

# Rapport Technique

Florian Barrois      Nicolas Devillers      Valentin Jeanroy      Mehdi Loisel  
Jean Mercadier      Ismail Taleb      Willeme Verdeaux

25 mars 2014

# Table des matières

<b>1</b>	<b>Installation</b>	<b>3</b>
1.1	Installation . . . . .	3
1.1.1	Prérequis . . . . .	3
1.1.2	Obtenir l'application . . . . .	3
<b>2</b>	<b>Structure</b>	<b>4</b>
2.1	Arborescence . . . . .	4
2.2	MVC . . . . .	5
2.2.1	Modèle . . . . .	5
2.2.2	Vue . . . . .	5
2.2.3	Contrôleur . . . . .	6
2.3	Détail de classe . . . . .	6
2.3.1	Setting . . . . .	6
2.3.2	Datas et DBTools . . . . .	6
2.3.3	ScriptRunner . . . . .	6
2.3.4	Message . . . . .	6
2.3.5	DynamicTree . . . . .	7
2.4	Test . . . . .	7
<b>3</b>	<b>Modélisation</b>	<b>8</b>
3.1	UML . . . . .	8
3.1.1	UML du modèle . . . . .	8
3.1.2	UML de la vue . . . . .	9
3.2	Base de donnée . . . . .	9

# Introduction

Dans ce rapport, nous allons expliquer la partie technique du projet, en commençant par les prérequis au lancement du programme puis en détaillant la procédure d'installation. Ensuite, nous étudierons la structure du projet avec le modèle vue contrôleur et nous finiront par nous intéresser au à la modélisation avec l'UML.

# Chapitre 1

## Installation

### 1.1 Installation

#### 1.1.1 Prérequis

Pour lancer l'application, il est obligatoire d'avoir à disposition un serveur MySQL. Ainsi au démarrage le programme chargera ses informations dans la base de données. Mais si celle-ci n'existe pas, elle sera créée avec le nom de stcal.

Cette base de données permet ainsi à l'application de sauvegarder et recharger ses données au lancement et à la fin de l'exécution du programme.

**note :** Il est conseillé de créer un utilisateur spécifique au programme qui aura tous les privilèges sur la base de données stcal.

#### 1.1.2 Obtenir l'application

Intégralité de l'application se trouve sur GitHub sur le repo stcal : <https://github.com/Ricain/stcal>

**Exécutable** Pour obtenir un exécutable de l'application, soit un fichier jar il suffit de le télécharger sur GitHub à l'URL suivant : <https://raw.githubusercontent.com/Ricain/stcal/Main/stcal.jar>

**Code source** Pour obtenir le code source d'une manière "propre" (sans le télécharger directement dans un fichier zip sur GitHub), il suffit de cloner le projet :

```
$ git clone https://github.com/Ricain/stcal
```

L'intégralité du projet sera clonée dans un répertoire stcal sous le répertoire courant. Cela nécessite d'avoir git d'installé. Dans un environnement Linux, le gestionnaire de paquet permet de l'installer. Sur un mac, il faut installer executé la commande suivante après avoir installé Xcode :

```
$ xcode-select -install
```

Une fois git installé il est possible de faire des "commit" et des "pull request".

**IDEA** Il est cependant possible de cloner directement le projet à partir d'IntelliJ IDEA. Pour cela il suffit d'importer un projet à partir d'un VCS (Version Control System). L'IDE vous demandera le lien GitHub de l'application énoncé plus haut.

L'environnement de développement doit être configuré de telle manière que les fichiers *jar* soient bien inclus en tant que *library*.

# Chapitre 2

## Structure

### 2.1 Arborescence

Ci dessous ce trouve arborescence du projet. On y compte 6 dossiers et 52 classes.

```
stcal
|-- Stcal.java
|-- control
|   |-- CALsettings.java
|   |-- CustomRenderer.java
|   |-- DBTools.java
|   |-- DBsettings.java
|   |-- Datas.java
|   |-- ListTools.java
|   |-- Message.java
|   |-- OSplitCsv.java
|   |-- Outics.java
|   |-- ScriptRunner.java
|   |-- Settings.java
|   |-- exceptions
|       |-- MaxSoutenanceException.java
|       |-- NoSuchSettingException.java
|       |-- \-- NothingToSaveException.java
|   |-- parserDate.java
|   |-- \-- parserPeriod.java
|-- don
|   |-- DAgenda.java
|   |-- DCandide.java
|   |-- DCouple.java
|   |-- DCreneau.java
|   |-- DEtudiant.java
|   |-- DListe.java
|   |-- DPersonne.java
|   |-- DProf.java
|   |-- DSalle.java
|   |-- Soutenance.java
|   |-- Type.java
|   |-- \-- manager
|       |-- DCandideManager.java
|       |-- DCreneauManager.java
|       |-- DEtudiantManager.java
|       |-- DProfManager.java
|       |-- DSalleManager.java
|       |-- Manager.java
|       |-- SalleManager.java
|       |-- Singleton.java
|       |-- \-- SoutenanceManager.java
```

```

\-- fen
  |-- CreneauTableModel.java
  |-- DCoupleTransferHandler.java
  |-- DynamicTree.java
  |-- DynamicTreeOperator.java
  |-- FCal.java
  |-- FChooser.java
  |-- FCsv.java
  |-- FExportIcs.java
  |-- FInterface.java
  |-- FLier.java
  |-- FMenu.java
  |-- FSalles.java
  |-- FSettings.java
  |-- FStage.java
  \-- FTab.java

```

## 2.2 MVC

Arborescence du projet a été pensé pour respecter au mieux le **m**odèle **v**ue **c**ontroleur. Sous le répertoire *src* on trouve trois autres répertoire :

**control** contient l'ensemble des classes moteur à l'application (contrôleur).

**fen** abréviation de fenêtre, contient l'ensemble des classes permettant de dessiner l'application (vue).

**don** abréviation de donné, contient l'ensemble des classes concernant les données de l'application (modèle).

### 2.2.1 Modele

Le modèle constitue les données de l'application. L'ensemble des classes prévue à cette effet sont stocké sous le répertoire *don*. On y trouve des classes comme :

- Étudiant
- Soutenance
- Prof
- Agenda
- etc

Sauvegarder les données de l'application permet non seulement de garder un historique des stages mais également de réouvrir l'application et de la retrouver tel qu'on l'a fermé. A cet effet on trouve un sous répertoire *manager* qui contient un ensemble de classe permettant de faire le lien entre la base de donné et le modèle de donné de l'application. Chaque manager **doit** implémentent l'interface manager afin de respecté le concept de JBDD (enseigné en S4).

Le script qui permet d'installer la base de donné se trouve dans le répertoire *res* (ressource).

Les classe constituant le répertoire *don* on était pensé sur un modèle bien précis. On verra le MCD et l'UML dans la partie modelisation de ce rapport.

### 2.2.2 Vue

La vue est constitué des classes sous le répertoire *fen* elles permettent de dessiner les fenêtre. La bibliothèque graphique utilisé ici est *Swing*.

La fenêtre principale est dessiné par la classes *FInterface*. On lui ajoute des objet de type *FTab* afin de créer des onglets. Il est important de noter que aucune des classes fenêtres (classes dans le répertoire *fen*) n'hérite d'un quelconque objet de la bibliothèque graphique, elles sont plutôt composé de ces objets.

La classe *Stcal* contient une méthode *mac*. Cette méthode est importance pour adapter la partie graphique au système OS X.

### 2.2.3 Contrôleur

Les classes concernant le contrôleur sont situées dans le répertoire *control*. Ces classes font le moteur de l'application et servent de lien entre la partie modèle et la partie vue. La classe *Main* est une exception car elle fait partie du contrôleur mais ne se trouve pas dans le même répertoire que les autres classes.

Les classes *Datas*, *DBTools*, *DBSetting* et *ScriptRunner* permettent de gérer la partie donnée de l'application. Les méthodes `load()` et `save()` dans la classe *Datas* permettent respectivement de charger et sauvegarder le modèle dans la base de donnée. Ces deux méthodes sont respectivement appelées au début et à la fin du programme. La classe *DBTools* permet de faire des opérations sur la base de donnée et *DBSetting* contient les informations de connexion à la base de donnée et permet d'obtenir une connexion valide à celle-ci.

## 2.3 Detail de classe

### 2.3.1 Setting

Les classes héritées de *Setting* permettent de stocker des informations, celles-ci sont stockées dans le répertoire `/.stcal` dans le répertoire personnel de l'utilisateur. Même si l'application est lancée sur Windows, dans ce cas le dossier ne sera pas caché.

Ce système de stocker des informations est utilisé pour les identifiants de connexion à la base de donnée (*DBSetting*) et sauvegarder les choix de l'utilisateur rentrés dans le formulaire du calendrier (*CALsetting*). Il est important de noter que tout mot de passe dans ces fichiers ne sont pas cryptés. C'est pour quoi il est préférable d'avoir un utilisateur de base de donnée propre à l'application et surtout de ne pas mettre le mot de passe root.

### 2.3.2 Datas et DBTools

*Datas* est le lien entre la partie donnée et la partie contrôle. Elle contient donc des listes d'objets qui caractérisent la partie modèle. Ces listes sont statiques et publiques afin d'être utilisés par l'ensemble de la partie contrôle. La méthode `load()` et la méthode `save()` sont appelées au début du programme afin de charger intégralement les données et les stocker dans cette classe. Et l'inverse à la fermeture du programme.

*DBTools* est une classe qui effectue des opérations sur la base de donnée. Elle permet de (re)créer la base de donnée à partir du script SQL mais aussi de gérer les paramètres de connexion et donc la connexion.

### 2.3.3 ScriptRunner

*ScriptRunner* est une classe provenant du projet Ibatis et permet d'exécuter des scripts SQL comme le script d'installation de la base de donnée dans le répertoire *res*. Elle a été modifiée pour enlever les dépendances de son projet d'origine et être un peu adaptée à ce projet.

### 2.3.4 Message

Cette classe gère les popups, la sortie standard, et la sortie d'erreur.

Le principe des popups est pensé pour se rapprocher du javascript. En effet, afficher un popup consiste juste à appeler une méthode avec le message en paramètre. Pendant que le message est à l'écran, l'application s'arrête. De plus il est possible de passer une exception en paramètre afin d'afficher le message de cette dernière. Il y a quatre types de popup :

**notice** est un message d'information

**warning** affiche un avertissement

**error** affiche un message d'erreur

**question** affiche une question à l'utilisateur et celui-ci peut cliquer oui ou non

Dans cette classe on récupère également la sortie standard et la sortie standard d'erreur. Cela permet si besoin de rediriger tout ce qu'écrit l'application dans des fichiers comme des logs.

### 2.3.5 DynamicTree

Il s'agit d'un Jpanel dans lequel est placé un Jtree permettant d'envoyer la liste des stages et de l'afficher triée. Ainsi on peut avoir une arborescence dans la partie fenêtre.

Pour plus d'information sur les classes, allez dans le répertoire *doc* dans le quelle se trouve le résultat de *javadoc*.

## 2.4 Test

A part une classe symbolique, il n'y a pas de test sur l'ensemble du projet. Il est donc très facile d'avoir des pertes de fonctionnalité.



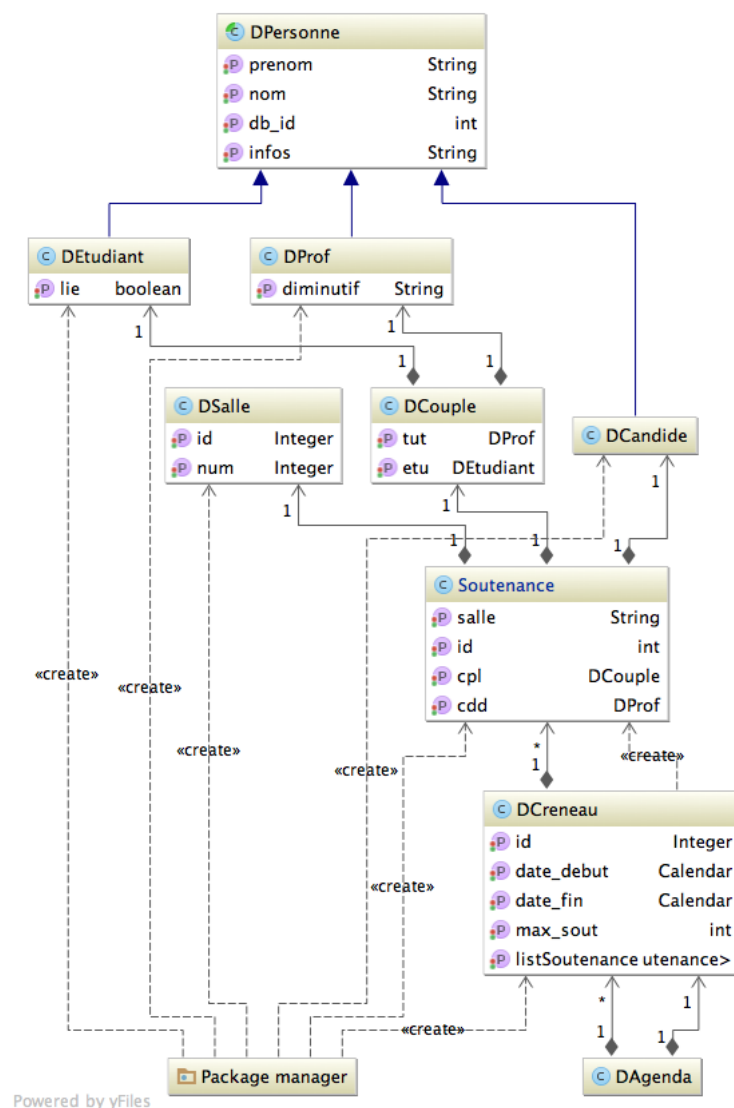
# Chapitre 3

# Modélisation

### 3.1 UML

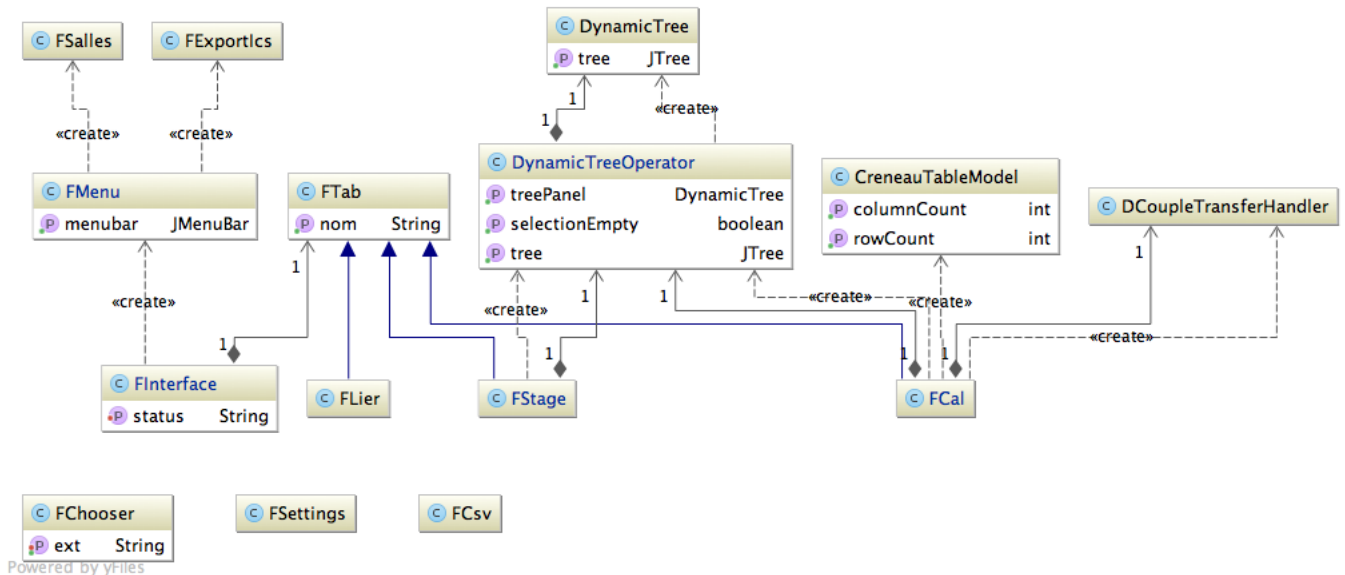
L'UML est divisé en trois parties, graphé sur le MVC. La partie contrôleur est majoritairement statique il n'est pas intéressant de voir sa représentation graphique. Dans les deux parties suivantes se trouvent les graphes UML du modèle et de la vue.

### 3.1.1 UML du modèle



Dans ce schéma voit bien l'enchaînement des objets. On commence par les personne et on fini par les agendas. On remarque également que le paquet *manager* crée la totalité objets. Cela vient du fait que les classes de manager instancie ces objets à partir de la base de donnée.

### 3.1.2 UML de la vue



Dans ce graphe, on remarque bien que la fenêtre principale (*FInterface*) es composé de d'onglets (*FTab*). On remarque également trois objet hérité de *FTab*, ces dernier constitue les trois onglets de l'application.

## 3.2 Base de donnée

La base de donnée est a été conçue à partir du schéma UML du modèle montré précédemment. Elle concerne que les données en non la vue ou le contrôleur.

# Conclusion

En espérant avoir été assez claire sur la modélisation de l'application et sur les grande ligne de sa structure. Nous espérons que l'application sera repris facilement par un future groupe de projet tuteuré.