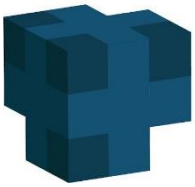

solveIT

Consider IT Solved.

Charte de Codage

Livrable de groupe

Rédigée par :

- LAGHOUB Nassim (Equipe 01)
- YOUSFI Kenza (Equipe 02)
- KESSI Romaissa (Equipe 03)
- DOUMI Athmane (Equipe 04)

Table des matières

Introduction

Le projet AutoLibDz réalisé par l'entreprise SolveIT sera partagé entre quatre équipes de développement. Les membres de ces dernières auront donc à collaborer et travailler sur le même code. Face à cette situation et compte tenu de l'ampleur du projet, il est plus que nécessaire d'instaurer des conventions de codage.

La notion de conventions de codage désigne l'ensemble de règles et conseils adoptés lors du développement d'un projet logiciel pour écrire et mettre en forme du code. Ces conventions visent donc à améliorer la lisibilité du code, son écriture, sa maintenance et aident même à éviter certaines erreurs.

Préambule

Le présent document présente les différents langages et outils utilisés dans le projet AutoLibDz. Il contient également les règles et conventions générales de codage ainsi que les meilleures pratiques à suivre selon le langage.

Langages et outils

Pour la réalisation du projet AutoLibDz, notre entreprise a opté pour une panoplie de langages et outils. Nous les listons dans ce qui suit :

- Développement mobile
  - Langage : Kotlin/Dart
  - Framework : Flutter
  - IDE : Android Studio/Visual Studio Code
- Développement web front-end
  - Framework : React.Js
  - IDE : Visual Studio Code
- Développement back-end

- Environnement: Node.Js
- IDE : Visual Studio Code

Conventions de codage

Conventions générales

Cette partie est générale à la programmation dans la plupart des langages. Elle se doit d'être la plus générale possible sans rentrer dans certaines conventions des différents langages.

1. Formatage du code

1.1. Indentation

Une bonne indentation du code permet de mieux s'y retrouver et d'éviter souvent de nombreuses erreurs.

Tous les éditeurs n'interprètent pas les tabulations de la même façon. Certains font qu'une tabulation vaut trois espaces, d'autres cinq espaces. Donc pour faire une indentation, il vaut mieux utiliser les espaces (généralement quatre) plutôt qu'une tabulation.

Un code écrit comme ci-dessous est quasi illisible !

```
if ($comment == FALSE)
{
echo 'Error';
}
else
{
```

```
echo 'Fine';
}
```

Un code sous cette forme est bien plus aéré et compréhensible :

```
if ($comment == FALSE)
{
  echo 'Error';
}
else
{
  echo 'Fine';
}
```

1.2. Accolades

Il existe deux façons très utilisées pour placer vos accolades.

Soit on les place après le nom de la fonction (comme dans l'exemple ci-dessous), soit on les place en dessous.

À côté

```
function checkpoints($x, $y) {
  ...
}
```

En dessous

```
function checkpoints($x, $y)
{
```

```
...

}
```

1.3. Interligne

Aérer le code avec des espaces blancs améliore souvent la structure de votre code. Prenons l'exemple de commentaires.

```
// Une variable

$myVar = 0;

// Une autre variable

$otherVar = 1 ;
```

1.4. Longueur d'une ligne

Veillez à ne pas écrire des lignes de code trop longues. Être obligé d'utiliser la barre de défilement pour voir la fin du code n'est pas pratique.

80 caractères sur une ligne semblent être la limite pour convenir à toutes les résolutions actuelles sans devoir utiliser la barre de défilement.

2. Convention d'écriture des commentaires

Les commentaires permettent d'expliquer au lecteur de votre code le fonctionnement de votre programme. Il peut vous sembler ennuyeux de les écrire, mais ils vous seront d'une précieuse aide si vous devez vous relire après quelques mois ou si quelqu'un essaye de comprendre votre code.

Considérez aussi que la pause de commentaires fait partie du temps développement.

Veuillez éviter aussi des commentaires inutiles.

2.1. Commentaires sur une ligne

```
// My comment

if ($comment == FALSE)

{

  echo 'Error';

}


// My other comment

if ($variable)

{

  echo 'Bon';

}
```

2.2. Commentaires en fin de ligne

```
$myVar = array();   // Array

$test = 0;        // Test variable
```

2.3. Commentaires multi-lignes

```
/* My comment is too long to be written on a simple line

then I think that I'll write it on multi-line */


// else ...

/*

$variable = 1;
```

```
$test = false;

if ($test)

{

  $variable = 0;

}

*/

echo $variable;
```

2.4. Documentation du code

Nombreux développeurs ne connaissent pas cette pratique pourtant bien utile. Elle consiste à documenter son code de telle façon qu'un programme externe puisse fournir les informations utiles concernant le code rien qu'en examinant les commentaires. Pour cela, les commentaires doivent s'écrire d'une certaine façon, par exemple pour les fonctions :

```
/**

 * Check if points are in the ship

 * @param int x the height

 * @param int y the width

 * @return bool TRUE / FALSE

 */

function checkpoints($x, $y)

{

  ...

}
```

Convention Node.JS (Conception des APIs en général)

1. Terminologie

Ressource: Une ressource est un élément de données, par exemple, un utilisateur.

Collection: un groupe de ressources est appelé une collection. Exemple: une liste d'utilisateurs

URL : Identifie l'emplacement de la ressource ou de la collection. Exemple: / user

2. Règles de Nommage

- **Utiliser kebab-case pour les URLs et noms de ressources dans les URLs**

```
/system-orders
```

- **Utiliser camelCase pour les paramètres d'une requête ou dans les champs de ressources:**

```
/system-orders/{orderId}
```

- **Utiliser un nom pluriel pour pointer vers une collection**

```
GET /users
```

- **L'URL commence par une collection et se termine par un identifiant**

```
GET /shops/:shopId/ or GET /category/:categoryId
```

- **Ne pas utiliser les verbes dans l'URL ressources**

  À la place, utilisez les méthodes HTTP appropriées pour décrire l'opération.

```
PUT /user/{userId}
```

Au lieu de :

```
POST /updateuser/{userId} or GET /getuser
```

- Utiliser les verbes dans l'URL non ressources

  Vous avez un endpoint qui ne renvoie rien d'autre qu'une opération. Dans ce cas, vous pouvez utiliser des verbes.

```
POST /alerts/245743/resend
```

- Utiliser camelCase pour JSON property

  Si vous créez un système dans lequel le corps de la requête ou la réponse est en format JSON, les noms des propriétés doivent être en camelCase.

```
{
  userName: "Omar Abil"
  userId: "1"
}
```

Inclure le nombre total d'une ressource dans la réponse:

```
{
   users: [
      ...
   ],
   total: 34
}
```

Les messages de réponse doivent être auto-descriptifs.

Utiliser le bon status code avec votre réponse pour décrire si tout a fonctionné ou l'application a bloqué.

3. Sécurité

- **Ne pas passer les tokens d'authentification au niveau des URLs, mais plutôt au niveau du header.**

```
Authorization: Bearer xxxxxx, Extra yyyyy
```

- **Utiliser les variables d'environnement.**

4. Séparer Express "app" et "server"

Séparer la définition d'Express en au moins deux fichiers: la déclaration de l'API(app.js) et les responsabilités

5. Format

Commencer les accolades d'un bloc de code sur la même ligne

```
// Do
function someFunction() {
  // code block
}
// Avoid
function someFunction()
{
  // code block
}
```

6. Conventions générales

- **Utiliser --save-dev sur la ligne de commande lors de l'installation d'un package afin de le rajouter dans la liste des devDependencies du package.json;**

```
npm install <package-name> [--save-dev]
```

- **Utiliser Async Await, évitez les callbacks**

- Utiliser un logger mature pour augmenter la visibilité des erreurs comme Winston, Bunyan ou Pino au lieu de console.log.
- Utiliser la syntaxe ES6 (“import” au lieu de “require”)

Conventions Kotlin

1. Conventions principales

- Utiliser les modificateurs de visibilité uniquement pour spécifier une visibilité différente de « public »
- Utiliser les Data class pour encapsuler les données
- Ne pas utiliser les points virgules
- Favoriser l’utilisation des symboles { ? , !! } pour faciliter la détection des éléments « nullable » par le compilateur

2. Class Layout

- Le contenu des classes doit être dans cet ordre
    1. Déclaration des attributs et blocs d’initialisation
    2. Constructeurs secondaires
    3. Méthodes
- Ne pas ordonner les méthodes alphabétiquement mais préférer le groupement des méthodes similaires.

3. Règles de nommage

Les noms des packages sont toujours en minuscule et on n’utilise pas de tirets (ni tiret du haut).

Les noms des classes et objets commencent par une majuscule