



Junior Enterprise

GUIDE DE LA PARTIE TECH

Pôle Projets Mandat 2020-2021

SOMMAIRE

CE DOCUMENT CONTIENT:

Introduction

Qualité de code demandée

Le Processus de test

La Documentation demandée

Travail en équipe demandé (Scrum)

Les bases de Git

Les normes demandées sur Git

Conseils UX et Ergonomie

INTRO

À PROPOS DE CE DOCUMENT

Le document suivant, élaboré dans le cadre de l'amélioration de la qualité du rendu du Processus Projet de la JEI, va fixer les standards qualité à suivre:
Considérez-le comme un guide !

Ce document doit être respecté durant les mini-projets de la phase de formation et tous les projets réalisés après sauf mention contraire !

Le contrôle de la qualité de l'avancement doit se faire en se basant sur ce qui est mentionné dans ce guide.

QUALITÉ DE CODE

PARTIE 1



N.B: Tous les noms doivent être en anglais !

- **Structure du Projet :**

Il est primordial de respecter une bonne structuration du projet et de diviser et même subdiviser vos fichiers en dossiers et d'attribuer les bons noms au fichiers.

Par exemple, en travaillant avec une architecture MVC (ou MVCS), créer un dossier pour chaque qui seront respectivement "models", "views" et "controllers" et un quatrième dossier peut être ajouté qui contiendra les requêtes qui sera appelé "services" ou "api". Dans chacun de ces dossiers, il est possible de subdiviser davantage en se basant sur la logique du projet. Par exemple, une plateforme ayant deux acteurs peut avoir deux dossiers sous "controllers" qui regroupent les contrôleurs de chaque acteur et qui portent respectivement les noms "actor1-controllers" et "actor2-controllers" avec dans l'un d'eux, par exemple, les fichiers "login_controller.extension" et "dashboard_controller.extension".

Il est à noter qu'il faudra utiliser le **kebab-case** pour nommer les dossiers et le **snake_case** pour les fichiers de votre projet.

QUALITÉ DE CODE

PARTIE 2

 **N.B:** Tous les noms doivent être en anglais !

- **D'autres conventions de nommage :**

Toujours donner les noms les plus simples qui apportent un sens complet. Eviter l'utilisation des mots de liaison en tout genre:

callTheNumber(); 

callNumber(); 

D'abord, pour les Classes, il faut utiliser le **TitleCase** (aussi appelé **UpperCamelCase**).

Exemple: class Student.

Ensuite, les variables et méthodes, à nommer en utilisant le **lowerCamelCase**.

Exemples: var studentAge; | function calculateScore() {...};

De plus, pour les routes favoriser l'utilisation d'un seul mot. Le cas échéant, utiliser le **lowerCamelCase**.

Exemples: www.url.com/home | www.url.com/editProfile

Enfin, il est très important d'englober son code dans des blocs try...catch. Les noms des exceptions personnalisées doivent être **UpperCamelCase** et se finir par le mot "Exception".

Exemple: DivideByZeroException;

QUALITÉ DE CODE

PARTIE 3

 **N.B:** Les commentaires doivent être en anglais !

- **Commenter son code :**

Les différentes formes de commentaires:

```
//une ligne de commentaire | /* zone de commentaire */ |  
<!-- commentaire html -->
```

Au début de chaque fichier, écrire quelques lignes de commentaires contenant: nom du module, son rôle, une petite description, nom de celui qui l'a écrit. En cas de modifications, ajouter description des modifications et le nom de celui qui les réalise.

Avant chaque méthode, écrire deux à trois lignes de commentaires expliquant le rôle de cette méthode, son input et son output.

Exemple:

```
/* input: student id | output: student name  
 * Function that takes a student's id and return his name  
 */  
function findNameStudent(idStudent) {...}
```

Après une variable, si son nom n'est pas suffisant pour la reconnaître ajouter un commentaire.

Exemple: var port = 3000; //server hosting port

QUALITÉ DE CODE

PARTIE 4

- **Diviser pour régner :**

Il est toujours conseillé de décomposer un problème complexe en étapes moins complexes, les résoudre séparément, afin de trouver une solutions globale. De même il est conseillé de diviser votre plateforme / application en modules et ce module en fichiers et comme ça, la résolution sera par étape et plus simple.

- **La gestion d'erreurs :**

Il y a trois points d'amélioration:

Premièrement, créer les fichiers "Logs" nécessaires qui vont recevoir les historiques d'erreurs que va rencontrer le projet.

Deuxièmement, veiller à ce que les requêtes/réponses http contiennent les bon "status codes" et même un petit message de confirmation.

Finalement, comme déjà brièvement mentionné, créer des exceptions personnalisées et utiliser les blocks

```
try {...}  
catch(CustomException) {...}  
catch(exception) {...}  
finally {...}
```

LE TEST

Définition:

Régression=une fonctionnalité qui marchait avant mais ne marche plus à cause des changements dans la nouvelle version du logiciel.

• **Le Test Manuel :**

Le principe est, lorsqu'une personne termine sa tâche, une autre personne de cette même équipe prend le travail et execute des scénarios de test pour voir si le résultat trouvé est celui attendu ou pas. Le but sera après de reproduire les bugs pour chercher des solutions.

Exemple: Scénario 1:

Ouvrir Site --> Ok

Consulter liste des produits --> Ok

Voir détails d'un produit --> Ok

Ajout au panier --> NOk (Not Ok)

• **Les Tests Automatiques :**

Lorsque le projet prend de l'ampleur, la recherche de régressions à chaque nouvelle version devient difficile, c'est pourquoi il est conseillé de programmer des scripts qui vérifient tout automatiquement à chaque fois.

Exemples de technologies: Selenium, Robot Framework, Cucumber, ...

LES DOCUMENTS

- **La Documentation Projet :**

Cahier de Charges et Livrable. Chaque projet effectué, externe ou interne, doit impérativement avoir ces deux documents. Le cahier de charges est préparé au début du projet mais il risque de recevoir des changements et le livrable doit se faire au fur et à mesure de l'avancement du projet.

- **La Documentation Technique :**

Tout d'abord, avant le début de chaque projet, il faut que l'équipe se mette d'accord sur les technologies à utiliser et leurs versions exactes. Cet accord doit être rédigé dans un document détaillant les étapes d'installation de cet environnement et validé avant de passer (template du document existe).

Ensuite, chaque étape où il y a installation d'un nouvel outil doit être documentée (version, installation, brève explication de l'utilisation...) et chaque document doit être validé.

Enfin, un rapport hebdomadaire (vers la fin de la semaine) est demandé durant chaque mini-projet/projet qui montre les pistes d'avancement de la semaine en question.

SCRUM

- **Le Product Owner :**

Le chef d'équipe sera le product owner, son rôle sera de comprendre le projet, le diviser en petites tâches et les répartir sur le reste de l'équipe tout en précisant l'ordre d'importance.

- **L'équipe de développement + test :**

Travail par Sprint (généralement de 1 semaine) durant lequel il y a le développement et test des fonctionnalités. Il est aussi à noter que le travail doit être par objectif puisque qu'une petite démo sera livrée au client à la fin de chaque objectif.

- **Travail d'équipe:**

Insister sur la communication interne de chaque équipe, la répartition des tâche équitablement et le respect des deadlines.

GIT: QU'EST- CE QUE C'EST ?

**POURQUOI EST-CE AUTANT
UTILISÉ ?**



Définition de GIT

Git est un logiciel de gestion de versions décentralisé.

Pour expliquer plus simplement, c'est un logiciel permettant de gérer un code sur lequel plusieurs personnes ont réalisé des modifications simultanément.

GIT

PARTIE 1

• Les bases :

Si c'est un nouveau projet, lancer ces commandes:

```
git init      #initialise dossier vide  
git remote set-url origin URL    #avec URL= url du gitlab ou github  
git add .   #préparer elements à l'envoi  
git commit -m "message expliquant les changements effectués"  
git branch -M main    #renommer la branche master à main  
git push --set-upstream origin main #envoi des changements
```

Si le projet existe déjà:

```
git clone URL  
git add .  
git commit -m "message"  
git push
```

• Les branches :

Un projet git se compose de branches qui sont l'équivalent d'arborescences:

La branche principale est Main, juste après Dev et ensuite vos branches. On va travailler sur la branche Dev, pour cela on va vers cette branche et on crée une branche en dessous:

```
git checkout dev  
git checkout -b sprintX/nom_branche  
(après travail)  
git add .  
git commit -m "message"  
git push --set-upstream origin sprintX/nom_branche
```

Il est maintenant temps de réintégrer cette branche dans dev:

```
git checkout dev  
git merge sprintX/nom_branche
```

Voilà! après cela, refaire le même cycle avec une nouvelle branche !

GIT

PARTIE 2

- **Nommer ses branches :**

Vu que le projet sera décomposé en sprints et chaque sprint comporte des fonctionnalités spécifiques, le nom de la branche doit être comme suit:

sprintX/feature_name

Exemple: sprint2/login_client

UX & ERGONOMIE

- **L'unique conseil :**

Ce conseil est de suivre les bases déjà présentes dans la plupart des plateformes.

Exemple: Le bouton Login est toujours en haut à droite pour les sites web.