Projet LO54 - A2018

Sommaire

Introduction

Le projet consiste à créer un site web afin de gérer l'offre de formation dans une école privée. Un catalogue est édité pour lister les formations disponibles. Les dates de sessions prévues pour chacune de ces formations sont également renseignées. Le site doit pouvoir permettre à un utilisateur de filtrer la liste des formations soit par un mot clé (contenu dans le titre de la formation), soit par les sessions disponibles à une date donnée, ou alors en fonction du lieu de la session (lieu affiché à partir d'une liste déroulante).
Pour notre cas spécifique, la technologie ajoutée pour la réalisation du projet est Javamail pour envoyer un mail de confirmation d'inscription à un cours.

I. Présentation de la technologie utilisée

Javamail est l'API standard de gestion des courriers électroniques de J2EE. Les spécifications de l'API sont définies, mais elle n'est pas fournie avec le JRE, ce qui offre la possibilité pour de tierces parties de fournir leur propre implémentation. Ceci dit, Sun fournit aussi une mise en oeuvre de référence de Javamail, qu'il est possible de télécharger à partir de l'adresse https://www.oracle.com/technetwork/java/javamail/index-138643.html.

Envoyer un message avec javamail n'est pas très compliqué. Néanmoins, il est nécessaire de comprendre un certain nombre de concepts clés. Nous allons donc détailler au fur et à mesure le code.

II. Tutoriel d'installation

1. Ajout de la dépendance dans un projet Maven

Pour utiliser Javamail, il est d'abord nécessaire d'ajouter la dépendance de cet outil dans le fichier pom.xml de notre projet Maven, la dernière version Javamail est 1.6.2.

```
<dependency>
    <groupId>com.sun.mail</groupId>
    <artifactId>javax.mail</artifactId>
    <version>1.6.2</version>
</dependency>
```

Figure 1 - Dépendance Maven dans pom.xml

2. Utilisation de JavaMail

A. Le protocole

Plusieurs protocoles sont pris en charge par l'API Javamail:

- SMTP : Acronyme de Simple Mail Transport Protocol. Ce protocole permet l'envoi de mails vers un serveur de mails qui supporte ce protocole.

- POP3 : Acronyme de Post Office Protocol. Ce protocole permet la réception de mails à partir d'un serveur de mails qui implémente ce protocole.

- IMAP : Acronyme de Internet Message Acces Protocol. Ce protocole permet aussi la réception de mails à partir d'un serveur de mails qui implémente ce protocole.

- *NNTP : Acronyme de Network News Transport Protocol. Ce protocole est utilisé par les forums de discussions (news).*

Le seul protocole pour l'envoi de mail est SMTP, c'est donc celui-ci que l'on va utiliser dans le cadre du projet.

B. SMTP

Pour envoyer un e-mail via SMTP, il faut suivre les étapes suivantes :

- Instancier un objet Session en définissant les variables d'environnement nécessaires
- Instancier un objet Message
- Mettre à jour les attributs utiles du message (Objet et corps du message)
- Appeler la méthode send() de la classe Transport

Création d'une classe qui comportera la méthode qui sera par la suite utilisée pour l'envoi de mail dans un main() ou pour notre cas dans un servlet.

```
public class EmailUtil {
    /**
     * Utility method to send sim
     *
     * @param toEmail
     * @param subject
     * @param body
     */
    public static void sendEmail(
```

Figure 2 - Classe EmailUtil

La méthode sendEmail() permettant l'envoie d'un mail sera utilisé directement dans notre servlet "Inscription" permettant l'inscription à une session d'un cours : l'utilisateur qui s'inscrit recevra un mail de confirmation d'inscription grâce à JavaMail.

1. Session

Tout d'abord, toute application utilisant l'API javamail doit commencer par ouvrir une session. Cette classe encapsule les données liées à la connexion.

C'est donc à partir de cet objet que toutes les actions concernant les mails sont réalisées. Les paramètres nécessaires sont fournis dans un objet de type Properties et de type Authenticator. Une session peut être unique ou partagée par plusieurs entités.

La méthode getInstance permet la création d'une session unique.

```
Session session = Session.getInstance(props, auth);
```

Figure 3 - Création d'une session

Pour utiliser l'objet Session, il est nécessaire de faire un import :

```
import javax.mail.Session;
```

Figure 4 - Import Session

Pour créer une session, 2 paramètres sont nécessaires :

- un objet Properties qui contient les paramètres d'initialisation. Un tel objet est obligatoire.
- un objet Authenticator optionnel qui permet d'authentifier l'utilisateur auprès du serveur de mails.
-

a) L'objet Properties

L'objet Properties contient les propriétés d'environnement utilisés par JavaMail, il est nécessaire de définir certaines propriétés pour utiliser le protocole SMTP :

=> mail.smtp.host va prendre en valeur le serveur auquel se connecter, dans notre cas on a décidé d'utiliser gmail, donc mail.smtp va prendre la valeur smtp.gmail.com.
=> mail.smtp.port c'est le port du serveur de connexion : la valeur par défaut est 25 hors le port dont nous avons besoin est le 465 ou le port 587 : tout dépend de ce que nous choisissons ssl ou starttls.
=> mail.smtp.ssl.enable (si port 465) ou mail.smtp.starttls.enable (si port 587) : booléen
=> mail.smtp.auth : comporte par défaut le booléen false

mail.smtp.host	smtp.gmail.com
mail.smtp.port	465 ou

	587
mail.smtp.auth	true
mail.smtp.ssl.enable (si port 465) ou mail.smtp.starttls.enable (si port 587)	true

```
Properties props = new Properties();
props.put("mail.smtp.host", "smtp.gmail.com"); //SMTP Host
props.put("mail.smtp.port", "587"); //TLS Port
props.put("mail.smtp.auth", "true"); //enable authentication
props.put("mail.smtp.starttls.enable", "true"); //enable STARTTLS
```

Figure 5 - Définition de l'objet Properties

Importation nécessaire pour l'utilisation de cet objet :

```
import java.util.Properties;
```

Figure 6 - Importation de Properties

b) L'objet Authenticator

L'objet Authenticator permet l'authentification de l'utilisateur par qui le mail sera envoyé. Pour utiliser cette classe, il est nécessaire de créer une classe fille pour récupérer l'adresse mail et le mot de passe de l'utilisateur.

```
final String fromEmail = "inscription.courslo54@gmail.com"; //gmail address mail
final String password = "projetecoleLO54"; // password of gmail address to send email
```

Figure 7 - Récupération de l'adresse mail et du mot de passe de l'utilisateur

La méthode de classe Authenticator utilisé pour obtenir ces informations est getPasswordAuthentication() renvoyant un objet PasswordAuthentication contenant ces informations.

```
Authenticator auth = new Authenticator() {
    @Override
    protected PasswordAuthentication getPasswordAuthentication() {
        return new PasswordAuthentication(fromEmail, password);
    }
};
```

Figure 8 - Définition de l'objet Authenticator

Des importations sont nécessaires pour cet objet :

```
import javax.mail.Authenticator;
import javax.mail.PasswordAuthentication;
```

Figure 9 - Import pour l'objet Authenticator

2. Message

Ici, nous utilisons un message MimeMessage qui est une classe fille de la classe Message qui comporte plusieurs méthodes permettant l'initialisation des données du message qui vont être utiles à l'écriture de notre mail.

Il faut donc créer un objet MimeMessage avec en paramètre la session précédemment créer

```
MimeMessage msg = new MimeMessage(session);
```

Figure 10 - Définition de l'objet MimeMessage

Il faut ensuite initialiser les données du message :

```
msg.setFrom(new InternetAddress(fromEmail, "Inscription Cours"));
msg.setSubject(subject);
msg.setText(body);

msg.addRecipients(Message.RecipientType.TO, InternetAddress.parse(toEmail, false));
```

Figure 11 - Ajout des données du message

- La méthode setFrom() permet de préciser dans le mail le mail et le nom de la personne ayant envoyé le mail. Un objet InternetAddress est nécessaire pour chaque émetteur et destinataire du mail, il permet de définir une adresse mail et le nom de l'expéditeur (ici :"Inscription Cours").
- La méthode setSubject() permet d'ajouter l'objet du mail à envoyer, ce texte sera un objet String contenu en paramètre de cette méthode.
- La méthode setText() permet d'ajouter le corps, le contenu du mail à envoyer, ce texte sera un objet String contenu en paramètre de cette méthode.

- La méthode addRecipient() permet d'ajouter un destinataire et le type d'envoi. Le type d'envoi est précisé grâce à une constante pour chaque type :
  - destinataire direct : Message.RecipientType.TO
  - copie conforme : Message.RecipientType.CC
  - copie cachée : Message.RecipientType.BCC

Dans notre cas, le destinataire est direct nous allons donc utiliser Message.RecipientTO.
Pour pouvoir utiliser cette classe, il est nécessaire de faire des imports des classes nécessaires :

```
import javax.mail.internet.MimeMessage;
import javax.mail.Message;
import javax.mail.Transport;
import javax.mail.internet.InternetAddress;
```

Figure 12 - Import pour l'utilisation de MimeMessage

3. Transport

Cette classe permet de réaliser l'envoi du message grâce à sa méthode send() contenant en paramètre le message à envoyer, cette méthode ouvre et ferme la connexion à chaque appel.

```
Transport.send(msg);
```

Figure 13 - Transport du message

Pour pouvoir utiliser cette classe, il est nécessaire de faire un import :

```
import javax.mail.Transport;
```

Figure 14 - Importation de la classe Transport

```
public class EmailUtil {
    public static void sendEmail(String toEmail, String subject, String body) {

        final String fromEmail = "inscription.courslo54@gmail.com"; //gmail address mail
        final String password = "projetecoleLO54"; // password of gmail address to send email

        try {

            Properties props = new Properties();
            props.put("mail.smtp.host", "smtp.gmail.com"); //SMTP Host
            props.put("mail.smtp.port", "587"); //TLS Port
            props.put("mail.smtp.auth", "true"); //enable authentication
            props.put("mail.smtp.starttls.enable", "true"); //enable STARTTLS

            Authenticator auth = new Authenticator() {
                @Override
                protected PasswordAuthentication getPasswordAuthentication() {
                    return new PasswordAuthentication(fromEmail, password);
                }
            };

            Session session = Session.getInstance(props, auth);

            MimeMessage msg = new MimeMessage(session);

            msg.setFrom(new InternetAddress(fromEmail, "Inscription Cours"));
            msg.setSubject(subject);
            msg.setText(body);

            msg.addRecipients(Message.RecipientType.TO, InternetAddress.parse(toEmail, false));


            Transport.send(msg);

        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

Figure 15 - Code classe EmailUtil complet

4. Utilisation de la méthode créée

Enfin, il ne reste qu'à utiliser la méthode créée : sendEmail(String toEmail, String subject, String body) contenant en paramètre l'adresse du destinataire du mail, le sujet du mail et le corps.

Dans notre cas : on utilise cette méthode dans notre servlet Inscription :

```
EmailUtil.sendEmail(email,object,body);
```

Figure 16 - Utilisation de la méthode sendEmail

grâce à l'import du package contenant la classe et la méthode précédemment créée :

```
import fr.utbm.projetlo.util.EmailUtil;
```

Figure 17 - Importation de la classe EmailUtil

III. Retour d'expérience

Ce projet nous a permis non seulement d'acquérir des compétences sur les applications web Java Entreprise mais aussi sur la capacité d'intégrer et de développer avec différents frameworks Java Entreprise.

Javamail a été correctement implémenté en utilisant un compte Gmail, les mails envoyés à l'utilisateur s'inscrivant à un cours sont complets avec un mail comportant un objet et le message comportant le nom du cours et les dates de la session.

Nous avons, grâce à ce projet, pu comprendre, comment se déroule l'envoie de mail via l'API javamail à travers la session, le message et le transport.

La principale difficulté rencontré est l'utilisation de NetBeans sur un mac.

Les tâches étaient bien réparti pour le codage et le rapport a été géré ensemble via un google docs créé.

Conclusion

Nous en avons maintenant terminé de ce tour d'horizon de Javamail. Vous l'avez constaté, l'API est très facile à appréhender et aucune configuration fastidieuse n'est requise avant de pouvoir transmettre un message.

Nous avons ici utilisé Javamail pour transmettre un message mais cet API ne se réduit pas à cette fonctionnalité. En effet, il est aussi possible de consulter sa messagerie avec javamail à l'aide d'autres protocoles que SMTP comme IMAP. Il serait possible pour ce projet de catalogue de cours d'ajouter une partie utilisateur (avec connexion) récupérant les mails de l'utilisateur connecté et les afficher dans cette partie utilisateur grâce aux autres protocoles de Javamail.

Bibliographie

https://javaee.github.io/javamail/

https://www.jmdoudoux.fr/java/dej/chap-javamail.htm

https://docs.oracle.com/

https://atatorus.developpez.com/