

# Rapport Technique

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

22 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	MVC . . . . .	4
2.1.1	Modele . . . . .	4
2.1.2	Vue . . . . .	4
2.1.3	Controleur . . . . .	5
2.2	Détail de class . . . . .	5
2.2.1	Setting . . . . .	5
2.2.2	Datas et DBTools . . . . .	5
2.2.3	Message . . . . .	5
<b>3</b>	<b>Modelisation</b>	<b>6</b>
3.1	UML . . . . .	6
3.2	Base de donnée . . . . .	6

# Introduction

# Chapitre 1

## Installation

### 1.1 Installation

#### 1.1.1 Prérequis

Pour lancer l'application, il est obligatoire d'avoir a disposition un serveur MySQL. Ainsi au lancement le programme chargera ses information dans la base de donn  . Mais si la base de donn   n'existe pas celui ci sera cr     avec le nom de stcal.

Cette base de donn     permet ainsi    l'application de sauvegarder et recharger ses donn     au lancement et    la fin de l'ex  cution du programe.

**note :** Il est conseill   de cr    r un utilisateur sp  cifique au programme qui aura tout les privileges sur la base de donn     stcal.

#### 1.1.2 Obtenir l'application

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

**Executable** Pour obtenir un ex  cutable de l'application, soit un fichier jar il suffit de le telecharger sur GitHub    l'url suivant : <https://raw.githubusercontent.com/Ricain/stcal/Main/stcal.jar>

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

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

L'int  gralit   du projet sera clon   dans un repertoire stcal sous le repertoire courant. Cela n  cessite d'avoir git d'install  . Dans un enviroment linux, le gestionnaire de packet permet de l'installer. Sur un mac, il faut installer executer la commande suivante apres avoir install   Xcode :

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

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

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

# Chapitre 2

## Structure

### 2.1 MVC

L'arborescence du projet a été pensée pour respecter au mieux le modèle vue contrôleur. Sous le répertoire *src* on trouve trois autres répertoires :

**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).

#### 2.1.1 Modèle

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

- Etudiant
- 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 fermée. A cet effet on trouve un sous-répertoire *manager* qui contient un ensemble de classes permettant de faire le lien entre la base de données et le modèle de données de l'application. Chaque manager **doit** implémenter l'interface *manager* afin de respecter le concept de JBDD (enseigné en S4).

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

Les classes constituant le répertoire *don* ont été pensées sur un modèle bien précis. On verra le MCD et l'UML dans la partie modélisation de ce rapport.

#### 2.1.2 Vue

La vue est constituée des classes sous le répertoire *fen* elles permettent de dessiner les fenêtres. La bibliothèque graphique utilisée ici est *Swing*.

La fenêtre principale est dessinée par la classe *FInterface*. On lui ajoute des objets 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 composées de ces objets.

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

### 2.1.3 Controleur

Les classes concernant le controleur sont situ   dans le repertoire *control*. Ces class font le moteur de l'application et servent de lien entre la partie modele et la partie vue. La class *Main* est une exception car elle fait partie du controler mais ne se trouve pas dans le meme repertoire que les autres class.

Les class *Datas*, *DBTools*, *DBSetting* et *ScriptRunner* permettent de gerer la partie donn  e de l'application. Les methodes `load()` et `save()` dans la class *Datas* permettent respectivement de charger et sauvegarder le model dans la base de donn  e. Ces deux methodes sont respectivement appell  es au debut et    la fin du programme. La class *DBTools* permet de faire des operation sur la base de donn  e et *DBSetting* contient les information de connection    la base de donn  e et permet d'obtenir une connection valide    celle ci. *ScriptRunner* est une classe provenant du projet Ibatis et permet d'executer des scripts SQL comme le script d'installation de la base de donn  e dans le repertoire *res*.

## 2.2 Detail de class

### 2.2.1 Setting

Les class herit  es de *Setting* permettent de stocker des information, celles-ci sont stock  es dans le repertoire `/.stcal` dans le repertoire personnel de l'utilisateur. M  me si l'application est lanc  e sur Windows, dans ce cas le dossier ne sera pas cach  .

Ce systeme de stocker des informations est utilis   pour les identifiants de connection    la base de donn  e (*DBSetting*) et sauvegarder les choix de l'utilisateur rentr  s dans le formulaire du calendrier (*CALsetting*). Il est important de noter que tout mot de passe dans ces fichiers ne sont pas crypt  s. C'est pour quoi il est preferable d'avoir un utilisateur de base de donn  e propre    l'application et surtout de ne pas mettre le mot de passe root.

### 2.2.2 Datas et DBTools

*Datas* est le lien entre la partie donn  e et la partie controle. Elle contient donc des listes d'object qui caracterise la partie model. Ces listes sont static et publique afin d'etre utilis  es par l'ensemble de la partie controle. La methode `load()` et la methode `save()` sont appell  es au debut du programme afin de charger l'integralit   des donn  es et les stocker dans cette class. Et l'inverse    la fermeture du programme.

*DBTools* est une class qui effectue des operations sur la base de donn  e. Elle permet de (re)creer la base de donn  e    partir du script SQL mais aussi de gerer les param  tres de connection et donc la connection.

### 2.2.3 Message

Cette class gere les popups, la sortie standart, et la sortie d'erreur.

Le principe des popup est pens   pour se rapprocher du javascript. En effet, afficher un popup consiste juste    appeler une methode avec le message en param  tre. Pendant que le message est    l'ecran, l'application s'arrete. De plus il est possible de passer une exception en param  tre afin d'afficher le message de cette derniere. Il y a quatre type de popup :

**notice** est un message d'information

**warning** affiche un avertissement

**error** affiche un message d'erreur

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

Dans cette class on recupere egalement la sortie standart et la sortie standart d'erreur. Cela permet si besoin de rediriger tout ce qu'ecrit l'application dans des fichiers comme des logs.

## Chapitre 3

# Modelisation

### 3.1 UML

### 3.2 Base de donnée

# Conclusion