PromiseAll che sfruttano l'async-await di javascript.

- **Operazioni CRUD:** Per gestire le richieste e i servizi ho utilizzato una REST-API.
REST è un paradigma per trasferire e rappresentare lo stato di una risorsa nel backend. Si basa sull'architettura Client/Server e su una connessione HTTP, dove c'è una request di una risorsa e una response dello stato di quella risorsa. Per gestire le varie operazioni http, come GET,POST,PUT e DELETE, ho definito delle routes.



Ogni route prevede anche un middleware per il controllo sul tipo di utente attraverso il controllo del JWT, in quanto, dopo aver verificato la presenza e la validità del token, determina se l'utente loggato è un admin o utente normale. In base a ciò abilita o disabilita le varie routes.

- **Autenticazione:** p Innanzi tutto ho utilizzato la libreria bcryptJs per hashare la password: nel momento in cui l'utente si registra creo un salt casuale univoco per ogni utente con cui ci hasho la password inserita. Ora la password è pronta per essere salvata nel db in tutta sicurezza perché comunicando in una connessione HTTPS non si corre il rischio di avere un man-in-the-middle.

To hash a password:

Technique 1 (generate a salt and hash on separate function calls):

```
const salt = bcrypt.genSaltSync(saltRounds);
const hash = bcrypt.hashSync(myPlaintextPassword, salt);
// Store hash in your password DB.
```

To check a password:

```
// Load hash from your password DB.
bcrypt.compareSync(myPlaintextPassword, hash); // true
```

Una volta che l'utente viene autenticato il backend mi restituisce un TOKEN JWT, creato attraverso una chiave SECRET. Per far navigare l'utente come utente registrato, ad ogni chiamata invio questo TOKEN con il quale il backend controlla se l'utente è autorizzato a compiere quella richiesta.
Ho utilizzato i token JWT (JsonWebToken), in quanto è sicuro e customizzabile, cioè posso memorizzarci le info che mi interessano verificare. Il JWT è composto da 3 parti:
- **Header**: che contiene le due informazioni base: la tipologia del token, che nel nostro caso è JWT, e quella

dell'algoritmo utilizzato per la cifratura;
- **Payload**: il blocco che contiene le informazioni di scambio tra le parti, come ad esempio se sei admin o no e la data di scadenza del token.
- **Signature**: è la concatenazione delle due parti precedenti hashate con la Secret memorizzata nel backend. Questo viene utilizzato dal backend per verificare la veridicità del token, in quanto prende l'header e il payload e li hasha con la secret, poi verifica che sia uguale alla signature. Da notare però che è possibile decifrare i token attraverso semplici tools online e quindi modificarli. Questo però non è un problema dato che non conoscendo la Secret, se si modifica il payload la signature non sarà mai valida. Lo scopo del JWT è quello di consentire alla parte ricevente di fidarsi che i dati ricevuti sono rimasti inalterati durante il trasporto.