2JAVA

Documentation Technique

ISTORE

Ryan Dordain & Séraphin Dubail

Binary

Table des matières

[I. Gestion des données 2](#_Toc126830745)

[Schéma de la base de données : 2](#_Toc126830746)

[Sécurité : 3](#_Toc126830747)

[II. Affichage Graphique 4](#_Toc126830748)

[III. Fonctionnement général 6](#_Toc126830749)

[Class User 6](#_Toc126830750)

[Class WhiteMail 7](#_Toc126830751)

[Classes Shops et Items 8](#_Toc126830752)

# Gestion des données

Pour des raisons de portabilité et de sécurité, nous avons choisis d’héberger notre base de données sur un serveur distant.

L’avantage est que grâce à cette solution, tous les appareils disposant du logiciel IStore auront accès aux données peu importe leur emplacement mais l’inconvénient est que ces appareils doivent obligatoirement être connectés à internet.

Les codes d’accès à la DB sont dans le fichier main.java

## Schéma de la base de données :

Notre base de données est divisée en 4 tables :

-items

-shop

-users

-white\_list

Voici un la composition détaillée des tables de la DB :

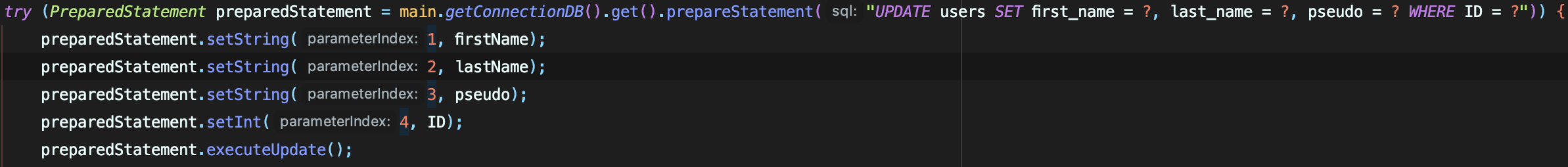
Une image contenant table

Description générée automatiquement

## Sécurité :

Pour sécuriser au maximum les données des utilisateurs, les mots de passes sont hachés dans le fichier *PassewordHasher.java,* pour y parvenir, nous avons utilisés la librairie native de java « Security ».

De plus, nous avons également utilisé des requêtes préparées afin d’éviter les injections SQL évitant ainsi de forcer l’entrer ou la suppression de données dans la base de données.



Requête préparée

Pour finir, nous avons un système de vérification des conditions très poussé afin d’éviter au maximum que l’utilisateur puisse faire ce qu’il veut ou qu’il ne puisse pas envoyer d’informations incomplètes. Par exemple quand un utilisateur essaye de créer un compte :

* On vérifie que l’adresse mail qu’il renseigne est dans la white List
* On vérifie que l’adresse mail n’est pas déjà utilisée
* On vérifie que l’adresse mail contient au minimum 12 caractères, un "@" et un "."
* On vérifie que le nom fait au moins 3 caractères
* On vérifie que le prénom fait au moins 3 caractères
* On vérifie que le pseudo fait au moins 6 caractères
* On vérifie que les deux mots de passes entrés font au moins 8 caractères
* On vérifie que les deux mots de passes entrés sont les mêmes

Et seulement si toutes ses conditions sont remplies, alors l’utilisateur va pouvoir valider et créer son compte. Si il valide alors qu’il ne rempli pas une des conditions, un message d’erreur précisant ce qu’il manque s’affichera.

Une image contenant texte

Description générée automatiquement

# Affichage Graphique

Pour réaliser la partie graphique de notre logiciel, nous avons utilisés la librairie native de java «Swing ». Son intégration native à notre IDE intelliJ permet d’organiser l’affichage graphique sans grande difficulté notamment grâce à la visualisation en temps réel de notre affichage.

L’instanciation des fenêtres de fait dans le fichier *Display.java*

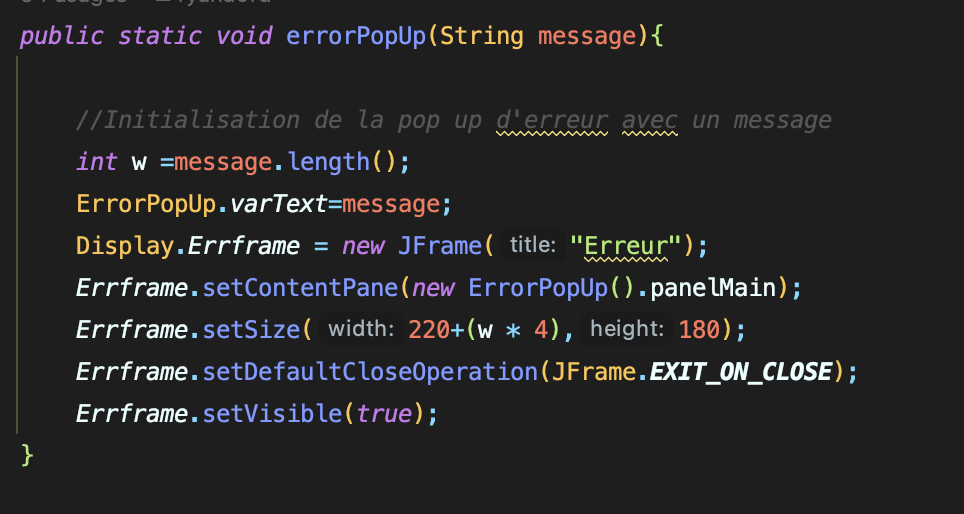
Une image contenant texte

Description générée automatiquement

Instanciation de la fenêtre

Nous avons instancié une JFrame par défaut à laquelle nous venons modifier le contenu à afficher ainsi que la taille de la fenêtre en fonction des besoins.

Nous avons procédé de la même manière pour générer une fenêtre popup que l’utilisateur va fermer à chaque fois : si besoin, on appelle la fonction *errorPopUp* avec en paramètre le message qu’on souhaite afficher. Elle définit au passage sa taille en fonction du message à afficher.



Fonction errorPopUp

Afin d’afficher le contenu de tableaux, nous avons créé des modèles de table pour chaque tableau. Nous en avons donc un pour les items, un pour les shops, un pour les utilisateurs et un pour les emails en whitelist.

Une image contenant texte

Description générée automatiquement

# Fonctionnement général

## Class User

Lors de la connexion de l’utilisateur, nous allons récupérer tous ses informations comme son nom et prénom, son rôle ou encore son magasin assigné s’il est vendeur et les stocker dans un objet user.

Pour simplifier l’utilisation de l’objet User et le sécuriser, nous lui avons ajoutés plusieurs Getter et setters.

Une image contenant texte

Description générée automatiquement

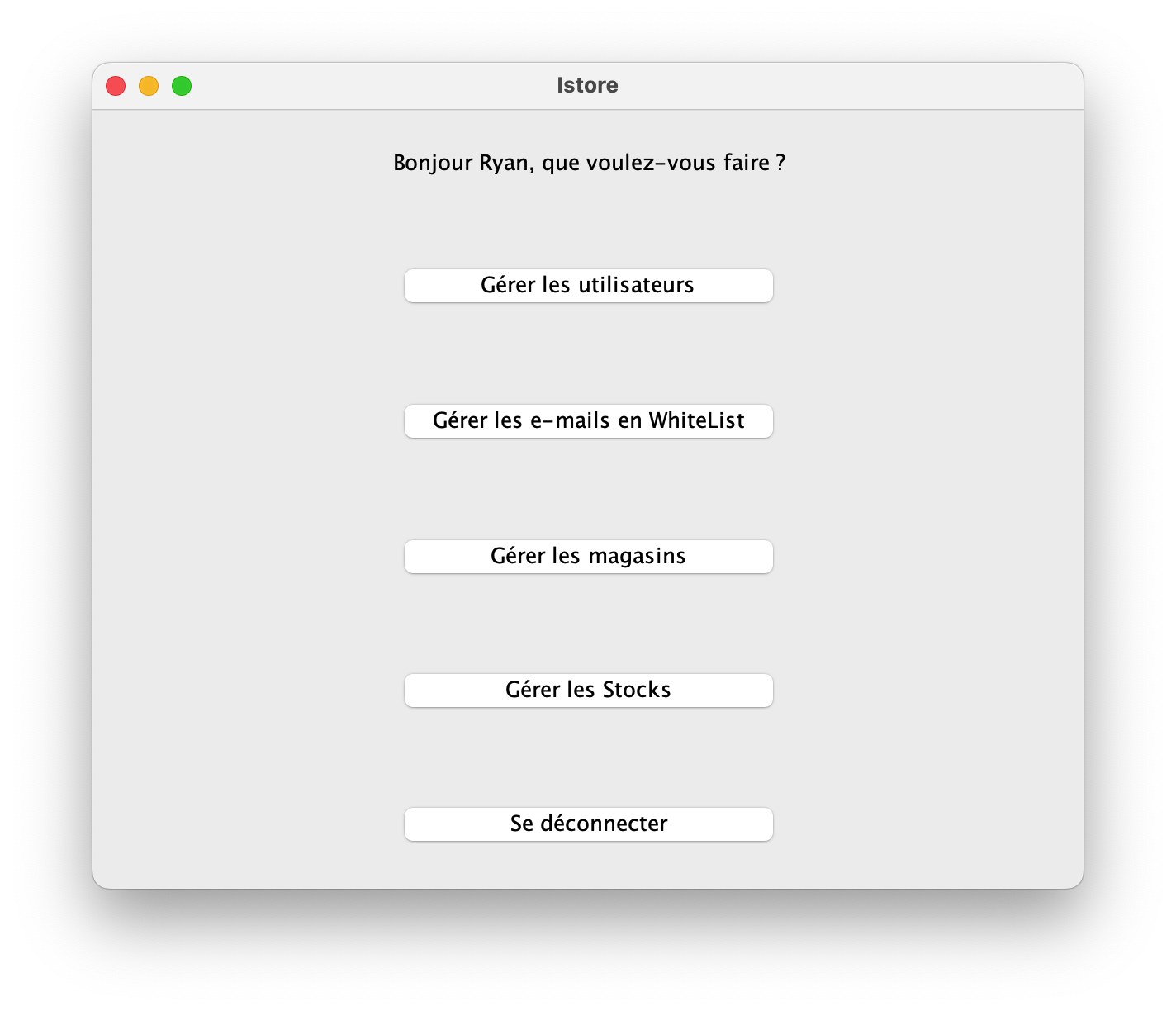
Les getters d'un User

Ils peuvent par exemple être appelés pour l’affichage du nom de l’utilisateur dans un message de bienvenue lorsque celui-ci se connecte.





Exemple d'utilisation d'un getter



Affichage sur la fenêtre d'accueil

# Class WhiteMail

Pour pouvoir manipuler les adresses mails sur liste blanche, nous avons mis en place une classe dédiée.

Ceci nous permet par exemple d’ajouter ou de supprimer une adresse mail de notre base de données.

Une image contenant texte

Description générée automatiquement

Insertion de mail

Une image contenant texte

Description générée automatiquement

Suppression de mail

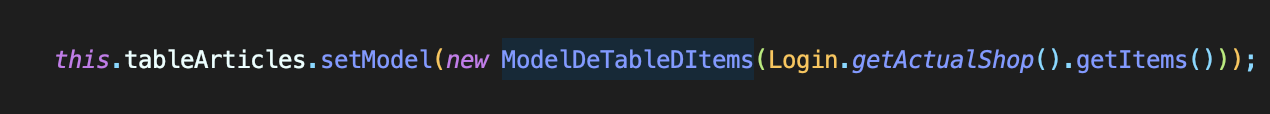
# Classes Shops et Items

Nous avons suivi le même procédé pour la gestion des stocks et des magasins. Ceux-ci sont groupés sous forme de liste pour simplifier leurs affichages sous forme de tableau dans l’interface utilisateur

Une image contenant texte

Description générée automatiquement

Getter Items



Utilisation des Getters pour générer une liste d'items