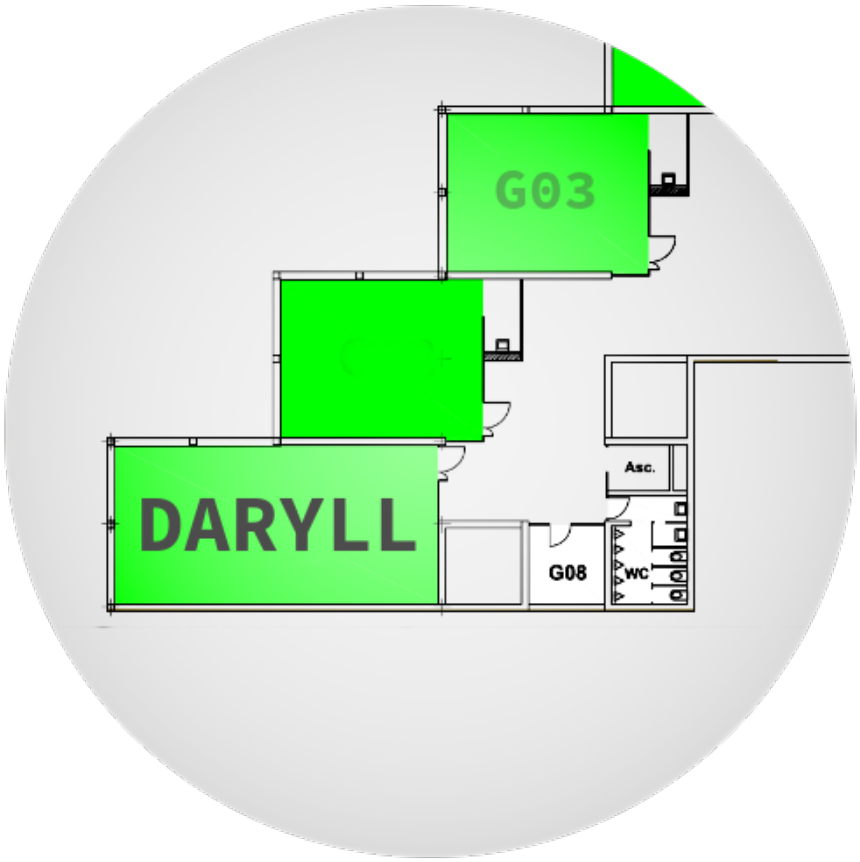
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud (HEIG-VD)

---

Projet de semestre (PRO)




# DARYLL
## Localisateur de salles disponibles

---

*Auteurs :*
Dejvid Muaremi
Aurélien Siu
Romain Gallay
Yohann Meyer
Loïc Frueh
Labinot Rashiti

*Client :*
René Rentsch
*Référent :*
René Rentsch

23 mai 2018

# Table des matières

**1 Introduction** **1**

**2 Objectifs** **1**

**3 Conception & Architecture** **1**
3.1 Technologies et outils utilisés . . . 1
3.1.1 Java 8 . . . 1
3.1.2 JavaFX 8 . . . 1
3.1.3 Scène FXML . . . 2
3.1.4 Scene Builder 8.3.0 . . . 2
3.1.5 Maven . . . 2
3.1.6 Git . . . 2
3.1.7 GitHub . . . 2
3.1.8 MySQL . . . 3
3.2 Docker . . . 3
3.3 Comparaison de l'interface finale avec la maquette . . . 3
3.4 Architecture . . . 4
3.4.1 Définition de la base de données . . . 4
3.5 Schemas BDD + commentaires . . . 5

**4 Description technique** **6**
4.1 Interface graphique . . . 6
4.1.1 Fonctionnement de JavaFX . . . 6
4.1.2 Scène principale . . . 6
4.2 Scènes secondaires . . . 6
4.2.1 Gestion des évènements selon le composant ciblé . . . 7
4.2.2 Redimensionnement . . . 8
4.2.3 Transformation ICS . . . 8
4.2.4 Transformation des plans des bâtiments . . . 9
4.2.5 Communication client - serveur . . . 9
4.2.6 Génération du fichier imprimable . . . 9

**5 Difficultés rencontrées** **9**
5.1 Interface graphiques . . . 9
5.2 Restructurer les plans . . . 10
5.3 Gestion des dates . . . 10

**6 Problèmes connus dans le programme final** **10**

**7 Conclusion** **10**
7.1 projet . . . 10
7.2 groupe . . . 10
7.3 avis des membres . . . 10

**Glossaire** **11**

**8 Annexes** **13**

# Table des figures

1 Maquette de l'interface . . . . . 3
2 Interface finale du programme . . . . . 4

# 1 Introduction

Daryll est un localisateur de salles disponibles se basant sur les possibilités graphiques de JavaFX ainsi que d'autres technologies, telles que MySQL. Réalisé pendant le cours de PRO (projet de semestre) de deuxième année auxquels sont astreints les étudiants en TIC. La durée de développement du programme fût de 10 semaines sur les 13 totales, avec un délai de remise au 24 Mai 2018.
Dans le cadre du projet, l'équipe de développement est composée du chef de projet Yohann Meyer, de son suppléent Loïc Frueh et des développeurs Aurélien Siu, Dejvid Muaremi, Labinot Rashiti, et Romain Gallay.

# 2 Objectifs

À l'heure actuelle, aucun outil interne de l'HEIG-VD ne permet d'obtenir une information claire et d'obtenir une rapide vue générale des disponibilités des salles de cours de la HEIG. L'objectif de notre projet est de régler ce problème en proposant un utilitaire ergonomique, complet et multi-plateforme.

Se basant sur les données GAPS, Daryll proposera une interface claire et rapide aux utilisateurs à la recherche d'une salle libre pour un ou plusieurs horaire(s) donné(s), ainsi que la liste des disponiblités pour une classe donnée. Il permettra dans ce second cas d'imprimer l'horaire selon les spécifications de l'utilisateur.

# 3 Conception & Architecture

## 3.1 Technologies et outils utilisés

### 3.1.1 Java 8

Parmi les deux langages de haut niveau proposés pour le projet (Java ou C++), nous avons choisi d'utiliser Java pour sa portabilité, sa sécurité et ses performances, ainsi que l'avantage d'une gestion de la mémoire par le garbage collector ; spécialement vu l'utilisation extensive que nous faisons d'image SVG, rapidement chères en mémoire. De plus, notre équipe disposait d'une expérience plus grande en développement JAVA que C++, et disposait même d'une préexpertise en JavaFX.

### 3.1.2 JavaFX 8

Dans le cadre du développement de Daryll , trois technologies à choix s'offrait à nous : Swing, JavaFx et Qt.

Notre choix s'est porté sur JavaFX, successeur de Swing en tant que librairie de création d'interfaces graphiques officielles du langage. Le SDK de JavaFX est intégré au JDK standard Java SE et dispose de plusieurs fonctionnalités remarquablement adapté à notre projet, spécialement les images JavaFX et la possibilité de les obtenir depuis un SVG.

### 3.1.3 Scène FXML

Le FXML est un langage inspiré du HTML ou XML et indique la définition de l'interface graphique utilisateur. Dans notre cas, nous avons plusieurs fichiers FXML qui contiendront toutes les balises nécessaires à la représentation d'une "view". Chaque balise représente donc un composant de cette vue. Grâce au langage FXML, il est possible de personnaliser chacune des balises avec des attributs et des valeurs.

### 3.1.4 Scene Builder 8.3.0

Scène Builder de Gluon permet de manipuler des objets JavaFX graphiquement et d'exporter ceux-ci dans un fichier .fxml interprétable et modifiable par la librairie graphique. Il nous permet également de voir l'interface comme si le programme était en cours d'exécution et de pouvoir y personnaliser directement les composants graphiques. Cette application simplifie donc considérablement la personnalisation, car il n'est pas nécessaire de réécrire tout le FXML à la main, et permet de plus de créer facilement un mockup de l'objectif.

### 3.1.5 Maven

La démultiplication des diverses librairies nécessaires à notre projet nous a vite poussé vers un outil de gestion de dépendances, et Maven s'est imposé de par l'habitude de notre équipe dans son utilisation, sa facile implémentation avec notre IDE de préférence (intelliJ), la gestion des tests unitaires et surtout l'aisance avec laquelle il est possible de génerer un .jar du projet.

### 3.1.6 Git

Git, le gestionnaire de version décentralisé libre, utilisé afin de gérer la totalité du projet ainsi que toutes ses modifications.

Nous avons créé une nouvelle branche par thème. Plus précisément, une branche pour le client, le serveur, l'administration (gantt, etc.) et une pour les rendus.

De pouvoir tirer parti d'une structure de développement commune aussi précise nous a permis de garantir une efficacité dans le travail en commun.

### 3.1.7 GitHub

GitHub, le service web permettant de parcourir visuellement l'historique Git de Daryll et qui nous a également permis de l'héberger en ligne.

GitHub offre également de nombreuses fonctionnalités ainsi que des outils de gestion pour le projet.

### 3.1.8 MySQL

Le système de gestion de bases de données relationnelles (SGDBR), faisant partie des logiciels de gestion des bases de données les plus utilisés au monde et dont l'entiereté de notre équipe connaît avec suffisamment de détails pour être facilement utilisé dans un projet de cette envergure.

## 3.2 Docker

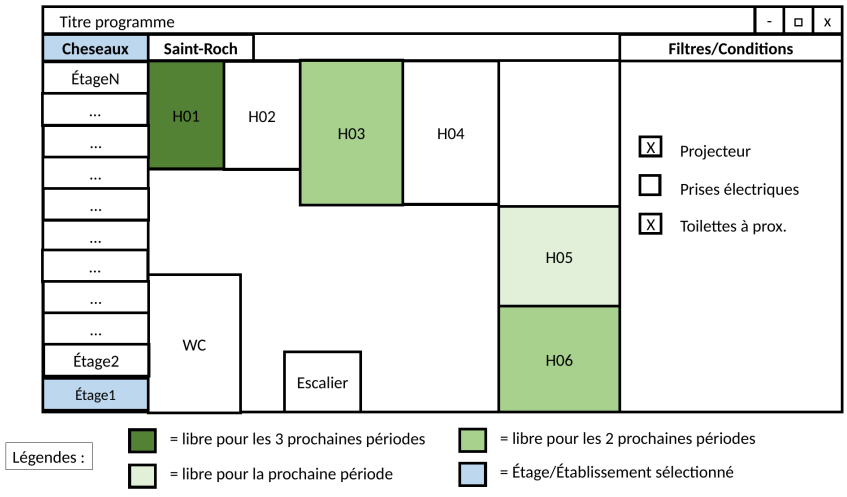
Docker, le logiciel libre qui automatise le déploiement d'application dans des conteneurs logiciels. Ces conteneurs sont isolés et peuvent être exécutés sur n'importe quel système qui prend en charges Docker.

Ceci nous permet d'étendre la flexibilité et la portabilité de notre serveur avec un minimum de travail de configuration supplémentaire.

## 3.3 Comparaison de l'interface finale avec la maquette

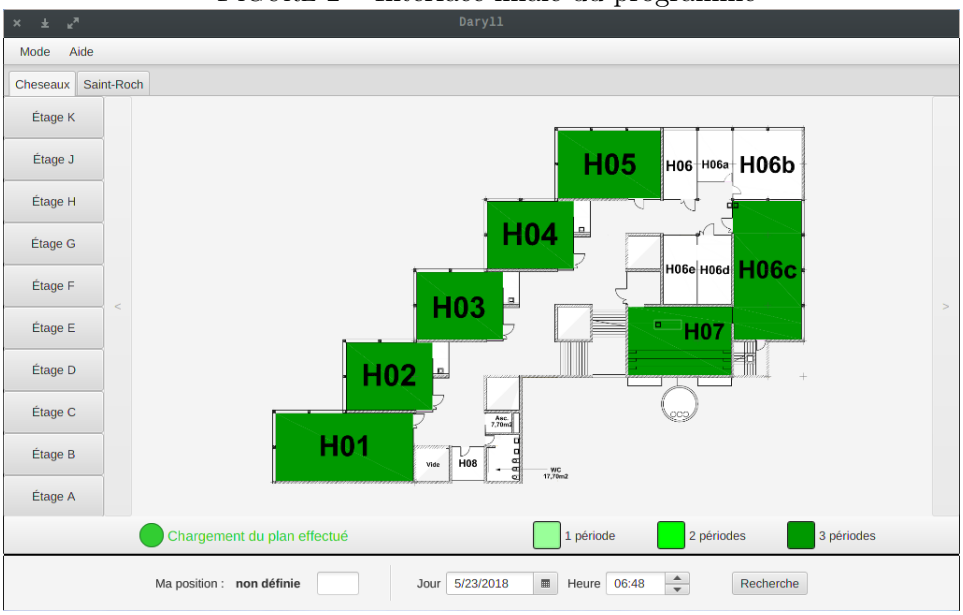
La maquette de l'interface à été conçue au début du projet en tenant compte de toutes les fonctionnalités que le programme devait fournir. Les figures 1 et **??** ci-dessous proposent une comparaison entre la maquette et le résultat final.

FIGURE 1 – Maquette de l'interface




Comme le montrent les figures 1 et 2, l'interface conçue lors de l'élaboration du cahier des charges à été repris presque entièrement pour notre programme. Les différences majeurs se retrouve dans l'absence de la partie filtre/condition (voir plus loin), la console GUI permettant d'obtenir des informations sur l'état du logiciel ainsi que la barre de recherche en bas de la fenêtre.

FIGURE 2 – Interface finale du programme




## 3.4 Architecture

### 3.4.1 Définition de la base de données

#### Period

Cette table contient une liste de 15 périodes qui représente les différentes périodes possibles dans l'horaire GAPS.
Une période est identifiée par un numéro unique allant de 0 à 15 et représentant le numéro de la période sur l'horaire journalier de GAPS, se caractérise par une heure de début et une heure de fin.
Chaque période est reliée à une ou plusieurs salles de classe.

#### Classroom

Cette table contient la liste des salles des campus de Cheseaux et Saint-Roch de l'HEIG-VD.
Une salle est identifiée par son nom (A01, A02, ...), et se caractérise par un booléen qui indique si elle est verrouillée ou non.
La salle est reliée à une ou plusieurs périodes, possède un équipement qui lui est propre, et se trouve dans un étage du bâtiment.

#### TakePlace

Le numéro d'une période P est relié à une salle de classe S par un TakePlace, cette relation se caractérise par une date.

#### Floor

Cette table contient la liste des étages des campus de Cheseaux et Saint-Roch de l'HEIG-VD.

Un étage est identifié par son nom (A, B, ...), et se caractérise par le campus auquel il appartient.
Un étage est relié à plusieurs salles et possède un équipement qui lui est propre.

**ClassroomEquipments**

Cette table permet de spécifier en détail l'équipement présent dans une salle.
L'équipement de la salle est identifié par un numéro unique, et se caractérise par des booléens qui indique la présence d'un beamer, de prises électriques, d'ordinateurs et d'un tableau blanc ou noir.
Un équipement est relié à une et une seule salle de classe.

**FloorEquipments**

Cette table permet de spécifier en détail l'équipement présent dans un étage.
L'équipement d'un étage est identifié par un numéro unique, et se caractérise par des booléens qui indique la présence de toilette pour hommes, de toilette pour femmes, d'une machine à café, d'un distributeur Selecta ou équivalent et d'un accès à une sortie du bâtiment.
Un équipement est relié à un et un seul étage.

## 3.5 Schemas BDD + commentaires

# 4 Description technique

## 4.1 Interface graphique

### 4.1.1 Fonctionnement de JavaFX

Afin de comprendre comment marche JavaFX, il faut imaginer notre programme comme étant une pièce de théâtre. JavaFX utilise cette image afin de structurer le programme ainsi que son interface graphique.

Nous nous retrouvons donc avec des termes comme "Stage", "Scene" et différents composants animant cette scène. Nous vous expliquerons nos différents composants et scènes plus tard dans la documentation.

JavaFX intègre également la notion de MVC (Modèle-vue-contrôleur). Ils sont créés par défaut lors du commencement d'un projet. La classe principale étant le modèle, la vue étant le fichier FXML et le contrôleur étant le fichier Java gérant les interactions avec le fichier FXML.

### 4.1.2 Scène principale

Lors du début du projet, le programme était une simple application JavaFX. Le projet DARYLL possédait donc uniquement un fichier contenant sa classe, une vue vide (fenêtre principal ou scène principale) et le contrôleur de cette vue (donc selon le modèle MVC vu précédemment).

La première tâche a été de choisir le conteneur principal de DARYLL et pour cela, il y a plusieurs choix. Certains ont des avantages que d'autres n'ont pas. En réalité, le choix dépend de l'utilisation de l'application et du rendu final.

Nous avons essayé plusieurs conteneurs tel que le Anchor Pane, le Grid Pane ou le simple Pane, mais notre choix a penché pour le Border Pane. Il aurait été possible de réaliser notre programme avec les autres conteneurs mentionnés, mais étant donné que nous avons décidé d'intégrer le redimensionnement de la fenêtre pour les plans des étages, l'implémentation aurait été plus compliquée et le Border Pane nous facilitait beaucoup plus la tâche à ce niveau-là.

Le Border Pane est donc plus intéressant, car il est déjà séparé en différentes zones (top, bottom, center, left, right), et grâce à ces zones, les composants prennent automatiquement la bonne taille avec le redimensionnement. L'autre aspect également important pour le Border Pane est le fait qu'il respecte le plus possible au mockup défini au début du projet.

## 4.2 Scènes secondaires

Par scènes secondaires, nous entendons les autres fenêtres ou pop-up qui s'ajoutent à la scène principale (fenêtre principale) de DARYLL. À la base, il n'y avait qu'une seule scène et un seul contrôleur qui gérait ladite scène. Les autres scènes étaient créées directement dans le contrôleur principal via du code.

Cette solution fonctionnait bien, mais ne donnait pas un rendu comme nous l'avions désiré. De plus, le contrôleur de la fenêtre principale devenait facilement énorme vu les nombreux pop-up que nous avons rajoutés pour les options du programme. Donc au final, cette solution ne donnait pas un bon rendu et n'était pas très évolutive, car il est difficile de retrouver la bonne vue dans un seul fichier.

Après des recherches et réflexions, nous avons décidé de dispatcher le FXML en plusieurs parties. Dorénavant, chaque fenêtre graphique aura son propre fichier FXML et son propre contrôleur gérant les interactions avec l'utilisateur de ladite fenêtre.

Cette nouvelle solution permet d'avoir un rendu nettement meilleur à l'affichage et rend le code beaucoup plus propre. En effet, lorsqu'il faudra mettre à jour ou dépanner une fenêtre du programme, il suffira d'aller dans son fichier FXML et contrôler pour effectuer les changements !

### 4.2.1 Gestion des évènements selon le composant ciblé

Cet aspect du projet mérite son propre titre de par la démarche réalisée afin d'optimiser le code.

Si nous regardons le mockup de base, nous apercevons que la zone de gauche contient tous les boutons des étages. La démarche basique aurait été de créer une fonction pour chaque bouton d'étage dans le contrôleur de la vue et ensuite de les assigner une à une via Scene Builder. Cela veut dire que pour x boutons nous aurons x fonctions.

Alors évidemment, cela fonctionnera, mais n'oublions pas que le contrôleur est déjà très conséquent en termes de code. S'il faut encore rajouter une fonction par étage en sachant que Cheseaux en a environ dix, le code deviendrait illisible et donc plus difficile à maintenir.

La nouvelle solution adoptée a été de factoriser ce code. En effet, les boutons d'étages ont le même but : afficher le plan de l'étage avec ses salles disponibles au moment X.

L'idée est donc de faire une fonction permettant de faire ce changement pour chaque bouton d'étage. Idéalement la fonction devra prendre le bouton ciblé en paramètre, mais malheureusement Scene Builder ne le permet pas...

Après plusieurs recherches, nous avons trouvé une alternative que nous avons utilisée tout au long du projet pour l'interface graphique. En fait, cette alternative permet de retrouver le composant ciblé via un objet "Event". Cet objet "Event" nous permet de retrouver la scène où se trouve le composant ciblé puis de récupérer le composant via son ID.

Cela veut dire que la fonction gérant le changement de plan (showFloor) ne va pas prendre un bouton en paramètre, mais un objet "Event" qui comprend le clic de la souris. C'est grâce à cette méthode que nous avons pu factoriser le code et le rendre plus propre.

### 4.2.2 Redimensionnement

L'interface graphique de DARYLL affiche les plans des étages de Cheseaux ou de Saint-Roch. Ces plans ont une certaine taille et cela nous amène à une question problématique sur l'affichage des plans sous différentes résolutions. Si nous imaginons notre application dans un cas concret d'utilisation, alors il se pourrait que l'utilisateur ait une mauvaise qualité d'image à cause de sa résolution d'écran.

C'est pour cela que DARYLL offre la possibilité d'agrandir ou de rapetisser la fenêtre afin que l'affichage du plan soit en adéquation avec la résolution de l'utilisateur. Alors bien sûr, cette fonctionnalité paraît toute simple, mais en réalité la mise en place est très difficile. Nous allons vous expliquer la démarche que nous avons eue tout au long du projet dans les paragraphes qui suivent.

Suite à des réunions concernant cette question, le groupe s'est mis d'accord sur le fait qu'il était préférable d'intégrer le redimensionnement. L'idée initiale était d'afficher le plan sur un conteneur puis d'incruster des composants au-dessus de chaque salle de l'étage afin d'indiquer le niveau de disponibilité de la salle avec un teint de vert. L'implémentation fut difficile, car il fallait faire attention à ce que les composants restent correctement placés sur les étages lorsque l'utilisateur agrandissait la fenêtre.

Cette idée a été mise de côté, car les tests n'ont pas été concluants et que le rendu final n'était pas propre. Nous avons donc eu une autre idée qui était d'utiliser les propriétés CSS du FXML. En effet, il suffisait d'aller dans les propriétés CSS du conteneur, de mettre la CSS correspondante en indiquant le lien du plan ainsi d'autres propriétés CSS et le plan affiché devenait redimensionnable. Malheureusement nous avons dû laisser tomber cette idée, car elle nous limitait à simplement afficher un plan alors que nous étions partis du principe d'afficher un plan avec des salles colorées en vert pour indiquer le niveau de disponibilité de la salle. Chose qui était impossible avec le CSS, car nous ne pouvions pas indiquer quelle salle colorier avec quel teint de vert.

Après multiples recherches, nous avons eu l'idée de contourner le problème. En effet, au lieu de rajouter des éléments sur le plan puis de les modifier au fur et à mesure selon les disponibilités, nous allons modifier directement le plan lui-même, colorier chaque salle et afficher le résultat. La différence étant que nous ne travaillons plus avec des formats d'images ordinaires, mais bel et bien avec le format SVG.

Le format SVG nous permet de définir des zones sur un plan. Chaque zone définissant un étage contiendra un ID permettant de l'identifier dans le programme. Les plans nous ont été fournis en PDF, mais grâce à l'outil Inkscape (éditeur d'images open source), il est possible de créer des fichiers SVG à partir de ces PDF. Il faudra donc, pour chaque étage de chaque établissement, définir les zones de chaque salle... Énorme tâche, mais pour un rendu final convainquant !

### 4.2.3 Transformation ICS

À partir du calendrier ICS de GAPS, nous avons dû implémenter un parseur afin de pouvoir parcourir ce fichier et récupérer la liste des salles occupées. Cette liste a été ensuite ajoutée à la base de données afin que l'on puisse y effectuer des requêtes dessus.

#### 4.2.4 Transformation des plans des bâtiments

Les plans fournis n'étaient pas parfaitement adaptés à notre application, ils étaient au format PDF et il y avait beaucoup d'informations superflues qui rendaient leur lecture très difficile. Par conséquent, il a fallu les modifier à la main un par un afin d'obtenir des fichiers SVG que l'on utilisera. Les modifications ont dû être faites sur chaque bâtiment, chaque étage, chaque salle et surtout sur les fragments de salles qui ont rendu la manipulation particulièrement difficile.

#### 4.2.5 Communication client - serveur

Nous avons appliqué ce que nous avons appris lors du cours de RES. Nous avons donc défini un paquet "network" qui va effectuer la sérialisation et l'envoie des données par le réseau en utilisant des sockets TCP.

Nous avons mis en place un protocole de communication entre le client et le serveur pour l'échange d'informations, à savoir le type, le contenu, et la fin des requêtes.

#### 4.2.6 Génération du fichier imprimable

L'implémentation d'une commande "imprimer" est extrêmement compliquée par conséquent nous avons choisi de créer un fichier texte temporaire et imprimable, qui sera ouvert automatiquement par l'éditeur de texte par défaut de l'utilisateur. Celui-ci devra ensuite décider s'il veut sauver ou imprimer ledit fichier. Si l'utilisateur ne sauvegarde pas le fichier, celui-ci sera écrasé lors de la prochaine requête.

Cette méthode nous permet de répondre à deux demandes à la fois, la possibilité de sauver un fichier contenant les disponibilités ainsi que la possibilité d'imprimer un fichier.

# 5 Difficultés rencontrées

## 5.1 Interface graphiques

Il y a de fortes chances que suite aux informations données dans les précédents points du rapport, les points critiques ont déjà été remarqués.

Pour l'interface graphique, la plus grosse difficulté a été la mise en place de l'affichage des plans des étages en gérant le redimensionnement et le fait que les salles doivent être colorées pour indiquer leur niveau de disponibilité.

Cela a été vraiment le plus gros souci dans cette partie. Nous voulions offrir à l'utilisateur cette fonctionnalité afin que l'utilisation de DARYLL soit des plus agréables et c'est chose faîte.

Bien sûr, nous sommes passés par plusieurs chemins pour avoir ce rendu.

Nous nous sommes tout de même dit que, dans les pires des cas, nous afficherons un simple tableau avec les salles et les informations nécessaires si l'affichage des plans était trop compliqué. Cependant, il faut avouer que cela aurait été moins remarquable.

Nous avons eu d'autres difficultés comme le fait de gérer les interactions entre vues et contrôleurs, certains composants graphiques ne répondant pas à nos demandes, mais au final ce n'est rien de très grave.

### 5.2 Restructurer les plans

Les salles étant représentées par des triangles sur le plan, il a fallu les regrouper afin d'obtenir une salle dont on pourra modifier sa couleur en temps réel.

La modification n'est pas particulièrement difficile, mais elle prend énormément de temps, mais le résultat final est très beau.

### 5.3 Gestion des dates

Malheureusement, les dates SQL ne sont pas parfaitement supportées par Java, ceci nous a donné beaucoup de fil à retordre afin de générer une requête SQL, qui sera envoyée à la base de données

## 6 Problèmes connus dans le programme final

## 7 Conclusion

### 7.1 projet

### 7.2 groupe

### 7.3 avis des membres

Dejvid Muaremi :

Ce projet fut très intéressant à faire, j'ai pu découvrir de nouvelles technologies et mettre en pratique tout ce que j'ai appris jusqu'à maintenant. Parfois, j'ai pris du retard sur mes tâches, mais j'ai pu le rattraper au final.

Aurélien Siu :

Romain Gallay :

Yohann Meyer :

Loïc Frueh :

Labinot Rashiti :

# Glossaire

**AnchorPane** AnchorPane est une classe de JavaFX permettant (entre autres) de disposer des noeuds selon des coordonnées (x,y).. 7, 22

**Canvas** Type de Node spécialisé permettant de dessiner des lignes, des formes et des éléments graphiques dans un canevas dans le Scene Graph.. 5, 8, 22

**Color** Classe JavaFX permettant de représenter une couleur.. 5, 22

**ColorAdjust** Effet JavaFX permettant d'ajuster la couleur d'un nœud en modifiant son contraste, saturation, teinte et luminosité.. 22

**GaussianBlur** Effet JavaFX permettant d'affecter à un nœud un effet de flou gaussien d'intensité variable.. 22

**GEMMSCanvas** Classe spécialisant la classe Canvas de JavaFX pour les besoins de l'application.. 22

**GEMMSImage** Classe spécialisant la classe ImageView de JavaFX pour les besoins de l'application.. 8, 22

**GEMMSText** Classe spécialisant la classe Text de JavaFX pour les besoins de l'application.. 8, 22

**ImageView** Type de Node personnalisé permettant de représenter une image dans le Scene Graph. 5, 8, 22

**LayerList** LayerList est une classe implémentée pour l'application GEMMS qui est utilisée pour afficher les calques d'un document sous la forme d'une liste verticale de cellules.. 9, 10, 22

**ListView** Classe JavaFX permettant de représenter une liste d'éléments dans une liste verticale ou horizontale de cellules correspondantes.. 8, 22

**Node** La classe Node est une classe de JavaFX. Il s'agit de la classe de base de tout élément présent dans le Scene Graph. 5, 22

**Scene** La Scene, dans une application JavaFX, est le container pour un Scene Graph. Chaque scène doit avoir un et un seul nœud racine du Scene Graph. 4, 22

**Scene Graph** Le Scene Graph (graphe de scène en français) est une structure en arbre qui garde une représentation interne des éléments graphiques de l'application. 4, 22

**SepiaTone** Effet JavaFX permettant d'affecter à un nœud un effet sépia d'intensité variable.. 22

**Stackpane** Classe JavaFX permettant de positionner des nœuds sur une pile de l'arrière vers l'avant.. 7, 22

**Stage** Le Stage est le container haut-niveau d'une application JavaFX. Il doit contenir tout le contenu d'une fenêtre. 4, 22

**Sérialisation** La sérialisation, c'est rendre un objet persistant afin de pouvoir le stocker ou l'échanger de manière de textuelle. 5, 22

**Text** Type de Node spécialisé permettant de représenter un texte dans le Scene Graph.. 5, 8, 22

**Workspace** Workspace est une classe de l'application GEMMS représentant la zone de travail de l'utilisateur dans un Document GEMMS. Un objet contient notamment la liste de tous les calques utilisés dans le document. 7, 22

## Références

[1] Oracle. Working with the javafx scene graph. https://docs.oracle.com/javafx/2/scenegraph/jfxpub-scenegraph.htm, 2013.

[2] Oracle. Node(javafx). https://docs.oracle.com/javase/8/javafx/api/javafx/scene/Node.html, 2015.

[3] Oracle. Overview(javafx 8). https://docs.oracle.com/javase/8/javafx/api/, 2015.

[4] Oracle. Scene (javafx). https://docs.oracle.com/javase/8/javafx/api/javafx/scene/Scene.html, 2015.

[5] Oracle. Stage (javafx). https://docs.oracle.com/javase/8/javafx/api/javafx/stage/Stage.html, 2015.

[6] Stackoverflow. Why isn't javafx.scene.text.font serializable. https://stackoverflow.com/questions/17678999/why-isnt-javafx-scene-text-font-serializable, 2013.

[7] Wikipedia. Bresenham-algorithmus. https://de.wikipedia.org/wiki/Bresenham-Algorithmus, 2007.

# 8 Annexes

Haute Ecole d'Ingénierie et de Gestion du
Canton de Vaud (HEIG-VD)

# Projet de semestre (PRO)
# Journal de travail

*Auteurs :*
*Auteurs :*
Dejvid Muaremi
Aurélien Siu
Romain Gallay
Yohann Meyer
Loïc Frueh
Labinot Rashiti

*Client :*
René Rentsch
*Référent :*
René Rentsch

23 mai 2018

# A Journal de travail - Edward Ransome

## A.1 Semaine 1 : 20 février - 24 février

Création du groupe et recherche d'idées. Discussion et proposition des deux principaux sujets au professeur, un éditeur d'images et un programme de manipulation de graphes.

## A.2 Semaine 2 : 27 février - 3 mars

Programme d'édition d'images accepté, début d'élaboration du cahier des charges. Discussion sur le fonctionnement, l'architecture, les fonctionnalités voulues ainsi que des mock-ups d'interface. Travail simultané de tout le groupe.

## A.3 Semaine 3 : 6 mars - 10 mars

Fin de rédaction du cahier des charges, avec mock-ups finaux et une liste des fonctionnalités indispensables et optionnelles. Rendu de celui-ci ainsi qu'un planning du travail sous forme de diagramme de Gant.

## A.4 Semaine 4 : 13 mars - 17 mars

Retour sur le cahier des charges, pas de problèmes majeurs. Planification rédigée sous forme d'un tableau Excel pour faciliter la compréhension et mieux voir le travail effectué à chaque membre du groupe. Début du travail individuel selon le planning.

## A.5 Semaine 5 : 20 mars - 24 mars

Étude de JavaFX. Tutoriel JavaFX 8 de Code Makery effectué, qui comprends une introduction à Scene Builder, création de fenêtres, effets et autres fonctionnalités de cette librairie. Étude de la documentation Oracle de JavaFX.

## A.6 Semaine 6 : 27 mars - 31 mars

Début de création de l'interface avec Guillaume Milani en se basant sur les mock-ups élaborés pour le cahier des charges.

## A.7 Semaine 7 : 3 avril - 7 avril

Fin de l'interface Scene Builder avec contrôleurs des boutons dans le code ainsi que des références aux éléments du fichier « .fxml » dans le code pour pouvoir les modifier hors de Scene Builder.

## A.8 Semaine 8 : 10 avril - 14 avril

Aide à l'implémentation du Workspace dans l'interface graphique principale avec déplacement et zoom.

### A.9 Semaine 9 : 24 avril - 28 avril

Continuation de l'implémentation du Workspace dans l'interface.

### A.10 Semaine 10 : 1 mai - 5 mai

Début d'élaboration de la zone « Effet » de l'interface, recherche sur l'implémentation des effets en JavaFX ainsi que leur application sur divers éléments de la librairie (Image, texte, canvas).

### A.11 Semaine 11 : 8 mai - 12 mai

Création de différents boutons permettant d'augmenter l'opacité, saturation et l'effet sepia.

### A.12 Semaine 12 : 15 mai - 19 mai

Changements dans l'implémentation des effets, un « Slider » JavaFX par effet incrémentable, permettant de modifier l'opacité, saturation, contraste et sepia sur un ou plusieurs calques. Implémentation d'une remise à zéro des effets. Début de réfection sur la sérialisation des effets.

### A.13 Semaine 13 : 22 mai - 26 mai

Ajout de l'effet de flou, ainsi que un bouton « Tint » qui créé une nuance de couleur sur un calque. Implémentation de la sérialisation des effets et correction de l'ordre d'application des effets pour permettre leur application dans un ordre quelconque sans recréer tous les effets.

### A.14 Semaine 14 : 29 mai - 2 juin

Élaboration du rapport.