Creazione di una semplice applicazione Web con CRUD usando Node.js e TDD

In questo progetto ho creato un semplice pannello di login utilizzando Node.js & Express per il backend, Pug per la visualizzazione, MongoDb Atlas per il database, Passport per l'autenticazione e lo testeremo utilizzando Mocha e Chai. Mentre per poter visualizzare anche le nostre API ho usato Swagger.

CRUD è l'acronimo di Create, Read, Update, Delete. È un insieme di operazioni che facciamo eseguire ai server (rispettivamente richieste POST, GET, PUT e DELETE). Ecco cosa fa ogni operazione:

Create (POST) - Crea qualcosa;
Read (OTTIENI) - Ottieni qualcosa;
Update (PUT) - Cambia qualcosa;
Delete (CANCELLA)- Rimuovi qualcosa.

Vediamole più nel dettaglio le tecnologie usate:

- Node.js è un ambiente di runtime Javascript multipiattaforma open source per lo sviluppo di applicazioni lato server.
- Express è un framework per applicazioni Web in Node.js minimale e flessibile che fornisce un solido set di funzionalità per applicazioni Web e mobile.
- Pug è un motore di modelli per convertire i dati inseriti e tradurli in sintassi html.
- MongoDB è un database.
- MongoDB Atlas è un servizio di database multi-cloud.
- Mocha è un framework di test JavaScript.
- Chai è una libreria di asserzioni BDD / TDD per Node che può essere abbinata a qualsiasi framework di test javascript.
- Passport.js è un middleware di autenticazione per Node.js.
- Swagger è uno strumento utile per progettare, creare, documentare e utilizzare le API REST.

Procedimento:
Creo una directory di progetto.
Eseguire il comando "*npm init*" che creerà il file "package.json" che memorizza le dipendenze per il progetto.
Come passaggio successiovo ho installato Express con il comando
npm install express -- save
in questo modo viene installata la libreria Express e salverà la dipendenza nel file package.json con il flag *"--save*". Quindi posso installare tutte queste dipendenze in un secondo momento semplicemente eseguendo il comando "*npm install*".

Dopo aver installato express, posso iniziare ad usarlo. Nel file principale, che ho chiamato server.js, questa app avvia un server e rimane in ascolto sulla porta 3000.

A questo punto posso eseguire l'applicazione con il comando "node server.js".

La richiesta risponderà con il seguente messaggio: “. Per ogni altro percorso, risponderà con un errore 404 non trovato.

Per usare i CRUD è necessaria una connessione al database.
In questo progetto, utilizzo MongoDB Atlas per il database.
Ho bisogno di una stringa di connessione per utilizzare questo db cloud nel progetto.
A questo punto mi sono registrata su MongoDB ed ho creato il mio db.

Ho installato "*mongoose*" .
Ed ho creato il mio file di configurazione del database . Nella cartella del progetto troviamo una cartella "config" ed al suo interno un file chiamato "db.config.js"

Recupero la stringa di connessione MongoDB, e la inserisco come parametro per *mongoose.*

Ora inserisco i **CRUD.**
Per la struttura del mio progetto ho creato queste directory e file.
Models > dove ho stabilito lo schema di definizione dei modelli;
Routes: > ho definito le mie rotte API ;
Controller: > dove viene gestita tutta la logica della validazione dei parametri di richiesta, query, invio di risposte, ecc.
Services: contiene le query del database e la restituzione di oggetti o la generazione di errori.
In maniera più approfondita:

- Ho creato la directory models e al suo interno un file chiamato user.js.
- Ho creato lo schema del modello utente con:
- un nome, un indirizzo e-mail e una password. Tutti gli utenti avranno questi dati.
- Ho usato il modulo "*mongoose-unique-validator*" per rendere unico il parametro email.

Poi ho definito i controller nel file user.controller.js.
Voglio che la mia API possa fare questi lavori:

- Recupera un elenco di utenti: findUsers;
- Recupera un singolo utente per ID: findUserById;
- Recupera un singolo utente per nome: findUserByName;
- Crea un nuovo utente: createNewUser;
- Modifica utente per ID : updateById;
- Modifica utente per nome: updateByName;
- Elimina utente per ID: deleteUserById;

A questo punto ho definito le rotte.
Ho inserito db.config e user.routes nel file server.js.

Sono passata a Swagger
Per usare swagger, ho dovuto installare i moduli *swagger-jsdoc* e *swagger-ui-express*.
Ho creato un file di configurazione swagger. Dopo aver definito i campi in questo file ho implemantato swagger nel file user.routes.js.

Ho definito swagger nel file server.js.
Quando verrà chiamato l'endpoint *"/api-docs*", verrà visualizzata l'interfaccia utente di swagger e si potrà utilizzare le API.

Si possono eseguire tutte le operazioni CRUD per il nostro servizio web utilizzando questa interfaccia utente.

Sono passata allo sviluppo **Frontend**
Questa app avrà 2 pagine web; login e welcome.
Ho utilizzato il modello già pronto preso da qui ->……
Ho convertito questo modello html in pug.js
Login.pug è la pagina principale. Ci sono funzioni di login e registrazione.
Quando si fa clic sul pulsante di registrazione, apparirà il modulo di registrazione.
Dopo la registrazione riuscita, apparirà di nuovo il modulo di login.
Quando si accede con successo, la pagina di Welcome verrà aperta e mostrerà i dati (nome utente) come messaggio di benvenuto.

Ho creato i file "public/css/style.css" e "public/js/main.js". In cui si trovano il design e le azioni della nostra pagina web di accesso/registrazione.

A questo punto sono passata allo sviluppo del **Backend.**
Quando si fa clic sul pulsante di invio della registrazione con nome utente, password ed e-mail compilati, verrà chiamata la richiesta di post.
Nella funzione di registrazione, ho utilizzato il modulo bcrypt.js per eseguire l'hash della password.
I dati degli utenti registrati con successo verranno salvati su MongoDb.
Si andrà alla fase di accesso. Qui, controllerò che le informazioni sull'utente immesse e le informazioni sull'utente nel database siano le stesse. Per questa autenticazione ho utilizzato *passport.js*

Sono passata poi alla fase di test con Mocha & Chai.