**GUIDE DE RECOMMANDATIONS**

**- GitmeMoney -**

****

**\_**

**David EVAN**

**06/09/2022**

**Version 1.0**

**Crypto Plateforme – GitMeMoney**

**Historique des révisions**

|  |  |  |  |
| --- | --- | --- | --- |
| **Numéro de version** | **Auteur** | **Description** | **Date de modification** |
| 1.0 | EVAN David  *(Architecte logiciel)* | Livraison initiale | 06/09/2022 |
|  |  |  |  |
|  |  |  |  |

Tableau 1 - Historique des révisions

**Objectif du document**

Dans le cadre du projet « Crypto-plateforme » initié par GitmeMoney, ce guide de recommandation a pour objectif de fournir des recommandations à destination des développeurs visant à améliorer les pratiques de développement logiciel et la qualité de code.

Il rappelle les standards de développement à respecter, les méthodes de travail à appliquer et les réunions associées à la méthodologie de travail.

**Note :**

Ce document vise à présenter de manière synthétique les recommandations en matière de développement logiciel et de méthodologie de gestion à appliquer pour ce projet. Il ne s’agit pas d’une documentation sur les concepts présentés. Des références seront indiquées pour l’approfondissement des concepts.

**Table des matières**

[Titre 1 4](#_Toc94255535)

[Titre 2 4](#_Toc94255536)

[Tableau 4](#_Toc94255537)

[TABLES DES RÉFÉRENCES 5](#_Toc94255538)

[Figures 5](#_Toc94255539)

[Tableaux 5](#_Toc94255540)

# ORGANISATION DU PROJET

## Méthodologie « SCRUM »

Rappel de la méthodologie (en synthèse) et de l’outillage utilisé (JIRA, scrum board)

## Les cérémonies

# ORGANISATION DES DÉVELOPPEMENTS

## Méthodes de travail

### Peer-programming

### Revue du code par les paires

### Tests unitaires

# STANDARDS DE DÉVELOPPEMENTS

## Introduction

Cette section vise à présenter un certain nombre de bonnes pratiques pour harmoniser l’écriture de code. Dans un esprit d’amélioration continue, ces recommandations peuvent être amenées à évoluer en fonction des pratiques et des usages.

## Conventions de code à respecter

### Frontend (HMTL / JavaScript / Angular …)

Le développement frontend en langage Javascript devra respecter les conventions décrites dans le guide de style et bonnes pratiques Angular :

* [Angular coding style guide](https://angular.io/guide/styleguide)

Notons que tout code HTML produit devra respecter les normes proposées par le W3C, notamment en matière d’accessibilité :

* [Validation HTML 5](https://validator.w3.org/)
* [WCAG3 : Web Content Accessibility Guidelines](https://www.w3.org/TR/WCAG22/)

### Backend (Java / Spring)

Le développement backend en langage Java s’appuiera sur les conventions de code standards à Java (habituellement maitrisés par tout développeur) et le guide de style Google pour Java pour la présentation :

* [Java Code Conventions](https://www.oracle.com/technetwork/java/codeconventions-150003.pdf)
* [Google Java Style Guide](https://google.github.io/styleguide/javaguide.html)

## Architecture API REST & HTTP

La présente section vise à présenter les conventions communes à la construction des API REST. Elle est issue d’un ensemble de bonnes pratiques communément appliquées et d’expérience accumulés sur la création des API REST.

### Architecture REST

#### Manipulation des ressources

Lors de la conception des API, les règles suivantes s’appliquent :

* Les URL doivent être construites conformément aux règles & bonnes pratiques de l’architecture REST. Les identifiants des ressources doivent être passés en route param.
* Les ressources présentes dans les URLS seront **systématiquement** écrites au pluriel, même si une seule ressource est accessible.  
   *Exemple :* /contract/v1/contracts/123456789
* Toutes les ressources d’un même service doivent **impérativement** partager un vocabulaire commun. Un champ représentant une donnée (exemple : Prix HT) doit disposer de la même “traduction”, peu importe l’API utilisé au sein de ce service et / ou le modèle utilisé, tant que la donnée possède le même sens.

#### Convention de nommage des URLs

Les conventions de nommages s’appliquent principalement à la nomenclature des URLs accessibles et composant les services API.

* Utilisation de la convention de nommage [Kebab Case](https://medium.com/better-programming/string-case-styles-camel-pascal-snake-and-kebab-case-981407998841).
* Utilisation de la langue **anglaise** pour le nommage des services, fonctions, attributs, ressources …
* Des **ressources** (et non des fonctions) doivent être utilisés dans les URLs (exemple /contracts et non /getallcontracts)
* Le nom des attributs composant une ressource devrait être différents des noms des champs de la base de données auxquelles ils font référence.

### Utilisation HTTP

#### Verbes HTTP

L’utilisation des verbes HTTP devra respecter la spécification ci-dessous, et, plus généralement, le sens de chaque méthode HTTP tel que décrit dans [la section 4.3 de la RFC 7231](https://tools.ietf.org/html/rfc7231#section-4.3) :

|  |  |
| --- | --- |
| **VERBE** | **DESCRIPTION** |
| **PUT** | Modification totale d’une ressource |
| **POST** | Création une ressource |
| **PATCH** | Modification partielle d’une ressource |
| **GET** | Récupération d’une ressource |
| **DELETE** | Suppression d’une ressource |

Tableau 2 : Utilisation des verbes HTTP pour la construction des APIs

#### Entêtes HTTP

Pour chaque réponse retournée, celle-ci doit inclure, à minima :

* La description du format de réponse : Ajout de l’entête Content-Type
* La définition de l’encodage utilisé : Ajout de l’entête Charset

#### Code statut HTTP

L’utilisation des codes de retour HTTP devra respecter la spécification suivante, et, plus généralement, le sens de chaque code de retour tel que décrit dans [la section 6 de la RFC 7231](https://tools.ietf.org/html/rfc7231#section-6) :

|  |  |
| --- | --- |
| **CODE** | **DESCRIPTION** |
| **2xx** | Succès |
| **200** | Succès. Des informations de retour sont disponibles. |
| **201** | Succès. Une ressource a été créer. Généralement, la réponse contient la ressource qui vient d'être créée. |
| **204** | Succès. La réponse ne contient aucune donnée. |
| **4xx** | Échec à cause d’un problème dans la requête (exemple : création d’un utilisateur avec un e-mail déjà existant ou paramètre de requête manquant). |
| **5xx** | Échec dû à une erreur du serveur |

Tableau 3 : Utilisation des codes de statut dans les réponses HTTP

### Convention de création pour les API

Afin de maintenir une cohérence forte entre tous les services, certains besoins doivent utiliser une syntaxe commune décrite ci-dessous.

#### Règles communes

* Le résultat de la requête devra **toujours** être retourné dans le champ data d’un objet JSON. Les autres attributs peuvent servir à ajouter des métadonnées à la requête.

#### API Paginées

Les API paginées acceptent toujours deux paramètres optionnels :

* page - Numéro de la page à retourner (défaut : 1)
* size - Nombre de résultats par page à retourner (défaut : dépend de l’API, généralement 20)

Il doit être possible de manipuler ces paramètres pour obtenir les pages suivantes ou augmenter le nombre de résultat dans une seule page.

#### API de recherche

Dans le cas d’une API permettant d’effectuer une recherche (Recherche exclusive) sur une ressource :

* Il doit être possible, en spécifiant des valeurs en query params, de filtrer les résultats uniquement sur un critère précis de la ressource.

*Exemple :* users/?email=john.doe@contoso.com - Liste des utilisateurs dont le nom est égal à la valeur indiquée.

* Les arguments de recherches de type string peuvent être préfixés/suffixés d'un \* pour rendre la recherche non-limitative.

*Exemple :* users/?username=Sandbob\* - Tous les utilisateurs dont le nom commence par "Sandbob".

* Une logique similaire existe pour les champs de type date, avec les préfixes : < et >.

*Exemple :* users/?createdAt=>2020-01-15 - Tous les utilisateurs créés après le 15/01/2015.

Il est aussi possible de créer des APIs permettant de sélectionner un ensemble de ressources, correspondant aux différentes valeurs des éléments indiqués dans les query params de la requête HTTP (Recherche inclusive).

* Une syntaxe basée sur des crochets ([ ]) permet de spécifier la liste des différentes valeurs séparées par des virgules (,).

Exemple : contracts/?id=[1124521,1124550,2102450] - Obtient une liste des contrats indiqués.

* Une syntaxe supplémentaire peut être implémentée, permettant une sélection sur un range de valeurs, en utilisant le séparateur ...

Exemple : users/?id=[1…5] (Les utilisateurs dont l'id est contenu entre 1 et 5).

# TABLES DES RÉFÉRENCES

## Figures

**Aucune entrée de table d'illustration n'a été trouvée.**

## Tableaux

[Tableau 1 - Historique des révisions 2](#_Toc94255517)

[Tableau 2 : Objectifs de chaque phases de l'ADM TOGAF 4](#_Toc94255518)