# 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 |
|---|---|---|---|
| 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 Objectifs

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

## 2 Réalisation du code sous Android Studio

### 2.1 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-Vue-Controller) et regroupera donc principalement ces trois packages, la vue sera traduis un fichier xml dans un package layout.

#### 2.1.1 Package control

Le package control est le cœur du code avec plusieur classes :

- ChangePasswordActivity.java
- Configuration.java
- GameStartActivity.java
- LoginActivity.java
- QuizGameActivity.java
- RecapGameActivity.java

Ici chaque classe du package control a sa vue correspondante, avec la classe configuration qui a pour but de crypter les données présentés dans le fichier de « cfg »

#### 2.1.2 Package Model

Le package Model quant à lui contiendra les classes :

- Answer.java
- Player.java
- Question.java
- QuizGame.java

Ici chaque classe 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.

### 2.1.3 Package Layout

Le package Layout contiendra toutes les classes faisant offices de frame ou d'interface graphique, on y retrouve les classes :

- activity_change_password.xml
- activity_game_start.xml
- activity_login.xml
- activity_quiz_game.xml
- activity_recap_game.xml

Ici les classe qui détermine la vue sont traduis par des fichier xml dans le package layout, cela diffère du client lourd vue plus haut.

### 2.1.4 Package DAO

Le package DAO servira à établir la connexion avec la base de données, on y retrouve trois classes :

- DAOsqlAnswer.java
- DAOsqlQuestion.java
- DataFileUser.java

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.

### 2.1.5 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.

## 2.2 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 sous android studio. Comme dit précédemment, les classes seront créés afin de remplir un rôle.

### 2.2.1 Package Layout

Les classes disponibles dans le package layout sont les vue, en effet il possède des fichiers xml qui permette de créer la vue et d'afficher la partie graphique de l'application

**Figure 14 : Frame de changement de Password**

Par exemple une classe activity_change_password.xml qui permet d'afficher la vue pour changer de password : Voici ci-dessous comment est représenté la vue pour la partie New password et Confirm password :

**Figure 15 : Partie de code de la vue**

### 2.2.2 Package Control

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.

### 2.2.3 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.

On y retrouve les variables « codeAnswer » qui constitue l'identifiant lorsque l'on créera une réponse, « descriptionAnswer » qui donnera la description de la réponse et « isCorrect » qui détermine si cette réponse est vraie ou fausse.

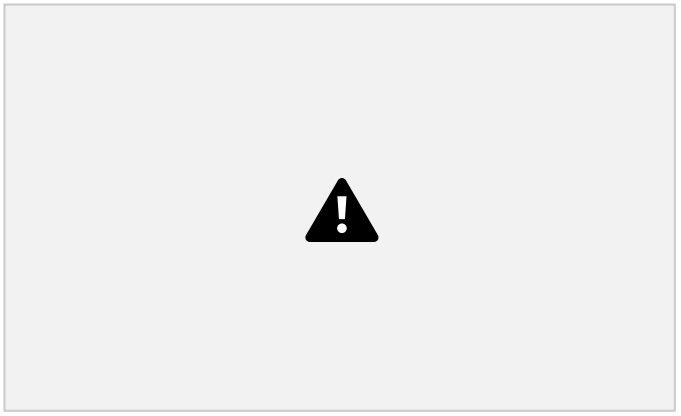

**Figure 16 : Partie spécification de la classe Answer**

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

**Figure 17 : Partie Implémentation (Constructeur) de la classe Answer**

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

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

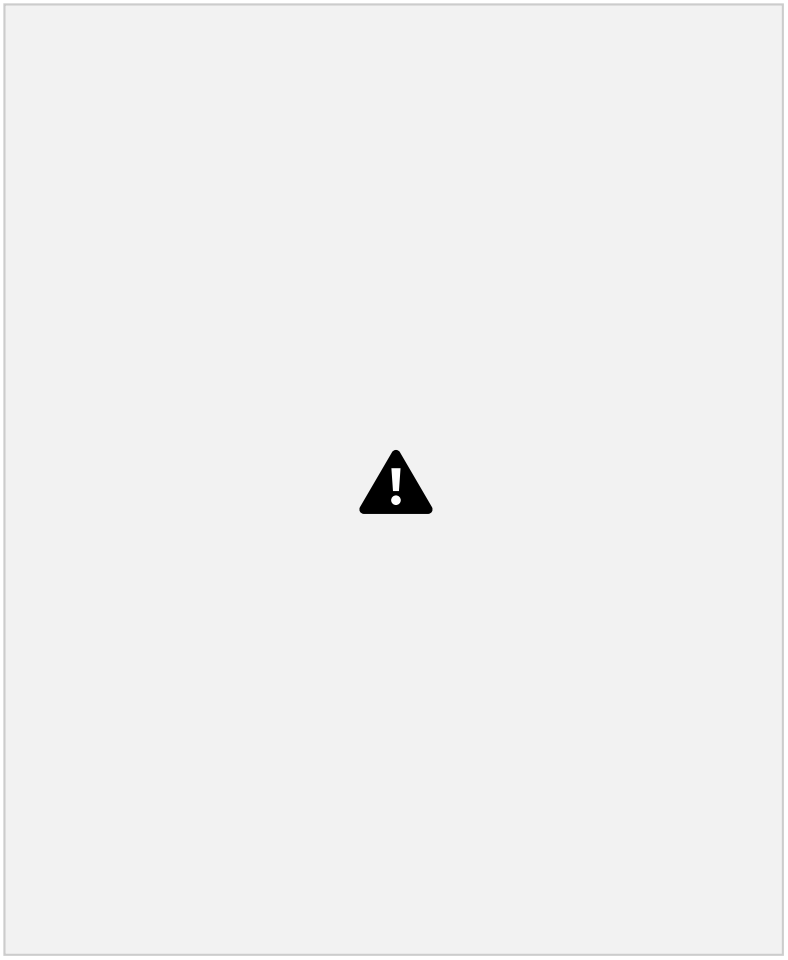

**Figure 18 : Partie Implémentation (Getter & Setter) de la classe Answer**

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

# 3 Conclusion

# 4 Index des illustrations