

Spritebot-v3

Programmer-Handbook

v.1.0.0



PAGE DE SERVICE

Référence : Spritebot

Plan de classement : stadium-technic-analyse-conception-spritebot

Niveau de confidentialité : confidential

Mises à jour

Version	Date	Auteur	Description du changement
1.0.0	20-10-2023	Thibaud MARCQ	Introduction, Objectifs, Architecture du code
2.0.0	18-03-2024	Enzo MARION	Réalisation du code sous Android Studio

Validation

Version	Date	Nom	Rôle

Diffusion

Version	Date	Nom	Rôle

SOMMAIRE

INTRODUCTION	1
1 OBJECTIFS	2
2 REALISATION DU CODE EN JAVA	2
2.1 Utilisation de Figma	2
2.2 ARCHITECTURE DU PROGRAMME	2
2.2.1 PACKAGE CONTROL	2
2.2.2 PACKAGE MODEL	2
2.2.3 PACKAGE VIEW	2
2.2.4 PACKAGE DAO	2
2.2.5 PACKAGE TOOLS	3
2.3 PROGRAMME PRINCIPAL	3
2.3.1 CLASSE CONTROLLER	3
2.3.2 CLASSES MODEL	3
2.3.3 CLASSES VUES	4
2.4 CAS D'UTILISATION « SE CONNECTER »	5
2.5 CONNEXION A LA BASE DE DONNEES	7
3 Bibliothèque utilisée :	8
3.1 MISE EN PLACE D'UN WEB SOCKET	9
3.1.1 QU'EST-CE QU'UN WEB SOCKET	9
3.1.2 LES DIFFERENCES ENTRE WEB SOCKET ET API	9
3.2 Utilisation de JBCrypt	9
3.3 Utilisation de Passay	10
3.4 Utilisation de Swing	10
3.5 Utilisation de Jasypt	11
3.6 Utilisation de JfreeChart	11
4 CONCLUSION	12
5 INDEX DES ILLUSTRATIONS	13

1. INTRODUCTION

Ce projet, nommé Spritebot, a pour objectif de fournir une plateforme ludique permettant aux utilisateurs d'améliorer leurs compétences en matière de sécurité informatique à travers des activités interactives. Cette documentation détaille les différentes phases de conception et de développement du jeu, en mettant particulièrement l'accent sur la réalisation du code sous l'environnement Android Studio. Elle offre également un aperçu de l'architecture du programme, des différentes classes et packages utilisés, ainsi que des technologies employées telles que les WebSockets. En somme, elle constitue un guide exhaustif pour comprendre et mettre en œuvre le projet Spritebot.

1 OBJECTIFS

Les objectifs sont de créer un jeu permettant aux employés de Vinci de s'améliorer en sécurité de l'informatique.

2 REALISATION DU CODE EN JAVA

La réalisation du code qui permettra de remplir les objectifs et de créer un jeu de quizz en application se fera tout d'abord en langage JAVA sur l'IDE d'Eclipse. On commencera par la structuration de notre programme avant de passer à l'écriture du code où l'on s'intéressera à la conception de frame, de connexion à une BDD et la protection lors de la connexion de l'utilisateur.

2.1 UTILISATION DE FIGMA

Nous avons alors créé une maquette sur figma qui permet de voir à quoi devrait ressembler notre jeu à la fin :

<https://www.figma.com/file/fUUMmTdekrvVylljadEPfR/Untitled?type=design&node-id=0-1&mode=design>

Ici la maquette montre plusieurs frame, la première correspond au Login du collaborateur de chez Vinci, ensuite on a une frame qui montre deux mode, monoplayer et multiplayer, en monoplayer on joue seul avec des série de question ; Cependant en multiplayer le joueur arrive dans une frame de sale d'attente, puis une fois la game lancé il joue normalement comme en solo et ensuite le joueur arrive sur une frame de récapitulation de la game avec le rank lors de la game le nom et le rank générale des participants.

2.2 ARCHITECTURE DU PROGRAMME

Tout d'abord, on va structurer notre programme, afin qu'il soit plus lisible à lire et à comprendre. Pour cela, nous diviserons en plusieurs packages les différentes classes en fonction de leurs rôles dans le code. Nous verrons en quoi consiste ces packages et leurs classes associées. L'architecture utilisée est sous format MVC (Model-View-Controller) et regroupera donc principalement ces trois packages.

2.2.1 PACKAGE CONTROL

Le package Control est celui qui sera au cœur du code dans ce package, se trouvera les classes Start et Controller. L'une servant à instancier l'application et l'autre permettant de contrôler les classes provenant d'autres packages. On note aussi la présence de la classe Configuration qui aura pour but de crypter les données présentes dans le fichier de configuration.

2.2.2 PACKAGE MODEL

Le package Model quant à lui contiendra les classes Answer, Player, Question et QuizGame qui sont comme des entités que l'on trouvera dans la base de données. Ce sont ces classes qui font en sorte que ces entités existent dans le programme au niveau Java.

2.2.3 PACKAGE VIEW

Le package View contiendra toutes les classes faisant offices de frame ou d'interface graphique, on y retrouve les classes Login, ChangePassword, GameStart, QuizGameGUI et RecapGame. Le code de ces classes se traduira par ce que l'on retrouvera lors du lancement de l'application.

2.2.4 PACKAGE DAO

Le package DAO servira à établir la connexion avec la base de données, on y retrouve trois classes, DAOsqlAnswer, DAOsqlQuestion et DataFileUser. Toutes ces classes auront pour but de se connecter à la base de données afin de récupérer l'identifiant de l'utilisateur, les questions ainsi que leurs réponses.

2.2.5 PACKAGE TOOLS

Le package Tools ne contiendra que la classe BCrypt, une classe utilisée pour le cryptage de données, c'est ce qu'on utilisera pour sécuriser la connexion utilisateur afin d'empêcher toutes tentatives de vols de données.

2.3 PROGRAMME PRINCIPAL

Maintenant que nous avons une vision de l'architecture que prendra notre code, nous pouvons maintenant nous intéresser ce en quoi consistera le programme que nous coderons en Java. Comme dit précédemment, les classes seront créés afin de remplir un rôle, par exemple, la classe Questions va permettre l'instanciation de variables questions. Ou encore la classe DataFileUser qui permettra de vérifier si le login et le password de l'utilisateur sont correcte avant de pouvoir permettre la connexion à l'application.

2.3.1 CLASSE CONTROLLER

Le controller est une classe essentielle dans notre code car c'est lui qui va interagir avec toutes les classes sources. Il permet notamment la liaison entre les classes Model et View.

2.3.2 CLASSES MODEL

Pour les classes Model, on retrouvera le même schéma, c'est-à-dire, tout d'abord, une phase de spécifications des variables de classe. Si on prend l'exemple de la classe Answer, le code ressemble à l'image ci-dessous.

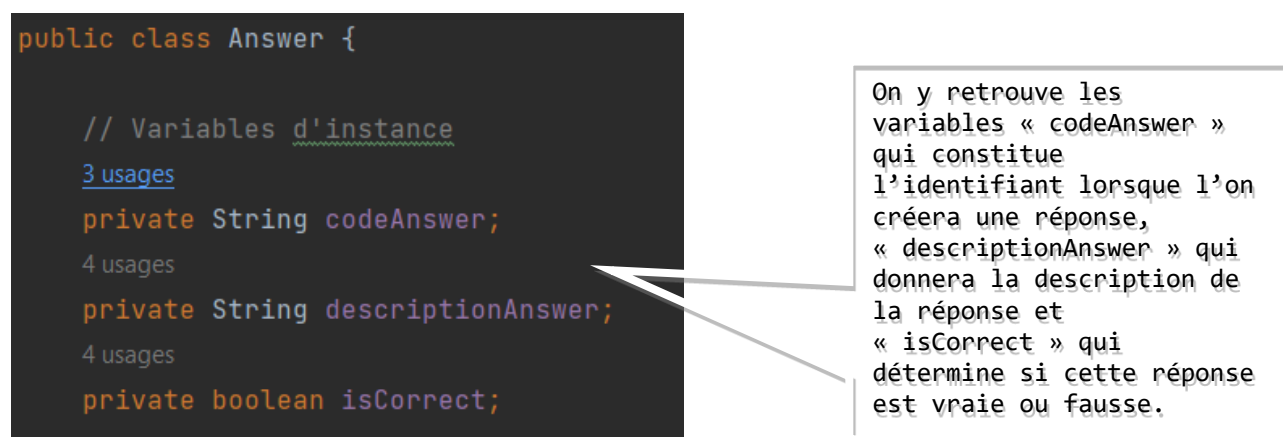


Figure 1 : Partie spécification de la classe Answer

S'ensuit la mise en place d'un constructeur, on y mettra en paramètres les variables « descriptionAnswer » et « isCorrect ».

```
// Constructeur  
public Answer(String descriptionAnswer, boolean isCorrect) {  
    this.descriptionAnswer = descriptionAnswer;  
    this.isCorrect = isCorrect;  
}
```

Figure 2 : Partie Implémentation (Constructeur) de la classe Answer

Puis pour terminer, on y ajoute aussi les getters et setters, qui nous seront utiles lorsque l'on voudra obtenir les variables à partir d'autres tables.

On fait un getter / setter par variables créés dans la classe, donc toutes les variables que l'on peut trouver dans la partie spécifications.

```
// Getter pour la description de la réponse
public String getDescriptionAnswer() {
    return descriptionAnswer;
}

// Setter pour la description de la réponse
public void setDescriptionAnswer(String descriptionAnswer) {
    this.descriptionAnswer = descriptionAnswer;
}

// Getter pour savoir si la réponse est correcte
public boolean getIsCorrect() {
    return isCorrect;
}

// Setter pour définir si la réponse est correcte
public void setCorrect(boolean correct) {
    isCorrect = correct;
}

// Getter pour le code de la réponse
public String getCodeAnswer() {
    return codeAnswer;
}

// Setter pour définir le code de la réponse
public void setCodeAnswer(String codeAnswer) {
    this.codeAnswer = codeAnswer;
}
```

Figure 3 : Partie Implémentation (Getter & Setter) de la classe Answer

Une fois ces trois bouts de codes réalisés, on peut délaissier les classes Model afin de se concentrer sur la partie graphique du code.

2.3.3 CLASSES VUES

Les classes que l'on mettra dans le package **View** sont celles qui permettront d'afficher l'interface graphique de l'application.

Tout comme les autres classes, il faut spécifier les variables, en différence que nous utilisons Swing et que cela nous demande de créer des variables comme **JPanel** ou **JButton**.

```
public class GameStart extends JFrame {

    // Contrôleur associé à la vue
    private Controller myController;

    // Panneau de contenu
    private JPanel contentPane;
```

Figure 4 : Partie spécifications de la classe GameStart

On se sert ici de la classe Vue « GameStart » qui permet de lancer le jeu une fois sur la page principale du programme. On spécifie donc que nous avons besoins du Controller mais aussi de créer un Panel (ou JPanel) que l'on nommera « contentPane ».

S'ensuit donc la partie constructeur avec l'initialisation des variables spécifiées.

```
// Constructeur
public GameStart(Controller unController) {
    // Initialise la vue avec le contrôleur spécifié
    this.myController = unController;

    // Paramètres de la fenêtre
    setResizable(false);
    setIconImage(Toolkit.getDefaultToolkit().getImage("img\\sb-logo-monogram-circle.jpg"));
    setTitle("Sprite bot");
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setBounds(100, 100, 300, 200);

    // Panneau de contenu
    contentPane = new JPanel();
    contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));
    setContentPane(contentPane);
    contentPane.setLayout(null);

    // Bouton "Start"
    JButton btnStart = new JButton("Start");
    btnStart.setBounds(160, 82, 89, 23);
    contentPane.add(btnStart);

    // Ajout d'un écouteur d'événements pour le bouton "Start"
    btnStart.addActionListener(new ActionListener() {
```

Figure 5 : Partie implémentation de la classe GameStart

Ce qui change d'avec les autres classes, c'est que nous n'avons pas besoin de getters & setters mais qu'il faut paramétrer les éléments Swing. On peut voir sur l'image ci-dessus que nous paramétrons certaines valeurs avec la JFrame comme la taille, le titre, les bordures, le fait de pouvoir changer sa taille, l'ajout d'une icône et l'opération de fermeture par défaut.

En plus, nous avons le paramétrage du JPanel, puis l'ajout d'un bouton que l'on nomme « Start » et qui servira à lancer le jeu une fois cliqué, amenant à écrire un **ActionListener** dans notre code, comme celui que l'on voit en dessous.

```
public void actionPerformed(ActionEvent e) {
    // Appelle la méthode du contrôleur pour créer l'IHM du quiz
    myController.CreateQuizGameGUI();

    // Ferme la fenêtre actuelle
    dispose();

    // Affiche un message dans la console
    System.out.println("Game started!");
}
```

Figure 6 : Méthode actionPerformed

La méthode fait que l'on appelle le Controller à créer une IHM du quiz, ce qui ferme la fenêtre actuelle et affiche le message « Game started ! ». Cette méthode permet de faire comprendre au joueur que le jeu est lancé.

2.4 CAS D'UTILISATION « SE CONNECTER »

Comme on l'a vu sur l'Analyse-Conception, nous devons permettre à l'utilisateur et donc joueur de pouvoir avoir son propre compte et donc de pouvoir se connecter à celui-ci.

Pour cela, nous créerons une classe Login dans le package View, on spécifiera les variables (Controller, JPanel et JTextField) afin que l'utilisateur puisse rentrer son identifiant et mot de passe.

```
package View;

import javax.swing.JFrame;

public class Login extends JFrame {

    // Contrôleur associé à la vue
    private Controller myController;

    // Panneau de contenu
    private JPanel contentPane;

    // Champs de texte pour le login et le mot de passe
    private JTextField txtLogin;
    private JTextField txtPwd;
```

Figure 7 : Partie spécification de la classe Login

Puis la partie implémentation va permettre de paramétrer les variables, on y mettra les labels pour login et password, les zones de textes où l'on rentre les informations et le bouton login afin de faire vérifier si les données entrées par l'utilisateur sont correctes.

```
// Labels pour le login et le mot de passe
JLabel lblLogin = new JLabel("Login : ");
lblLogin.setBounds(10, 11, 46, 14);
contentPane.add(lblLogin);

JLabel lblPwd = new JLabel("Password :");
lblPwd.setBounds(10, 36, 62, 14);
contentPane.add(lblPwd);

// Champs de texte pour le login et le mot de passe
txtLogin = new JTextField();
txtLogin.setBounds(87, 11, 86, 20);
contentPane.add(txtLogin);
txtLogin.setColumns(10);

txtPwd = new JPasswordField();
txtPwd.setColumns(10);
txtPwd.setBounds(87, 36, 86, 20);
contentPane.add(txtPwd);
```

On peut voir que les labels possèdent leurs bordures ainsi qu'un texte en plus d'être ajouté dans le « contentPane ». Pour les JTextField, on paramètre le nombre de colonnes, les bordures et l'ajout dans le « contentPane ».

Figure 8 : Partie implémentation (JLabel & JTextField) de la classe Login

Sur le JButton, en plus du paramètre, on y ajoute la méthode « actionPerformed » qui lui permettra de vérifier l'authentification et si les données inscrites par l'utilisateur sont correctes, notamment en utilisant le « .getText() » et que si les données sont correctes, cela change la fenêtre et affiche la page **GameStart**.

```

// Bouton "Login"
JButton btnLogin = new JButton("Login");
btnLogin.setBounds(23, 82, 89, 23);
contentPane.add(btnLogin);

// Bouton "Change Password"
JButton btnChangePassword = new JButton("Change Password");
btnChangePassword.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        // Récupère le nom d'utilisateur et le mot de passe
        String nomUtilisateur = txtLogin.getText();
        String motDePasse = txtPwd.getText();

        // Vérifie l'authentification
        if (unController.verifyUserLogin(nomUtilisateur, motDePasse)) {
            // Affiche la fenêtre de changement de mot de passe
            myController.CreateFrameChangePassword(nomUtilisateur);
            dispose(); // Ferme la fenêtre actuelle
        } else {
            // Affiche un message d'erreur
            JOptionPane.showMessageDialog(null, "Nom d'utilisateur ou mot de passe incorrect.");
        }
    }
});
btnChangePassword.setBounds(134, 82, 118, 23);
contentPane.add(btnChangePassword);

```

Figure 9 : Partie implémentation (JButton & ActionListener) de la classe Login

```

// Ajout d'un écouteur d'événements pour le bouton "Login"
btnLogin.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        // Récupère le nom d'utilisateur et le mot de passe
        String nomUtilisateur = txtLogin.getText();
        String motDePasse = txtPwd.getText();

        // Vérifie l'authentification
        if (unController.verifyUserLogin(nomUtilisateur, motDePasse)) {
            // Affiche la fenêtre de démarrage du jeu
            myController.CreateGameStart();
            dispose(); // Ferme la fenêtre actuelle
        } else {
            // Affiche un message d'erreur
            JOptionPane.showMessageDialog(null, "Nom d'utilisateur ou mot de passe incorrect.");
        }
    }
});

```

Figure 10 : Partie implémentation ActionListener de la classe Login

2.5 CONNEXION A LA BASE DE DONNEES

Afin de pouvoir rentrer les informations tels que les données utilisateurs ainsi que les questions et réponses, nous allons créer une base de données. Cette base de données sera sous MySQL et portera le nom de « sbcg » soit SpriteBot Cybersecurity Game.

Pour cela, on va créer une classe **Configuration** ainsi qu'un fichier de configuration que l'on nomme « vtg.cfg », ce fichier comportera les données nécessaires à la connexion à la base de données.

La capture d'écran ci-dessous montre comment fonctionne la configuration avec le fichier ainsi que la lecture. On voit aussi que l'on paramètre un mot de passe par défaut « P@ssw0rdsio ».


```
// Constructeur
public Configuration() {
    // Initialise les propriétés et l'encrypteur
    this.properties = new Properties();
    this.encryptor = new StandardPBEStringEncryptor();

    // Définit le mot de passe pour l'encrypteur
    this.encryptor.setPassword("P@ssw0rdsio");

    try (FileInputStream input = new FileInputStream("./data/vtg.cfg")) {
        // Charge les propriétés de configuration à partir du fichier
        properties.load(input);
    } catch (IOException e) {
        // Gère une éventuelle IOException si le chargement du fichier échoue
        e.printStackTrace();
    }
}

// Méthode pour lire une propriété et la décrypter
public String readProperty(String theKey) {
    // Décrypte la valeur de la propriété en utilisant l'encrypteur
    return encryptor.decrypt(properties.getProperty(theKey));
}
}
```

Figure 11 : Configuration de la connexion à la base de données

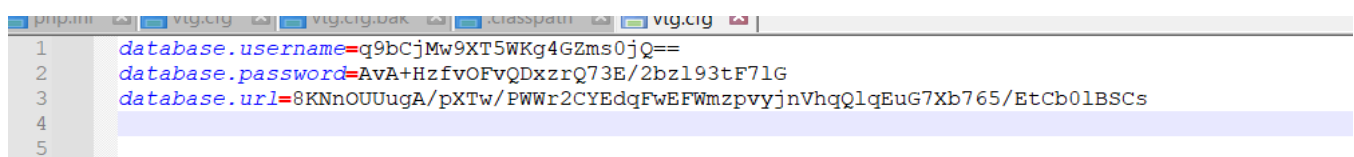
Pour utiliser la base de données, on prend par exemple la classe **DAOsqlQuestion** qui sert à récupérer les questions inscrites dans la base de données. La classe ci-dessous fait appel à la configuration (le nom de la base de données, l'username et la mot de passe).

```
// Implémentation
public DAOsqlQuestion(Controller theController) {
    try {
        // Initialise le contrôleur et établit la connexion à la base de données
        this.myController = theController;
        String dbname = this.myController.getMyConfiguration().readProperty("database.url");
        String username = this.myController.getMyConfiguration().readProperty("database.username");
        String password = this.myController.getMyConfiguration().readProperty("database.password");

        this.connection = DriverManager.getConnection(dbname, username, password);

        this.statement = connection.createStatement();
    } catch (SQLException e) {
        // Gère les exceptions liées à la connexion à la base de données
        e.printStackTrace();
    }
}
```

Figure 12 : Obtention des données de la base de données (DAOsqlQuestion)



```
1 database.username=q9bCjMw9XT5WKg4GZms0jQ==
2 database.password=AvA+HzfvOFvQDxzrQ73E/2bz193tF71G
3 database.url=8KNnOUUugA/pXTw/PWwR2CYEdqFwEFWmzpvynVhqQlqEuG7Xb765/EtCb01BSCs
4
5
```

Figure 13 : Fichier de configuration (vtg.cfg)

Sur la capture d'écran ci-dessus, on voit ce qu'on retrouve dans le fichier de configuration « vtg.cfg ». On y retrouve trois variables, essentielles à la connexion. Tout d'abord, le « database.username » où on y inscrit le nom d'utilisateur de la base de données. Puis le « database.password » avec le mot de passe et enfin le « database.url » avec le lien menant vers la base de données (comme le host, le nom de la base de données ou le port utilisé ».

3 BIBLIOTHEQUE UTILISEE :

3.1 MISE EN PLACE D'UN WEB SOCKET

3.1.1 QU'EST-CE QU'UN WEB SOCKET

Un Web Socket est un moyen de communiquer entre le client et le serveur. Elle permet notamment d'envoyer un message au serveur et de recevoir une réponse de celui-ci sans avoir à le consulter.

Elle s'exécute sous forme de protocole TCP (Transmission Control Protocol) et est souvent utilisé dans des applications nécessitant des mises à jour en temps réel (application de chat, tableaux de bord en direct, jeux en ligne).

On retrouve généralement un Web Socket au niveau de la couche Application

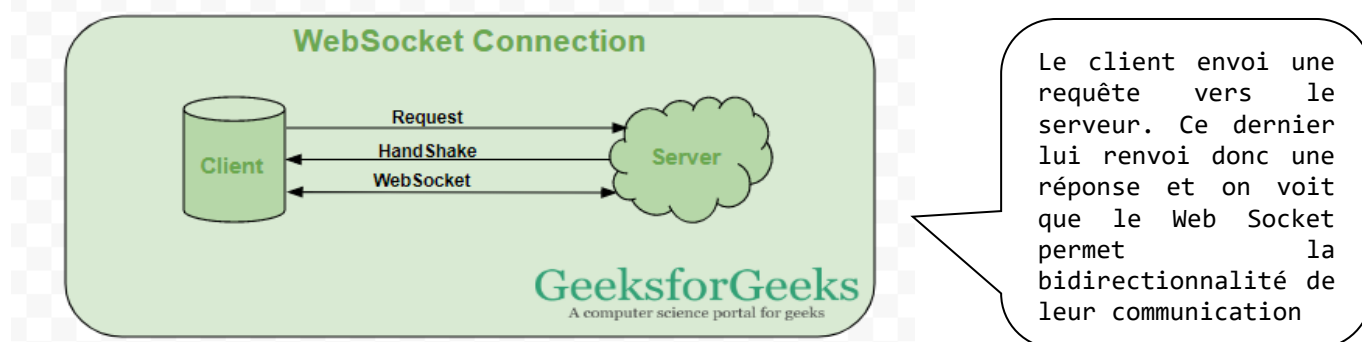


Figure 14 : Schéma de fonctionnement d'un Web Socket

Il existe de multiples outils permettant l'utilisation d'un Web Socket, on peut citer Kryonet, HumbleNet, Socket.IO.

Dans notre cas on a utiliser Kryonet :

```
import com.esotericsoftware.kryo.Kryo;  
import com.esotericsoftware.kryonet.Connection;  
import com.esotericsoftware.kryonet.Listener;  
import com.esotericsoftware.kryonet.Server;
```

Figure 15 : Import de Kryonet

3.1.2 LES DIFFERENCES ENTRE WEB SOCKET ET API

Une API (Interface de Programmation d'Application) quant à elle est un ensemble de règles et de protocoles qui va permettre à différents logiciels de communiquer entre eux. Utilisant souvent des protocoles http (HyperText Transfert Protocol), elle nécessite des requêtes afin de récupérer ou envoyer des données du côté client au côté serveur.

Les principales différences entre les API et un Web Socket se trouve dans la communication. Là où les Web Sockets communiquent de manières bidirectionnelles, les API utilisent une communication unidirectionnelle (le client envoie une demande au serveur et attend sa réponse).

3.2 UTILISATION DE JBCRYPT

JBCrypt est une bibliothèque open-source en Java qui implémente l'algorithme de hachage de mots de passe bcrypt. Cet algorithme est conçu pour être résistant aux attaques par force brute et est largement utilisé pour sécuriser les mots de passe dans les applications.

L'utilisation de JBCrypt est simple : elle permet de hacher les mots de passe avant de les stocker dans une base de données et de vérifier si un mot de passe entré correspond au hachage stocké. Cela garantit que même si la base de données est compromise, les mots de passe des utilisateurs restent sécurisés.

JBCrypt génère des hachages de mots de passe en utilisant un sel aléatoire et un certain nombre de tours de chiffrement, ce qui rend les attaques par force brute très coûteuses en termes de temps et de ressources.

3.3 UTILISATION DE PASSAY

Passay est une bibliothèque open-source en Java utilisée pour la validation des mots de passe. Elle fournit des fonctionnalités pour générer, évaluer et valider les mots de passe selon des règles spécifiées.

Fonctionnalités principales :

1. Validation de mots de passe : Passay permet de valider les mots de passe selon des règles prédéfinies telles que la longueur minimale, la présence de caractères spéciaux, de lettres majuscules ou minuscules, etc.
2. Génération de mots de passe sécurisés : Elle propose des méthodes pour générer des mots de passe aléatoires conformes à des critères de sécurité spécifiques.
3. Personnalisation des règles de validation : Passay offre la possibilité de définir des règles de validation personnalisées pour répondre aux besoins spécifiques de votre application.
4. Intégration facile : La bibliothèque est conçue pour être facilement intégrée dans des projets Java existants, offrant ainsi une solution rapide et efficace pour la gestion des mots de passe.

Nous on l'utilise pour vérifier les règles de mots de passe :

```
// Vérification des règles de mot de passe avec Passay
PasswordValidator validator = new PasswordValidator(
    new LengthRule(8, 30),
    new CharacterRule(EnglishCharacterData.UpperCase, 1),
    new CharacterRule(EnglishCharacterData.LowerCase, 1),
    new CharacterRule(EnglishCharacterData.Digit, 1),
    new CharacterRule(EnglishCharacterData.Special, 1),
    new WhitespaceRule()
);
```

Figure 16 : Utilisation de Passay

3.4 UTILISATION DE SWING

Swing est un ensemble de bibliothèques Java utilisées pour créer des interfaces graphiques (GUI). Il fournit des composants graphiques riches et une architecture flexible pour la création d'applications desktop interactives en Java.

Principales fonctionnalités de Swing :

1. Composants graphiques : Swing offre une large gamme de composants graphiques prêts à l'emploi, tels que des boutons, des champs de texte, des étiquettes, des listes, des tableaux, des barres de défilement, etc. Ces composants peuvent être utilisés pour construire des interfaces utilisateur complexes.
2. Personnalisation : Les composants Swing peuvent être personnalisés en termes d'apparence et de comportement. Vous pouvez spécifier des polices, des couleurs, des tailles, des bordures et d'autres propriétés pour adapter l'apparence des composants à vos besoins.

3. Gestion des événements : Swing utilise un modèle d'événements pour gérer les interactions utilisateur. Vous pouvez associer des écouteurs d'événements à des composants pour répondre aux actions de l'utilisateur, comme cliquer sur un bouton ou saisir du texte dans un champ de texte.

Exemple d'utilisation de Swing dans notre code :

```
// Configuration de la fenêtre
setResizable(false);
setIconImage(Toolkit.getDefaultToolkit().getImage("img\\sb-logo-monogram-circle.jpg"));
setTitle("Quiz Game");
setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
setSize(1422, 742);
setLocationRelativeTo(null);

// Initialisation des composants d'interface utilisateur
questionLabel = new JLabel();
questionLabel.setFont(new Font("Arial", Font.BOLD, 18));
answerRadioButtons = new JRadioButton[4];
ButtonGroup buttonGroup = new ButtonGroup();
JPanel answerPanel = new JPanel(new GridLayout(4, 1));
```

Figure 17 : Utilisation de Swing avec une frame et des composants d'interface

3.5 UTILISATION DE JASYPT

Jasypt est une bibliothèque Java utilisée pour la simplification du chiffrement de données sensibles, telles que les mots de passe, les informations d'identification et autres données confidentielles. Elle offre des fonctionnalités de chiffrement et de déchiffrement de manière transparente, ce qui facilite l'intégration de la sécurité dans les applications Java.

Principales fonctionnalités de Jasypt :

1. Chiffrement des données : Jasypt fournit des outils simples pour chiffrer des données sensibles à l'aide d'algorithmes de chiffrement robustes tels que AES, DES, Blowfish, etc.
2. Déchiffrement des données : Elle permet également de déchiffrer les données chiffrées à l'aide de la même clé et du même algorithme utilisés pour le chiffrement, offrant ainsi une solution complète pour la gestion des données chiffrées

Exemple d'utilisation de Jasypt dans notre code :

```
// Initialise les propriétés et l'encrypteur
this.properties = new Properties();
this.encryptor = new StandardPBEStringEncryptor();

// Définit le mot de passe pour l'encrypteur
this.encryptor.setPassword("mdp");
```

Figure 18 : Utilisation de Jasypt pour crypter avec un mdp définit

3.6 UTILISATION DE JFREECHART

JFreeChart est une bibliothèque open-source en Java utilisée pour créer des graphiques et des diagrammes de données de manière simple et flexible. Elle offre une grande variété de graphiques prêts à l'emploi, ainsi que des fonctionnalités avancées pour personnaliser et manipuler les graphiques selon les besoins spécifiques de l'application.

Principales fonctionnalités de JFreeChart :

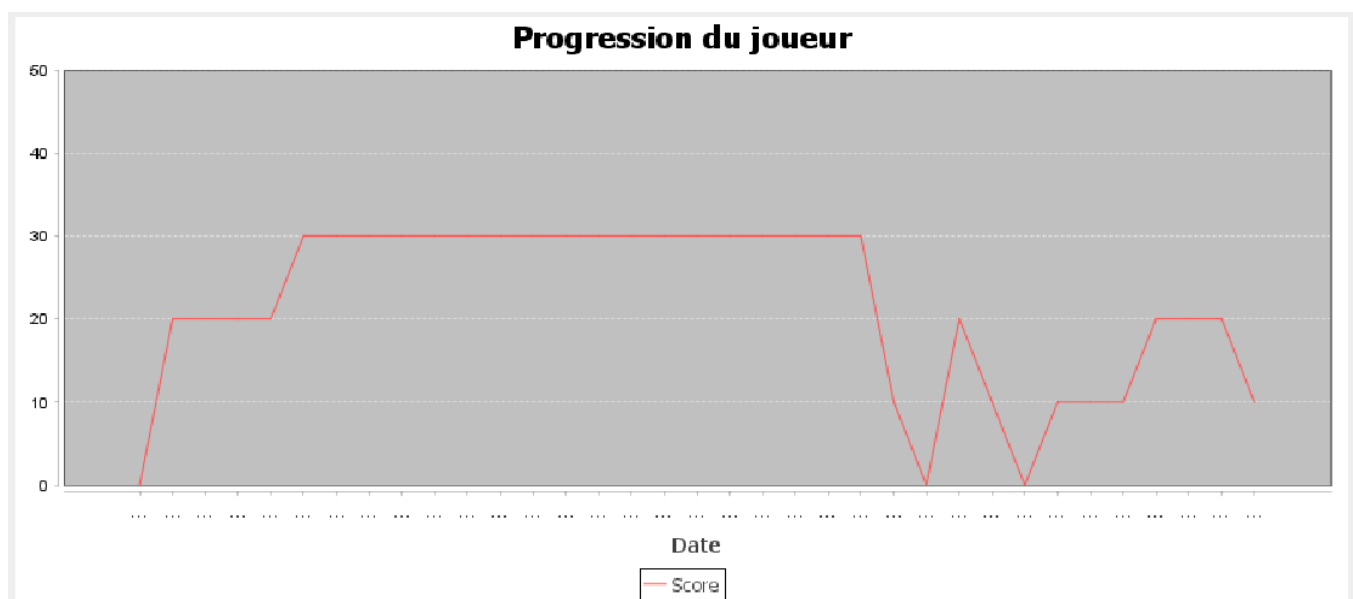
1. Types de graphiques variés : JFreeChart prend en charge une large gamme de types de graphiques, y compris les graphiques en barres, les graphiques en courbes, les graphiques en secteurs, les diagrammes de dispersion, les diagrammes XY, les diagrammes à bulles, les diagrammes de Gantt, etc.
2. Personnalisation des graphiques : Elle offre des fonctionnalités avancées pour personnaliser l'apparence des graphiques, y compris la modification des couleurs, des polices, des styles de ligne, des formes de marqueurs, des légendes, des titres, des axes, etc.
3. Manipulation des données : JFreeChart permet de manipuler facilement les données affichées dans les graphiques, notamment en ajoutant, en supprimant ou en mettant à jour des séries de données, en définissant des plages d'axes, en filtrant les données, etc.

Exemple d'utilisation de JfreeChart dans notre code :

```
DefaultCategoryDataset dataset = createDataset();
JFreeChart chart = ChartFactory.createLineChart(
    "Progression du joueur",
    "Date",
    "Score",
    dataset,
    PlotOrientation.VERTICAL,
    true, true, false);|
```

Figure 19 : Création d'un graphique qui montre la progression du joueur en fonction de la date et du score

Résultat visuel du graphique :



4 CONCLUSION

Le projet Spritebot représente une avancée significative dans la sensibilisation à la sécurité informatique chez Vinci. En développant un jeu interactif et éducatif, l'équipe a créé un outil ludique pour renforcer les compétences en cybersécurité des employés. L'utilisation de technologies modernes telles que les WebSockets et des bibliothèques comme JBCrypt et JFreeChart garantit une expérience utilisateur sécurisée et immersive. En conclusion, Spritebot offre une solution innovante pour sensibiliser et former les employés aux enjeux de la cybersécurité, renforçant ainsi la posture de sécurité de l'entreprise.

5 INDEX DES ILLUSTRATIONS

Figure 1 : Partie spécification de la classe Answer	3
Figure 2 : Partie Implémentation (Constructeur) de la classe Answer	3
Figure 3 : Partie Implémentation (Getter & Setter) de la classe Answer	4
Figure 4 : Partie spécifications de la classe GameStart	4
Figure 5 : Partie implémentation de la classe GameStart	5
Figure 6 : Méthode actionPerformed	5
Figure 7 : Partie spécification de la classe Login	6
Figure 8 : Partie implémentation (JLabel & JTextField) de la classe Login	6
Figure 9 : Partie implémentation (JButton & ActionListener) de la classe Login	7
Figure 10 : Partie implémentation ActionListener de la classe Login	7
Figure 11 : Configuration de la connexion à la base de données	8
Figure 12 : Obtention des données de la base de données (DAOsqlQuestion)	8
Figure 13 : Fichier de configuration (vtg.cfg)	8
Figure 14 : Frame changement de password	10
Figure 15 : Partie de code de la vue	11
Figure 16 : Partie spécification de la classe Answer	11
Figure 17 : Partie Implémentation (Constructeur) de la classe Answer	11
Figure 18 : Partie Implémentation (Getter & Setter) de la classe Answer	12
Figure 19 : Schéma de fonctionnement d'un Web Socket	13
Figure 20 : Import de Kryonet	13