METODOLOGIE DI PROGRAMMAZIONE PER IL WEB

COMPLETARE IL TASK MANAGER

In questo sesto e ultimo laboratorio, aggiornerai l'applicazione web per la gestione dei task per supportare molteplici pagine (attraverso le route o passando a un rendering server-side) e per aggiungere il login.

ESERCIZIO 1 – PIÙ PAGINE

Aggiorna il task manager che hai sviluppato nei laboratori precedenti per supportare più pagine". In particolare, **scegli** se mantenere la stessa applicazione ma implementare le *route client-side* oppure se passare al rendering server-side (con *ejs* come motore di template).
In entrambi i casi, implementa tutti i filtri e il form per aggiungere un nuovo task come pagine separate.

Aggiungi un'ulteriore pagina con un form, che dovrà essere utilizzato per eseguire il login di un utente. Il form dovrà avere due campi *obbligatori*: e-mail e password, entrambi validati in maniera appropriata nel client.

Il prossimo esercizio si concentrerà sul processo di login: per ora, aggiungi solo la pagina di login e le route appropriate per supportare il seguente caso d'uso: quando un utente sottomette un form debitamente compilato, re-indirizzalo alla pagina contenente la lista dei task e mostra un messaggio "*Benvenuto, <email>!*" all'inizio della pagina.

Suggerimenti:

1. *Puoi utilizzare la soluzione del lab 5 come punto di partenza, se vuoi: https://github.com/luigidr/2020-metweb-lab5-express-fetch*
2. *In aggiunta, puoi dare un'occhiata agli esercizi sviluppati durante le lezioni:*
    a. *client-side routing, https://github.com/luigidr/2020-metweb-es-js-routing*
    b. *oppure server-side app, https://github.com/luigidr/2020-metweb-es-server-rendering*

ESERCIZIO 2 – LOGIN

Implementa il processo di login, come mostrato durante le lezioni, per permettere a un utente autorizzato di vedere i propri task. In particolare, sfruttando le sessioni, dovresti realizzare i seguenti cambiamenti e aggiornamenti al database, al server, e al client.

1. [**Database**] Crea una nuova tabella utenti ("user", colonne obbligatorie: *id, name, email, hash*), con almeno un utente. Usa `bcrypt` per generare l'hash di una password a tua scelta, dato che non memorizziamo password in chiaro nel database.
2. [**Database**] Modifica la tabella dei task in modo tale che un utente possa avere uno o più task. Associa l'utente appena creato a tutti i task presenti nella tabella.
3. [**Express**] Crea un metodo `getUser()` e un `checkPassword()` per recuperare, rispettivamente, le informazioni su uno specifico utente dal database e per controllare se la password ricevuta dal form di

login corrisponde all'hash memorizzato nel database per lo stesso utente. Installa e usa il modulo `bcrypt` per controllare gli hash.

4. [**Express**] Usando il modulo passport, implementa una route per il login che, in caso di successo, invii al client la mail dell'utente. Configura passport in modo da utilizzare le sessioni e i cookie.
5. [**Express**] Proteggi le alter route così che siano accessibili solo se l'utente è autenticato (cioè, il session id passato nel cookie è valido).
6. [**Client**] Modifica la logica dietro al form di login per inviare la mail e la password inserite nel form al server (per esempio via fetch).
7. [**Client**] Gestisci cosa capita in caso di login errato o corretto. Nel primo caso, il client dovrebbe mostrare un messaggio di errore appropriato (per esempio, "Mail non riconosciuta", "Password errata" o simili) e continuare a mostrare il form di login. Nel secondo caso, il client dovrebbe gestire la risposta ricevuta e mostrare un messaggio "Benvenuto, <email>!" quando si fa il redirect alla pagina che contiene la lista dei task.
8. Handle the receipt of a wrong or successful login. In the former case, the client should display a suitable error message (e.g., "*Wrong username*", "*Wrong password*" or similar) and continue to show the login form. In the latter case, the client should handle the received JWT token, and show a "*Welcome, {name of the user}*" message upon redirect to the page containing the list of tasks.

OPZIONALE

Aggiungi la funzionalità di logout, sia nel client che nel server. Dopo il logout, il sito dovrà visualizzare la pagina di login.

Suggerimenti:

1. *Puoi usare il seguente sito per generare l'hash di una password, secondo bcrypt:* https://www.browserling.com/tools/bcrypt*. Se l'hash generata iniziasse con $2y$, rimpiazza quella parte con $2b$, poiché il modulo node di bcrypt non supporta hash del tipo $2y$.*
2. *Puoi installare il modulo bcrypt con npm. La documentazione è disponibile all'indirizzo https://github.com/kelektiv/node.bcrypt.js*
3. *Per il login, sfrutta passport: http://www.passportjs.org*
4. *Per manipolare il database, puoi usare DB Browser for SQlite: https://sqlitebrowser.org*
5. *Come punto di partenza, puoi dare un'occhiata agli esercizi sviluppati durante le lezioni, entrambi con un database simile a quanto richiesto nell'esercizio:*
    a. *con routing client-side, https://github.com/luigidr/2020-metweb-es-login*
    b. *oppure con server-side app, https://github.com/luigidr/2020-metweb-es-server-rendering*