Université de Lille

Projet Non Alternants
Systèmes de gestion de rendez-vous en ligne

S5.A.01 - Frameworks Web
BUT3 - Info
Philippe Mathieu
2024–2025

Objectifs

A la manière de prendreunrendezvous, de Doctolib ou de nombreux sites de prise de rendez-vous mis en place durant la crise `COVID-19`, l'objectif de ce projet consiste à réaliser un site internet de gestion de rendez-vous multi-utilisateurs. Le site doit permettre d'une part de montrer aux utilisateurs les créneaux libres, d'autre part de permettre aux utilisateurs de saisir et gérer leurs rendez-vous, et évidemment de n'autoriser des rendez-vous que s'ils respectent les contraintes souhaitées pour ce site.

Le site doit donc impérativement s'appuyer sur de nombreux paramètres permettant l'expression de contraintes afin de pouvoir être adapté à toutes les situations. Par exemple le planning de réservation des créneaux de piscine (avec la contrainte "pas plus de 30 personnes par heure") ou le planning de réservation de créneaux chez le médecin (avec la contrainte "pas plus d'1 personne toutes les 15mn"). C'est un framework général : on ne cherche pas un site qui permet de créer plusieurs plannings. Si on souhaite faire deux plannings avec des contraintes différentes (par exemple l'un pour la piscine, l'autre pour le médecin), on créera 2 sites WEB en changeant à l'initialisation leurs paramètres.

Ce projet se fait en binôme.
Rdv hebdomadaire : le jeudi 13h30

Semaine 1 : les outils de base

Objectif : La modélisation

1. Réfléchir à la structuration de l'application et à ses fonctionalités
2. Réfléchir aux paramètres génériques qui seront utilisés
3. Proposer un modèle conceptuel de données adapté (Power AMC ? ?)
4. Créer les tables correspondantes (et conserver le script)
5. Remplir de quelques données d'exemples pertinents (et conserver le script)
6. Tester quelques requêtes
   - Nombre de réservations aujourd'hui ?
   - Créneaux libres dans le mois ?
   - Personnes impliquées dans le créneau ?
   - etc ...

Semaine 2 : le calendrier

Objectif : L'affichage d'un calendrier avec des cases réactives

Étapes à suivre :

1. Créer un projet web MAVEN
2. La gestion des dates :
   Pour ce type de projet, il est impératif de bien maitriser le travail sur les dates, et notamment de savoir manipuler l'objet `LocalDate` et savoir obtenir le 1er jour du mois et le nombre de jours du mois avec un objet de type `LocalDate`

```
//Récupérer le jour de la semaine actuel sous forme d'entier.  Lundi = 1; Mardi = 2 ...
this.maDate.withDayOfMonth(1).getDayOfWeek().getValue();
//Récupérer le nombre de jours dans un mois
this.maDate.lengthOfMonth();
```

3. La page de base :
   Réaliser une page web qui permet d'afficher un planning mensuel (mois calendaire) en 2D (jours en colonnes). Cette page contiendra un bouton "`Précédent`" et un bouton "`Suivant`" permettant de changer de mois. On doit donc avec cette simple page dynamique pouvoir parcourir l'ensemble du calendrier.
   Pour cette première semaine on se contentera d'une servlet/jsp simple, inutile pour l'instant d'utiliser des technologies sophistiquées qui seront vues en cours les prochaines semaines.
4. La mise en place de la CSS :
   Une feuille de styles permet de fixer la taille des cases et une couleur particulière pour le mois courant. On aura surement avantage à utiliser un framework CSS genre Bootstrap ou Pure pour assurer le côté "`responsive`".
5. L'affichage dans une case :
   Chaque case affiche maintenant le numéro du jour qu'elle représente.
6. Gestion des paramètres :
   Par défaut, si aucune date n'est précisée en paramètre, la page s'ouvre sur le mois courant, sinon sur le mois passé en paramètre.
7. La réactivité des cases.
   Chaque case doit maintenant être "`réactive`" (hyperlien). En guise de test on permettra un minimum d'interactivité avec une case : chaque case est associée à un compteur de clicks qui lui est propre ; chaque click sur une case permet d'incrémenter ce compteur. La case affiche donc la date dans un coin avec au milieu la valeur de son compteur.
8. Persistance.
   Faire en sorte que les compteurs soient persistants. Si on arrête le serveur et qu'on le redémarre, les compteurs doivent toujours être corrects.
   (Ce compteur est une base de travail, il sera bien sûr remplacé par la suite par des Rendez-Vous.)

Semaine 3 : La prise de rendez-vous

On souhaite maintenant pouvoir faire des réservations à des créneaux particuliers du jour choisi. En cliquant sur une case, on ouvre maintenant le planning du jour avec les créneaux de réservation possibles selon les contraintes imposées (notamment durée du créneau et nb de personnes par créneau). La sélection d'un créneau valide doit permettre d'indiquer le créneau comme réservé de manière persistante. Il doit être possible de retourner sur n'importe quel planning de n'importe quel jour afin de voir les réservations effectuées.

Le planning général affiche maintenant dans chaque case non plus un compteur mais une synthèse du planning du jour, par exemple le nombre de créneaux encore disponibles et le nombre total de personnes pour la journée. On pourra éventuellement utiliser des codes couleur (plus ou moins rouge selon l'occupation des créneaux ou de la journée)

Rien n'est encore vérifié concernant les personnes. Il n'y a pas encore d'authentification. Du coup n'importe qui peut réserver n'importe quoi, on n'a pas encore de compte utilisateur à créer et les rendez-vous ne contiennent pas encore d'identifiant utilisateur.

3 étapes

1. Techniquement, le projet doit maintenant être transformé en un projet Maven, si possible en respectant les principes d'un MVC Web, et avec des DAO pour l'accès aux données.
2. Réfléchir aux contraintes qui seront gérées et au moyen de les coder. On considèrera au minimum :
   — titre et logo
   — jours ouverts dans la semaine (pour la piscine c'est tous les jours, pour le médecin c'est fermé le Week-End).
   — heures de début, heure de fin de journée
   — durée d'un créneau (pour le médecin c'est 15mn, pour la piscine c'est 1h ou 2h)
   — nombre de personnes acceptées par créneau (pour le médecin c'est 1, pour la piscine c'est 30)
   Il y a différentes manières de faire .... via un fichier `properties` ; via une table `parameters` de la BDD, via une procédure d'init à lancer au préalable et qui insère les données nécessaires dans les tables sur SGBD ou via un objet Java . A vous de choisir.

```java
// exemple de Bean Constraints
import java.time.LocalTime;
import java.util.ArrayList;
import java.util.Arrays;

public class Piscine implements Constraints
{
  public static ArrayList<Integer> enabledDays =
         new ArrayList<>(Arrays.asList(1,2,3,4,5,6,7));
  public static int maxPerSlot = 30;
  public static long minutesBetweenSlots = 60;
  public static LocalTime start = LocalTime.of(11, 0);
  public static LocalTime end = LocalTime.of(18, 0);
}
```

3. Vérifiez que votre site permet de gérer le scenario `medecin` et le scenario `piscine`.
4. L'ensemble doit toujours être "`responsive`" et fonctionner sur smartphone.

Semaine 4 : L'authentification

Objectif : Authentification par filtre, JPA

Le planning général anonyme reste public, mais on ne peut pas agir dessus. Quelqu'un qui n'est pas authentifié ne peut que voir les occupations sur le planning général. Il ne peut prendre aucune réservation s'il n'est pas authentifié.

On met en place deux rôles : `Admin` et `User`

— Admin : peut tout faire : afficher les plannings généraux avec les noms, gérer les créneaux, notamment en supprimer en cas de pb , etc ...
— User : voit, prend et gère ses propres rendez-vous

Un utilisateur quelconque doit pouvoir se créer un compte sur l'application avec un certain nombre de données personnelles qui seront conservées, notamment (tel, email ,.... voire photo). On lui affecte dans le

cas le rôle `User`. Les `Admin` ne peuvent qu'être rentrés directement via un accès au SGBD (en dur).

1. Tous les accès à la base passent maintenant par JPA. Ajoutez un `persistence.xml`. Vous aviez déjà les POJO, il suffit de les annoter pour assurer la persistance. Vos DAO sont maintenant supprimés et leur appel remplacé par un appel au `entityManager`.
2. L'ensemble du projet doit maintenant fonctionner avec H2 en mode `Memory`. Au démarrage du projet un fichier `data.sql` doit assurer un certain nombre d'insertions dans les tables.
3. Le login est maintenant impératif. Une fois identifié, créez un objet nommé `Principal` contenant le login et le rôle, que l'on range en session.
4. Les mots de passe sont encodés dans la base (inutile de prendre une méthode sophistiquée, md5 suffira pour l'instant)
5. A partir de la page d'accueil, un nouvel utilisateur doit pouvoir créer un nouveau compte. Un utilisateur déjà connu du système doit aussi pouvoir modifier ses infos personnelles.
6. Un utilisateur doit maintenant pouvoir voir une synthèse de ses propres rendez vous, et en annuler un si nécessaire.

Rôle `User`. Actions possibles :
— modifier son profil avec ses données personnelles
— prendre une réservation
— Lister ses propres réservations (passées et futures)
— administrer ses réservations (modifier, supprimer)

Rôle `Admin`. Actions possibles :
— Toutes les actions possibles sur les réservations de tout le monde

Semaine 5

Objectif : Passage à Spring MVC

1. L'ensemble des actions réalisées dans le projet doit maintenant uniquement passer par des controleurs. On veillera à exposer au minimum l'API Servlets dans les signatures des méthodes de composants (donc éviter au maximum `HttpServletRequest`, `HttpServletResponse`, `HttpSession` etc ...
2. L'authentification se fait toujours à l'aide d'un filtre fait "à la main". Sous Spring un filtre se déclare comme dans le TP2 , il suffit juste d'ajouter l'annotation `@ServletComponentScan` sur l'application principale (juste sous `@SpringBootApplication`)
3. La gestion du mail
   — Ajoutez la dépendance `spring-boot-starter-mail`.
   — Dans `application.properties` ajouter les caractéristiques sur serveur de mail
   ```
   spring.mail.host=smtp.univ-lille.fr
   spring.mail.port=587
   spring.mail.username=prenom.nom.etu
   spring.mail.password=xxxx
   spring.mail.properties.mail.smtp.starttls.enable=true
   ```
   — Il suffit ensuite d'injecter là où on en a besoin un objet de type `JavaMailSender` puis rédiger et envoyer le message à l'aide des méthodes associées. la methode `send` se charge d'envoyer le message. Par exemple :

```
@Autowired
private JavaMailSender sender;

MimeMessage message = sender.createMimeMessage();
MimeMessageHelper helper = new MimeMessageHelper(message);
helper.setFrom("xxx@univ-lille.fr");
helper.setTo("dest@gmail.com");
helper.setSubject("Hi");
helper.setText("How are you?");
sender.send(message);
```

— Pour l'Université ça devrait fonctionner tel quel. Pour utiliser le serveur SMTP de Gmail, il faut d'abord créer un mot de passe d'application : Une explication se trouve ici : Aller sur son compte Google, puis rubrique Securité, puis validation en 2 étapes, puis tout en bas "mot de passe des applications". Générer un nouveau mot de passe et utiliser ce mot de passe dans `application.propertie`

— Dans le champs `From` attention à bien utiliser une adresse mail correspondant au fournisseur que vous avez déclaré ! (adresse ulille avec `smtp.univ-lille.fr` , adresse google avec `smtp.gmail.com`)

4. L'upload de documents
   On souhaite notamment que les utilisateurs puissent charger leur photo dans leur profil. Pour cela on pourra s'inspirer de nombreux exemples sur internet comme celui-ci .

Semaine 7

Objectif : Un controleur REST

Pour l'organisateur, réalisez un controleur REST qui fournit aussi bien en XML qu'en JSON (selon l'entête `Accept`) la liste des rdv d'une journée avec les personnes concernées :
`http://localhost:8080/todayslist/{date}`
Cette méthode renverra une erreur si la date n'est pas au bon format.

Pour les clients, réalisez un controleur REST qui fournit aussi bien en XML qu'en JSON (selon l'entête `Accept`) la liste des rdv futurs de la personne passée en paramètre :
`http://localhost:8080/myappointments/{name}`
Cette méthode renverra une erreur si plusieurs personnes ont le même nom.

On ne s'occupera pas de l'authentification pour ces deux end-points. On considère qu'ils sont publics.