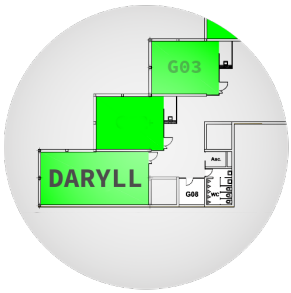
HAUTE ÉCOLE
D'INGÉNIERIE ET DE GESTION
DU CANTON DE VAUD
www.heig-vd.ch

Haute Ecole Spécialisée
de Suisse occidentale
Fachhochschule Westschweiz
University of Applied Sciences and Arts
Western Switzerland

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

24 mai 2018

Table des matières

Table des figures

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 ainsi qu'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.

De plus, nous désirons offrir au power-user la possibilité de faire des recherches composées plus pointues, et dans ce cas aussi d'en imprimer le résultat.

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 adaptées 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

Scene 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 que l'entièreté de notre équipe connaît avec suffisamment de détails pour être facilement utilisé dans un projet de cette envergure.

3.1.9 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.

3.2 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 2 ci-dessous proposent une comparaison entre la maquette et le résultat final.

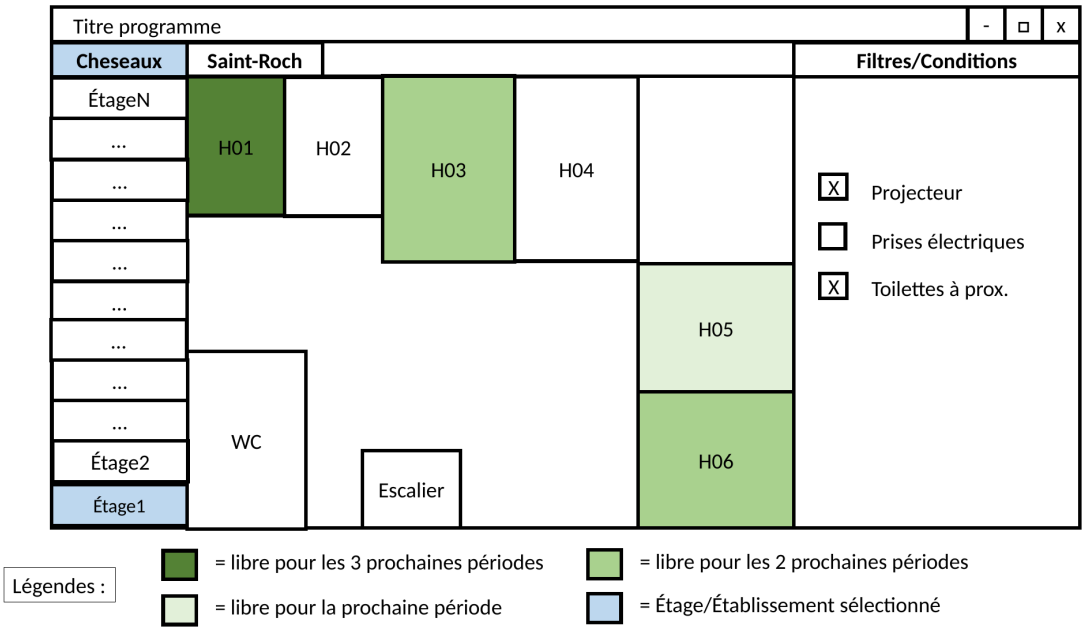


FIGURE *1 – Maquette de l'interface*

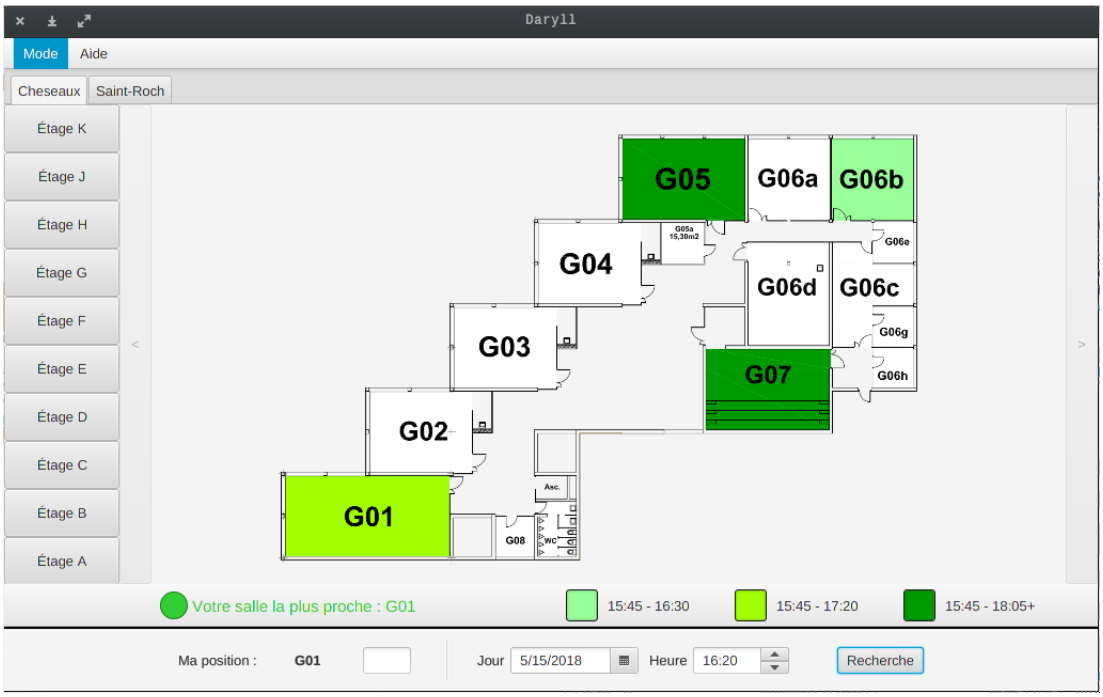


FIGURE *2 – Interface finale du programme*

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.

3.3 Architecture

Ce projet est basé sur un modèle MVC classique et une sérialisation en JSON. Ci-dessus, une vue par packages de l'architecture du client, du serveur et de leurs moyens de communication. Nous offrons ici une vision générale des différents packages qui composent notre application, tant client que serveur.

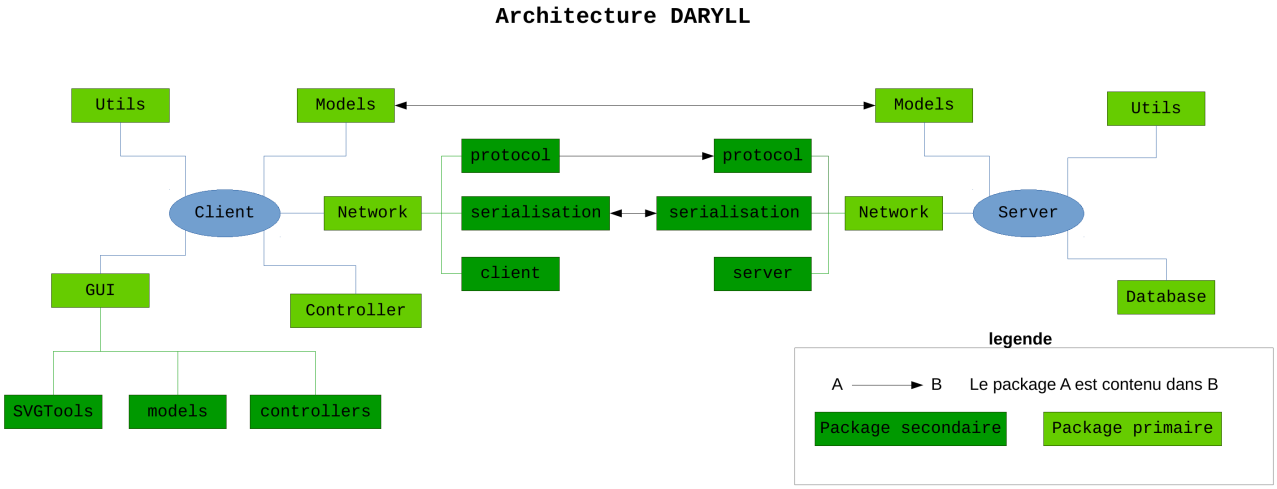


FIGURE *3 – Architecture du programme*

3.3.1 Architecture serveur, vue d'ensemble

- DATABASE : Interface Java <-> MySQL
- UTILS : Parsing ICS, mapping entre les période (représentation personnelle) et les heures
- MODELS :les modèles nécessaires pour s'adresser à la base de donnée.
- NETWORK : l'entièreté de la gestion des communications entre serveur et clients.
  - Server : Main et gestion de threads/connexion
  - Serialisation : transformation POJO <-> JSON
  - protocol : définition du protocole de communication

3.3.2 Architecture client, vue d'ensemble

- NETWORK :
  - Client : gestion connexion

- UTILS : constantes d'utilisation et mapping
- CONTROLLER : contrôleur du MVC
- GUI
- SVGTOOLS : SVG -> image JavaFX
- MODEL : modèle graphique
- CONTROLLERS : classes gérant les interactions des éléments graphiques

3.3.3 Architecture serveur

3.3.3.1 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.

3.3.3.2 Schéma de la base de données et commentaires

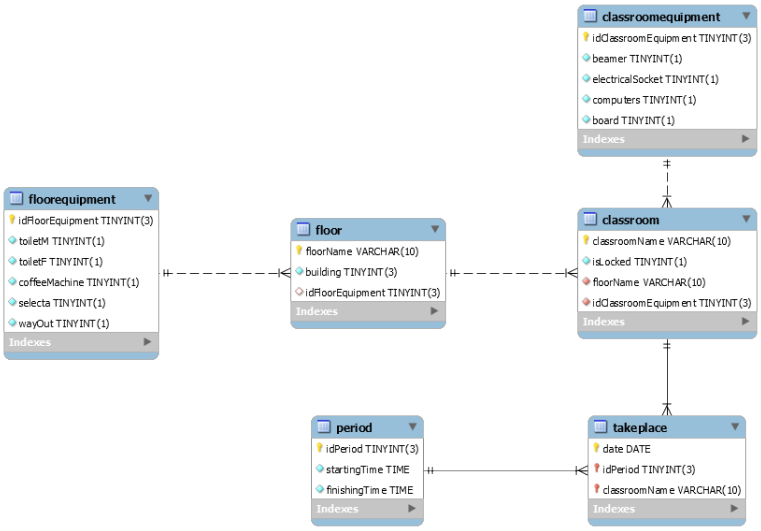


FIGURE *4 – schéma de la BDD*

Extrêment épurée et disposant de toutes les informations nécessaires avec un minimum de coût d'utilisation, notre base de données est adaptée à nos besoins.

Actuellement, seuls les tables Floor, Classroom, Takeplace et Period sont effectivement utilisées. En effet, l'utilisation des tables equipment implique l'accès à des données précises qui n'ont pas pu être fournie par les entités concernées de la HEIG.

Nous avons donc décidé de ne pas proposer la fonctionnalité de disponibilité de matériel pour l'instant, préférant une garantie d'exactitude à plus de possibilités. Les tables suscitées sont toujours disponible dans un souci d'évolutivité. Si un jour, DARYLL obtient un accès à l'inventaire de l'HEIG-VD, il n'y aura pas besoin de grandement modifier la base de données.

3.3.4 Architecture client

Lorsque l'utilisateur fait une requete quelconque via le GUI, celle-ci est récupérée par ce dernier, prétraitée et envoyée au controller via une de ses méthodes de traitement (ex. handleClientFloorRequest()). Le controller envoie la requete au serveur via la classe ClientSocket qui permet de se connecter au serveur et d'y envoyer des données sous forme

sérialisée en JSON. Les méthodes askFor... sont là pour cela (ex. askForClientFloorRequest).

Une fois la requête utilisateur envoyée au serveur, le controller utilise une nouvelle fois la classe ClientSocket et ses méthodes pour récupérer le résultat renvoyé. Ce résultat est ensuite traité en fonction du type de handler utilisé et retourne si necessaire ce dont le GUI a besoin pour l'affichage d'un résultat.

3.3.4.1 Résumé schématique

User –> GUI –> Controller –> ClientSocket –> Serveur –> ClientSocket –> Controller (–> GUI) –> User

3.3.4.2 Controller

Le controller est muni de 3 méthodes principales :

- handleClientFloorRequest()
- handleClientClassroomRequest()
- handleClientAdvancedRequest()

handleClientFloorRequest() sert à traiter la requête principale de notre logiciel, à savoir celle affichant les salles libres pour un étage sur un plan du GUI avec la coloration adaptée en fonction du nombre de périodes de disponibilité.

Cette méthode reçoit l'étage, la date et l'horaire demandé sous la forme d'un seul objet. Elle envoie cet objet à la classe ClientSocket puis attend une réponse. Lorsque ClientSocket a transmis la réponse du serveur, cette méthode la traite puis génére une map permettant au GUI d'afficher le plan de l'étage demandé muni des différentes couleurs.

handleClientClassroomRequest() traite la requête utilisateur pour une classe spécifique. Cette méthode ne reçoit qu'un objet représentant une classe, et envoie l'information au serveur via ClientSocket et se met en attente de la réponse.

Une fois la réponse obtenue, traite celle-ci et génére un fichier conternant toutes les plages horaires libres pour la journée et ouvre un éditeur de texte. De là, l'utilisateur peut l'enregistrer sur sa machine ou l'imprimer, avant ou après modifications.

Note : Le fichier généré porte toujours le même nom et est donc écrasé à chaque nouvelle requête.

handleClientAdvancedRequest() gère les requetes avancées effectuées par l'utilisateur dans le GUI. Cette méthode reçoit une liste des requêtes avancées effectuées ensuite elle traite une à une chacune des requêtes comme suit :

1. Envoie la requête au serveur
2. Reçoit la réponse du serveur et analyse le type de requête
3. En fonction du type de requête, sélectionne un canevas d'écriture afin d'écrire ce résultat dans un fichier
4. Passe à la requête suivante

A la fin de toutes les requêtes, ouvre le fichier final généré contenant toutes les requêtes de l'utilisateur. L'utilisateur peut alors l'enregistrer sur sa machine et/ou l'imprimer.

Note : Le fichier généré porte toujours le même nom et est donc écrasé à chaque nouvelle requête.

Attention : Certains fichiers générés par le menu recherche avancée ne sont pas forcement adaptés à une impression, de par leur contenu (trop) exhaustif.

3.3.4.3 ClientSocket

Les méthodes principales sont :

connect() Etabli la connection avec le serveur

disconnect() Déconnecte le client du serveur

askForFloor() transmet les données reçue par le Controller au serveur. Les données reçue sont serialisée au format JSON avant d'être envoyé par connection TCP au serveur.

askForClassroom() idem (seul le type d'objet transmis et quelque menu traitement change)

askForAdvancedRequest() idem

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.

L'élément principal est le Stage. Il nous permet d'interagir avec la fenêtre. Il y a ensuite un objet de type Scene contenant les différents composants animant cette scène et 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 représente le modèle, le FXML représente la vue auquel est attribué un contrôleur.

Ce contrôleur étant le fichier Java gérant les interactions avec le fichier FXML.

4.1.2 Scène principale

Java FX propose différents conteneurs graphiques comme des AnchorPane, des GridPane, des BorderPane.

Tous ces conteneurs héritent de l'objet conteneur de type Pane.

La première tâche a été de choisir le conteneur principal de DARYLL et pour cela, il y avait plusieurs possibilités. 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. Après quelques essais nous avons finalement choisi l'objet de type BorderPane. Il se prêtait bien à notre application, notamment pour le redimensionnement des éléments dans la fenêtre comme les plans d'étages.

Il est séparé en différentes zones (top, bottom, center, left, right), et grâce à ces zones, les composants prennent automatiquement la bonne taille avec le redimensionnement.

La vue principale est contrôlée par le MainViewController. Elle nous permet de charger un étage directement dans l'interface graphique.

La méthode principale dans cette classe est showFloor Elle s'occupe d'obtenir les informations des salles libres, de mettre à jour le plan correspondant et enfin de le charger dans l'objet ImageView (Onglet Cheseaux ou St-Roch).

4.2 Scènes secondaires

Par scènes secondaires, on entend les autres fenêtres ou pop-up qui sont créés à partir de la scène principale (fenêtre principale) de DARYLL.

Ces scènes ont chacune leur fichier FXML avec leur propre contrôleur gérant les interactions avec l'utilisateur de ladite fenêtre. Nous en avons 3 :

AboutView.fxml : Vue du menu à propos

TimeslotView.fxml : Vue du menu Recherche avancée

RoomScheduleView.fxml : Vue du menu Horaire d'une salle

4.2.1 Gestion des évènements selon le composant ciblé

Grâce à JavaFX, nous pouvons gérer l'action des boutons des étages avec une seule fonction.

En fonction du fx :id du bouton, on récupére directement l'étage correspondant à afficher. 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.

De récupérer le bouton nous 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 et réutilisable.

4.2.2 Redimensionnement

L'interface graphique de DARYLL affiche les plans des étages de Cheseaux ou de Saint-Roch.

Nous avons jugé important pour l'utilisateur de pouvoir redimensionner la fenetre.

DARYLL offre donc la possibilité d'agrandir ou de réduire la taille de la fenêtre afin que l'affichage du plan soit en adéquation avec la résolution de l'utilisateur.

L'idée générale est 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.

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...

4.2.3 Transformation SVG

4.2.3.1 Coloration SVG

Pour effectuer une coloration des salles des étages de l'HEIG, il faut avoir des fichiers SVGs qui contiennent une zone par salle et que chaque zone soit identifiée via le nom de la salle.

Les plans des étages nous ont été fournis dans des documents PDF. Ce PDF contenait donc tous les étages de Cheseaux et de Saint-Roch. Il a fallut donc extraire de ces fichiers PDF, des fichiers utilisables au format SVG (format le plus simple à obtenir et à utiliser après-coup), d'ou l'utilisation du logiciel Inkscape.

Le problème vient de la transformation de PDF en SVG qui crée énormément de fragments de zones SVG. De ce fait, aucune salle n'est définie par une zone uniquement, mais pas plusieurs petites zones. Il a fallut donc regrouper chaque fragment de zones SVG pour en faire qu'une et que celle-ci puisse définir la salle via un identifiant.

Nous avons donc dû regrouper chaque fragment de zones SVG pour chaque étage de chaque bâtiment (ceux fournis par GAPS). Lorsque nous avons regroupé les fragments de zones nous avons en réalité ajouté une balise de type "groupe" et un identifiant.

La classe SVGToolBox va donc colorer les plans de manière automatique en parcourant tous les groupes définis du fichier SVG puis indiquer la couleur dans son attribut "style".

Cette opération est très coûteuse et prend du temps pour charger l'image dans l'interface graphique. Nous avons donc implémenté l'utilisation de threads dans la coloration afin de ne pas figer l'interface graphique et de garantir une ergonomie logique.

Il y a au final 3 nuances de vert pour indiquer si une salle est libre : pour 1 période, 2 périodes, ou 3 périodes et +.

4.2.3.2 Transcodage des plans

Pour l'affichage des images SVG, JavaFX ne fournit pas de méthode toute prête pour les afficher directement.

Nous nous sommes rendus compte qu'il n'y avait pas beaucoup de documentation concernant l'utilisation de SVG en Java. Il y a bien un objet de type WebView capable de lire

les fichiers SVG directement, mais le rendu graphique n'était pas concluant (car trop cher en ressources).

Nous avons donc du trouver un moyen de transcoder ces images pour obtenir des objets de type BufferedImage, directement compréhensible par JavaFX. Tout cela est effectué dans la class PlanLoader. Elle utilise le transcodeur de type BufferedImageTranscoder pour la transformation.

4.2.4 Composants graphiques spécifiques

Pour obtenir une date, JavaFX propose un composant tout prêt : l'objet DatePicker

On obtient donc directement la date indiquée en la sélectionnant dans un calendrier.

Quand il n'existe pas un module adapté, il a fallut créer soit-même des objets graphiques.

Ainsi dans le paquetage model du GUI, se trouve 2 objets :

Un objet TimeSpinner de type Spinner
Un objet AdvancedRequestForm, contenant tous les élements graphiques nécessaires pour demander une requête avancée à l'utilisateur.

Pour le temps, rien n'est proposé de base par JavaFX. Nous avons donc du chercher un moyen d'obtenir l'heure avec tout de même un contrôle de saisie.

La solution a donc été de recréer un élement Spinner (qui permet de faire défiler des nombres) et de lui renseigner une LocalTime. Cet objet appelé TimeSpinner, nous a permis d'obtenir un joli rendu de l'heure avec tous les contrôles de saisie nécessaires.

L'objet AdvancedRequestForm est créé dynamiquement dans la fenêtre de recherche avancée (TimeslotView), jusqu'au nombre maximum de 7. Nous avons aussi créé un objet GuiLogger, qui nous permet d'afficher des informations à l'utilisateur en fonction de ce qu'il recherche.

4.2.5 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.

4.2.6 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.

4.2.7 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.

4.2.8 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 graphique

Bien qu'ayant déjà soulevé les principaux problèmes plus tôt, nous allons reprendre ici avec plus de détails nos explications.

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 la coloration des salles en fonction de leur niveau de disponibilité.

Bien que représentant un défi de taille, nous nous sommes rapidement mis d'accord sur l'importance de la fonctionnalité de redimensionnement.

Nous avons donc exploré de nombreuses possibilités de rendu avant de trouver une version suffisamment précise, que nous proposons dans cette première version de Daryll .

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 (par exemple les webview, capalbe d'afficher des SVG mais bien trop coûteux).

5.2 Restructuration des plans

Les salles étant représentées par des triangles sur les plans (dans le SVG), il a fallu les regrouper afin d'obtenir des objets manipulables permettant l'affichage précis des salles disponibles en modifiant la couleur du groupe nouvellement créé.

Un travail de fourmi peu gratifiant, mais au résultat à la hauteur de nos attentes.

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

6.1 Plan étage B corrompu

Le plan secondaire de l'étage B ne peut pas s'afficher correctement : floor-B2.svg pose problème. On l'a donc retiré de la liste pour que le programme ne plante pas, mais on perd donc l'information sur ces salles (Toujours disponibles en advance request et horaire d'une salle). C'est un problème mineur car concernant une zone peu fréquentée de l'école par les étudiants.

Nous supposons l'origine du problème au niveau de l'encodage du PDF fourni.

6.2 Impression trop volumineuse

Pas réellement un problème mais plus une mise en garde. Certains résultat de recherche -comme par exemple demandé toute les salles libre d'un batiment pour le semestre- crée un fichier de résultat très volumineux et par conséquent très peu adapté à une impression.

Ceci est obligatoire si l'on veut garder la cohérence de l'outil (toujours faire ce que l'utilisateur demande), mais on pourrait rajouter une alerte pour éviter le problème.

6.3 Sanitarisation des entrées utilisateurs

Toutes ne sont pas vérifiées dans le détail, par exemple une date de fin qui prédate la date de début dans une recherche sera effectivement envoyée au serveur. On obtiendra une réponse vide.

On pourrait prévenir l'utilisateur de son erreur, souvent simple mégarde.

6.4 Optimisation

A notre grand regret, nous n'avons pas trouvé moyen de traiter le nombre et la taille de nos plans avec une efficacité suffisante pour faire disparaître tout délai d'attente.

Il serait intéressant de se pencher sur les possibilités d'optimisations du code.

7 Possibles améliorations

7.1 Optimisation

Utiliser un pool de threads plutôt que de les créer/détruire à chaque nécessité, moins de coût.

7.2 SVG

trouver un moyen de réduire la taille en mémoire des SVG. Piste potentielle : compression/decompression.

7.3 Saisie utilisateur

Augmenter la sanitarisation de nos entrées utilisateurs, et lui donner plus d'informations visuelles.

7.4 Mode daltonien/mode nuit

Permettre à l'utilisateur de définir un/des thème(s) graphiques et de les utiliser.

7.5 Recherche de la salle la plus proche

Notre implémentation actuelle s'avère moins robuste qu'un algorithme qui a fait ses preuves telle que Dijkstra. En changer serait idéal

7.6 React interface

Il est obligatoire pour l'instant d'appuyer sur enter ou le bouton de validation accordé pour validé une saisie. Il serait intéressant de récupèrer les valeurs dès que l'on détecte un changement.

8 Conclusion

8.1 Projet

Bien que n'ayant pas réussi à implémenter l'entièreté des fonctions imaginées, le cahier des charges a été respecté et nous avons une application robuste, simple d'utilisation et utile dans la recherche d'une salle libre à la HEIG.

Les problèmes principaux ont trouvés leur source soit dans la sérialisation des donnnées (de ICS à BDD, de BDD a POJO, de POJO à JSON et l'inverse et de JavaFX à POJO), soit dans la gestion de la communication client-serveur, soit au final dans la génération d'un GUI regonomique et réactif. Chaqun de ces problèmes s'est trouvé assigné à un membre de l'équipe, ce qui a permis leurs résolutions efficaces.

8.2 Groupe

Tous impliqués dans ces objectifs, notre équipe a su faire montre d'une réelle volonté de bien faire, ainsi que d'une remarquable résistance au stress.

La régularité de nos réunions a su garantir la communication nécessaire au progrès des différentes parties, et un système de décisions communes basé sur la discussion et le bon sens a permis l'appropriation réelle du projet par chacun de ses membres et en conséquence de quoi, son succès.

8.3 Avis des membres

8.3.1 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.

8.3.2 Aurélien Siu

J'ai eu beaucoup de plaisir à participer dans ce projet. J'avais un peu d'appréhension à l'idée de réaliser un projet de cette envergure en groupe. Finalement, que du positif, il y avait pour moi je pense une bonne entente au sein du groupe. J'ai aussi pu augmenter mes connaissances en interface graphique, ce fut très intéréressant.

8.3.3 Romain Gallay

Participer à la réalisation d'un projet de programmation impliquant une équipe de cette taille est une première dans mon parcours académique. Il est évident que gérer un travail de groupe à 6 nécessite certaines compétences différentes de ce dont nous avons l'habitude à 2 ou 3 personnes, c'est donc avec curiosité et intérêt que j'ai abordé ce projet.

La collaboration au sein de notre équipe s'est très bien passée et j'ai pu rapidement trouver un rythme de travail en phase avec les membres du groupe. J'ai eu l'opportunité d'améliorer divers compétences, aussi bien techniques que de gestions et planifications de projet. Malgré quelques passages plus difficiles, notamment de debug ou de questionnement d'implémentation, je suis pleinement satisfait de notre collaboration et du travail accompli.

8.3.4 Yohann Meyer

Sans aucun doute l'expérience la plus formatrice de mon parcours académique ici. Bien qu'impardonnable dans son exigence, cette équipe a su surmonter avec brio tout les écueils qui attendent l'inexpérimenté.

Un projet dont je suis plus que satisfait bien que peu étonné, disposant d'une task force aussi compétente et volontaire. Le seul risque se trouvait au niveau d'un potentiel manque de cohésion, je peux dire maintenant que c'était une crainte bien mal fondée.

8.3.5 Loïc Frueh

Ce projet à été très stimulant et formateur. J'ai pu faire appel à plusieurs notions théorique vues dans plusieurs cours (RES, POO, SER, ...) et je suis assez fier du résultat même si il n'est évidamment pas parfait.

La collaboration au sein du groupe c'est extrêmement bien passée, et ce même malgré le manque de sommeil à la fin du projet, lors des longues phases de debugging. Chaque personne a su trouver sa place et sa contribution au sein du groupe.

Dans l'ensemble, j'ai apprécié travaillé sur ce projet même si cela n'a pas été toujours facile.

8.3.6 Labinot Rashiti

Je pense qu'on peut retenir une bonne expérience de ce projet. Le groupe s'entendait bien et les différentes réunions de groupe chaque semaine nous ont permis de diriger un cap, d'attribuer des tâches pour chacun et enfin avancer dans le projet.

Il est vrai que le début était un peu poussif, car nous n'avons pas eu les informations nécessaires concernant les plans des bâtiments ou les horaires GAPS. Heureusement, nous avons été intelligents dans la manière de gérer ces retards et nous nous retrouvons donc avec un projet terminé.

Pour ma part, j'ai pu travailler dès le début sur l'interface graphique vu que je n'étais pas atteint par ces retards et j'ai continué sur ce thème jusqu'à la fin du projet.

Par la suite, Aurélien m'a rejoint pour l'interface graphique et notamment pour le redimensionnement des fenêtres.

Je tiens à le féliciter pour son idée sur l'utilisation des SVGs pour les plans qui était une très bonne idée ainsi que son implication durant tout le projet.

Références

[1] Docker. https://www.docker.com/.

[2] Apache. Maven. https://maven.apache.org/.

[3] Software Freedom Conservancy. Git. https://git-scm.com/.

[4] GAPS. Calendrier ics.

[5] GitHub. Github. https://github.com/.

[6] Projet Inkscape. Inkscape. https://inkscape.org/fr/.

[7] Labri.fr. Introduction à javafx. http://www.labri.fr/perso/johnen/pdf/IUT-Bordeaux/UMLCours/IntroductionJavaFX-V1.pdf.

[8] Olivier Liechti. Cours de res de l'heig-vd par olivier liechti. https://github.com/wasadigi/Teaching-HEIGVD-RES.

[9] Philippe Moser. Plan des bâtiments.

[10] Oracle. Designing gui with the librarie panel (release 2). https://docs.oracle.com/javase/8/scene-builder-2/user-guide/library-panel.htm.

[11] Oracle. Getting started with javafx. https://docs.oracle.com/javafx/2/get_started/form.htm.

[12] Oracle. Java 8. https://docs.oracle.com/javase/8/docs/.

[13] Oracle. Javafx scene builder user guide. https://docs.oracle.com/javafx/scenebuilder/1/user_guide/library-panel.htm.

[14] Oracle. Mysql 5.7. https://dev.mysql.com/doc/refman/5.7/en/.

[15] Stackoverflow. How to find an element with an id in javafx ? https://stackoverflow.com/questions/12201712/how-to-find-an-element-with-an-id-in-javafx.

[16] Stackoverflow. Javafx getting scene from a controller. https://stackoverflow.com/questions/26060859/javafx-getting-scene-from-a-controller.

9 Annexes

9.1 Discussion planification initiale/finale

Bien qu'ayant été obligé de reprendre notre planification initiale depuis le début une semaine après le rendu du cahier des charges et souffrant donc d'un retard d'une semaine, notre nouvelle planification a été suffisamment bien conçue pour nous permettre de la suivre presque à la lettre jusqu'à aujourd'hui.

Cela est dû sans aucun doute aux réunions hebdomadaires hors heures de cours auxquels notre équipe s'est astreinte.

9.2 Cahier des charges

9.3 Gantt

PRO: cahier des charges
DARYLL

Groupe 1-B

March 2018

1 Description du projet

En tant qu'étudiant suivant le cours de PRO, nous avons reçu la tâche importante de réaliser un projet complet suivant quelques instructions. Après d'intenses discussions et une réflexion propre à amener une vision claire des besoins actuels de l'étudiant, nous sommes arrivé à la conception de **DARYLL, un utilitaire permettant l'obtention des disponiblités des salles de cours de la HEIG.**
N'ayant pas l'opportunité d'étudier le marché pour déterminer où se trouve le besoin du public général, nous avons préféré les besoins d'un public connu, à savoir celui des étudiants de la HEIG et dont une des demandes proéminentes se trouve être un moyen de trouver une salle libre pour y réviser. Se basant principalement sur les données GAPS, il proposera une interface claire et rapide aux étudiants à la recherche d'une salle libre pour un ou plusieurs horaire(s) donné(s). L'objectif principal est l'obtention d'une information utile, instantanée et aussi précise que le permettront les données initiales. Techiquement, ce logiciel se basera sur une architecture client-serveur avec une conception MVC. Pour plus de détails techniques, se référer à la section **fonctionnement**.

2 Environnement de développement envisagé

1. Langage de programmation : Java
2. Base de données : MySQL
3. Librairie graphique : JavaFX
4. API tierces : GSON, JUnit
5. Framework pour gestion client serveur : Netty, Apache MINA, CoralReactor, autre ?

3 Fonctionnalités

3.1 de base

1. Affichage des salles libres sous forme de plan (voir Mockup) selon les paramètres suivants, choisis par l'utilisateur :
   (a) Position de l'utilisateur (granularité: salle)
   (b) Bâtiment : St-Roch ou Cheseaux
   (c) Numéro d'étage
   (d) Horaire, à choix : maintenant, ou sur une ou plusieurs plages horaires
   (e) Présence de prises électriques
   (f) Présence de tableaux
   (g) Présence d'ordinateurs

   (e) (f) (g) soumis à la contrainte d'obtention d'une information précise encore incertaine.
2. Affichage de divers commodités sur le plan d'étage : toilettes, Selectas et machines à café

3.2 facultatives

1. MODE SUPPLEMENTAIRE: Obtention des horaires en fonction d'une salle libre donnée (inverse fonctionnalité de base)
2. Mode daltonien
3. Mode nuit
4. Génération de fichiers .csv ou .txt contenant les salles libres
5. Possibilité d'indiquer si une salle est occupée par un utilisateur de DARYLL, soit par user input soit par détection de la borne wi-fi utilisée.

4 Fonctionnement

La solution est constituée de deux entités : le client et le serveur.

4.0.1 Serveur

Contient

la base de données `[MySQL]` contenant une table d'étage et une de salle, chacune remplie des informations utiles (toilettes h/f pour les étages, horaires de cours pour les salles, ...)

une application `[Java]` basée sur un framework [A DEFINIR: voir 2.5], responsable de la réception des demandes de la part du client, de la transformation en demande SQL adaptée selon les heuristiques et le renvoi du tout au format `[JSON]`.

l'heuristique sera codée afin de présenter à l'utilisateur la meilleure vue possible. Salles les plus proche en fonction de la position entrée (étage), exclusion des salles fermées ou privatisées, affichage des salles libres selon le temps encore disponible avant la prochaine occupation (trois mode de couleur, voir Mockup).

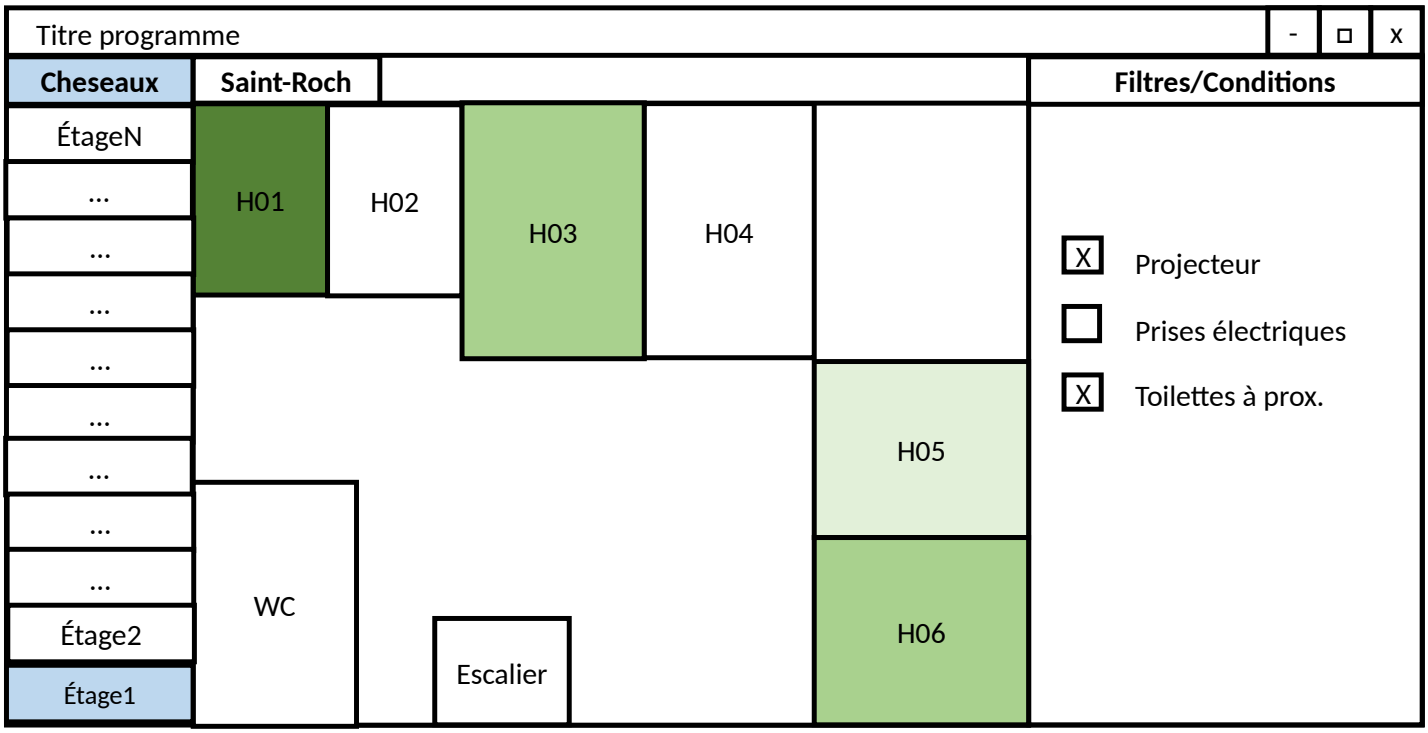
4.0.2 Client

Construit sur un motif Modèle-Vue-Contrôleur, codé intégralement en java.

Modèle Gestion de l'interaction avec le serveur, plus la logique de répresentation des objets.

Vue utilisation de `[JavaFX]` pour présentations des données du modèle. Objectifs primaires: rapidité et ergonomie. L'interface se présentera sous la forme d'un plan d'un étage (séléctionné ou le plus proche selon la demande) dont les salles seront décorées en fonction de leurs disponibilités. Voir le mockup pour plus de détails. Idealement nous nous baserons sur les plans existants.

Contrôleur Même objectif et technologies que dans le modèle.



Légendes :

= libre pour les 3 prochaines périodes

= libre pour les 2 prochaines périodes

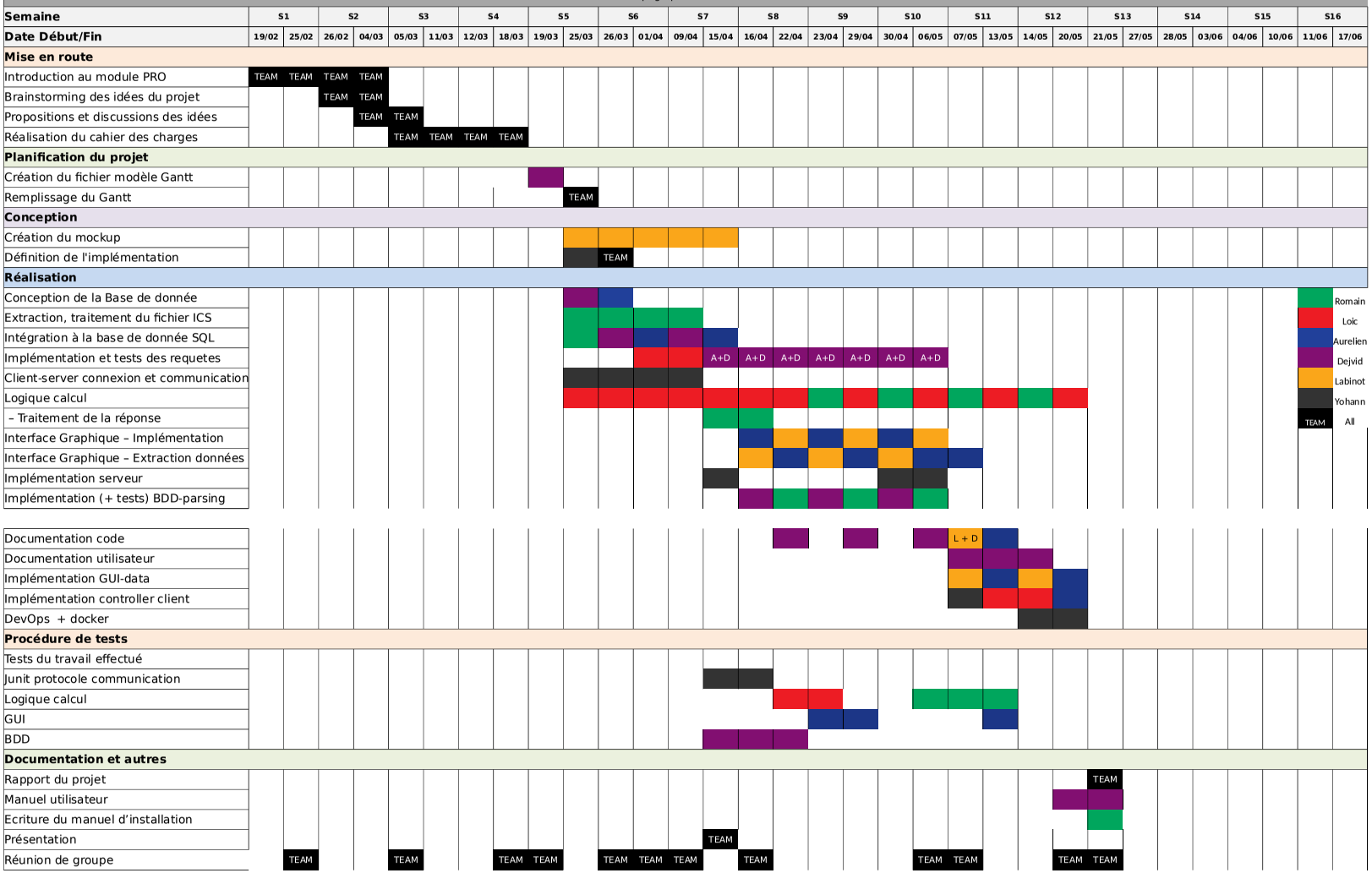
= libre pour la prochaine période

= Étage/Établissement sélectionné

PRO

Planification du projet

Découpage par demi-semaine



heig-vd
HAUTE ÉCOLE
D'INGÉNIERIE ET DE GESTION
DU CANTON DE VAUD
www.heig-vd.ch

Hes·so
Haute Ecole Spécialisée
de Suisse occidentale
Fachhochschule Westschweiz
University of Applied Sciences and Arts
Western Switzerland

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

Projet de semestre (PRO)
Journaux de travail personnels

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

Client :
René Rentsch
Référent :
René Rentsch

24 mai 2018

Journal de travail

Dejvid Muaremi

Date	Activité	Heures
22/02/18	Introduction du module. Constitution du groupe avec nomination d'un chef de groupe et de son remplaçant.	3
23/02/18	Réunion: choix de proposition projet	2
01/03/18	Feedback des propositions. Discussion pour compléter le projet choisi: DARYLL	2
03/03/18	Rédaction du modèle de Gantt sous Excel, ainsi que des répartitions des heures	2
06/03/18	Réunion: Discuter des fonctionnalités du projet	2
08/03/18	Mise à jour des répartitions des heures	1
08/03/18	Changement de modèle de Gantt	2
15/03/18	Feedback du cahier des charges et de la planification	2
17/03/18	Réunion: Refaire la planification initiale	2.5
20/03/18	Réunion: Définir les tâches à réaliser des vacances	2
24/03/18	Conception de la base de données. (Conceptuel et relationnel)	8
27/03/18	Réunion: Définir les tâches à réaliser des vacances	2
30/03/18	Implémentation de la base de données	4
03/04/18	Réunion durant les vacances	2
07/04/18	Création des classes pour la communication Java — MySQL	6
10/04/18	Réunion: avancement après vacances	2
14/04/18	Ajout de méthodes sur la base de données.	4
15/04/18	Call Java sur les méthodes de la base de données	3
18/04/18	Réunion de groupe pour préparer la présentation	2
19/04/18	Préparation de la présentation et présentation	2.5
25/04/18	Correction de bug sur les méthodes SQL et Java	2
03/05/18	Réunion de groupe	2
08/05/18	Réunion de groupe afin de définir les tâches à faire durant la période de l'ascension	2
13/05/18	Ajout de fonctionnalité à la BDD	2
15/05/18	Documentation de la BDD	4
18/05/18	Réunion du petit groupe: debugging du projet	4
19/05/18	Réunion du petit groupe: Ajout de méthodes sur la base de données.	6
20/05/18	Réunion du petit groupe: Correction de bug Java sur les calls des méthodes SQL	6
21/05/18	Réunion de groupe afin de définir les dernières tâches	2
Projet	Rédaction du rapport	10
Projet	Rédaction de la documentation utilisateur	8
24/05/18	Finalisation et rendu	2
	Total	**106**

Journal de travail

Aurélien Siu

Date	Activité	Heures
22/02/18	Introduction du module. Constitution du groupe avec nomination d'un chef de groupe et de son remplaçant.	3
23/02/18	Réunion: choix de proposition projet	2
01/03/18	Feedback des propositions. Discussion pour compléter le projet choisi: DARYLL	2
03/03/18	Rédaction du modèle de Gantt sous Excel, ainsi que des répartitions des heures	2
06/03/18	Réunion: Discuter des fonctionnalités du projet	2
08/03/18	Mise à jour du Gantt	1
08/03/18	Changement de modèle de Gantt	2
15/03/18	Feedback du cahier des charges et de la planification	2
17/03/18	Réunion: Refaire la planification initiale	2.5
18/03/18	Première version de la conception de la base de données	2
20/03/18	Réunion: Définir les tâches à réaliser des vacances	2
24/03/18	Conception de la base de données. (Conceptuel et relationnel)	8
27/03/18	Réunion: Définir les tâches à réaliser des vacances	2
30/03/18	Implémentation de la base de données	4
03/04/18	Réunion durant les vacances	2
07/04/18	Création des classes pour la communication Java — MySQL	6
10/04/18	Réunion: avancement après vacances	2
12/04/18	Début de réalisation de l'interface graphique. Verrouillage de la taille minimum de la fenêtre, ajout de package	2
15/04/18	Préparer une interface pour la présentation	
18/04/18	Réunion de groupe pour préparer la présentation	2
19/04/18	Préparation de la présentation et présentation	2.5
21/04/18	Debugging de l'interface graphique avec les tailles des fenêtres. Dispatching de tout le FXML. Création d'un fichier FXML et ainsi que d'un contrôleur par fenêtre. Puis mise à jour du rendu des fenêtres «pop-up».	2
28/04/18	Le client est devenu un projet Maven et peut utiliser un fichier .svg. L'étage K peut être chargé via un transcodeur.	8
	modification mineure de l'interface graphique	1
	Correction du contrôleur, chemin des pop-up, ajout d'une barre d'outils, ajout d'un package pour les plans	4
03/05/18	Réunion de groupe	2
07/05/18	Ajout et intégration d'une classe pour la gestion de fichiers SVG	3
08/05/18	Réunion de groupe afin de définir les tâches à faire durant la période de l'ascension	2
10/05/18	Ajout des boutons pour la sélection de l'étage, des plans de Cheseaux. Affichage des étages G et J. Réfléxion sur l'affichage des étages qui sont séparés en plusieurs images.	2
13/05/18	Gestion des plans sur des images multiples pour Cheseaux. Résolution de bug sur St-Roch. Appel explicite sur le garbage collector afin de réduire la consommation de ram	5
17/05/18	Ajout de l'option «Salle courante», gestion basique de la résolution de l'écran et de la position de la fenêtre. Ajout d'une console de log sur l'interface	2
18/05/18	Réunion groupe réduit: Ajout de l'option pour créer un jar exécutable du client + debugging	4

19/05/18	Réunion groupe réduit: Restructuration des packages du serveur et nettoyage de l'arborescence.	1
19/05/18	Implémentation de l'envoi des requêtes du client vers le serveur. Gestion d'exception, et modification des adresses IP	2
19/05/18	Première communication client-serveur	3
19/05/18	Première communication client-serveur modifiant les couleurs du plan	6
20/05/18	Réunion groupe réduit: Conversion heure <-- -> période fonctionnelle + debugging	4
21/05/18	Réunion de groupe afin de définir les dernières tâches	2
21/05/18	Requêtes client-serveur avancées fonctionnelles, ajout d'une fenêtre pour les requêtes avancées et meilleure gestion de l'heure	6
22/05/18	Gestion d'exception, mise à jour des plans, mise à jour de l'interface graphique	6
24/05/18	Finalisation et rendu	2
	Total	**120**

Journal de travail

Romain Gallay

Date	Activité	Heures
22/02/18	Introduction du module. Constitution du groupe avec nomination d'un chef de groupe et de son remplaçant.	3
23/02/18	Réunion choix de proposition projet	2
01/03/18	Feedback des propositions. Discussion pour compléter le projet choisi : DARYLL	2
06/03/18	Réunion : Discuter des fonctionnalités du projet	2
08/03/18	Rédaction des technologies et fonctionnalités	1
17/03/18	Réunion : Refaire la planification initiale	2.5
20/03/18	Réunion : Définition des tâches à réaliser pendant le week-end prolongé	2
27/03/18	Réunion : Définition des tâches à réaliser pendant vacances	2
01/04/18	Recherche d'informations sur la construction et le parsing d'un fichier ICS	2
03/04/18	Réunion durant les vacances	2
04/04/18	Début d'implémentation du parseur ICS	2
05/04/18	Implémentation du parseur ICS	3.5
10/04/18	Réunion avancement après vacances	2
12/04/18	Test du parser ICS et debugging	3
18/04/18	Réunion de groupe pour préparer la présentation	2
19/04/18	Préparation de la présentation et présentation	2.5
21/04/18	Définition et schéma du protocol de traitement de requêtes de recherche	3
22/04/18	Implémentation de gestion de requêtes d'étage côté serveur	8
28/04/18	Redéfinition du mapping entre période et heures	1
03/05/18	Réunion de groupe	2
06/05/18	Implémentation de gestion des requêtes d'horaire d'une classe côté server	5
08/05/18	Réunion de groupe afin de définir les tâches à faire durant la période de l'ascension	2
13/05/18	Implémentation de gestion des requêtes avancées d'une classe côté server	6
15/05/18	Nettoyage du code côté serveur	3
16/05/18	Ajout de la javadoc côté serveur	4
18/05/18	Réunion du petit groupe : debugging	4
19/05/18	Reunion du petit groupe : Début d'implémentation de salle la plus proche à l'aide de l'algorithm de Dijkstra	6
20/05/18	Réunion du petit groupe : Implémentation de salle la plus proche avec méthode simplifiée	6
21/05/18	Réunion de groupe afin de définir les dernières tâches	2
22/05/18	Tests et refactorisation de code côté serveur	5.5
24/05/18	Finalisation et rendu	2

	Total	**95**

Journal de travail

Yohann Meyer

Date	Activité	Heures
22/02/18	Introduction du module. Constitution du groupe avec nomination d'un chef de groupe et de son remplaçant.	3
23/02/18	Réunion choix de proposition projet	2
01/03/18	Feedback des propositions. Discussion pour compléter le projet choisi : DARYLL	2
03.03.2018 -	Rédaction du modèle de Gant sous Excel, ainsi que des répartitions des heures	2
06/03/18	Réunion : Discuter des fonctionnalités du projet	2
15/03/18	Feedback du cahier des charges et de la planification	2
05/04/18	divers organisation de projet (git, ..)	2
17/03/18	Réunion : Refaire la planification initiale	2.5
20/03/18	Réunion : Définir les tâches à réaliser des vacances	2
22/03/18	Cleanup, structuration des commentaires	4
27/03/18	Réunion : Définir les tâches à réaliser des vacances	2
31/03/18	Implémentation du protocole client-serveur	4
03/04/18	Réunion durant les vacances	2
05/04/18	divers organisation de projet	2
10/04/18	Réunion avancement après vacances	2
	Implémentation de la connexion client-server	4
18/04/18	Réunion de groupe pour préparer la présentation	2
19/04/18	Préparation de la présentation et présentation	4.5
03/05/18	Réunion de groupe	2
05/04/18	divers organisation de projet	2
04/06/18	cleanup et optimisation code	2
08/05/18	Réunion de groupe afin de définir les tâches à faire durant la période de l'ascension	2
	Separation projet client/server	2
21/05/18	Réunion de groupe afin de définir les dernières tâches	2
21/05/18	Mise au point de la procédure d'installation – docker	8
22/05/18	Automatisation du build docker	4
	Rédaction du rapport	6
	Rédaction du manuel utilisateur	1
24/05/18	Finalisation et rendu	2
	Total	**79**

Journal de travail

Frueh Loïc

Date	Activité	Heures
22/02/18	Introduction du module. Constitution du groupe avec nomination d'un chef de groupe et de son remplaçant.	3
23/02/18	Réunion choix de proposition projet	2
01/03/18	Feedback des propositions. Discussion pour compléter le projet choisi : DARYLL	2
05/03/18	Recherche d'un moyen d'obtenir les plans des batiments. Envoi d'un mail au sercretariat.	0,5
06/03/18	Réunion : Discuter des fonctionnalités du projet	2
07/03/18	Rédaction du paragraphe fonctionnement	1,5
08/03/18	Relecture cahier des charge, correction et rendu	1
15/03/18	Feedback du cahier des charges et de la planification	2
17/03/18	Réunion : Refaire la planification initiale	2.5
20/03/18	Réunion : Définir les tâches à réaliser durant le week-end prolongé	2
23/03/18	Reflexion et conception de l'architecture client	3
27/03/18	Réunion : Définir les tâches à réaliser des vacances	2
31/03/18	Tentatives de connection TCP entre client et serveur, premier essai de serialisations (Recherche au sujet de la classe JsonMapper), experimentations.	4
03/04/18	Réunion durant les vacances	2
06/04/18	Tentatives de transfert d'objet entre client et server via une connection TCP, application des notions vu en cours de RES	5
10/04/18	Réunion avancement après vacances	2
18/04/18	Réunion de groupe pour préparer la présentation	2
18/04/18	Préparation personnelle de la présentation	2
19/04/18	Préparation de la présentation et présentation	2.5
21/04/18	Création de la classe Controller et début de son implémentation avec la méthod handleClientRequest()	5
24/04/18	Création de la classe ClassroomByFloor implémentant des listes de classe et une map permettant de trouver les classes relatives à un étage	2
26/04/18	Correction de la classe ClientSocket afin d'envoyer les bonnes données attendue au server et implémentation de la reception	2
03/05/18	Réunion de groupe	2
06/05/18	Correction de la classe Controller, gestion de la demande de l'horaire disponible d'une classe particulière, séparation de la method handleClientRequest() en 2 méthods spécifiques : handleClientFloorRequest pour les requetes d'étage et handleClientClassroomRequest() pour les requetes de salle spécifique	6
08/05/18	Réunion de groupe afin de définir les tâches à faire durant la période de l'ascension	2
11/05/18	Conception et création des Objets requetes avancées	3
12/05/18	Création dans le controller de la méthode handleClientAdvancedRequest() afin de récupérer les requetes avancées du GUI, de les envoyés au serveur et d'en receptionner la réponse.	7
13/05/18	Correction du handleClientAdvancedRequest() pour correctement envoyé les infos au serveur	3
18/05/18	Réunion du goupe réduit : debugging du projet	4
19/05/18	Réunion du goupe réduit : debugging du projet	6

20/05/18	Réunion du goupe réduit : debugging du projet	6
21/05/18	Réunion de groupe afin de définir les dernières tâches	2
22/05/18	Refactoring du code et tests divers	4
23/05/18	Relecture du rapport et ajout de contenu concernant l'architecture client	2
24/05/18	Finalisation et rendu	2
	Total	**99**

Journal de travail

Labinot Rashiti

Date	Activité	Heures
22/02/18	Introduction du module. Brainstorming des idées de projets.	3
23/02/18	Rédaction des propositions de projets.	2
01/03/18	Feedback des propositions. Discussion pour compléter le projet choisi.	2
03/03/18	Réalisation du modèle de Gant sous Excel.	2
04/03/18	Réalisation du mockup du projet selon les discussions du groupe	2
06/03/18	Réunion de groupe pour discuter des fonctionnalités du projet	2
08/03/18	Changement de modèle de Gant et du mockup selon les nouvelles indications	2
15/03/18	Feedback du cahier des charges et de la planification	2
17/03/18	Refaire la planification initiale suite aux indications	2.5
20/03/18	Réunion pour définir les tâches à réaliser durant les vacances	2
22/03/18	Recherches et lectures de la documentation des outils à utiliser (JavaFX, FXML, Scenebuilder)	1.5
31/03/18	Installation des modules nécessaires pour faire tourner JavaFX, FXML et Scenebuilder)	4
01/04/18	Création d'une première interface graphique en essayant de respectant le mockup le plus possible	3.5
02/04/18	Création des fonctions pour chaque boutons/options/menu de l'interface graphique en JavaFX	2.5
02/04/18	Lier les fonctions à chaque boutons/options/menu via Scenebuilder puis ajouter des modes supplémentaires dans le menu	3
03/04/18	Réunion puis implémentation d'une fonction "clean" afin de voir si la GUI gère bien les évènements via Scenebuilder	2
03/04/18	Regrouper les différentes fonctions sous une seule fonction pour les étages. Trouver un moyen de retrouver le bouton ciblé directement dans la fonction	3
04/04/18	Ajout des composants graphiques supplémentaires suite à la discussion de groupe (filtres de recherches en bas et partie St-Roch)	2.5
11/04/18	Ajout des fenêtes "pop-up" de chaque mode dans le contrôleur MainViewContrôleur	4
18/04/18	Réunion de groupe pour préparer la présentation	2
19/04/18	Préparation de la présentation et présentation	0.5
21/04/18	Debugging du la GUI avec les tailles des fenêtres. Dispatching de tous le FXML. Créer un FXML et contrôleur par fenêtre. Puis mise à jour du rendu des fenêtres "pop-up"	4
03/05/18	Réunion de groupe et debugging des fonctions des boutons d'étage	2
06/05/18	Création des fichiers SVG pour chaque étage de St-Roch à partir des PDFs fournis. Définition des zones pour chaque salle de chaque étage.	3.5

06/05/18	Rédaction de la documentation pour l'interface graphique	2.5
08/05/18	Réunion de groupe afin de définir les tâches à faire durant la période de l'ascension	2
10/05/18	Mise à jour de tous les plans SVGs pour donner un format plus adapté à l'imageView.	4.5
10/05/18	Ajout des boutons de déplacements des plans, restructuration du code avec commentaires	3.5
12/05/18	Rédaction de la documentation pour l'interface graphique	4
16/05/18	Modification des pop-ups notamment pour les créneaux horaires. Changement des fx-id de chaque composants dans le FXML afin d'être plus cohérent avec les noms	2
16/05/18	Debugging du la GUI et mise à jour du rendu de la fenêtre principale et St-Roch	2
19/05/18	Recherches d'imprimeries pour faire les reliures IBICO	2.5
20/05/18	Compléter et structurer la documentation.	3
21/05/18	Debug des fichier SVGs, redéfinition complète des labels, des zones des salles pour chaque salle de chaque étage de chaque batîment, ajout du lien pour l'aide utilisateur dans le menu	8
22/05/18	Debug du GUI avec Aurelien, modification des SVGs pour Saint-Roch de certaines salles puis nouveau format pour l'étage U et R	3.25
	Total	**96.75**