Programmation Web Avancée Astuces, Javascript, React, REST

Rémi Emonet – 2016
Université Jean Monnet – Laboratoire Hubert Curien

UNIVERSITÉ JEAN MONNET SAINT-ÉTIENNE

Programmation Web Avancée 4 : Plan

- Server-side : HTTPS
- Server-side : Spring Security
- Server-side : gestion d'utilisateurs
- Javascript
- React : les bases
- Principes REST
- Serveur REST en Spring
- Client REST JSON en ReactJS
- Conclusion

Programmation Web Avancée 4 : Plan

- Server-side : HTTPS
- Server-side : Spring Security
- Server-side : gestion d'utilisateurs
- Javascript
- React : les bases
- Principes REST
- Serveur REST en Spring
- Client REST JSON en ReactJS
- Conclusion

HTTPS

- Les communication HTTP sont non-sécurisées
- Un service en ligne doit utiliser HTTPS/SSL
- Comment configurer HTTPS
  - générer certificate, au choix
    - self-signed
    - signé par https://letsencrypt.org/
    - signé par une autre autorité (e.g., video https://www.gandi.net/ssl)
  - configurer le serveur ou l'application
    - configure SSL in Spring

```
server.port=8443
server.ssl.key-store=classpath:keystore.jks
server.ssl.key-store-password=secret
server.ssl.key-password=another-secret
```

- NB : alternative
  - avoir une application HTTP
  - déployer un serveur façade qui gère la communication HTTPS (e.g., nginx ssl ou heroku, ...)

Demo ou Visite !

Programmation Web Avancée 4 : Plan

- Server-side : HTTPS
- Server-side : Spring Security
- Server-side : gestion d'utilisateurs
- Javascript
- React : les bases
- Principes REST
- Serveur REST en Spring
- Client REST JSON en ReactJS
- Conclusion

Spring Security

- `spring-boot-starter-security` dans le `pom.xml`
  - gestion de l'authentification des utilisateurs
  - restrictions d'accès
  - redirection vers une page de login si nécessaire
- Besoin de créer une class Java pour la configuration
  - extend `WebSecurityConfigurerAdapter`
  - annotée avec `@Configuration` and `@EnableWebMvcSecurity`
  - une méthode: `configure(HttpSecurity http)`
    - choisir quelles urls sont public/private
    - choisir les urls de login/logout, etc
  - configuration du back-end d'authentification
    - une méthode annotée avec `@Inject`
    - qui prends `AuthenticationManagerBuilder` comme paramètre
    - choisir où/comment sont stockés les utilisateurs
    - permet aussi de créer des utilisateurs de test
- Guide Spring : https://spring.io/guides/gs/securing-web/
- Username grâce à un paramètre de type `Authentication`

Spring Security : Configuration Typique

```java
@Configuration
@EnableWebMvcSecurity
public class WebSecurityConfig extends WebSecurityConfigurerAdapter {

  @Override
  protected void configure(HttpSecurity http) throws Exception {
     http.authorizeRequests()
            .antMatchers("/info", "/").permitAll()
            .anyRequest().authenticated() //.hasAnyRole("ADMIN")
     .and().formLogin().permitAll()
     .and().logout().permitAll();
  }

  @Override
  protected void configure(AuthenticationManagerBuilder auth) {
      auth.inMemoryAuthentication()
        .withUser("robert").password("toto").roles("USER", "ADMIN")
        .and().withUser("bob").password("toto").roles("USER");
  }
}
```

- Utilisateurs stockés en mémoire (test)
- Comprendre roles/authorities (on stack overflow)

À partir de cette configuration, 2 utilisateurs existent. Seules les pages `/` et `/info` sont accessible par les utilisateurs non connectés. Toute autre page renvoie vers un formulaire de login.

Demo ou Visite !

Spring Security: redéfinir la page login, etc

```java
...
   http.authorizeRequests()
          .anyRequest().authenticated()
     .and().formLogin().permitAll().loginPage("/MYlogin")
     .and().logout().permitAll();
...
```

- + controller to route to a thymeleaf login page

```html
    ... xmlns:th="http://www.thymeleaf.org" ...

    <!-- the POST is handle by spring -->
    <form th:action="@{/MYlogin}" method="post">
        <input name="username" type="text">
        <input name="password" type="password">
        <div><input value="Sign In" type="submit"></div>
    </form>
...
    <!-- same here -->
    <form th:action="@{/logout}" method="post">
        <input value="Sign Out" type="submit">
    </form>
```


Programmation Web Avancée 4 : Plan

- Server-side : HTTPS
- Server-side : Spring Security
- Server-side : gestion d'utilisateurs
- Javascript
- React : les bases
- Principes REST
- Serveur REST en Spring
- Client REST JSON en ReactJS
- Conclusion

Un `UserDetailsService` Maison

- Objectifs
  - remplacer la gestion en mémoire des utilisateurs
  - choisir comment et où l'on sauve dans le profil utilisateur
- Principe
  - faire une entité JPA pour les utilisateurs
  - faire une repository pour y accéder
  - implémenter un « user manager » (`UserDetailsService`)
  - et **stocker correctement les mots de passe** (`BCryptPasswordEncoder`)
  - enregistrer son `UserDetailsService` dans Spring
  - dire à Spring quel encodeur on a utilisé pour les mots de passe
  - utiliser JPA pour ajouter/modifier les utilisateurs

Demo ou Visite !

Programmation Web Avancée 4 : Plan

- Server-side : HTTPS
- Server-side : Spring Security
- Server-side : gestion d'utilisateurs
- Javascript
- React : les bases
- Principes REST
- Serveur REST en Spring
- Client REST JSON en ReactJS
- Conclusion

JavaScript : Bases

- Seul langage impératif dans le navigateur
- Surnom de ECMAScript
- Première version : 1995
- ECMAScript 6 (ES6): 17 juin 2015 ⇒
- Pas de lien (fort) avec Java
- Multi-paradigm: objet, impératif, fonctionnel, prototype
- Aussi hors du navigateur (sans DOM)
  - Rhino, Nashorn (JSR 223: Scripting for the Java Platform)
  - V8: NodeJS, MongoDB, Couchbase

JavaScript dans le Navigateur

- Avec un élément HTML `script`
- `<script></script>` avec un contenu

```
1   <script>
2     alert("Hello World.");
3   </script>
```

- `<script src="..."></script>` pour inclure un fichier `.js`

```
1   <script src="myFile.js" />
```

- Exécuté quand l'élément est parsé ⇒
- Pour expérimentations : utiliser la console du navigateur

Syntaxe et Typage JS

- Syntaxe ressemblant à C/Java
- Pas de typage statique (fonctions, variables)
- Tout est objet (y compris les types de base) `rrr = 3.14;`
- Accès dynamique au type via `typeof(rrr)`
- Types par défauts
  - `String` (ex `"hello"`, `'hello'`)
  - `Number` (ex `12`, `42.10`)
  - `Boolean` (ex `true`, `false`)
  - `Function` (ex `function (a) { return a*a; }`)
  - `Object` (pour les tableaux et les autres objets)

Variables et Portées JS

- Variables locales aux fonctions, définies avec `var`
  - comme si définie au début de la fonction
  - ≠ C/Java : JS n'a pas de portée « bloc »
- portée
  - portée fonction (= python, ≠ java)
  - portée globale, gérée par l'environnement (navigateur, ...)
- NB
  - `null` est une valeur (`typeof(null) === 'object'`)
  - variable non définie: `typeof(rrr) === 'undefined'`

Objets JS (et JSON)

- Les objets JavaScript sont des « maps »
  - les clés sont des chaînes
  - les valeurs sont de n'importe quels types
- Les objets JavaScript n'ont pas de classe (fixe)
  - possibilité d'ajouter des propriétés dynamiquement
  - possibilité d'enlever ou changer le type de propriétés
- Notation pour la création et l'accès

```
var o = {       first: 123,
                "second": 456,
                'third': {firstname: "Bob", age: "99"}        };
alert(o.first);
alert(o["first"]);
var k = "second";
alert(o[k]);
```

- Représentation JSON (JavaScript Object Notation)
  - proche de JavaScript, mais que des constantes, que des `"..."`
  - guillemets obligatoires (ex, erreur avec `first` et `'third'`)

Tableaux JS (et JSON)

- Tableaux JavaScript
  - type spécial d'objet JavaScript
  - sucre syntaxique « syntactic sugar »
  - possible en JSON
- Notation pour la création et l'accès

```
var freshFruits = ["banana", "pear", "apple", "tomato"];

alert(freshFruits[0]);
alert(freshFruits.length);

freshFruits.push("blueberries")
alert(freshFruits.length);

for (k in freshFruits) { // itère sur les clés
   alert(freshFruits[k]);
}
```


Fonctions JS

- Fonction nommée « classique » en JavaScript

```
function toto(a,b) {
   return a+b;
}
alert(toto(1,3));
```

- Fonction JavaScript : objet de première classe

```
var f = function(a) {
   return a*a;
}
alert(f(2));
var squares = [1,2,3].map(f);
```

- Fonction anonyme

```
var squares = [1,2,3].map(function(a) {
   return a*a;
});

setTimeout(function() { alert("Hello"); },
           3000);
```


Objets et Classes JS

- JS n'est pas un langage orienté classe (Java)
- JS est un langage à prototype
- Très dynamique
- Attention à `this` (≠ Java)
  - un objet « contexte »
  - pas nécessairement d'un type donné (comme la classe)
- Plus flexible que les langages à classes
- Le mot clé `class` arrive en ES6 ;-(

Programmation Web Avancée 4 : Plan

- Server-side : HTTPS
- Server-side : Spring Security
- Server-side : gestion d'utilisateurs
- Javascript
- React : les bases
- Principes REST
- Serveur REST en Spring
- Client REST JSON en ReactJS
- Conclusion

React

- http://facebook.github.io/react/
  - JavaScript Framework, By Facebook
  - Initial release: 2013
  - Version 15.3.0: Juillet 2015
- Seulement la vue ; concept de virtual DOM
- One-way data flow
  - construire la vue à partir des données (en partant de zéro)
  - la vue est un DOM virtuel
  - React calcul de « diff » et fait seulement les changements
- Se combine bien avec l'architecture FLUX
  - e.g. https://facebook.github.io/flux/docs/overview.html
  - recommandé : Redux
- Also
  - React Native: https://facebook.github.io/react-native/
  - GlReact: https://github.com/ProjectSeptemberInc/gl-react/

React : JSX, composant, propriétés, état

- JSX : mélange js et xml
- Composant (visuel ou logique) : classe ES6
- Propriétés (d'un composant), passées sous forme d'attributs
- État (d'un composant/application), `.setState`
- Utilisation de React et JSX
  - soit précompilé et packé
  - soit en mode « léger » (inclure babel et react)
- En pratique : démo

Programmation Web Avancée 4 : Plan

- Server-side : HTTPS
- Server-side : Spring Security
- Server-side : gestion d'utilisateurs
- Javascript
- React : les bases
- Principes REST
- Serveur REST en Spring
- Client REST JSON en ReactJS
- Conclusion

Principes REST : *Representational State Transfer*

Representational State Transfer

- Thèse de Roy T. Fielding (et [fr])
- Un style d'architecture
  - un ensemble de ressources
  - des liens entre ces ressources
  - comme une bonne application Web
  - Hypermedia as the engine of application state (HATEOAS)
- Propriété et objectifs
  - performance et passage à l'échelle
  - simplicité et évolutivité
- Principes
  - client-serveur, *stateless* (sans état), couches (cache, sécurité, ...)
  - ressources auto-décrites avec identifiant et métadonnées, HATEOAS
- Résumé en Français

RESTful HTTP APIs : REST sur HTTP

- Ressource : identifiant et représentation
  - URI (*Unique Resource Identifier*), `https://github.com/apache/spark`
  - métadonnées, ex : `Content-Type: text/html`, `Content-Encoding: gzip`
  - données, ex : HTML gzippé
  - NB : pas de verbes ou actions dans les URI
    - ok : `https://github.com/repositories`
    - ko : `https://github.com/createNewRepo?name=pwa`
- Protocole *stateless* : HTTP avec méthodes HTTP
  - GET : sûre (sans effets de bords)
    - récupère la représentation d'une ressource
    - liste le contenu d'une collection
  - DELETE : idempotente (même résultat si appliqué plusieurs fois)
    - supprime une ressource existante
    - supprime une collection
  - PUT : idempotente
    - remplace ou met à jour une ressource
    - remplace ou met à jour une collection
  - POST :
    - crée une nouvelle entrée dans une collection

Programmation Web Avancée 4 : Plan

- Server-side : HTTPS
- Server-side : Spring Security
- Server-side : gestion d'utilisateurs
- Javascript
- React : les bases
- Principes REST
- Serveur REST en Spring
- Client REST JSON en ReactJS
- Conclusion

Tutoriel Spring+REST+React !

Création d'un Modèle REST en Spring

- But : partager un modèle JPA en REST, au format JSON
- Étapes
  - modèle : entités JPA
  - service : repository `JpaRepository` (ou `CrudRepository`)
  - `spring-data-rest-webmvc` (dépendance maven)
    - ou à la main avec `@RestController`, `@Valid`, `RequestMethod.*`
  - (configuration : `spring.data.rest.basePath=/api`)
  - si client à une autre url : *Cross-origin resource sharing* (CORS)
- Test simplifiés

```
1   curl -X GET    http://localhost:8080/api/hotels
2   curl -X POST   http://localhost:8080/api/hotels -H "Content-Type:
application/json" -d '{"name":"ibis", "nrooms":444}'
3   curl -X GET    http://localhost:8080/api/hotels
4   curl -X GET    http://localhost:8080/api/hotels/ibis
5   curl -X DELETE http://localhost:8080/api/hotels/ibis
6   curl -X GET    http://localhost:8080/api/hotels
```

  - Let's do it ! (hotels)
  - (plugin JSONView + cfg content `application/hal+json`)

Programmation Web Avancée 4 : Plan

- Server-side : HTTPS
- Server-side : Spring Security
- Server-side : gestion d'utilisateurs
- Javascript
- React : les bases
- Principes REST
- Serveur REST en Spring
- Client REST JSON en ReactJS
- Conclusion

Accès HTTP Avec AngularJS

- Service par défaut `$http`
- Étapes
  - injecter le service `$http`
  - appeler `$http.get(...)`, `$http.put(...)`, etc.
- Exemple
- Injection de dépendance
- ⇒

REST+HTTP+JSON Avec AngularJS

- Service `$resource` fourni par le module `ngResource`
- API de haut niveau pour les services REST
- Étapes
  - inclure `angular-resource.js`
  - ajouter la dépendance vers `ngResource`
  - injecter le service `$resource`
  - instancier/personnaliser le service
- Exemples
- Let's do it (hotels)

Programmation Web Avancée 4 : Plan

- Server-side : HTTPS
- Server-side : Spring Security
- Server-side : gestion d'utilisateurs
- Javascript
- React : les bases
- Principes REST
- Serveur REST en Spring
- Client REST JSON en ReactJS
- Conclusion

Points Clés