Configuration par une DB

Démonstration 2 du module 7

L'objectif de cette démonstration est l'utilisation d'une base de données pour déclarer les utilisateurs.

Contexte

- Continuer dans l'application de démonstration précédente
- Dans ressources, vous trouverez un script SQL pour créer les tables nécessaires et pour créer des comptes.

```sql
CREATE TABLE eni_users (
  name VARCHAR(50) NOT NULL,
  email VARCHAR(50) NOT NULL,
  password VARCHAR(100) NOT NULL,
  enabled TINYINT NOT NULL DEFAULT 1,
  PRIMARY KEY (email)
);

CREATE TABLE authorities (
  email VARCHAR(50) NOT NULL,
  authority VARCHAR(50) NOT NULL,
  FOREIGN KEY (email) REFERENCES eni_users(email)
);

CREATE UNIQUE INDEX ix_auth_email on authorities (email,authority);
```

- Une table pour gérer les utilisateurs et une pour les rôles

Déroulement

Configuration

- Modification de la classe de configuration SecurityConfig :
    - Pour utiliser la base de données

```
package fr.eni.demo.security;

import javax.sql.DataSource;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import
org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;
import org.springframework.security.core.userdetails.User;
import org.springframework.security.core.userdetails.User.UserBuilder;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.crypto.password.PasswordEncoder;

@Configuration
@EnableWebSecurity
public class SecurityConfig extends WebSecurityConfigurerAdapter {

	// add a reference to our security data source
	@Autowired
	private DataSource dataSource;

	@Override
	protected void configure(AuthenticationManagerBuilder auth) throws Exception {
		// use jdbc authentication
		auth.jdbcAuthentication()
		.dataSource(dataSource)
		 .usersByUsernameQuery("select email,password,enabled from eni_users "
					+ "where email = ?")
		.authoritiesByUsernameQuery("select email,authority from authorities "
					+ "where email = ?");}

	@Bean
	public PasswordEncoder passwordEncoder() {  return new BCryptPasswordEncoder();  }

	@Override
	protected void configure(HttpSecurity httpSecurity) throws Exception {

		httpSecurity.authorizeRequests()
		.antMatchers("/trainers/detail*").hasAnyRole("ADMIN")
		.antMatchers("/trainers").hasAnyRole("TRAINER", "ADMIN")
		.antMatchers("/trainers/c*").hasAnyRole("ADMIN")
		.antMatchers("/resources/**").permitAll()
		.antMatchers("/**").permitAll()
		.antMatchers("/index").permitAll()
	 .anyRequest()
	 .authenticated()
	 .and()
	 .formLogin()
	 .and()
	 .logout()
	 .logoutSuccessUrl("/");
	}
}
```

- Elle déclare les 4 utilsiateurs qui peuvent être reconnu
- Elle leur donne des rôles (1 ou plusieurs)
- Elle permet de chiffrer le mot de passe
- Et elle renseigne les URL et les rôles associés

Exécution :

- Par défaut Spring fourni une page de connexion avec le module Spring Security.
    - Lancer l'application.
- Vous devez retrouver le même comportement que précédemment