COSA STIAMO UTILIZZANDO

Sviluppare l’applicazione **in modalità SPA (2 tiers**):

Sviluppare un'applicazione web in modalità SPA (Single Page Application) a due livelli (2 tiers) significa creare un'applicazione web che funzioni principalmente come un'unica pagina, dove la maggior parte dell'interazione e del rendering dei contenuti avviene lato client.

In una SPA a due livelli, il primo livello è costituito dal client, che comprende l'interfaccia utente, la logica di presentazione e l'elaborazione dei dati. Il secondo livello è costituito dal server, che gestisce le richieste e le risposte del client, nonché l'accesso ai dati o alle risorse necessarie per l'applicazione.

La modalità SPA offre un'esperienza utente più fluida, poiché le pagine non vengono completamente ricaricate ad ogni interazione dell'utente. Invece, l'applicazione recupera dinamicamente i dati necessari dal server e aggiorna solo le parti dell'interfaccia utente che sono necessarie per visualizzare i nuovi contenuti o rispondere all'interazione dell'utente.

**Node.js, Express.js, GraphQL e PostgreSQL** sono tecnologie diverse che possono essere utilizzate insieme per creare applicazioni web complesse. Ecco una panoramica delle differenze tra di loro e come possono interagire:

* Node.js: E’ il runtime di javascript che permette lo sviluppo server-side e consente di scrivere ed eseguire codice javascript;
* Express.js: Un framework web per Node.js. Si occupa di gestire il routing delle richieste HTTP, la gestione delle risposte e fornisce un'architettura per creare applicazioni web. Express.js è principalmente responsabile della parte server-side dell'applicazione web.
* GraphQL: GraphQL è un linguaggio di query per le API che consente di definire in modo dichiarativo le richieste di dati dal server. A differenza degli endpoint REST tradizionali, in cui il client deve fare più richieste per ottenere i dati necessari, GraphQL permette al client di specificare esattamente quali dati vuole ricevere dal server e in quale formato. GraphQL consente di ridurre il sovraccarico di rete e di ottenere solo i dati richiesti, migliorando l'efficienza delle richieste. GraphQL viene utilizzato principalmente per la parte API dell'applicazione.
* PostgreSQL: PostgreSQL è un potente sistema di gestione di database relazionali (RDBMS). Offre un'ampia gamma di funzionalità per la gestione dei dati, compresi i tipi di dati avanzati, le transazioni ACID e la scalabilità. PostgreSQL è uno dei database più popolari e viene utilizzato comunemente nelle applicazioni web per archiviare e recuperare i dati.

Come possono interagire:

Express.js viene spesso utilizzato come framework per gestire il backend delle applicazioni web. Può essere utilizzato per definire i percorsi delle API, gestire le richieste HTTP e comunicare con il database PostgreSQL per l'accesso e la modifica dei dati. Express.js può utilizzare librerie specifiche per PostgreSQL per interagire con il database, eseguire query, gestire transazioni e recuperare i dati richiesti.

GraphQL può essere utilizzato come livello di astrazione sopra il backend Express.js e il database PostgreSQL. Express.js può essere configurato per esporre uno schema GraphQL che definisce i tipi di dati e le operazioni disponibili. GraphQL si occupa di elaborare le query inviate dal client, tradurle in query SQL per PostgreSQL e restituire solo i dati richiesti. Questo permette al client di ottenere esattamente i dati di cui ha bisogno in un'unica richiesta, migliorando l'efficienza e riducendo il traffico di rete.

In sostanza, Express.js viene utilizzato per gestire la parte server-side e comunicare con il database PostgreSQL, mentre GraphQL viene utilizzato per gestire le richieste dei client e fornire un'interfaccia efficiente per l'accesso ai dati. Insieme, queste tecnologie consentono di creare applicazioni web moderne e scalabili.

**Se si utilizza Angular lato client, come è possibile far comunicare client e server ?**

Quando si utilizza Angular come framework di sviluppo del lato client, la comunicazione tra il client (Angular) e il server (Express.js con GraphQL o un altro backend) può avvenire tramite richieste HTTP.

Ecco un'indicazione generale su come configurare la comunicazione tra client e server quando si utilizza Angular:

* Creazione di servizi: In Angular, è possibile creare servizi per gestire le richieste HTTP verso il server. I servizi possono essere utilizzati per eseguire chiamate GraphQL al server, inviare richieste di dati, effettuare aggiornamenti e ricevere le risposte. I servizi possono essere creati utilizzando il modulo Apollo fornito da Angular.
* Configurazione dell'URL del server: Nel servizio, è necessario specificare l'URL del server in cui è in esecuzione il backend (Express.js). Questo può essere un indirizzo IP locale (ad esempio [http://localhost:3000](http://localhost:3000/)) o un indirizzo IP remoto.
* Gestione delle risposte: Una volta inviata una richiesta al server, GraphQl consente al client di ricevere eventuali risposte. La risposta può contenere dati che possono essere utilizzati per aggiornare l'interfaccia utente o eseguire altre operazioni lato client.

In sintesi, Angular utilizza il modulo Apollo per eseguire richieste HTTP al server (Express.js) attraverso servizi personalizzati. Il server può essere configurato per gestire queste richieste e restituire le risposte, che vengono quindi elaborate da Angular per aggiornare l'interfaccia utente o eseguire altre operazioni necessarie.

**Come è possibile passare il client i dati di una base di dati per poterli visualizzare correttamente?**

Per passare i dati dal server al client in modo che possano essere visualizzati correttamente, puoi seguire questi passaggi generali:

* Recupero dei dati dal database: Nel tuo server (Express.js), devi implementare la logica per recuperare i dati dal database. Puoi utilizzare un'interfaccia di accesso al database (ORM) o un driver specifico per eseguire le query necessarie e ottenere i dati richiesti.

(Noi abbiamo utilizzato direttamente i database messi a disposizione da Pg Admin per poter visualizzare successivamente dati aggiornati o inseriti e anche perché il mapping fornito dall’interfaccia ORM risulta inefficiente in alcuni casi, meno performante di una query SQL).

* Elaborazione dei dati nel server: Dopo aver recuperato i dati dal database, puoi eseguire eventuali operazioni di elaborazione o formattazione dei dati nel server. Ad esempio, puoi eseguire filtri, ordinamenti o aggregazioni sui dati per ottenere il risultato desiderato.
* Invio dei dati al client: Dopo aver elaborato i dati nel server, puoi inviarli al client come parte della risposta HTTP (Sfruttandpo sempre le funzionalità di GraphQl). Puoi includere i dati come parte del corpo della risposta o come attributi specifici, a seconda delle tue esigenze.

Elaborazione dei dati nel client: Dopo aver ricevuto i dati nel client, puoi elaborarli ulteriormente per adattarli alla visualizzazione. Ad esempio, puoi formattare i dati, creare strutture dati aggiuntive o applicare filtri per visualizzare solo una parte specifica dei dati.

* Visualizzazione dei dati: Infine, puoi utilizzare le funzionalità di Angular per visualizzare i dati nel tuo componente o template. Puoi utilizzare ad esempio la direttiva \*ngFor per iterare attraverso i dati e mostrare ogni elemento, utilizzare le interpolazioni {{ }} per visualizzare i valori specifici dei dati, o utilizzare altre funzionalità di Angular per presentare i dati in modo appropriato.

In sintesi, i dati vengono recuperati e elaborati nel server, quindi inviati al client attraverso una risposta HTTP. Nel client, i dati ricevuti vengono elaborati ulteriormente e quindi visualizzati utilizzando le funzionalità di Angular.

GraphQL

COME FUNZIONA GRAPHQL? Il client chiede i dati al server GraphQL attraverso le query. Il formato della risposta è descritto nella query e definito dal client invece che dal server. La struttura dei dati non è hardcoded come nelle tradizionali API REST e questo permette il recupero dei dati dal server più efficiente per il client. Per esempio un client può chiedere risorse collegate senza bisogno di definire un nuovo endpoint. GraphQL è un punto di contratto tra il client e il server: viene definito uno schema che entrambi rispetteranno. Gli oggetti di tale schema devono essere tipizzati per cui non è possibile che ci siano incomprensioni tra team di sviluppo separati: lo schema è unico e comune, inoltre, per ogni campo viene definito il tipo. L’uso di GraphQL permette al client di cambiare il formato della risposta autonomamente e indipendentemente dal backend. In GraphQL lo strumento che usiamo per aggiornare i dati dal client al server si chiama mutations. Con GraphQL è possibile descrivere i dati e la richiesta degli stessi in maniera più naturale, permettendo quindi di velocizzare lo sviluppo. Così come il client, allo stesso modo anche il server GraphQL può essere scritto in qualsiasi linguaggio si voglia: PHP, nodejs, Typescript, etc.

DESCRIZIONE DEL PROGETTO

App.js

Questo file è un modulo di un progetto Angular che utilizza Express.js, GraphQL e PostgreSQL come database. Il codice è strutturato in diverse sezioni e utilizza diverse librerie per gestire le funzionalità del server. Ecco una descrizione strutturata delle varie parti del codice:

1. Importazione delle librerie:

- express: Libreria per creare un'applicazione Express.js.

- fs: Modulo del sistema per leggere file dal disco.

- graphqlHTTP: Middleware di Express.js per gestire le richieste GraphQL.

- buildSchema: Funzione di GraphQL per costruire uno schema.

2. Configurazione di Express.js:

- Creazione dell'applicazione Express.js con express().

- Utilizzo del middleware cors per gestire la comunicazione cross-origin.

- Utilizzo del middleware json per il parsing delle richieste JSON.

3. Lettura dello schema GraphQL:

- Caricamento del file "schema.graphql" utilizzando fs.readFileSync.

- Costruzione dello schema GraphQL utilizzando buildSchema e il contenuto del file caricato.

4. Connessione al database PostgreSQL:

- Creazione di due istanze di Pool per gestire la connessione al database dei dati (data\_dbms) e al database delle credenziali (credentials\_dbms).

- Le istanze di Pool vengono configurate con le informazioni di connessione al database (utente, host, database, password, porta).

5. Definizione di una chiave e opzioni per la generazione di token JWT (JSON Web Token).

6. Definizione delle funzioni di risoluzione per le query GraphQL:

- La funzione **films** recupera le informazioni sui film dal database e le restituisce come risultato della query. Vengono gestiti i parametri di offset e limit per la paginazione dei risultati. Abbiamo definito un limite di 12 poiché nella nostra interfaccia abbiamo scelto di avere 12 card per pagina. L’offeset si aggiorna ogni 12 film e c’è un count che conta il numero totale dei film presenti.

- La funzione **film\_information** recupera le informazioni di un film specifico dal database in base al titolo e le restituisce come risultato della query.

- La funzione **film\_store\_information** recupera le informazioni sugli store in cui un film è disponibile per il noleggio e le restituisce come risultato della query.

- La funzione **order\_information** recupera le informazioni sugli ordini di noleggio di un cliente specifico dal database e le restituisce come risultato della query.

- Le **funzioni film\_category\_search, film\_title\_search e film\_title\_and\_category\_search** eseguono ricerche sui film in base alle categorie e/o al titolo, restituendo i risultati corrispondenti. . Abbiamo definito un limite di 12 poiché nella nostra interfaccia abbiamo scelto di avere 12 card per pagina. L’offeset si aggiorna ogni 12 film e c’è un count che conta il numero totale dei film presenti.

7. La funzione **signUp** gestisce la registrazione di un nuovo utente nel database delle credenziali. Salva l'username e la password nel database utilizzando una query SQL di inserimento.

Questo è un breve riassunto del codice presente nel file "html app.js". Si tratta di un modulo che definisce le query GraphQL e gestisce la comunicazione con il database PostgreSQL utilizzando Express.js come framework server.

Descrizione di cosa è un middleware e dei tre middleware express.js, cors e json:

Un middleware è un concetto ampiamente utilizzato nelle applicazioni web per gestire le richieste HTTP e le risposte tra il client e il server. In sostanza, è una funzione o un insieme di funzioni che vengono eseguite tra il momento in cui viene ricevuta una richiesta dal client e il momento in cui viene inviata una risposta al client.

Nei framework web come Express.js, i middleware vengono utilizzati per estendere le funzionalità dell'applicazione, aggiungere funzioni di elaborazione delle richieste o delle risposte, manipolare i dati delle richieste o aggiungere controlli di sicurezza.

Quando una richiesta arriva al server, passa attraverso una pipeline di middleware che vengono eseguiti in sequenza. Ogni middleware può accedere all'oggetto della richiesta (request) e all'oggetto della risposta (response) e può eseguire operazioni come la modifica dei dati della richiesta, l'aggiunta di informazioni alla risposta, il controllo delle autorizzazioni, la gestione degli errori e così via.

MIDDLEWARE UTILIZZATI:

1 - Middleware in Express.js:Express.js è un framework web popolare per lo sviluppo di applicazioni web in Node.js. Express.js fornisce una varietà di middleware integrati che possono essere utilizzati per estendere le funzionalità dell'applicazione. Questi middleware possono essere aggiunti all'applicazione tramite il metodo app.use().

2 - Middleware CORS:

CORS (Cross-Origin Resource Sharing) è un meccanismo di sicurezza che consente a una pagina web di fare richieste a un dominio diverso da quello da cui è stata caricata. Il middleware CORS in Express.js gestisce la configurazione dei permessi di accesso alle risorse da parte dei client esterni.

3 - Middleware JSON:

Il middleware JSON in Express.js è utilizzato per analizzare i dati inviati nel corpo di una richiesta HTTP con il formato JSON e rendere questi dati disponibili come oggetto JavaScript nella richiesta (req.body). Questo middleware è essenziale per l'elaborazione dei dati inviati dai client.

Quando il middleware JSON è applicato a un'applicazione Express.js tramite app.use(express.json()), esso analizza il corpo delle richieste che contengono dati in formato JSON e li converte in un oggetto JavaScript che può essere facilmente manipolato nel codice dell'applicazione.

SCHEMA.GRAPHQL

1 - Query:

Il tipo di operazione "Query" rappresenta le richieste di dati al server.

Query non menzionate precedentemente:

"signIn": Effettua l'autenticazione dell'utente utilizzando il nome utente e la password.

"tokenTest": Effettua un test sul token di autenticazione fornito.

2 -Mutation:

Il tipo di operazione "Mutation" rappresenta le richieste di modifica dei dati sul server.

Le operazioni definite all'interno di "Mutation" includono:

"signUp": Registra un nuovo utente con un nome utente e una password.

"rent": Effettua una prenotazione per il noleggio di un film specificato fornendo la città, l'indirizzo e la data di noleggio.

3 - Tipi risultato:

Auth" rappresenta i dati di autenticazione e contiene campi come il flag di autenticazione, il token di accesso e la scadenza del token.

"TokenTest" rappresenta il risultato di un test sul token di autenticazione e contiene un campo booleano per indicare il successo o il fallimento del test.

DVD-STORE.service

Il file service.ts è un file di servizio Angular che fornisce metodi per interagire con un server GraphQL utilizzando Apollo Client. Questo file contiene diverse interfacce che definiscono i tipi di dati utilizzati nel progetto.

Il servizio DvdStoreService è un servizio Angular Injectable che viene fornito a livello radice (providedIn: 'root') e utilizza Apollo Client per effettuare richieste GraphQL al server.

Il costruttore del servizio inizializza le query GraphQL utilizzando apollo.watchQuery e definisce i parametri necessari per ogni query.

Il metodo tokenCheck viene utilizzato per controllare il token di autenticazione prima di eseguire ogni richiesta al server. Verifica se il token è presente, se è scaduto e se corrisponde al token memorizzato nel server. Se il token non è valido, reindirizza l'utente alla pagina di accesso.

Questo file di servizio è utilizzato dai componenti del progetto per ottenere i dati dal server GraphQL e interagire con esso. In particolare, sembra che i due file precedenti che hai inviato si integrino con il servizio DvdStoreService per ottenere i dati dei film, delle categorie e delle informazioni correlate.

COME AVVIENE IL PASSAGGIO DEI DATI TRA CLIENT E SERVER

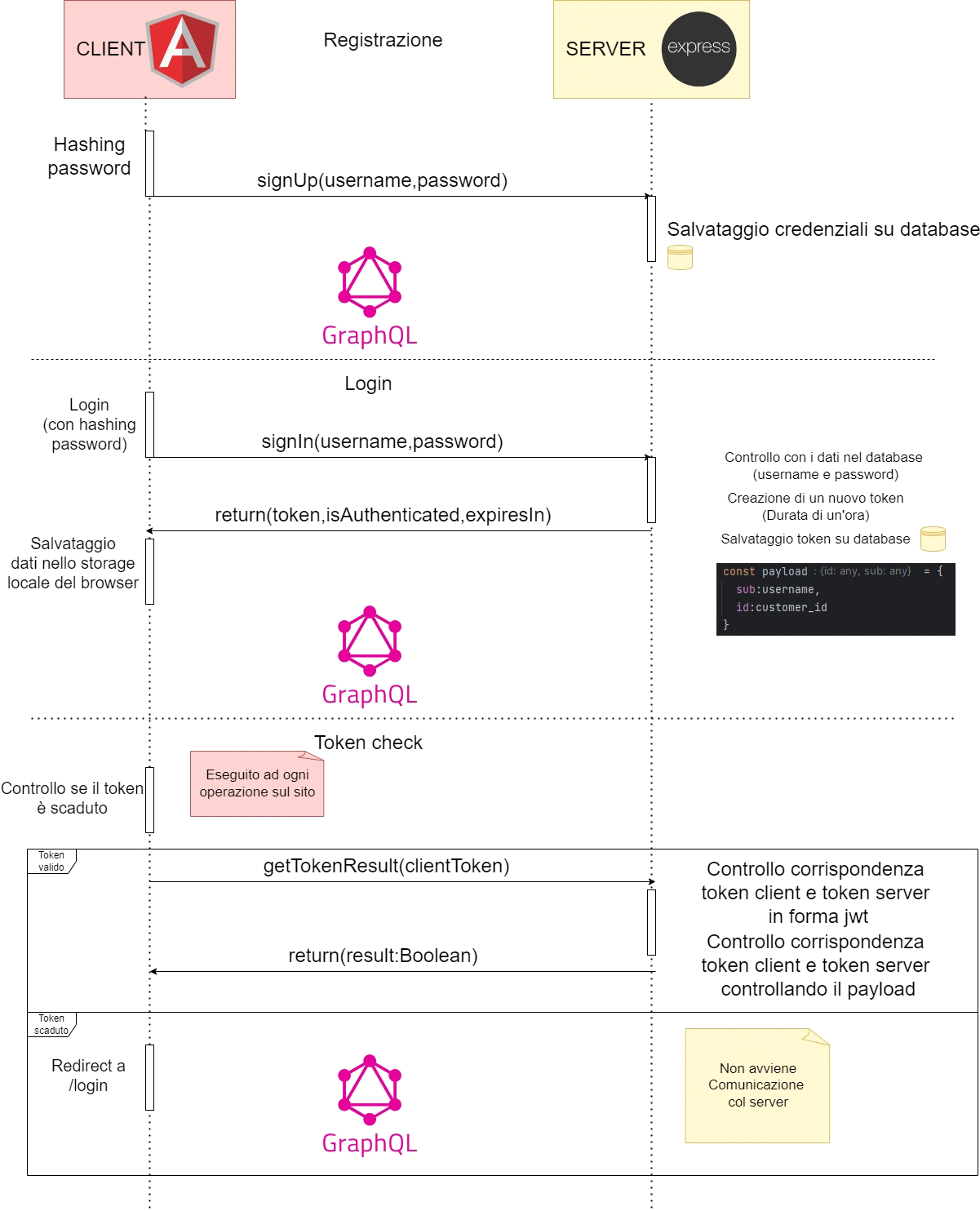
* Il client (Angular) invoca un metodo definito nel file service.ts per richiedere determinati dati dal server.
* Il metodo nel file service.ts utilizza Apollo Client per effettuare una richiesta GraphQL al server.
* La richiesta viene inviata al server Express tramite un'API GraphQL specificata nel file app.js.
* Il server Express riceve la richiesta e utilizza i resolver corrispondenti per ottenere i dati richiesti dal database Postgres tramite le query definite in app.js.
* I dati vengono restituiti dal server al client tramite la risposta GraphQL.
* Apollo Client nel file service.ts riceve la risposta dal server e restituisce i dati al metodo chiamante nel client-side dell'applicazione Angular.
* I dati vengono utilizzati nel client per mostrare le informazioni desiderate all'utente.

COME AVVIENE IL PASSAGGIO DEI DATI DAL SERVICE AI COMPONENT E TRA I VARI COMPONENTI

A partire dal file di servizio, sono definiti una serie di metodi che restituiscono una promise che sono il risultato della query effettuata in modo asincrono. Ogni componente che necessita dei dati ha un metodo all’interno del quale viene richiamato il servizio relativo a quella determinata query.

Per il passaggio tra i vari componenti, definiamo delle variabili nel service e queste saranno utilizzate nei vari componenti che ne avranno bisogno. Questi sono principalmente utilizzati nelle finestre di dialogo.

COME VIENE GESTITA L'AUTENTICAZIONE CON TOKEN



Hashing Password:

La password che viene fornita in fase di registrazione viene crittografata con l'algoritmo Sha3 utilizzando la libreria crypto a lato client

signUp(username,password):

manda i dati al server tramite Apollo e il server lì salva appena arrivano

Login:

l'utente tenta di accedere immettendo i dati e cliccando sull'apposito bottone html

signIn(username,password):

i dati vengono inviati dal client al server. La password è già crittografata in questa fase.

Il server controlla che i dati siano corretti e crea il token usando lo username e il customer\_id assegnato dalla sequenza seriale della tabella user\_credentials posto sul database dvd\_credentials.

Il token viene salvato sul database e inviato al client assieme al dato utente loggato(isAuthenticated) e il dato della durata del token(expiresIn).

salvataggio nello storage locale del browser:

le informazioni provenienti dal server vengono salvate nella local storage del browser.

Ogni token viene mantenuto nella locazione finchè è valido in termini di tempo.

controllo scadenza token:

ad ogni operazione compiuta da parte di un utente all'interno del sito corrisponde un controllo se il token è expired oppure se è ancora valido.

se il token è valido (ci troviamo ancora nella finestra temporale dell' expiresIn), allora effettuo una chiamata al server per controllare se l'operazione che si sta per compiere viene effettuata dal proprietario del token (customer\_id)

getTokenResult(clientToken):

la richiesta che viene effettuata al server ogni volta che si intende verificare l'identità di chi esegue una determinata operazione sul sito.

Notare il doppio controllo effettuato, uno testa se il token è lo stesso di quello salvato sul dbms, l'altro è quello del test dei singoli elementi del payload (la funzione jwt.verify(token,key) permette di ottenere indietro il payload utilizzando la chiave segreta in possesso solo del server).

Importante notare anche che se il client trova che il token è scaduto costringe l'utente a loggarsi nuovamente e quindi a sovrascrivere il token presente sia nel dbms del server, sia nel local storage del client di conseguenza

OSSERVAZIONI SECONDARIE CIRCA L'AUTENTICAZIONE TRAMITE TOKEN

- l'utente può entrare in /home o in /orders quando il token è ancora valido (utente già loggato in precedenza)

- se l'utente è sprovvisto di token non può accedere a /home o /orders causa mancaza di fase di login