ENSSAT
LANNION

UNIVERSITÉ DE RENNES 1

# Rapport technique

-

## Simulateur de Panne en JEE

Groupe Clafoutis aux Merises


Etudiants :

BRIARD Arthur
CLAVIER Léo
LE SCANFF STEVEN
LIU Yan
ROUSSEL ARTHUR


Semestre 3 – 2ème année Informatique
Année 2016 - 2017

# Généralités

La requête HTTP du navigateur est transmise au serveur d'application. Le fichier de configuration web.xml de ce serveur permet l'interfaçage entre la ressource web demandée et le servlet. Le servlet a donc pour rôle de comprendre la requête et de générer une réponse HTTP.

# I- Simulateur de pannes

Une application web qui permet de générer à la demande des simulations de pannes. Elle est accessible depuis localhost:8080/SimulateurPanne/PanneServlet. PanneServlet utilise différentes classes et deux fichiers jsp afin de permettre à l'utilisateur de générer des pannes.

### Brève description des principaux fichiers utilisés :

- panne.jsp : Page d'accueil du module de simulation de pannes. Elle est composée de deux formulaires HTML afin de générer des pannes aux caractéristiques voulues, ou aléatoires.
- PanneDAOImpl.java : Classe comportant différentes méthodes s'interfaçant avec la base de données. Il s'agit notamment dans cette classe que s'effectue les requêtes d'insertion de pannes dans la base de donnée.

### Difficultés rencontrées :

Nous avons réalisé que l'utilisateur pouvait, s'il n'était pas guidé, mal se servir de l'outil et provoquer des pannes sur des machines existantes. Deux choix s'offraient à nous : tester la valeur entrée par l'utilisateur pour voir si elle était présente dans la base de données, ou ne pas laisser à l'utilisateur la possibilité d'entrer une machine si elle n'existe pas. Nous avons opté pour la deuxième option en intégrant au formulaire principal de simulation de panne une liste déroulante comportant le nom des machines existantes. Pour cela, nous avons dû interroger la base de données pour récupérer l'ensemble des noms des machines dans panne.jsp. Une seconde liste déroulante permet de choisir le type de panne.

Afin de créer une panne aux caractéristiques aléatoires, il nous fallait pouvoir déterminer une machine et une cause de panne aléatoirement, par un simple clic sur un bouton dans panne.jsp. Il nous a été difficile de déterminer comment traiter différemment les créations aléatoires et les créations par formulaire, sans créer d'autres fichiers supplémentaires dans le seul but de

déterminer deux fichiers au hasard. Nous avons eu l'idée de créer un deuxième formulaire dans panne.jsp, mais aux propriétés "machine" et "type" cachés, ainsi, l'utilisateur n'a accès qu'au bouton de confirmation. l'intérêt de cette méthode est que nous avons pu attribuer des valeurs par défaut à ces champs, que l'utilisateur ne pourra pas modifier car ils sont inaccessibles. La génération aléatoire de la machine à attaquer et de la panne qu'elle subit se fait donc avant l'affichage du formulaire. De ce fait, nous n'avions plus qu'à insérer comme valeur par défaut ce que nous venions de générer aléatoirement. Pour récupérer une machine aléatoirement, nous avons récupéré la liste des machines disponibles et généré un nombre aléatoire compris entre 0 et nbMachines -1. Ce numéro sert à indexer la machine à récupérer.
Par la suite, nous avons cependant opté pour une manière plus propre en utilisant un nouveau servlet NewPanneAleatoireServlet afin que la génération des nombres aléatoires se fassent seulement si on clique sur le bouton « Aléatoire ».

Nous avons aussi rencontré des problèmes afin de laisser à l'utilisateur la date et l'heure d'activation de l'attaque. En effet, il était difficile de récupérer de façon certaine une date correspondant aux standards attendus par le type SQL Date. Nous avons donc abandonné cette fonctionnalité et mis pour chaque panne la date et l'heure correspondant à l'entrée de la requête.

Lors de l'entrée de pannes dans la base de données, notamment par création aléatoire, il pouvait arriver que nous tentions de créer pour une machine une panne qu'elle possédait déjà, ce qui nous amenait à une erreur car ces deux paramètres constituent la clé primaire de la table panne. De ce fait, cela rendait la poursuite du traitement impossible. Pour y remédier, nous avons pris la décision que lorsque cette situation se produisait, la nouvelle panne écrasait la précédente. Il s'avère que cette situation permet de se débarrasser dans notre base de données de pannes déjà résolues sans plus de complications.

Enfin, nous avons eu de grosses difficultés à faire en sorte que certaines machines ne puissent pas être affectées par certains types de pannes. Nous avons vite déterminer qu'il fallait interroger la base de données pour obtenir le type d'une machine en fonction de son nom, puis de tester cette valeur afin de pouvoir ou non accepter la requête. Cependant, des erreurs que nous n'avions pas à résoudre ont fait que nous n'avons pas pu implémenter cette fonctionnalité.

# II- Console de monitoring

La console de monitoring est accessible depuis le navigateur en recherchant la page localhost:8080/SimulateurPanne/MainServlet. MainServlet utilise plusieurs classes java et fichiers jsp pour générer une page dynamique affichant les pannes enregistrées dans une base de donnée.

## Un brève description de quelques-uns des fichiers utilisés :

- monitoring.jsp : c'est la page d'accueil de ce module. Elle présente sous le format d'un tableau HTML le nombre de pannes apparues à divers moments.

- PanneDAOImpl.java : est une classe qui propose des différentes méthodes s'interfaçant avec la base de donnée. Par exemple, une de ces méthodes effectue une requête SQL retournant les enregistrements contenus dans la table « panne ». Ces enregistrements sont traités et reformatés pour former une liste. Enfin, cette liste est retournée à l'entité appelante.

- PanneServiceImpl.java : est une classe faisant le lien entre les méthodes brutes en lien avec la base de donnée et les informations plus générales dont peuvent avoir besoin les servlets. Par exemple, lorsqu'un servlet veut afficher et organiser les pannes dans un tableau, elle appelle une unique méthode contenue dans cette classe. Celle-ci a pour but de choisir quelles méthodes de PanneDAOImpl.java sont adéquates.

## Difficultés rencontrées :

Rechercher les données pertinentes dans la base de donnée n'a pas été un gros problème. La difficulté a été d'actualiser ces données sans recharger la page. Pour cela, nous nous sommes servi de Jquery. Jquery permet entre autre d'afficher le contenu entier d'une page B, à l'intérieur d'une page A. Finalement, la page d'accueil du module de monitoring propose 4 boutons à l'utilisateur, ces boutons ont pour effet d'afficher le contenu de la page details.jsp. Le contenu de cette page change en fonction du bouton à l'origine de la demande.

Le sujet de projet nous a imposé l'affichage d'une checkbox représentant l'état de la panne (réparée ou non réparée). Nous avons eu l'idée d'une fonctionnalité supplémentaire permettant de simuler une réparation des pannes à partir de l'écran de monitoring. Celle ci a consommé une partie non négligeable du temps qu'il nous a été alloué pour ce projet. Chacune des cases *réparée/non réparée* est liée à un formulaire. Il est possible pour l'utilisateur d'envoyer le résultat de ce formulaire grâce à un bouton. Les données émises sont envoyées à un servlet utilisant le service de mise à jour de la base de donnée. Après un délai, l'utilisateur est redirigé vers la page

d’accueil du monitoring où il peut constater que ses modifications ont été prises en compte (il y a malheureusement un rechargement de la page).

Un délai supplémentaire pour terminer le projet nous aurait permis de produire une amélioration dont nous tenons à cœur : le rafraîchissement automatique de l’intégralité du module de monitoring. Pour l’instant, seul l’affichage des détails se fait sans rechargement de la page.