MADINA-TIC

CHARTE DE CODAGE

Par: AMAR BENSABER Mohammed

Date de création: 03/01/2019

Date de dernière modification: 13/03/2019

Remarque: la langue Anglaise doit toujours être utilisée, sauf pour les choses imposées par le projet.

SOMMAIRE

TABLE OF CONTENTS

CONVENTIONS JAVASCRIPT

NOM DE FICHIER

Les noms de fichiers doivent être en miniscule et descriptives.

FICHIER CODE SOURCE JAVASCRIPT

Les fichiers code source JavaScript ont la structure suivante:

- Commentaire de début
- Commandes **require** et **import**
- Code

La structure de commentaire de début:

```
/**
* fichier.js
* le rôle de ce fichier.
* @todo Ajouter function a.
* @todo Ajouter function b.
*/
```

INDENTATION

L'unité d'intentation est 4 spaces (peut utiliser aussi tabulation)

COMMENTAIRES

COMMENTAIRE DE DÉBUT.

Commentaire pour décrire une fonction avec **JSDoc**

```
/** * Represents a report.
* @function
* @param {string} title - The title of the report.
* @param {string} author - The author of the report.
*/
function Book(title, author) { ... }
```

COMMENTAIRES SUR UNE SEULE LIGNE

Des commentaires courts peuvent être placés sur une seule ligne indentée au niveau de code qui la suit. Si un commentaire ne peut pas être écrit sur une seule ligne, alors il devrait utiliser une mise en forme succession des commentaires sur une seule ligne. Une commentaires en ligne devrait être précédé d'une ligne blanche.

```
if (user.auth) {


      // redirect to homepage

      redirect("/home");

}
```

DÉCLARATION

NOMBRE DE DÉCLARATION PAR LIGNE

Ne mettre qu'une seule declaration par ligne.

```
int reportCount;

int fixedReportCount;
```

ORGANISATION

Les déclarations se font uniquement au début des blocs.

DÉCLARATIONS DES CLASSES

On utilisant la norme ES6

```
class Rectangle {

      constructor(height, width) {

            this.height = height;

            this.width = width;

      }

      // Getter

      get area() {

            return this.calcArea();

      }
```

```
        // Method
        calcArea() {
            return this.height * this.width;
        }
    }
```

voir aussi [un autre example.]

INSTRUCTIONS

INSTRUCTIONS SIMPLES

Chaque ligne ne devrait contenir qu'une unique instruction.

INSTRUCTIONS COMPOSÉES

Les accolades devraient toujours être écrites, même pour les instructions simples.

IF, IF-ELSE, IF-ELSE IF

Les instructions de type *if-else* devraient avoir la mise en forme suivante

```
// if
if (user.Auth) {
    redirect("/home");
}
// if-else
if (user.Admin) {
    redirect("/admin");
} else {
    redirect("/account");
}
 // if-else if
if (!user.Auth) {
    redirect("/login");
} else if (user.Admin) {
    redirect("/admin") ;
```

```
} else {
      redirect("/account");
}
```

FOR, WHILE

Les instruction *for* et *while* devraient avoir la forme suivante

```
// while
while (condition) {
      ...
}
// for
for (begin; condition; step) {
       ...
}
```

BLANCS

LIGNES BLANCHES

Une indentation verticale; un ligne blanche devrait toujours être utilisées dans les circonstances suivantes:

- Entre les blocs logiquement séparés d'une function.
- Entre des sections d'un fichier de source code.

ESPACES BLANCS

Aprés une virgule dans une liste d'arguments.

```
function (report, coord) { ... }
```

Tous les opérateurs binaires devraient être séparés de leurs opérandes par des espaces.

```
a = a + (b * c) / d;
```

CONVETIONS DE NOMMAGE

On s'inspire par **Google JavaScript Style Guide** et **Airbnb JavaScript Style Guide**

Type	Example	Règle
modules	nunjucks	Le nom d'un *module* est toujours écrit en caractères miniscules.
variables	reportCount	Les noms de *variables* doivent être des noms, avec les premières lettres des mots à partir du second en majuscules.
functions	redirectUser	Commencer par des verbes en infitif, avec les premières lettres des mots à partir du second en majuscules.
methods	getReport	-
classes	ReportList	Les noms de *classes* doivent être des noms avec les premières lettres de chaque mot interne en majuscules.
enumerations	CategoryCode	-
constants	SERVER_PORT	Les noms de *constants* doivent être des noms tout écrite en majuscule en séparant les mots par _
filenames	fileserver.js	Le nom d'un *fichier* est toujours écrit en caractères miniscules.

CONVETIONS JAVA

NOM DE FICHIER

Faire référence à 4

FICHIER CODE SOURCE JAVA

Les fichiers code source JavaScript ont la structure suivante:

- Commentaire de début
- Commandes **package** et **import**
- Déclaration de classe ou interface

La structure de commentaire de début:

```
/**
* fichier.java
* package nom de package
* Le rôle de ce fichier.
* @todo Ajouter function a.
* @todo Ajouter function b.
*/
```

INDENTATION

Faire référence à 4

COMMENTAIRES

- Commentaires en blocs
- Commentaire de début
- Commentaire pour décrire une class, interface ou méthode avec **javadoc**

```
/**
* Moves a chess piece.
*
* @param from position from
* @param fromRank rank from
* @param to position to
* @param toRank rank to
* @return true if the move is valid, otherwise false
* @since 1.0
* @version 1.5
*/
void doMove(int from, int fromRank, int to, int toRank) { // ...body }
```

- Commentaires sur une seule ligne: Des commentaires courts peuvent être placés sur une seule ligne indentée au niveau de code qui la suit. Si un commentaire ne peut pas être écrit sur une seule ligne, alors il devrait utiliser une mise en forme succession des commentaires sur une seule ligne. Une commentaires en ligne devrait être précédé d'une ligne blanche.

```
if (user.auth) {

    // send a GET request to the server
    req.sendGetReq();
}
```

DÉCLARATION

NOMBRE DE DÉCLARATION PAR LIGNE

Faire référence à 5

ORGANISATION

Les déclarations se font uniquement au début des blocs.

Déclarations des classes et interfaces

•L'accolade ouvrante { est placée à la fin de la ligne de déclaration.

•L'accolade fermante } entame une nouvelle ligne.

•Pas d'espace entre le nom d’une méthode et la parenthèse ouvrante (.

```
public class User {
    // class body
}

public interface IsDownloadable {
    public String URL = "https://www.madina.tic/api/";
    public void downloadFile(string fileName);
}
```

INSTRUCTIONS

INSTRUCTIONS SIMPLES

Chaque ligne ne devrait contenir qu'une unique instruction.

INSTRUCTIONS COMPOSÉES

Les accolades devraient toujours être écrites, même pour les instructions simples.

IF, IF-ELSE, IF-ELSE IF

Faire référence à 6

FOR, WHILE

Faire référence à 7

BLANCS

Faire référence à 7

LIGNES BLANCHES

ESPACES BLANCS

CONVETIONS DE NOMMAGE

Faire référence à 7

CONVETIONS GO

Nous utilisons l'outil gofmt fourni par l'équipe de Go pour formater le code strictement selon leurs normes.

NOM DE FICHIER

Les noms de fichiers doivent être en miniscule et descriptives.

FICHIER CODE SOURCE GO

Les fichiers code source Go ont la structure suivante:

- Commentaire de début
- Commandes **import**
- Déclaration des structure
- Code

La structure de commentaire de début: Peut être un commentaire bref si le package is simple.

```
/*
Package regexp implements a simple library for regular expressions.

The syntax of the regular expressions accepted is:

    regexp:
        concatenation { '|' concatenation }
    concatenation:
        { closure }
    closure:
        term [ '*' | '+' | '?' ]
    term:
        '^'
        '$'
        '.'
```

```
        character
        '[' [ '^' ] character-ranges ']'
        '(' regexp ')'
*/
// >>>> ou bien
// Package regexp implements regexp
package regexp
```

INDENTATION

L'unité d'intentation est 4 spaces (peut utiliser aussi tabulation)

COMMENTAIRES

La génération de documentation à l'aide de l'outil godoc

- Commentaires en blocs
- Commentaire de début
- Commentaire pour décrire une fonction

```
// NewGraphJSON loads graph from JSON file
func NewGraphJSON(r io.Reader) (*Graph, error) {
```

- Commentaire pour décrire les structure

```
// Graph represents a simple graph
type Graph struct {
    Type  string           `json:"type"`
    Nodes map[string]*Node `json:"nodes"`
    Edges []*Edge          `json:"edges"`
    IDs   []string
}
```

- Commentaires sur une seule ligne: Des commentaires courts peuvent être placés sur une seule ligne indentée au niveau de code qui la suit. Si un commentaire ne peut pas être écrit sur une seule ligne, alors il devrait utiliser une mise en forme succession des commentaires sur une seule ligne. Une commentaires en ligne devrait être précédé d'une ligne blanche.

```
...

// allocate memory for weightmat
g.WeightMat = make(WeightMatrix, len(g.Nodes))
```

DÉCLARATION

NOMBRE DE DÉCLARATION PAR LIGNE

Possible de mettre plusieurs declarations par ligne.

```
var x, v int
y, z := 0, 0
```

ORGANISATION

Il est possible de faire des déclarations imbriquées

INSTRUCTIONS

INSTRUCTIONS SIMPLES

Chaque ligne ne devrait contenir qu'une unique instruction.

INSTRUCTIONS COMPOSÉES

Les accolades devraient toujours être écrites, même pour les instructions simples.

IF, IF-ELSE, IF-ELSE IF

Les instructions de type if-else devraient avoir la mise en forme suivante

```
// if
if user.Auth {
        redirect("/home")
}

// if-else
if user.Admin {
        redirect("/admin")
} else {
        redirect("/account")
}

// if-else if
if !user.Auth {
        redirect("/login")
} else if user.Admin {
        redirect("/admin")
} else {
        redirect("/account")
}
```

FOR, WHILE

Les instruction for et while devraient avoir la forme suivante:

```
// while
for condition {
    ...
}
```

```
// for
for begin; condition; step {
    ...
}

// for
for key, value := range m {
    ...
}
```

BLANCS

LIGNES BLANCHES

Une indentation verticale; un ligne blanche devrait toujours être utilisées dans les circonstances suivantes:

- Entre des sections d'un fichier de source code.
- Entre les blocs logiquement séparés d'une function.

ESPACES BLANCS

- Aprés une virgule dans une liste d'arguments.

```
func nextInt(b []byte, i int) (int, int) {
```

- Les opérateurs binaires + - devraient être séparés de leurs opérandes par des espaces.

```
a = a + (b*c)/d - a
```

CONVETIONS DE NOMMAGE

On s'inspire par Effective Go

Type	Example	Règle
packages	`models`	Le nom d'un *module* est toujours écrit en caractères miniscules.
variables	`reportCount`	Les noms de *variables* doivent être des noms, avec les premières lettres des mots à partir du second en majuscules.
functions and private methods	`redirectUser`	Commencer par des verbes en infitif, avec les premières lettres des mots à partir du second en majuscules.
public methods	`GetReport`	-
structures	`ReportList`	Les noms de *structure des données*doivent être des noms avec les

Type	Example	Règle
		premières lettres de chaque mot interne en majuscules.
enumerations	`CategoryCode`	-
constants	`ServerPort`	-
filenames	`routes.go`	Le nom d'un *fichier* est toujours écrit en caractères miniscules.

CONVENTIONS SQL

CONVENTIONS DE NOMMAGE

Type	Example	Règle
Tableaux	`reports, users`	Les noms des tableaux doivent terminer par `s` (en pluriel) et en miniscule.
Primary Keys	`pk_userid, pk_reportid`	-
Foreign Keys	`fk_userid, fk_categoryid`	-
colonnes	`username, email`	Les noms des colonnes doivent être en minuscule séparé par des _

A ÉVITER

la langue anglaise doit toujours être utilisée, sauf pour les choses imposées par le projet.

FORMAT

- Utiliser les majuscules pour tous les mots-clés SQL. ex, SELECT, INSERT, UPDATE, WHERE, AND, OR, LIKE ...
- Utiliser les commentaires si nécessaire
- Utiliser les parenthèses. ex, WHERE (color='red' AND (size = 1 OR size = 2))
- Utiliser des requêtes optimisées
- Ecrit les requêtes comme ceci

```
SELECT ...
FROM ...
WHERE (...
AND ... IN (
    SELECT ...
    FROM ...
    GROUP BY ...
    HAVING ...
) AS ... );
```