Descrizione progetto Programmazione Web

BACK END

Il web server utilizzato è TomCat 9.0.56. Sono stati usati Maven (dipendenze prese dal Maven Repository) e Java 15.0.2. L'editor usato è Visual Studio Code, con le estensioni:

NOME	LINK
Extension Pack for Java	*https://marketplace.visualstudio.com/items?itemName=vscjava.vscode-java-pack*
Debugger for Java	*https://marketplace.visualstudio.com/items?itemName=vscjava.vscode-java-debug*
Maven for Java	*https://marketplace.visualstudio.com/items?itemName=vscjava.vscode-maven*
TomCat for Java	*https://marketplace.visualstudio.com/items?itemName=adashen.vscode-tomcat*

È stato utilizzato Hibernate (guida seguita) per connettersi con un database MySql. Il database presenta le tabelle roles, routes, types, users.

NOME	CAMPI	CONTENUTO
roles	*rolename*	Contiene i nomi dei tipi di utenti dell'applicazione. Gli unici due ruoli presenti sono admin e standarduser.
routes	*id, user, date, type, route, stages*	Contiene tutte le route di tutti gli utenti.
types	*id, type*	Ogni entry rappresenta uno dei tipi possibili per una route (ovvero il mezzo utilizzato).
users	*id, username, password, email, role*	Contiene gli utenti dell'applicazione.

È stato usato Jersey, autenticazione e autorizzazione sono stati gestiti con Token JWT, seguendo questo tutorial e una guida su filtri ed interceptor in Jersey. Nel database viene salvato l'hash di password assieme ad un salt (metodo hashed & salted). A differenza del tutorial, i JWT non vengono salvati in memoria quindi non possono essere revocati. Come suggerito nel tutorial invece, è stato implementato un semplice meccanismo per il token refresh. Questo prevede un refresh token dalla durata 8 volte maggiore rispetto a quella del normale access token (impostata a 30 minuti). Con questo refresh token è possibile ottenere un nuovo access token evitando la ritrasmissione delle credenziali.

Seguendo la documentazione di Jersey e la guida precedentemente citata sugli interceptor sono stati poi inseriti un WriterInterceptor ed un ReaderInterceptor, per implementare la trasmissione e ricezione del payload HTTP in gzip qualora gli header *accept-encoding* o *content-type* esplicitassero gzip.

API Test

Per i test delle api è stato utilizzato il programma PostMan.

API Endpoints

ENDPOINT	METODO	DESCRIZIONE	AUTORIZZAZIONE
api/user/register	POST	Registrare un nuovo utente	-
api/user/authenticate	POST	Autenticare un utente	-
api/user/refresh	GET	Fornire un nuovo access token	refresh token (standard user)
api/resource/routes/{d1}/{d2}	GET	Restituire le route di un utente nell'intervallo di date {d1} {d2}.	access token (standard user)
api/resource/userinfo	GET	Restituisce le informazioni dell'utente autenticato	access token (standard user)
api/resource/types	GET	Restituisce i tipi di route utilizzabili.	-
api/resource/route	POST	Inserire una nuova route	access token (standard user)
api/resource/route	PUT	Modificare una route esistente	access token (standard user)
api/resource/route/{id}	DELETE	Eliminare la route con l'id specificato	access token (standard user)

FRONT END

L'editor usato è stato Visual Studio Code. È stato utilizzato Vue.js 3 con Vue CLI (usando Node.js v16.13.1). È stato utilizzato il meccanismo di routing di Vue.js per realizzare una Single Page Application. Alcuni tutorial seguiti sono stati How to build a spa with Vue.js e Vue Router: building spa.

Le estensioni di Visual Studio Code usate sono state:

NOME	**LINK**
Vetur	*https://marketplace.visualstudio.com/items?itemName=octref.vetur*
Vue VSCode Snippet	*https://marketplace.visualstudio.com/items?itemName=sdras.vue-vscode-snippets*

Per la mappa principale è stato usato vue-leaflet, versione Beta compatibile con Vue.js 3. Per le mappe di inserimento e modifica delle route non sono stati utilizzati custom component (data la non totale compatibilità di vue-leaflet con Vue.js 3 nell'utilizzo di leaflet).

Nel browser informazioni come le route visualizzate sulla mappa, l'access token ed il refresh token, sono stati salvati in *Window.localStorage*. Il contenuto della mappa nella sezione "Routes" del sito si aggiorna quando l'array di route salvato nel local storage viene aggiornato. Per farlo viene creato un custom event ad ogni inserimento nel local storage, seguendo il metodo qui spiegato.

L'accesso alle API viene effettuato utilizzando la libreria AXIOS, la struttura del codice è stata realizzata prendendo spunto da A simple approach to Managing API Calls.

Sono stati utilizzati i componenti Vue 3 Datepicker e vueform/multiselect per i form di inserimento e modifica di una route.

Per le transizioni fra le view di Vue.js è stato seguito il tutorial Vue 3 Animations Tutorial #12 - Route Transitions.

ROUTED APP

Una demo del funzionamento dell'applicazione è stata caricata su youtube.