Théophile PUMAIN

IUT de Clermont-Ferrand
Département Informatique
Année 2015-2016

Rapport de stage

Stage réalisé du 4 avril au 10 juin 2016
au Centre Jean Perrin
58 Rue Montalembert, 63000 Clermont-Ferrand

sous la direction de Mme Manon SOURDEIX

Conception et développement d'une interface web PHP/MySQL
dédiée à la gestion des runs de séquençage haut débit

J'autorise la diffusion de mon rapport sur l'intranet de l'IUT.

Je tiens à remercier mon maître de stage, Mme Manon SOURDEIX, pour m'avoir accompagné pendant ces dix semaines et pour m'avoir accordé sa confiance pour la réalisation de ce projet.

Je souhaite ensuite remercier Pr. Yves-Jean BIGNON, directeur du Laboratoire de Biologie Médicale du Centre Jean Perrin, pour avoir autorisé mon accueil au sein de son laboratoire.

Je remercie également Yannick BIDET, maître de conférence, Flora PONELLE, bioinformaticienne, Laurence LAFARGE, responsable qualité et Maud PRIVAT, cadre technique, pour leur accueil chaleureux au sein du service.

Enfin, je souhaite remercier mon professeur référent, Mme Sylvie GUILLAUME pour m'avoir suivi et conseillé durant le stage.

Sommaire

Introduction

Du 4 avril au 10 juin 2016, j'ai réalisé mon stage de deuxième année de DUT Informatique au Centre Jean Perrin à Clermont-Ferrand. Cet établissement, situé sur le plateau Saint Jacques à proximité du CHU Gabriel Montpied, est spécialisé dans la lutte contre le cancer. J'ai été rattaché à l'unité fonctionnelle d'Oncogénétique* du Laboratoire de Biologie Moléculaire (LBM) du Centre Jean Perrin regroupant médecins, biologistes, chercheurs et bioinformaticiens. Cette unité est chargée de deux missions :

- établir des diagnostics pour des patients afin d'évaluer leurs prédispositions génétiques à certains cancers,
- assurer une activité de recherche et de développement de technologies innovantes.

Pour cela, le Centre Jean Perrin s'est récemment équipé d'un séquenceur* dit de *nouvelle génération* qui est un instrument capable de déterminer la séquence nucléotidique* d'un ou plusieurs gènes d'intérêt chez plusieurs patients à la fois. L'objectif est d'identifier les mutations délétères* portées par les patients afin de déterminer leur prédisposition à un cancer donné. Cette étape de séquençage réalisée par le séquenceur est appelée un run*. Les données brutes issues des runs sont inexploitables en l'état. Elles doivent donc être analysées par des programmes informatiques afin d'obtenir des résultats interprétables par les biologistes qui ont pour mission d'établir un diagnostic pour le patient.

Sous la direction de Mme Manon SOURDEIX, bioinformaticienne*, j'ai eu pour objectif de concevoir et de développer une interface web accessible aux biologistes et aux techniciens de laboratoire. A travers cette interface, les utilisateurs doivent pouvoir :

- consulter l'avancement des runs en cours de séquençage et obtenir des informations sur les runs terminés,
- exécuter les programmes d'analyses en spécifiant de manière intuitive différents paramètres,
- visualiser les résultats des analyses de manière ergonomique.

Une gestion des droits d'accès était également demandée afin de restreindre chaque utilisateur aux seules fonctionnalités autorisées par son service et sa fonction.

A la fin du stage, il est prévu de déployer cette interface à une plus grande échelle, sur la "Plateforme de génétique médicale" du site de Clermont-Ferrand. Cette structure est financée et dirigée à la fois par le CHU et le Centre Jean Perrin et dispose de deux autres séquenceurs nouvelle génération. L'interface doit donc pouvoir être facilement adaptable pour gérer ces autres séquenceurs.

Tout d'abord, je commencerai par présenter l'établissement et son organisation, puis je détaillerai les objectifs du stage. Suite à cela, je présenterai une partie Analyse dans laquelle j'exposerai l'étude et la conception préalable à la réalisation pour ensuite détailler le développement de l'application en lui-même. Enfin, je dresserai un bilan technique et personnel de ce stage.

Présentation de l'entreprise

I. Présentation générale

Le Centre Jean Perrin est l'un des 20 Centres français de Lutte Contre le Cancer (CLCC). C'est un Établissement de Soins Privés d'Intérêt Collectif (ESPIC), reconnu d'utilité publique et financé par l'État. Les CLCC assurent des missions de soins, de recherche et d'enseignement, avec la volonté permanente d'accroître la qualité et l'accessibilité aux soins.

Le Centre Jean Perrin est situé sur le plateau Saint Jacques à Clermont-Ferrand et est à proximité du CHU Gabriel Montpied. Il a été fondé en 1973 et doit son nom à Jean Baptiste Perrin (1870-1942), physicien, chimiste et homme politique français notamment reconnu pour ses travaux en physique corpusculaire et récompensé du prix Nobel de physique en 1926. En 2013, l'établissement prenait en charge 24 500 patients et en 2014, il était composé de plus de 630 médecins, soignants, chercheurs, techniciens et administratifs[1].

Le centre possèdent un Laboratoire de Biologie Médicale (LBM). C'est une structure où sont prélévés et analysés des échantillons d'ADN issus de patients sous la responsabilité de biologistes médicaux. Les biologistes médicaux interprètent ensuite les résultats dans le but de participer au diagnostic et au suivi de certaines maladies. Le LBM du Centre Jean Perrin est dirigé par le Pr. Yves-Jean Bignon et est spécialisé dans l'établissement de diagnostics chez des patients atteints de cancer ou susceptibles d'avoir des prédispositions génétiques à l'apparition d'un cancer. Le laboratoire assure également une activité de recherche et de développement de technologies innovantes pouvant par la suite être utilisées en diagnostic.

Le LBM est composé de deux unités fonctionnelles :

- l'unité d'Anatomo-pathologie, qui étudie la partie somatique du cancer, c'est-à-dire la tumeur en elle-même. Leur analyse permet notamment d'orienter les soins vers le traitement adapté en fonction du statut génétique du patient.
- l'unité d'Oncogénétique*, à laquelle j'ai été rattaché, est spécialisée dans l'étude de la partie constitutionnelle du cancer, à savoir l'impact des mutations génétiques et la prédisposition héréditaire des individus à développer un cancer.

II. Accréditation et démarche qualité[2]

Le LBM a récemment été accrédité par le COFRAC (Comité français d'accréditation) à la norme ISO 15189. Cette accréditation est une reconnaissance des compétences techniques du laboratoire. Pour répondre à cette exigence, le LBM s'est inscrit dans une démarche qualité très précise qui s'applique à

1. Source : http://www.unicancer.fr/centre/centre-jean-perrin-clermont-ferrand
2. Les informations détaillées dans cette partie sont tirées du manuel qualité du LBM, disponible à l'adresse suivante : http://www.cjp.fr/images/fr/pdf/Laboratoires/manuel-qualite-LBM.pdf

tous les niveaux de la structure. En premier lieu, l'exigence de qualité définit l'organisation du laboratoire. Ainsi celui-ci est régi par un *management par processus*. C'est un type d'organisation qui décompose les différents stades d'une opération en processus afin de placer sa réalisation au coeur de la structure. Ici l'opération est l'établissement d'un diagnostic pour un patient ou un prescripteur. La gestion du LBM par les processus permet l'organisation suivante :

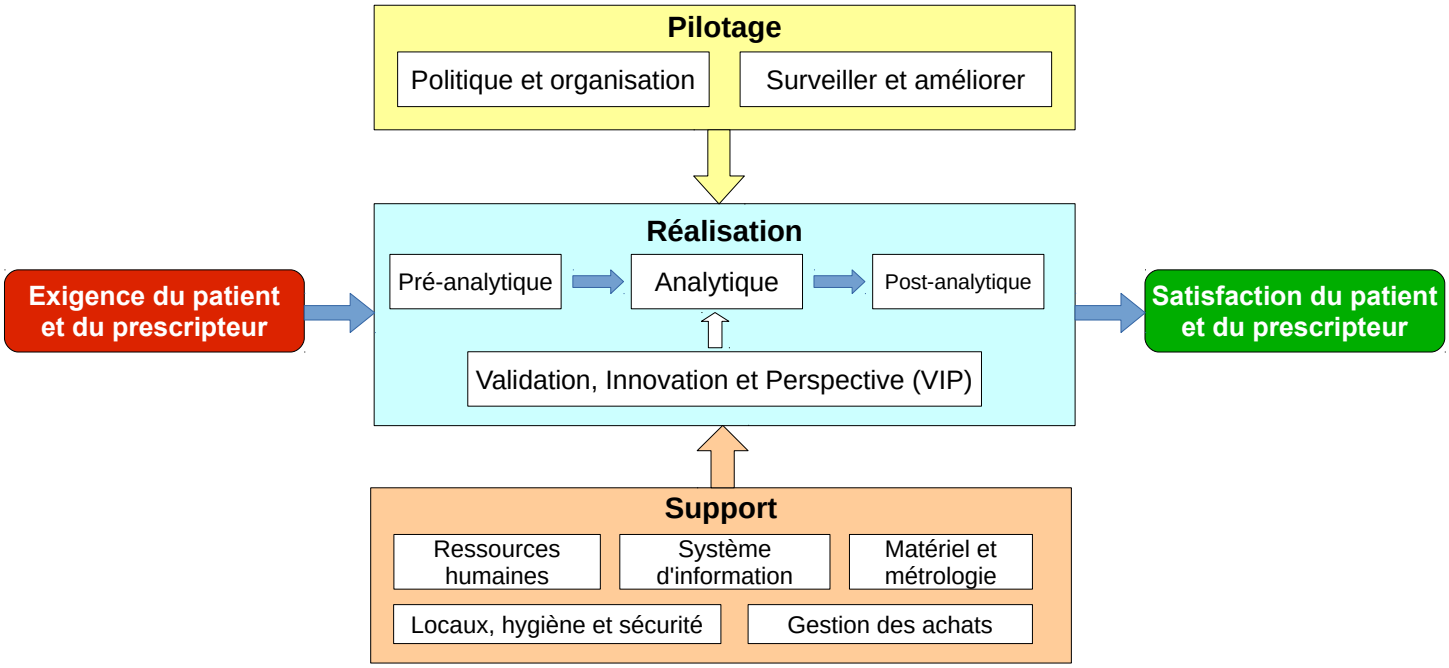


Figure 1 : Organigramme de l'organisation du LBM en processus

Il existe trois types de processus :

- les processus de pilotage qui définissent la politique, l'organisation ainsi que la surveillance et l'amélioration,
- les processus de réalisation qui regroupent les prestations du laboratoire et qui constituent le coeur de l'activité de la structure,
- les processus supports qui participent au bon fonctionnement des processus de réalisation en apportant l'ensemble des ressources nécessaires.

Tous les laboratoires accrédités à la norme ISO 15189 ont une organisation similaire car l'accréditation impose ce type de management. Toutefois, le LBM, de part son statut de laboratoire de recherche, possède une spécificité : les processus de réalisation sont soutenus par un processus nommé VIP (Validation, Innovation et Perspectives). Ce processus assure une veille technologique ainsi que la mise en place et la validation de nouvelles technologies ou examens qui pourront servir par la suite au processus analytique. Ce dernier est, quant à lui, chargé de réaliser directement l'analyse d'un prélèvement. Je me situe dans le processus VIP dans la mesure où l'interface que je développe fait partie d'une innovation technologique utile au personnel du processus analytique.

L'exigence de qualité s'applique également à d'autres niveaux :

- Au niveau matériel, le laboratoire doit s'assurer de manière permanente que l'ensemble du matériel est minutieusement sélectionné lors de l'achat et par la suite correctement entretenu et utilisé afin d'assurer performance et fiabilité. L'utilisation du matériel fait notamment l'objet de protocoles et de normes de sécurité qui doivent être respectés de manière stricte.

- Au niveau du personnel, le laboratoire doit garantir que chaque membre du personnel est formé et qu'il respecte ces mêmes protocoles. L'accent est également mis sur la communication interne. Dans cet objectif, une réunion qualité hebdomadaire rassemble l'ensemble du service. C'est notamment l'occasion pour le personnel d'exposer les difficultés rencontrées pendant la semaine et de trouver des solutions. Ces informations sont codifiées en fiches d'anomalie (FA) et l'ensemble des échanges et des propositions font l'objet de comptes-rendus, le tout dans le but d'inscrire le laboratoire dans une démarche constante d'amélioration.

- Au niveau des données, la démarche qualité assure également la fiabilité des informations concernant les patients ainsi que le respect de leur vie privée. Le traitement et la transmission rapide et fiable des diagnostics des patients doivent être assurés tout en limitant au maximum l'accès par le personnel aux données nominatives afin de garantir leur confidentialité. J'ai dû ainsi signer une charte de confidentialité, bien que je n'aie pas eu accès à des données nominatives durant mon stage. En effet, l'accès aux données est limité aux personnes autorisées à l'aide d'un identifiant et d'un mot de passe individuels. Des profils utilisateurs sont définis selon les fonctions de chacunet l'ensemble du système d'information est géré par le service informatique du Centre Jean Perrin. Les données des patients et les documents du personnel sont stockés sur des serveurs utilisant la technologie RAID*. Il s'agit d'une technique de répartition des données sur plusieurs disques durs sans que cela soit visible par l'utilisateur afin de réduire les risques de pertes de données en cas de défaillance d'un disque.

III. Matériel de séquençage

Pour analyser des prélèvements d'ADN et ainsi établir des diagnostics pour les patients, le LBM dispose d'un séquenceur. C'est un instrument capable de déterminer la séquence nucléotidique* d'un ou plusieurs gènes d'intérêt à partir d'un prélèvement ADN. Ce séquenceur (cf. Figure 2) est dit de "nouvelle génération" car il utilise des techniques de séquençage haut-débit (ou NGS pour *Next-Generation Sequencing*). Ces techniques se caractérisent par l'utilisation d'approches massivement parallèles, permettant de séquencer des centaines de milliers de fragments simultanément. Le LBM du Centre Jean Perrin fait parti des 3 laboratoires français à être accrédités pour ce type de séquençage.

Figure 2: Le séquenceur Illumina MiSeq du Centre Jean Perrin

Deux autres séquenceurs de ce type sont disponibles et situés sur la *Plateforme de Génétique Médicale* commune au Centre Jean Perrin et au CHU. Les deux unités du LBM que sont l'unité d'Oncogénétique et l'unité d'Anatomo-Pathologie font partie de cette plateforme ainsi que certains services du CHU, à savoir :

- Cytogénétique
- Anatomo-Pathologie d'Estaing
- Hématologie
- Bactériologie
- Biologie Moléculaire

Présentation synthétique du stage

I. Existant et origine du projet

Afin d'établir un diagnostic à partir d'un prélèvement chez un patient, les étapes suivantes sont nécessaires :

- extraction et préparation de l'ADN,
- séquençage de l'ADN par le séquenceur,
- obtention des données brutes du run de séquençage,
- analyse des données par des programmes informatiques,
- interprétation des résultats de l'analyse aboutissant à l'établissement d'un diagnostic.

Pour réaliser ces étapes, des moyens humains et informatiques sont mis en oeuvre. En premier lieu, un technicien de laboratoire va préparer les prélèvements puis lancer le séquençage. Au cours du run* (qui peut durer de plusieurs heures à plusieurs jours), le séquenceur déverse automatiquement des données brutes sur un serveur prévu à cet effet. A chaque étape, le séquenceur crée également différents fichiers à travers une arborescence standardisée. Au terme du séquençage, un logiciel nommé SeqNext accède aux données brutes et réalise une analyse bioinformatique. Les résultats de cette analyse sont ensuite affichées dans l'interface de SeqNext, puis recopiés manuellement par un technicien dans un tableur Excel. Ces informations peuvent alors être interprétées par un biologiste et permettent l'établissement d'un diagnostic qui sera ensuite délivré au patient.

Ce fonctionnement pose plusieurs problèmes :

- Pour consulter l'état d'avancement d'un run, le personnel doit accéder au serveur de données brutes. D'une part, cette consultation n'est pas intuitive puisqu'elle implique de repérer manuellement certains fichiers dans l'arborescence standardisée créée par le séquenceur (cf. Figure 1). D'autre part, chaque membre du personnel a alors accès aux runs des autres services qui concernent d'autres patients. Même si les données ne sont pas nominatives, cela entre en contradiction avec l'obligation d'un certain niveau de confidentialité quant aux données personnelles des patients[3].
- Le logiciel SeqNext ne permet pas d'exporter les résultats des analyses qu'il effectue. Ceux-ci ne sont accessibles que dans l'interface du logiciel et doivent être recopiés manuellement par un technicien de laboratoire. Cette étape engendre un risque d'erreur humaine important. Plusieurs relectures sont donc effectuées mais elles nécessitent du temps et de la disponibilité.

Pour résoudre ces problèmes, la mise en place d'une interface web a été envisagée afin de gérer tout le processus analytique. Ce système offrirait notamment la possibilité au personnel de visualiser de manière intuitive l'état d'avancement des runs et des analyses, mais également de limiter l'accès aux informations concernant les patients.

3. La CNIL considère un échantillon de la séquence d'ADN d'un individu comme données à caractère personnel. Ainsi le stockage de ces séquences dans une base de données reste une question sensible.

Pour pallier les inconvénients du logiciel SeqNext, les bioinformaticiens du service Oncogénétique développent en parallèle leurs propres scripts* d'analyse. Tout d'abord, les scripts permettent de réaliser des analyses supplémentaires non disponibles dans SeqNext. Ensuite, ils offrent un contrôle total sur le format de sortie des résultats permettant ainsi d'éviter les erreurs de recopie. Enfin, ils pourront à terme faire économiser au service le coût d'une licence annuelle, SeqNext étant un logiciel propriétaire payant.

Toutefois, ces scripts ne peuvent pour l'instant être lancés qu'en ligne de commande (cf. Figure 3). Ils ne sont donc pas adaptés à une utilisation par les techniciens de laboratoire ou les biologistes. L'application web pourrait ainsi remédier à cela en permettant aux biologistes d'exécuter les scripts d'analyse à travers une interface intuitive. La visualisation et l'exportation des résultats d'analyse pourraient également être facilitées par l'intégration des scripts dans l'interface.

```
[lom@l031el ~]$ perl script_analyse.pl -id 2107078355 -analysis Exome
-input /var/www/html/tmp/input/inputFile0038.txt -parallel 1 -cpu 1 -runFolder
/mnt/Donnees_Brutes/2016/160224_M70223_0013_000000000-G06HV
_multiplicom_run2_BB -bed /mnt/Analyses_Secondaires/Fichiers_input/Bed
/MedExome_hg19_capture_targets.bed
```

Figure 3 : Exemple de ligne de commande permettant d'exécuter un script d'analyse

II. Objectifs du stage

Les objectifs de mon stage sont donc la réalisation et le déploiement de cette interface qui doit ainsi servir d'intermédiaire entre l'utilisateur et les données tout en satisfaisant deux conditions :

- permettre à l'utilisateur de visualiser et de gérer de manière intuitive les données concernant les runs et les analyses,
- restreindre l'accès par l'utilisateur aux seules informations qui lui sont autorisées.

Pour permettre aux membres du personnel d'accéder à l'interface depuis leur poste, un serveur web doit être mis en place et s'intégrer à l'intranet du Centre Jean Perrin. Un modèle métier décrivant l'organisation des différentes entités doit être conçu et implémenté sous la forme d'une base de données.

Chaque run doit être affilié à un service et ses informations ne doivent être accessibles qu'au personnel de ce service. De plus, l'équipe bioinformatique doit également pouvoir gérer l'interface et accéder à des informations supplémentaires non accessibles aux autres utilisateurs. Ainsi une gestion sécurisée des utilisateurs doit être implémentée.

L'utilisateur doit pouvoir lancer des scripts d'analyse. Pour cela, il doit pouvoir spécifier les paramètres de l'analyse à travers une interface intuitive. Le programme doit alors pouvoir récupérer les paramètres spécifiés par l'utilisateur, générer la ligne de commande et exécuter le script en tâche de fond.

Toutes les informations accessibles depuis l'interface proviendront d'une base de données. La conception et la réalisation de cette base de données sont également une composante de mon stage. De plus, cette base doit être mise à jour en temps réel. Un module de l'interface doit donc pouvoir parcourir le système de fichiers des serveurs contenant les données brutes de séquençage et mettre à jour la base de données régulièrement, sans pour autant consommer trop de ressources.

III. Environnement matériel et logiciel

Pour réaliser cette interface, un ordinateur sous CentOS 6.7 est mis à ma disposition. CentOS est une distribution Linux éditée par la société Red Hat et principalement destinée aux serveurs.

L'interface sera accessible via l'intranet du Centre Jean Perrin grâce à un serveur web sous Apache. Apache est le logiciel le plus utilisé sur Internet pour gérer un serveur HTTP. L'ordinateur sous CentOS disposant de ressources limitées, il ne servira qu'à développer l'interface et ne sera pas utilisé à terme pour faire tourner le serveur Apache. Le déploiement final se fera à la fin du stage sur d'autres machines plus puissantes capables de faire fonctionner à la fois le serveur web et les scripts d'analyse en parallèle.

L'interface en elle-même est réalisée dans le langage PHP côté serveur et en HTML, CSS et Javascript côté client. L'application sera architecturée suivant le modèle MVC*. L'environnement de développement choisi est Sublime Text et l'interface est développée en utilisant le navigateur Mozilla Firefox. Toutefois, la majorité du personnel amenée à utiliser cette interface possède uniquement le navigateur par défaut de Windows : Internet Explorer. La compatibilité de l'interface avec ce navigateur doit donc être assurée.

Une base de données MySQL doit également être mise en place et pourra être gérée via PHPMyAdmin, un outil permettant, dans un navigateur, de gérer des bases de données MySQL via une interface graphique.

L'ordinateur est connecté à l'intranet du Centre Jean Perrin. J'ai ainsi accès à deux serveurs du réseau :

- un premier serveur nommé "Donnees_Brutes" contient les données brutes des runs générées par le séquenceur, accessible en lecture seulement,
- un deuxième serveur nommé "Analyses_Secondaires" contient les fichiers liés au lancement des analyses ainsi que leurs résultats.

Objectifs et organisation du stage

I. Indications

Pour débuter mon stage, mon maître de stage m'a donné des indications concernant la forme générale de l'application à réaliser. Ces indications décrivaient quatre fonctionnalités principales. Initialement, chacune d'entre elles devait être accessible via une page web. Ainsi l'application devait être composée des quatre pages décrites sur la figure 4 :

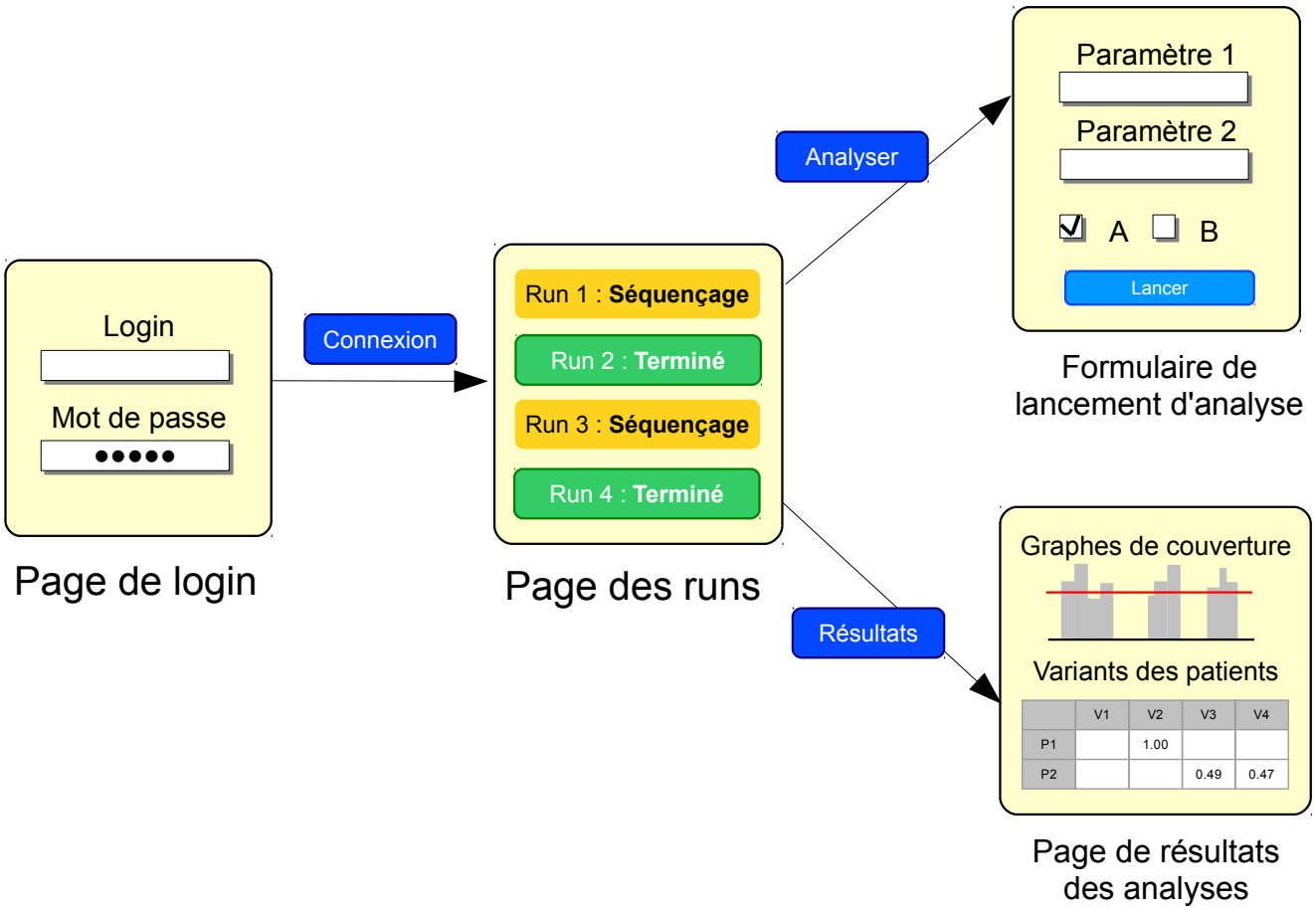


Figure 4 : Schéma de l'idée initiale de l'application

• Une page de login où l'utilisateur doit saisir un couple login/mot de passe pour accéder au reste de l'application.
• La page principale ou page des runs qui affiche la liste des runs*, avec pour chaque run son état d'avancement. A partir de cette page et pour chacun des runs, deux nouvelles pages sont accessibles :
  – si le run a terminé son étape de séquençage, un bouton permet d'accéder à une page contenant

un **formulaire de lancement d'analyse**[4]. A travers ce formulaire, l'utilisateur peut renseigner les paramètres du script d'analyse et le lancer en tâche de fond.

– si l'analyse d'un run est terminée, un bouton permet d'accéder à la **page de résultats des analyses**. L'utilisateur va ainsi pouvoir interpréter les résultats du séquençage en visualisant les informations suivantes :
  - les graphes de couverture*, permettant de déterminer la fiabilité des données issues du séquençage.
  - les variants* portés par les patients, permettant de connaître chez un patient donné, les positions génomiques qui diffèrent par rapport au génome de référence et qui peuvent engendrer une susceptibilité génétique à la maladie étudiée.

Par la suite, il s'est avéré que l'application devait contenir plus de quatre pages pour rendre ces différentes fonctionnalités facilement accessibles. Toutefois, ces indications servaient davantage de lignes directrices que de véritables contraintes précises. Ainsi j'ai bénéficié d'une grande liberté quant à la conception de cette application. Les fonctionnalités requises ainsi que l'utilisation des technologies PHP et MySQL étaient imposées par le sujet de stage mais j'ai pu choisir librement comment concevoir l'application et la base de données, dans la mesure où mes choix conduisaient à une architecture logique et facilement extensible.

II. Organisation prévisionnelle

A partir de ces indications, ma/mon maître de stage et moi-même avons défini un planning détaillant les différentes tâches à réaliser durant ce stage. Celui-ci est présenté ci-dessous sous la forme d'un diagramme de Gantt.

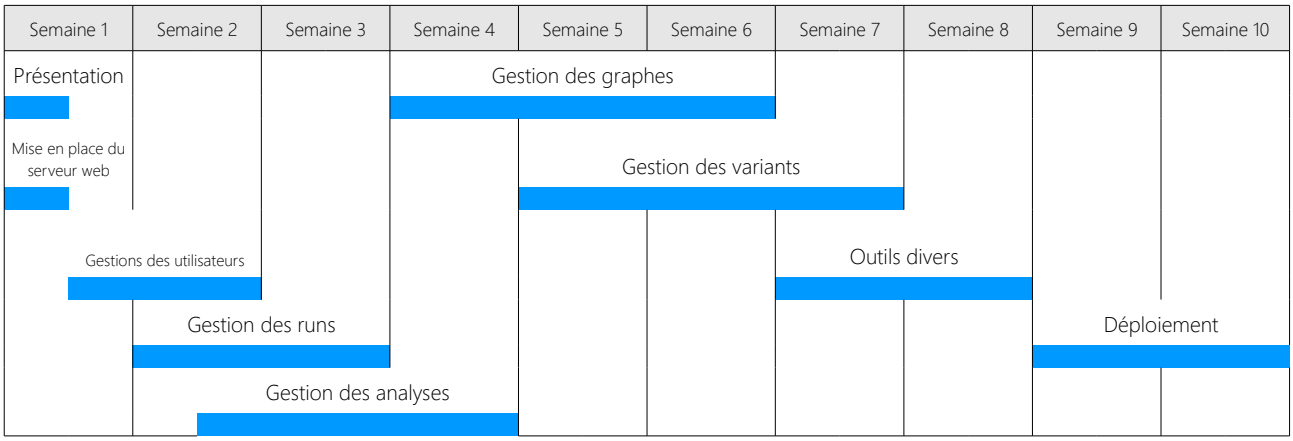


Figure 5 : Diagramme de Gantt prévisionnel réalisé en début de stage

4. Rappel : dans un run, le séquençage de l'ADN de plusieurs patients est effectué et des données brutes de séquençage sont déversées sur un serveur. L'analyse de ces données via un script d'analyse va permettre au biologiste de visualiser les résultats du séquençage et de déterminer si les patients sont porteurs de mutations délétères favorisant l'apparition d'un cancer.

Présentation :
Présentation de la structure et de l'organisation du LBM* par le personnel.

Mise en place du serveur web :
Installation des divers programmes servant à faire fonctionner l'interface et la base de données sur une machine sous CentOS 6.

Gestion des utilisateurs :
Implémentation de la partie dédiée à la gestion des utilisateurs et des droits d'accès dans la base de données et dans l'application.

Gestion des runs :

- Réalisation d'un script de détection des runs de séquençage capable de parcourir le système de fichier et de détecter l'apparition de nouveaux runs et le changement d'état des runs existants afin de mettre à jour la base de données,
- Intégration des runs dans la base de données et réalisation de l'affichage des runs dans l'interface.

Gestion des analyses :

- Réalisation d'un formulaire de lancement d'analyse permettant à l'utilisateur de saisir les paramètres requis et d'exécuter le script d'analyse,
- Détection d'une mise à jour de l'état des analyses dans le système de fichier et mise à jour de la base de données,
- Réalisation de l'affichage des analyses dans l'interface.

Gestion des graphes :
Réalisation d'un formulaire permettant de lancer le script d'analyse de couverture* et affichage des graphes de couverture générés par le script.

Gestion des variants* :
Réalisation d'un formulaire permettant de lancer le script de création du fichier des variants* et affichage intuitif des informations de ce fichier dans l'interface.

Outils divers :
Intégration d'outils supplémentaires utiles aux biologistes dans l'interface (ex. : permettre au biologiste d'écrire des commentaires, de valider des variants ou d'effectuer des recherches de patients).

Déploiement :
Mise en place de l'application sur la *Plateforme de génétique médicale* et vérification de son bon fonctionnement.

III. Déroulement réel

Au terme du stage, j'ai fait le diagramme de Gantt réel pour représenter les tâches réalisées :

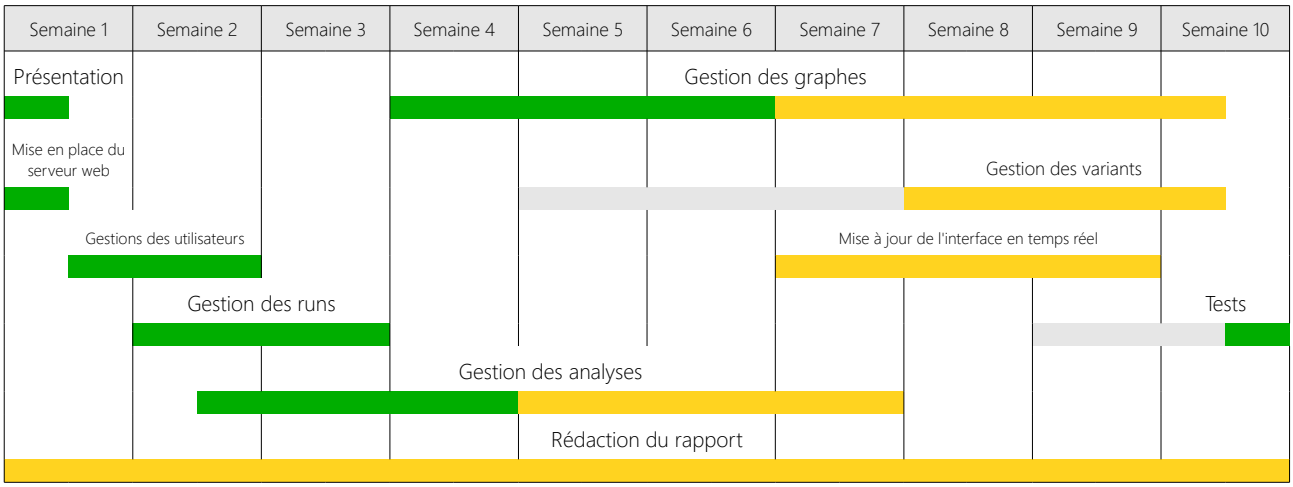


Figure 6 : Diagramme de Gantt réel réalisé en fin de stage

Légende :

: Partie d'une tâche réalisée dans le temps prévu par le planning prévisionnel

: Partie d'une tâche ayant nécessité du temps supplémentaire non prévu

: Partie d'une tâche ayant été décalée ou raccourcie par rapport au planning prévisionnel

Nous pouvons constater des différences entre le planning prévisionnel et le planning réel sur les tâches suivantes :

Gestion des analyses :

Cette partie a pris bien plus de temps que prévu pour les raisons suivantes :

- La création du formulaire de lancement d'analyse a nécessité l'utilisation du langage Javascript. Je n'avais jamais utilisé ce langage auparavant et un temps d'apprentissage et d'adaptation a donc été nécessaire.
- Le lancement du script en ligne de commande à partir de l'interface a posé un certain nombre de problèmes lié aux droits des utilisateurs Linux.
- J'ai sous-estimé le travail sur cette tâche qui a notamment inclus la création de trois pages web au lieu d'une seule prévue initialement.

Gestion des graphes :

J'ai réalisé cette partie en collaboration avec Stéphanie MONNERIE, stagiaire en deuxième année de master bioinformatique dans l'équipe de biologie moléculaire du CHU. Stéphanie devait créer un script d'analyse de couverture* et je devais l'intégrer dans mon interface pour que les utilisateurs puissent facilement l'exécuter et obtenir les résultats.

Cette partie n'a malheureusement pas pu être achevée à la fin de mon stage pour plusieurs raisons. D'une part, des modifications non prévues ont dû être apportées sur le script pour qu'il puisse être

intégré dans l'interface. D'autre part, le script a nécessité l'installation et la configuration de programmes spécifiques pour fonctionner conjointement à l'interface. Des problèmes de compatibilités ont également posé des problèmes. De plus, vers la fin de mon stage, Stéphanie a eu des problèmes de santé nous empêchant d'avancer sur cette partie.

Gestion des variants :

Cette tâche a été décalée en raison de la prolongation de la tâche "Gestion des analyses". En effet, la partie gérant les analyses devait être terminée avant de pouvoir commencer l'affichage des variants.

Mise à jour de l'interface en temps réel :

Cette tâche n'avait pas été prévue par le planning prévisionnel. Elle consiste à permettre aux pages web de l'application de se mettre à jour automatiquement sans réactualisation de la page par le client ou le navigateur. Cette partie a nécessité un travail d'implémentation non prévu, aussi bien du côté serveur que du côté client.

Tests :

Par manque de temps, je n'ai pu participer au déploiement de l'application sur la *Plateforme de génétique médicale*. Toutefois, l'application a pu être testée sur un autre ordinateur afin de vérifier son bon fonctionnement. L'objectif a donc été atteint puisque nous avons pu lancer des analyses depuis l'interface comme prévu.

Rédaction du rapport :

A la demande de notre professeur référent, la rédaction du rapport s'est déroulée sur toute la durée du stage. Cette tâche n'avait pas été prévue par le planning prévisionnel.

Ainsi, malgré des retards sur certaines tâches et à l'exception de la partie "Gestion des graphes", l'application est entièrement fonctionnelle et répond aux attentes formulées en début de stage.

Analyse

Avant de commencer le développement de l'application, une phase d'analyse a été réalisée afin d'identifier les besoins des utilisateurs, de comprendre les différents types de données à manipuler, de concevoir la base de données et de définir les vues de l'interface.

I. L'interaction avec l'interface

Suite aux indications de mon maître de stage, j'ai pu définir des rôles pour les utilisateurs ainsi que l'ensemble des interactions entre les acteurs et l'application. Ces informations sont formalisées dans le diagramme de cas d'utilisation suivant :

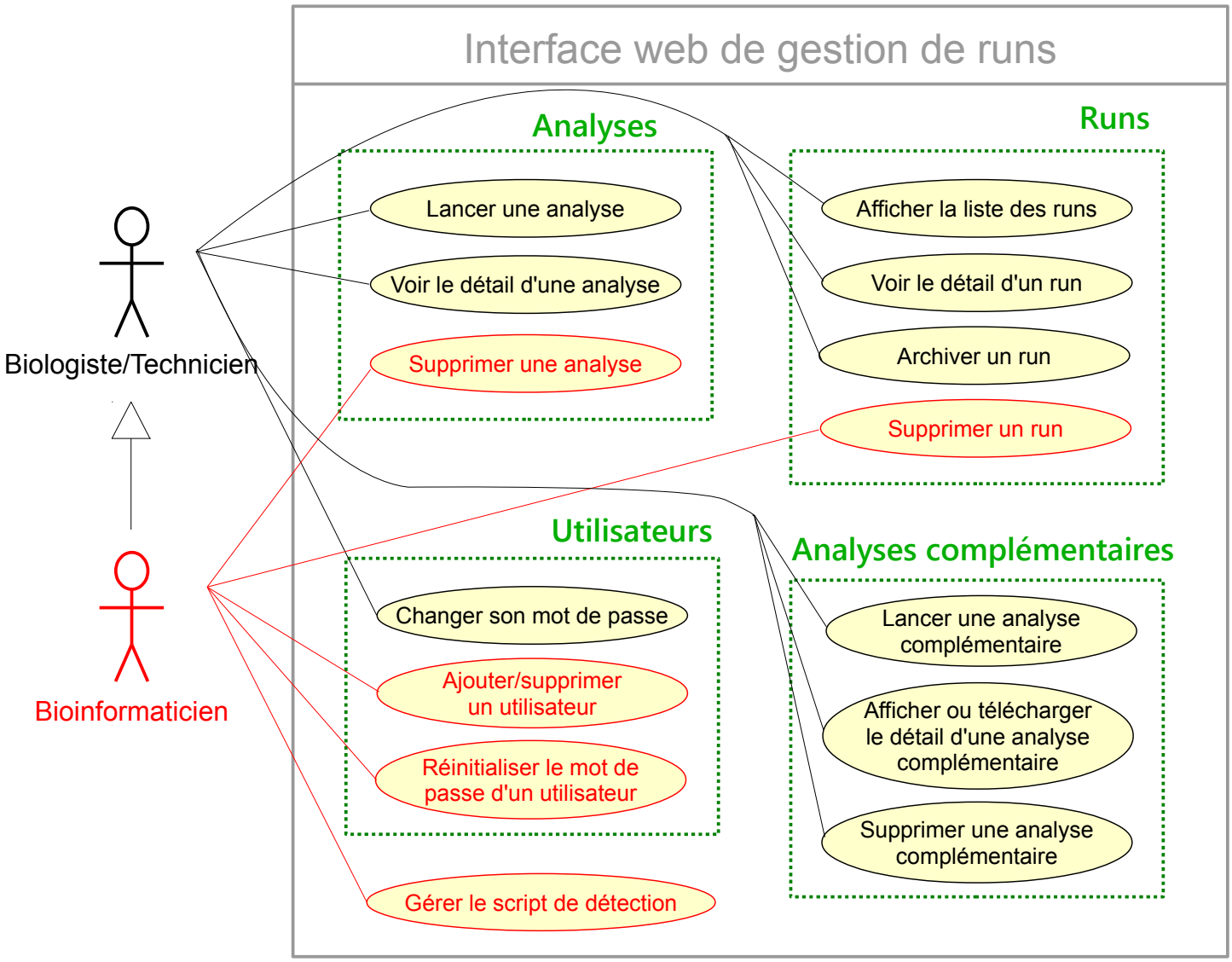


Figure 7 : Diagramme de cas d'utilisation de l'application

On distingue deux rôles pour les utilisateurs de cette application :
• les biologistes / techniciens
• les bioinformaticiens

Chaque rôle correpond à une fonction au sein de l'établissement. Pour chaque rôle est défini l'ensemble des actions pouvant être réalisées par l'utilisateur. Les actions en noir sont réalisables par les biologistes et les techniciens, tandis que les actions en rouge sont réservées aux bioinformaticiens. A l'heure actuelle, les biologistes et les techniciens peuvent réaliser les mêmes actions. Cependant, il est exigé que cette distinction soit faite afin que, par la suite, certaines actions réservées aux biologistes mais interdites aux techniciens puissent être facilement rajoutées. La flèche de généralisation située entre l'acteur "Biologiste/Technicien" et l'acteur "Bioinformaticien" indique que le bioinformaticien peut, en plus de ses propres actions, réaliser les actions des biologistes/techniciens. L'inverse n'est bien évidemment pas souhaitable et l'application doit être sécurisée à ce niveau afin que chaque acteur n'aie jamais la possibilité de réaliser une action qui ne lui est pas autorisée.

Chaque action est liée à un type de données : les runs, les analyses, ou les analyses complémentaires (la description de chacun de ces types est faite dans la partie Détails des données existantes ci-dessous). Pour simplifier le diagramme de cas d'utilisation, une limitation supplémentaire n'a pas été représentée : la gestion des services. En effet, les utilisateurs ainsi que les données de chaque type sont liés à un service. Or, les biologistes et les techniciens doivent être du même service que les données sur lesquelles ils veulent effectuer une action. Par exemple, un biologiste appartenant au service *Oncogénétique* ne pourra pas *voir les détails d'un run* lié au service *Anatomo-pathologie*. En revanche, les bioinformaticiens peuvent accéder à toutes les données quelque soit leur service.

II. Détails des données existantes

Les différents types de données que l'application aura à manipuler sont présentés dans cette partie. Elle détaille la structure de ces données, leur emplacement ainsi que les informations utiles que l'application devra prendre en compte et éventuellement afficher à l'utilisateur.

1) Les runs

Comme précisé dans l'introduction, un run désigne le processus de séquençage réalisé par le séquenceur. Lors du démarrage d'un run, le séquenceur va créer un dossier de run sur un serveur nommé *Donnees_Brutes* et dans lequel il déverse toutes les données issues du séquençage. L'arborescence du système de fichier du serveur de données est normalisée : à la racine du serveur, on trouve un dossier par année, puis dans chacun de ces dossiers sont stockés les dossiers de run. Le nom d'un dossier de run est de la forme suivante :

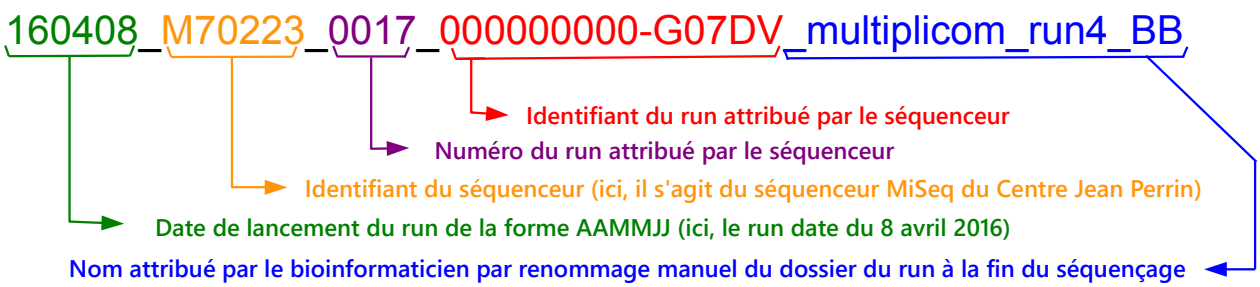


Figure 8 : Détail du format du nom d'un dossier de run

Ainsi un dossier de run sera accessible sous Linux par le chemin suivant :
`/mnt/Donnees_Brutes/2016/160408_M70223_0017_000000000-G07DV_multiplicom_run4_BB`

L'application doit permettre d'afficher la liste des runs mais également de connaître leur état. Un run peut avoir trois états :

- en cours de séquençage,
- en cours de génération des fichiers FastQ (fichiers utilisés par la suite lors de l'analyse du run),
- run terminé.

Un quatrième état a été rajouté pour l'application : l'état "Run introuvable", pour gérer le cas où le serveur de données est inaccessible ou bien que le dossier de run a été renommé ou supprimé. Certains fichiers ajoutés automatiquement par le séquenceur à la racine du dossier du run peuvent nous renseigner sur l'état de celui-ci. Les deux fichiers qui nous intéressent sont nommés *RTAComplete.txt* et *CompletedJobInfo.xml*. Leur présence ou leur absence doit permettre à l'application de connaître l'état du run.

Les transitions entre ces états sont formalisées à travers le diagramme suivant :

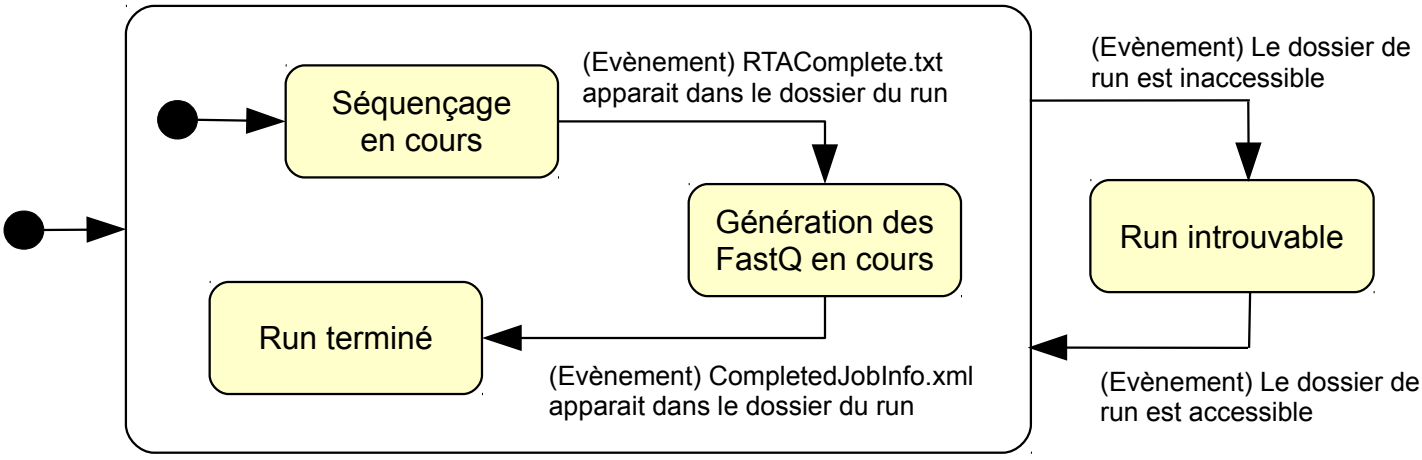


Figure 9 : Diagramme d'états-transitions d'un run

A noter qu'il n'y a pas d'état préalable à l'état "Séquençage en cours" car après avoir créé le dossier de run, le séquenceur commence immédiatement le séquençage. Il n'y a pas non plus d'état final qui indiquerait la fin du processus car si le dossier de run devient inaccessible, l'application doit être informée que le run est introuvable et ce même s'il était auparavant terminé.

Cela étant, il n'est ni utile ni souhaitable que l'application parcourt en permanence le système de fichier pour garder à jour l'ensemble des runs. D'une part, car cela risque de poser des problèmes de performances. D'autre part, parce que les anciens runs datant de plusieurs mois ou de plusieurs années ne sont plus utilisés. Pour cela, un système d'archivage doit être mis en place. Les utilisateurs pourront archiver les runs dont ils ne se servent plus et ceux-ci ne seront alors plus mis à jour. Les runs pourront à tout moment être désarchivés en cas de besoin et seront alors à nouveau mis à jour.

A titre d'information, voici quelques chiffres concernant les runs :

- La durée de séquençage d'un run varie de plusieurs heures à plusieurs jours.
- Actuellement, l'équipe du service Oncogénétique lance environ 3 à 4 runs par mois.
- Un dossier de run peut contenir de cinq à plusieurs dizaines de giga-octets de données.

2) Les patients

Lors du séquençage, la séquence nucléotidique* de certains gènes est déterminée chez plusieurs patients. Chaque run peut ainsi concerner entre 20 et 200 patients environ. Les données saisies dans le séquenceur sont anonymisées afin de protéger la vie privée des patients. Ainsi, l'application n'aura à manipuler aucune donnée personnelle mais uniquement un identifiant (exemple d'identifiant de patient : *14H4035*). L'application doit garder en mémoire la liste des patients liés à un run, information stockée dans un document nommé la *Sample Sheet*. Cette *Sample Sheet* est fournie au séquenceur par le technicien lors du lancement du run. Elle est consultable à la racine du dossier du run sous la forme du fichier *SampleSheet.csv* (un exemple de ce fichier est disponible en annexe à la page 48). Un fichier CSV (Comma-separated values) est un fichier texte dont les informations sont séparées par des virgules.

Le fichier *SampleSheet.csv* est consistué de plusieurs sections, dont la section [Data] qui contient sur chaque ligne les informations relatives à un patient :

```
15H1189,,,,p703,ATCAAAGG,p501,GTCAATAC,,Homo_sapiens\UCSC\hg19\Sequence\WholeGenomeFASTA,
```

L'information qui nous intéresse est ici l'identifiant du patient. En effet, afin que l'utilisateur puisse connaître les patients de chaque run, cette information doit être visible depuis l'interface. L'application devra donc être capable de parcourir ce fichier CSV et d'en extraire les identifiants de patients.

3) Les analyses

Lorsque le séquençage s'achève et donc que le run est à l'état "Terminé", les données brutes, qui sont inexploitables en l'état, doivent être analysées. Pour cela, les bioinformaticiens ont réalisé des scripts d'analyse. Un premier script peut être lancé dès la fin du séquençage afin générer des fichiers VCF (Variant Call Format). Ces fichiers contiennent les variants* portés par les patients, c'est-à-dire la partie d'un gène où la séquence est différente chez cet individu par rapport à la séquence de référence et potentiellement à l'origine d'une mutation délétère*. Ce script est écrit dans le langage Perl et peut être exécuté avec une ligne de commande Unix similaire à celle-ci :

```
perl parallel_global_pipeline.pl -analysis Panel_capture -input
inputFile0038.txt -runFolder /mnt/Donnees_Brutes/2016/160125_M70223_0010_
000000000-AHWH3_GroupeAU -bed Panel_LBM_2014_input.BRCA.bed -panel
Panel_LBM_2014_input.BRCA.transcripts
```

Figure 10 : Exemple de ligne de commande permettant d'exécuter le script d'analyse

La commande `perl parallel_global_pipeline.pl` appelle le programme `perl` qui va interpréter et exécuter le contenu du script d'analyse nommé `parallel_global_pipeline.pl`. Ce script requiert plusieurs arguments pour fonctionner :
`-analysis` : le type d'analyse à effectuer,
`-input` : un fichier contenant une liste des identifiants des patients du run à analyser,
`-runFolder` : le nom du run à analyser,

Les arguments suivants (-bed et -panel) indiquent l'emplacement des fichiers nécessaires à l'exécution de l'analyse. Le nombre et le type de ces fichiers dépendent du type d'analyse. Par exemple, une analyse de type *Exome* ne nécessitera qu'un fichier .bed (avec l'argument **`-bed`**), tandis qu'une analyse de type *Panel_capture* nécessitera en plus du fichier .bed, un fichier Panel.

Pour un run donné, plusieurs analyses peuvent être lancées. Chaque analyse peut être effectuée sur tout ou partie des patients du run. Les patients à analyser sont inscrits dans le fichier input qui est donné en argument du script.

A l'heure actuelle, l'utilisateur qui ne connait pas Linux ou qui n'est pas familier avec les interfaces en ligne de commande ne peut pas lancer d'analyse lui-même. L'application devra donc permettre de lancer une analyse à travers une interface intuitive en demandant le type d'analyse (argument **`-analysis`**), les différents fichiers requis selon ce type ainsi que la liste des patients à analyser. A partir de cette liste, le fichier d'input (argument **`-input`**) contenant les patients devra être généré et stocké de façon temporaire, puis la ligne de commande sera créée automatiquement et enfin exécutée en arrière-plan par l'application.

Les données issues des analyses sont stockées sur un deuxième serveur nommé *Analyses_Secondaires*. Le script d'analyse va créer un dossier pour chaque run suivant la même arborescence que pour le serveur Donnees_Brutes, à savoir un dossier par année à la racine, puis à l'intérieur un dossier par run nommé de la même manière que les dossiers de run créés par le séquenceur :

Exemple de dossier créé par le séquenceur :
`/mnt/Donnees_Brutes/2016/160408_M70223_0017_000000000-G07DV_multiplicom_run4_BB`

Exemple de dossier créé par le script d'analyse pour le même run :
`/mnt/Analyses_Secondaires/2016/160408_M70223_0017_000000000-G07DV_multiplicom_run4_BB`

Tous les résultats des analyses d'un run seront stockés dans ce dernier dossier.

Pour chaque analyse, un fichier nommé *log_Analyse_Secondaire.xml* est créé à la racine du dossier du run sur le serveur *Analyses_Secondaires*. Celui-ci est vide au début de l'analyse et se remplit quand elle est terminée. Il contient notamment les dates et heures de début et de fin de l'analyse ainsi que les temps d'exécution pour chaque patient. Ces informations devront être affichées dans l'interface.

Tout comme pour les runs, une analyse peut durer plusieurs heures. L'interface doit alors afficher l'état d'avancement des analyses en temps réel. Les différents états de l'analyse ainsi que les transitions entre les états sont détaillées dans le diagramme ci-dessous :

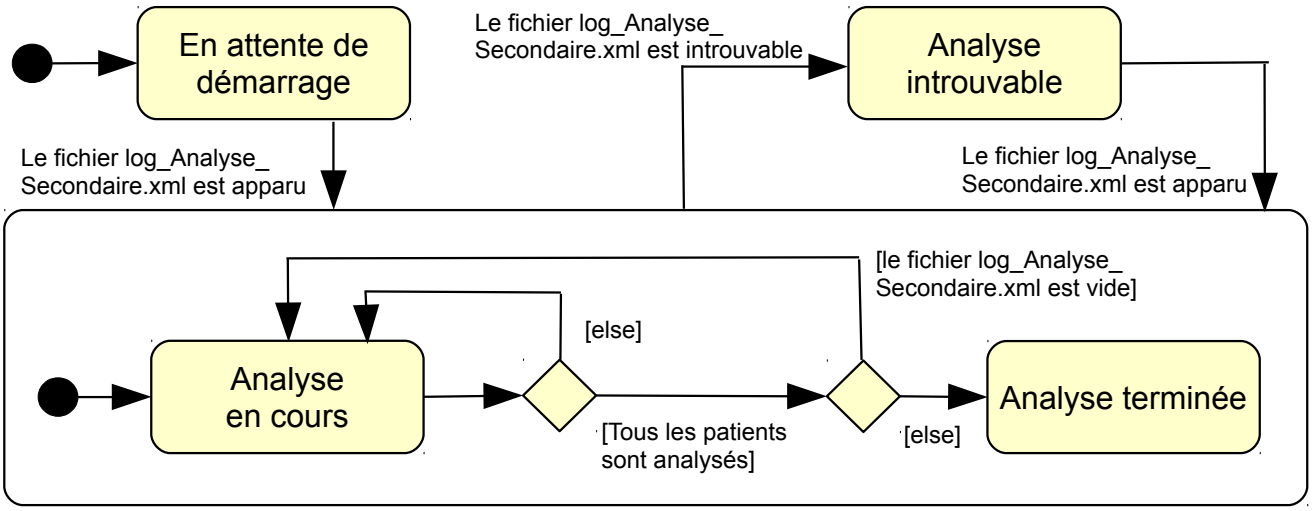


Figure 11 : Diagramme d'états-transitions d'une analyse

Lors de l'état "Analyse en cours", l'application doit prendre en compte l'état d'avancement de l'analyse de chaque patient. Cela permet de savoir lorsque tous les patients sont terminés mais également d'afficher la progression de l'analyse en cours dans l'interface. Les patients peuvent eux-même passer par 3 états différents :

- "En attente" : l'analyse est en cours mais elle n'a pas encore commencé sur ce patient.
- "En cours" : un dossier de patient apparait dans le dossier de l'analyse, cela signifie que l'analyse de ce patient a commencé.
- "Terminé" : un fichier nommé *out_vcf/[IdDuPatient].annotated.QC.vcf* est apparu dans le dossier du patient. L'analyse de ce patient est terminée.

Lorsque tous les patients sont terminés, le fichier *log_Analyse_Secondaire.xml* qui a été rempli peut alors être lu. A partir de ce moment, l'analyse est considérée comme "Terminée" et d'autres scripts peuvent alors être lancés afin de réaliser des analyses complémentaires.

4) Les analyses complémentaires

Lorsqu'une analyse est terminée, des scripts supplémentaires peuvent être exécutés afin d'extraire diverses informations des fichiers .vcf (fichiers contenant les variants*) générés par l'analyse : ce sont les scripts d'analyses complémentaires. Les analyses complémentaires sont également exécutables via une ligne de commande et peuvent être effectuées sur tout ou partie des patients de l'analyse. Elles sont stockées dans un dossier nommé "Analyses_complementaires" situé à la racine du dossier d'analyse. L'application doit pouvoir gérer deux types d'analyses complémentaires :

- L'analyse de couverture permet de déterminer pour chaque portion de gène ciblée la fiabilité du séquençage réalisé. Le script d'analyse de couverture peut générer des fichiers .pdf de graphes de couverture, qui contiennent des informations lisibles et interprétables par les biologistes.

- Le fichier global de résultat peut être créé par un autre script et résume les informations sur les variants* portés par chaque patient du run. C'est un fichier de type tableur Excel également interprétable par les biologistes.

L'exécution d'un script d'analyse complémentaire dure en moyenne 30 minutes. Tout comme pour les analyses, les utilisateurs doivent pouvoir lancer les scripts d'analyse complémentaire à travers l'interface en saisissant les différents paramètres requis. L'application devra là encore se charger de créer et d'exécuter la ligne de commande. Les utilisateurs doivent pouvoir télécharger les fichiers de résultats de chaque analyse complémentaire, mais également pouvoir visualiser ces résultats via une interface ergonomique.

5) Le modèle du domaine

A partir des informations recueillies à propos des données existantes, à savoir les runs, les patients, les analyses, et les analyses complémentaires, j'ai pu concevoir un premier modèle du domaine :

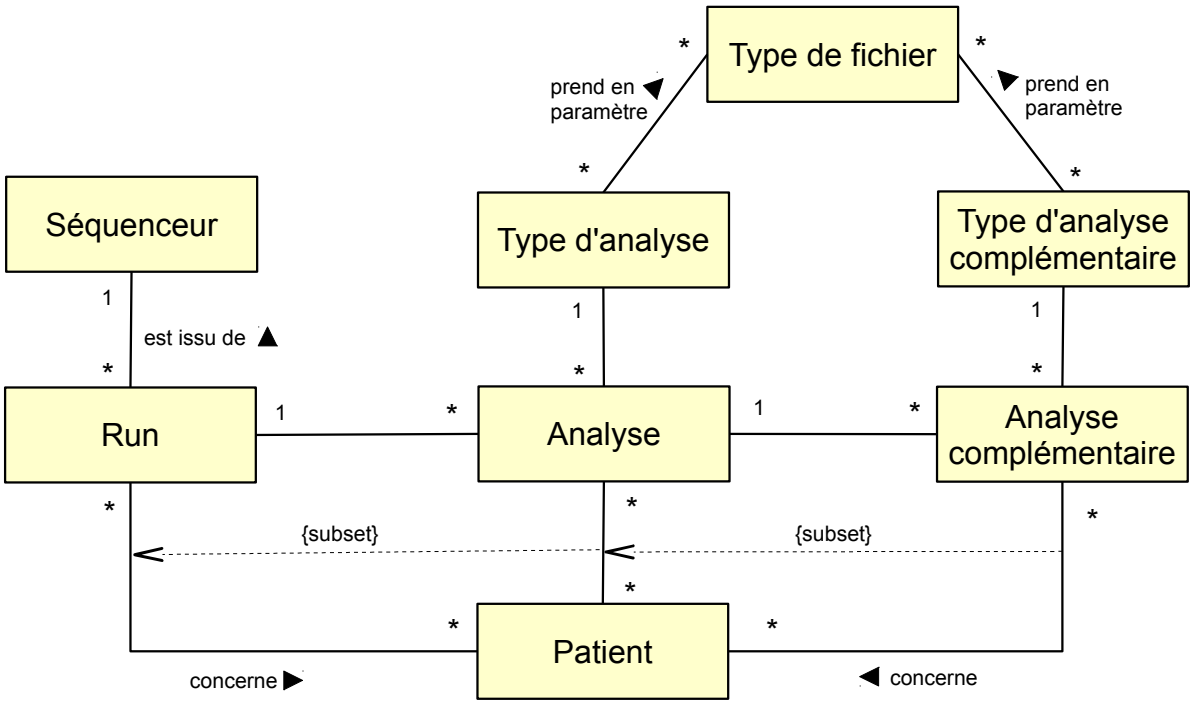


Figure 12 : Modèle du domaine concernant les données existantes

La mention {subset} signifie "sous-ensemble". Cela signifie par exemple que l'ensemble contenant les patients d'une analyse est inclus dans l'ensemble contenant les patients du run lié à cette analyse.

Ce diagramme décrit uniquement les relations entre les données déjà existantes. Les nouvelles données à intégrer dans le modèle et nécessaires à l'application comme les utilisateurs ou les services sont présentées ci-dessous.

III. Détails des données de l'application

Avant la création de l'application, les notions d'utilisateurs, de service et de fonction n'existait pas dans le système d'information lié aux runs de l'établissement. Pour les intégrer au modèle de l'application, il a fallu définir leur structure et leur interacton avec les autres types de données.

1) Les utilisateurs

Un utilisateur représente un individu membre du personnel de l'établissement. L'application devra connaître le nom, le prénom, l'e-mail, la fonction et le service de l'individu. L'utilisateur aura accès dans l'interface aux informations le concernant.

L'application ne doit être accessible qu'aux utilisateurs authentifiés à l'aide d'un couple login / mot de passe. Pour des raisons de sécurité, le mot de passe ne sera pas stocké en clair dans la base de données mais sera au préalable transformé par une fonction de hachage. Cette technique permet de comparer facilement un mot de passe saisi par l'utilisateur avec le mot de passe stocké tout en rendant très difficile l'obtention en clair du mot de passe stocké. L'application doit pouvoir permettre à un utilisateur de changer son mot de passe.

2) Les fonctions

Les fonctions correspondent aux rôles définis dans le diagramme de cas d'utilisation, à savoir *technicien*, *biologiste* et *bioinformaticien*. La fonction de chaque individu devra être renseignée et celle-ci définira les droits de l'utilisateur sur l'application. L'entité Fonction sera composée d'un identifiant numérique (actuellement 0 pour le technicien, 1 pour le biologiste et 2 pour le bioinformaticien) ainsi que du nom de la fonction.

3) Les services

Chaque utilisateur et chaque run doit être lié à un et un seul service afin que l'application puisse restreindre l'accès par les utilisateurs aux seuls runs de leur service. Seuls les utilisateurs ayant la fonction *Bioinformaticien* auront accès quelque soit leur service à tous les runs. L'entité Service est composé de l'identifiant et du nom du service (ex : *CJPLOM* est l'identifiant du service *Oncogénétique du Centre Jean Perrin*).

Auparavant, le service des runs n'était pas mentionné dans le système de fichier. Il n'y avait donc aucun moyen de savoir à quel service appartenait un run. Pour que l'application puisse obtenir cette information, l'identifiant du service doit désormais être inscrit dans la ligne *Description* de la *Sample Sheet* avant le démarrage du run :

```
Description,CJPLOM_multiplicom_run4_BB,,,,,,,,
```

Si cet identifiant n'est pas inscrit ou qu'il est incorrect, le run n'appartiendra à aucun service et ne sera accessible que par les bioinformaticiens.

IV. Modèle relationnel de la base de données

A partir des informations collectées sur les données, leur structure et leurs interactions, j'ai pu modéliser la base de données de l'application. Cette modélisation est formalisée à travers le diagramme suivant :

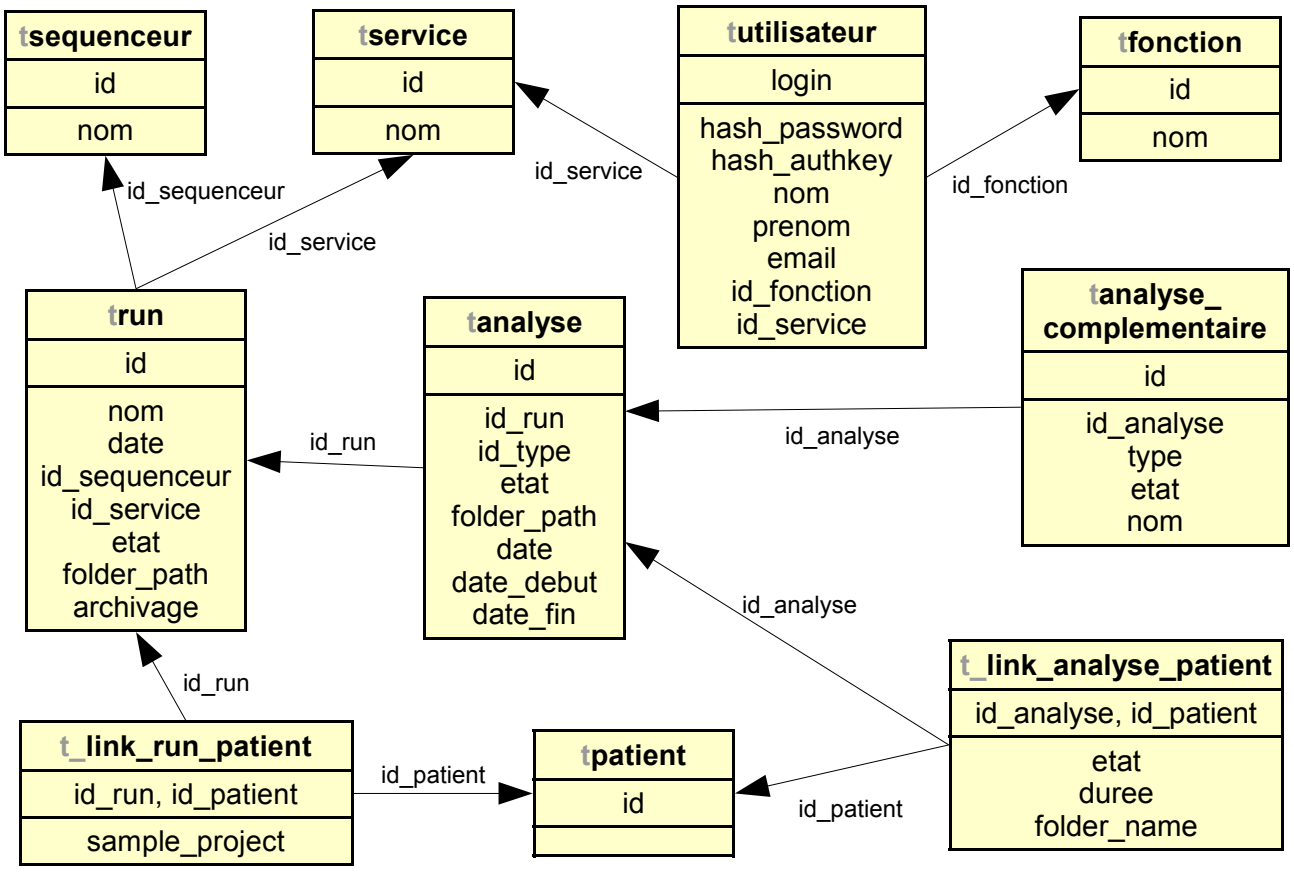


Figure 13 : Modèle relationnel partiel de la base de données

Pour alléger le diagramme et en simplifier la lecture, je n'ai reproduit ici qu'un modèle relationnel partiel. La partie concernant les types d'analyses et les types de fichiers ainsi que plusieurs tables n'ayant aucune relation n'ont pas été intégrées. Le modèle relationnel complet est disponible en annexe (page 49).

Par rapport au modèle du domaine, les tables d'associations t_link_run_patient et t_link_analyse_patient ont été rajoutées. Ces tables permettent à un patient d'être lié respectivement à un run et à une analyse. L'utilisation de ces tables est nécessaire pour permettre qu'un run contienne plusieurs patients et qu'un patient soit lié à plusieurs runs, tout en s'assurant que la base de données respecte la *première forme normale**.

L'autre différence avec le modèle du domaine réside dans le fait qu'il n'y ait aucune relation entre la table des analyses complémentaires et la table des patients. En effet, même si une analyse complémentaire

concerne effectivement plusieurs patients, il n'a pas été jugé utile pour l'application de conserver cette information.

Les détails des attributs de chaque table est consultable dans le dictionnaire des données en annexe (page 50).

V. Définition des vues de l'interface

Pour réaliser des actions, les utilisateurs vont avoir accès à plusieurs pages web qui vont constituer l'interface. Le fait d'être sur un page web peut être assimilé à un état pour l'interface web. Ainsi, l'ensemble des pages ainsi que les moyens de passer d'une page à l'autre peuvent être formalisés à travers le diagramme d'états-transitions suivant :

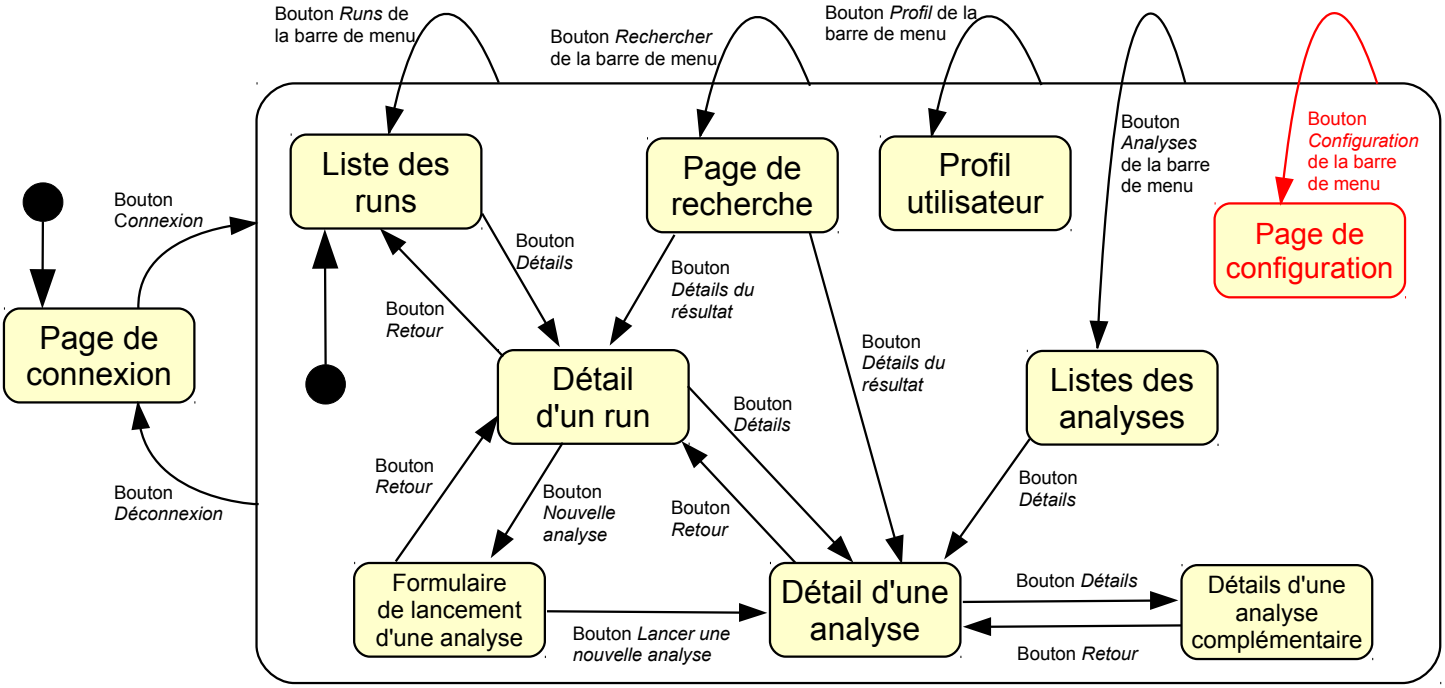


Figure 14 : Diagramme d'états-transitions des pages de l'interface

La page de connexion est le point d'entrée de l'application. Elle est la seule page à ne pas nécessiter d'être connecté pour être accessible et permet d'accéder au reste de l'application sous réserve de saisir un login et un mot de passe valide.

Dans la partie supérieure du diagramme se situent les pages accessibles depuis la barre de menu présente sur toutes les pages de l'interface, à l'exception de la page de connexion. La barre de menu permet d'accéder aux pages suivantes :

- la liste des runs : page d'accueil qui liste l'ensemble des runs du service de l'utilisateur. Ce dernier peut choisir d'afficher uniquement les runs issus d'un séquenceur, les runs archivés ou encore les runs d'une année donnée.
- la listes des analyses : affiche l'ensemble des analyses des runs non-archivés liés au service de l'utilisateur. L'état d'avancement de chaque analyse est indiqué et les analyses sont regroupées par runs.
- la page de recherche : permet à l'utilisateur de rechercher et d'afficher des runs et des analyses suivant différents critères comme leur nom, leur date ou encore leur état.
- la page de profil de l'utilisateur : affiche les informations de l'utilisateur actuellement connecté, à savoir le login, le nom, le prénom, l'e-mail, la fonction et le service de l'utilisateur. Cette page lui permet aussi de changer son mot de passe.
- la page de configuration : accessible uniquement par les bioinformaticiens, cette page permet de gérer l'application à travers différents paramètres. Le bioinformaticien peut ajouter ou supprimer des utilisateurs et gérer l'exécution du script de mise à jour en temps réel de la base de données.

Les quatres dernières pages sont accessibles depuis les pages précédentes :

- la page de détail d'un run affiche toutes les informations d'un run à savoir sa date, son état, la liste des patients qui lui sont liés ainsi que la liste des analyses le concernant.
- le formulaire de lancement d'analyse est accessible depuis la page de détail d'un run à condition que ce dernier soit à l'état *Terminé*. Cette page permet à l'utilisateur de saisir les paramètres du script d'analyse puis de l'exécuter.
- la page de détail d'une analyse est disponible au moment où l'analyse est lancée. Elle permet de suivre son état d'avancement ainsi que la progression de l'analyse de chaque patient. Cette page affiche également la liste des analyses complémentaires et permet, une fois l'analyse terminée, de lancer les scripts d'analyse complémentaire via des formulaires également disponibles sur cette page.
- les pages de détails d'analyse complémentaire regroupent deux pages différentes permettant d'afficher les résultats des analyses complémentaires des types "Analyse de couverture" et "Création du résultat global" respectivement. Ces pages permettent un affiche ergonomique des résultats et offrent également la possibilité à l'utilisateur d'exporter et télécharger ces résultats au format PDF ou Tableur Excel (.xls).

Développement

Une fois la phase d'analyse terminée, la développement de l'application peut débuter. Dans cette partie sont détaillés les technologies utilisés, l'architecture de l'application, ainsi que le fonctionnement de certaines fonctionnalités clés.

I. Présentation des technologies

Pour réaliser cette application, plusieurs technologies et langages ont été utilisés. S'agissant d'une interface web, l'application utilise tout d'abord le langage HTML. C'est le langage de base d'une page web, utilisé pour en représenter la structure. Il est combiné au langage CSS (Cascade Style Sheet) afin de définir le placement des éléments sur la page ainsi que de leur appliquer des effets de style.

Afin de réduire le temps de travail à réaliser l'aspect graphique de l'application, j'ai utilisé un framework CSS nommé Bootstrap qui permet d'une part, de facilement structurer une page web grâce à un système de grille, et d'autre part d'appliquer des styles pré-définis aux éléments d'une page. Bootstrap offre une palette d'éléments à la fois ergonomiques et visuellement agréables tout en restant sobres. L'aspect graphique de Bootstrap convient donc tout à fait à un établissement de santé.

Pour dynamiser les pages de web de l'application côté client, j'ai dû utiliser le langage Javascript. C'est un langage de script qui est exécuté par le navigateur du client et qui permet de modifier dynamiquement une page web en permettant par exemple d'afficher des messages d'erreur, de cacher des élements ou de vérifier les champs d'un formulaire avant qu'il ne soit envoyé au serveur. N'ayant jamais utilisé ce langage, j'ai dû l'apprendre spécialement pour ce stage.

Le langage Javascript offre également la possibilité au client d'envoyer et de recevoir des informations du serveur sans changer de page à travers la technologie AJAX. Les informations reçues du serveur en AJAX peuvent servir à modifier dynamiquement certains éléments de la page web sans avoir à la recharger. Par exemple, c'est de cette manière que la page d'un run peut afficher l'état de ce dernier en temps réel.

Pour faciliter l'envoi de requêtes AJAX et la modification dynamique d'éléments HTML, j'ai également utilisé un framework Javascript nommé Jquery.

Javascript simple

```javascript
// Initialisation de l'objet permettant l'envoi
var xhr = new XMLHttpRequest();
// On indique le type et l'URL de la requête
xhr.open('GET', '?action=ajax_getAllRuns');
// Déclenchera la fonction lors du changement
d'état de l'objet
xhr.addEventListener('readystatechange',function(){
    if(xhr.readyState == XMLHttpRequest.DONE){
       // La requête s'est terminée
       if(xhr.status == 200){
           //En cas de succès :
           displayRuns(xhr.responseText);
       }
       else{
           //En cas d'erreur :
           displayError(xhr.responseText);
       }
    }
});

xhr.send(null); //Envoi de la requête AJAX
```

Jquery

```javascript
$.ajax({
    url: '?action=ajax_getAllRuns',
    type: 'GET',
    success: function(response){
        //En cas de succès :
        displayRuns(response)
    },
    error: function(error){
        //En cas d'erreur :
        displayError(error.responseText)
    }
}); //Envoi de la requête AJAX
```

Figure 15 : Exemple d'une même requête Ajax envoyée en Javascript simple ou en Jquery et permettant de récupérer depuis le serveur la liste des runs afin de les afficher.

Dans le code ci-dessus, nous pouvons voir que la fonction d'exécution d'une requête AJAX est plus courte, plus rapide à écrire et plus facilement compréhensible que celle utilisant l'objet *XMLHttpRequest* qui permet de réaliser la même action en Javascript simple.

En lien avec le Javascript, j'ai eu à manipuler le format de données JSON (JavaScript Object Notation) qui permet notamment de facilement transférer des données en AJAX afin de les réutiliser dans d'autres langages.

Les informations à afficher dans l'interface sont stockées dans une base de données MySQL. Les données y sont structurées et organisées afin de permettre leur accès via des requêtes SQL. La ligne de commande suivante est une requête SQL permettant d'obtenir tous les runs de la base de données triés par ordre décroissant de date :

```sql
SELECT * FROM trun ORDER BY date DESC
```

Pour répondre aux requêtes HTTP du client et envoyer des pages web, le langage PHP est utilisé côté serveur. C'est le langage qui constitue le coeur de l'application et qui définit toute sa logique. Il sert d'interface entre les utilisateurs, la base de données et le système de fichiers. Contrairement au Javascript, le code PHP du serveur n'est ni accessible, ni modifiable par l'utilisateur final. C'est donc en PHP que doivent être réalisées toutes les actions pouvant présenterdes risques de sécurité.

Le fonctionnement de la partie PHP de l'application ainsi que celui de la base de données nécessite l'utilisation d'un serveur HTTP. Le logiciel libre Apache, disponible sous Linux, sera utilisé pour remplir cette fonction. En 2015, il était le serveur HTTP le plus populaire en étant utilisé par plus de 50% des sites

internets actifs du monde entier[5]. Apache permet d'exécuter des processus en arrière-plan, fonctionnant en permanence, afin de répondre à des requêtes. Ces processus sont appelés des daemons et sont généralement exécutés au démarrage de la machine. Pour faire fonctionner notre application, nous utiliserons deux daemons :

- un deamon HTTP (httpd), capable d'envoyer des pages webs via le protocole HTTP lorsque le serveur reçoit une requête d'un client,
- un deamon MySQL (mysqld), utilisé pour connecter le serveur web à la base de données MySQL.

II. Installation

Le développement de l'interface a été réalisé sur un ordinateur équipé du système d'exploitation CentOS 6.7. Pour installer les différents programmes nécessaires au fonctionnement de l'application, CentOS dispose d'un outil nommé **yum** (pour *Yellowdog Updater Modified*). C'est un gestionnaire de paquets permettant d'installer et de mettre à jour des programmes en ligne de commande. Par exemple, la commande suivante permet d'installer Apache :

```
yum install httpd
```

J'ai ainsi effectué l'installation et la configuration d'Apache, de PHP et de MySQL au début du stage. J'ai ensuite installé l'éditeur de texte *Sublime Text* afin de commencer le développement de l'application.

III. Architecture de l'application

1) Présentation générale du modèle MVC

Pour réaliser cette application, j'ai choisi d'implémenter le patron d'architecture MVC (Modèle-Vue-Contrôleur). Celui-ci sépare la structure de l'application entre trois parties :

- La partie Modèle va gérer l'accès et la modification des données, que ce soit avec une base de données ou un système de fichier.
- La partie Vue se chargera de générer et d'afficher les vues, c'est-à-dire les pages web à envoyer à l'utilisateur.
- La partie Contrôleur fera le lien entre le modèle et la vue. Elle interprétera la requête demandée par l'utilisateur, puis ira éventuellement chercher ou modifier les données par le biais du modèle pour enfin afficher ces données ou des messages à l'attention de l'utilisateur grâce à la vue.

Le patron MVC est massivement utilisé par les applications web ainsi que par de nombreux frameworks. Il favorise la lisibilité et l'extensibilité de l'application en découplant chacune de ses grandes fonctions. Ainsi chaque partie a un rôle bien défini et ne doit pas faire exécuter directement les actions d'autres rôles (par exemple, la vue ne doit jamais effectuer de requête SQL pour obtenir des données de la base de données car cette action est réservé au modèle).

5. Source : http://news.netcraft.com/archives/2015/01/15/january-2015-web-server-survey.html

2) Application du modèle MVC

Dans cette partie du rapport, je vais détailler la manière dont j'ai implémenté le modèle MVC pour cette application en suivant une requête initiée par le client d'un navigateur puis sa réponse par le serveur.

a) Le point d'entrée : index.php

Toutes les requêtes clients de l'application débutent en faisant appel au fichier *index.php* situé à la racine du serveur. A chaque requête, ce fichier va exécuter trois instructions principales. Il va tout d'abord initialiser la classe Autoloader, qui permet à PHP de charger automatiquement le code des classes qui vont être utilisées sans avoir besoin d'importer manuellement leur code. Ensuite, il va réaliser la connexion à la base de données, ce qui permettra ensuite au modèle de récupérer et modifier les données de la base. Enfin, il va diriger la requête du client vers la partie Contrôleur qui se chargera d'y répondre.

b) La partie Contrôleur

Avant de répondre à la requête du client, la partie contrôleur va devoir définir le rôle de l'utilisateur. Pour rappel, les utilisateurs de l'application peuvent avoir le rôle Biologiste/Technicien ou le rôle Bioinformaticien. Pour chaque rôle, une classe Contrôleur va lister toutes les actions que l'utilisateur est autorisé à réaliser. Les classes NormalController et SuperController contiennent la logique d'exécution des actions respectivement liés au Biologiste/Technicien et au Bioinformaticien. La classe Routeur va alors se charger de déterminer le rôle de l'utilisateur et de le rédiriger vers le contrôleur qui lui est lié. Les rôles de chaque utilisateur sont inscrits dans la base de données (ils correspondent à l'attribut *fonction* de la table *t_utilisateur*). La classe Routeur va donc faire appel au modèle afin d'obtenir le rôle de l'utilisateur courant.

L'action que le client souhaite réaliser est stockée dans l'URL sous la forme "index.php?action=nom_de_l_action". Chaque action correspond au nom d'une des méthodes d'une classe Contrôleur. Ainsi la classe Routeur va extraire l'action de l'URL puis va ensuite faire de l'introspection en appelant la méthode `get_class_methods()` (méthode interne de PHP) afin d'obtenir la liste des méthodes de la classe Contrôleur liée au rôle. Si l'action demandée correspond à une des méthodes récupérées, cette méthode est exécutée. Sinon la requête est refusée.

Exemple de requête :

- L'utilisateur est un biologiste et l'URL envoyé au serveur est "*index.php?action=displayRuns*". La classe NormalController possède bien une méthode *displayRuns* permettant d'afficher tous les runs, cette méthode est donc exécutée.
- L'utilisateur est un technicien et l'URL envoyé au serveur est "*index.php?action=config*". La classe SuperController possède une méthode *config* permettant d'afficher la page de configuration de l'application. Or, le rôle Technicien n'est pas lié à la classe SuperController mais à NormalController. Cette dernière ne possède pas de méthode *config*, la requête est donc refusée.

Cas particuliers :

- Si aucune action n'est spécifiée dans l'URL, la méthode defaultAction de la classe Contrôleur lié au rôle sera appelée.
- Si l'utilisateur ne possède aucun rôle, quelque soit l'action demandée, la classe Routeur va faire appel à la vue afin d'afficher la page de connexion.
- Comme mentionné dans le diagramme de cas d'utilisation, le bioinformaticien peut réaliser toutes les actions réalisables par le biologiste/technicien. Pour cela, au lieu de recopier toutes les méthodes du NormalController dans le SuperController, PHP nous offre la possibilité de faire hériter la classe SuperController de la classe NormalController. Ainsi cela signifie qu'en définissant la classe comme suit "`class SuperController extends NormalController`", la classe SuperController possédera toutes les méthodes de la classe NormalController, permettant au bioinformaticien d'exécuter les actions des deux classes contrôleurs.

Cette façon de procéder est claire et facilite l'extension puisqu'il suffit de rajouter une méthode dans la classe contrôleur voulue pour rajouter une action au rôle correspondant.

Selon l'action demandée, la méthode du contrôleur va ensuite faire appel au modèle afin de récupérer ou modifier des informations et à la vue afin de les envoyer au client.

c) La partie Modèle

Le modèle consistue l'unique point d'accès aux données de l'application. Pour les besoins de l'application, le modèle est découpé en trois parties :

- la partie *Gateway* permet à l'application d'accéder à la base de données. Elle contient une classe pour chaque entité de l'application (*RunGateway*, *AnalyseGateway*, *UtilisateurGateway*, etc ...). Ces classes sont les seules à être habilitées à manipuler et exécuter des requêtes SQL. Elles utilisent notamment l'objet **`PDO`** qui permet depuis PHP 5.1 de communiquer avec la base de données de manière sécurisée en prévenant tout risque d'injections SQL*.
- la partie *FromFile* (FF) permet à l'application d'accéder aux données du système de fichiers. Là encore, cette partie contient plusieurs classes représentant les entités de l'application (*RunFF*, *AnalyseFF*, *PatientFF*, etc ...). Ces classes doivent être les seules à utiliser les fonctions PHP d'accès aux fichiers telles que **`scan_dir()`** qui permet de lister les éléments d'un dossier ou encore **`file_get_contents()`** pour obtenir le contenu d'un fichier.
- la partie principale du modèle contient les classes commençant par *Model* (*ModelRun*, *ModelAnalyse*, etc ...). Ce sont les seules classes à pouvoir accéder aux classes des deux parties précédentes et pourront ainsi accéder aux données de la base de données et du système de fichier. Par exemple, la méthode **`ModelRun::getAll()`** va appeler la méthode **`RunGateway::getAll()`** afin d'obtenir tous les runs de la base de données. Autre exemple, la méthode **`ModelRun::refresh()`** va appeler entre autres les méthodes **`RunFF::getAll()`** et **`RunGateway::add()`** afin de récupérer des runs du système de fichier puis de les ajouter dans la base de données. Ce seront les méthodes des classes *Model* qui seront appelées par les classes *Controller*.

Une fois les données récupérées via le modèle, le contrôleur va faire appel à la partie Vue pour envoyer ces données au client.

d) La partie Vue

Selon le format de données attendu par le client, le contrôleur va faire appel soit à la classe *HtmlClientSender* permettant d'envoyer une page HTML, soit à la classe *AjaxClientSender* pour envoyer un message texte ou des données au format JSON. La classe *HtmlClientSender* possède une méthode par vue. Les vues (V) sont stockées dans le dossier *view* et chaque vue correspond à une page. Ainsi `HtmlClientSender::sendRunView()` affichera *runV.php*, `HtmlClientSender::sendRunListView()` affichera *run_listV.php*, etc ... . La classe *AjaxClientSender* quant à elle possède les méthodes `sendDataJson()` et `sendText()` selon le format d'envoi choisi.

Au-delà du format de données envoyé, j'ai voulu attribuer à chacune de ces classes un rôle bien distinct. La classe *HtmlClientSender* doit fournir au client le squelette de la page web demandée en ne contenant qu'un minimum de données issues du modèle. Le navigateur du client se chargera alors d'exécuter le Javascript fourni avec la page afin d'envoyer une ou plusieurs requêtes AJAX au serveur. Ces requêtes seront ensuite gérées en tant qu'actions par la partie Contrôleur, puis la classe *AjaxClientSender* sera appelée pour envoyer les données demandées afin de remplir la page du client.

L'affichage d'une vue et la complémentarité HTML/AJAX sont détaillées à travers l'exemple suivant : l'utilisateur est sur la page "Liste des runs" et souhaite aller sur la page "Détails d'un run" pour obtenir la liste des patients de ce run. Les interactions réalisées par cette action sont présentées dans le diagramme ci-dessous :

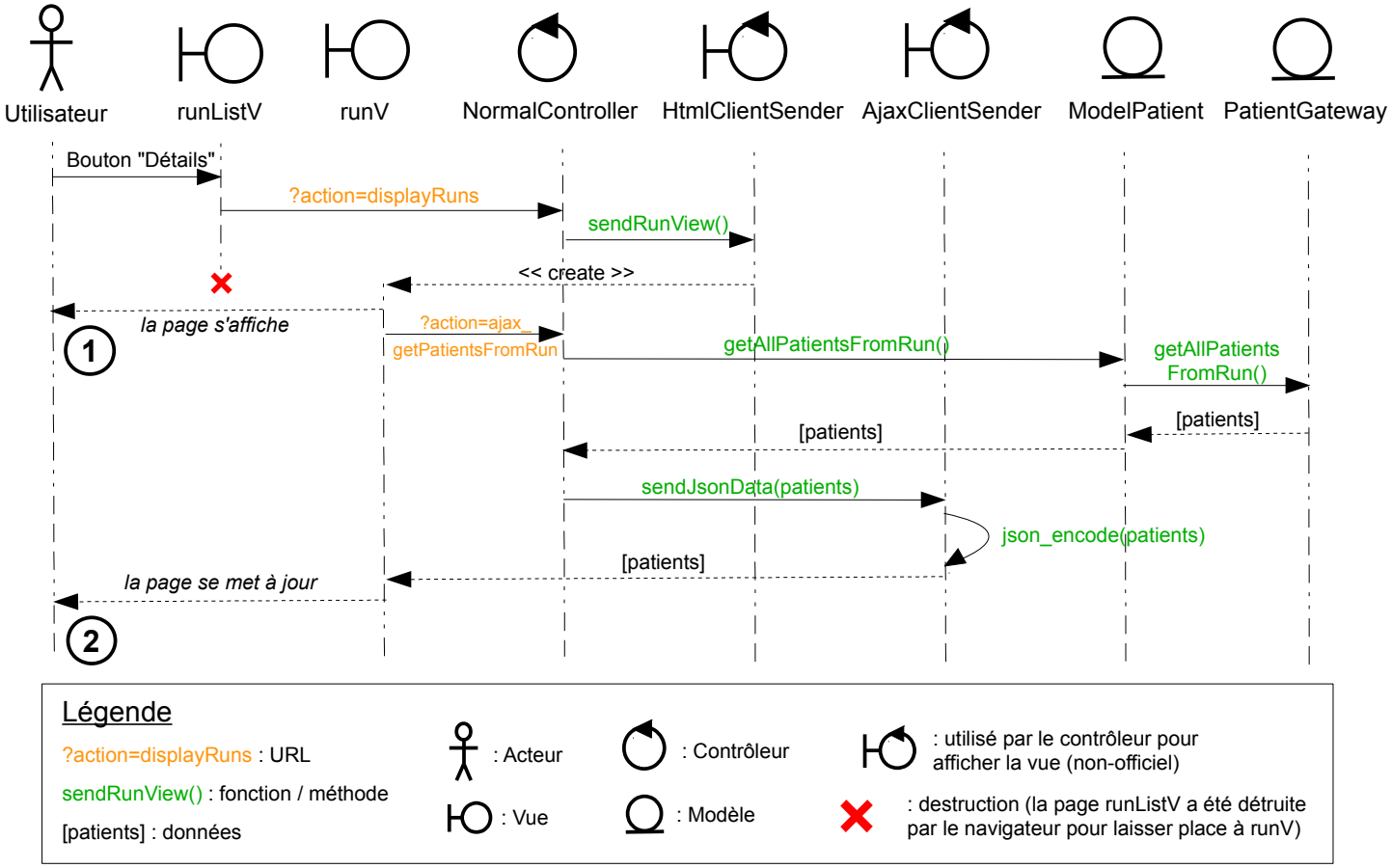


Figure 16 : Diagramme de séquence représentant les interactions entre l'utilisateur et les différentes classes du système lors de l'affichage de la page de détail d'un run

La page de détail de run (ainsi que la plupart des pages de l'interface) est affichée en deux fois. Comme nous pouvons le voir sur la capture d'écran n°1 ci-dessous (figure 17), la page HTML est apparue entièrement chargée dans le navigateur mais des informations sont encore manquantes (indiquées par les cadres bleus). L'affichage des patients vient donc dans un second temps, lors de la réception des données de la requête AJAX. La deuxième capture d'écran affiche la page finale obtenue après quelques secondes.

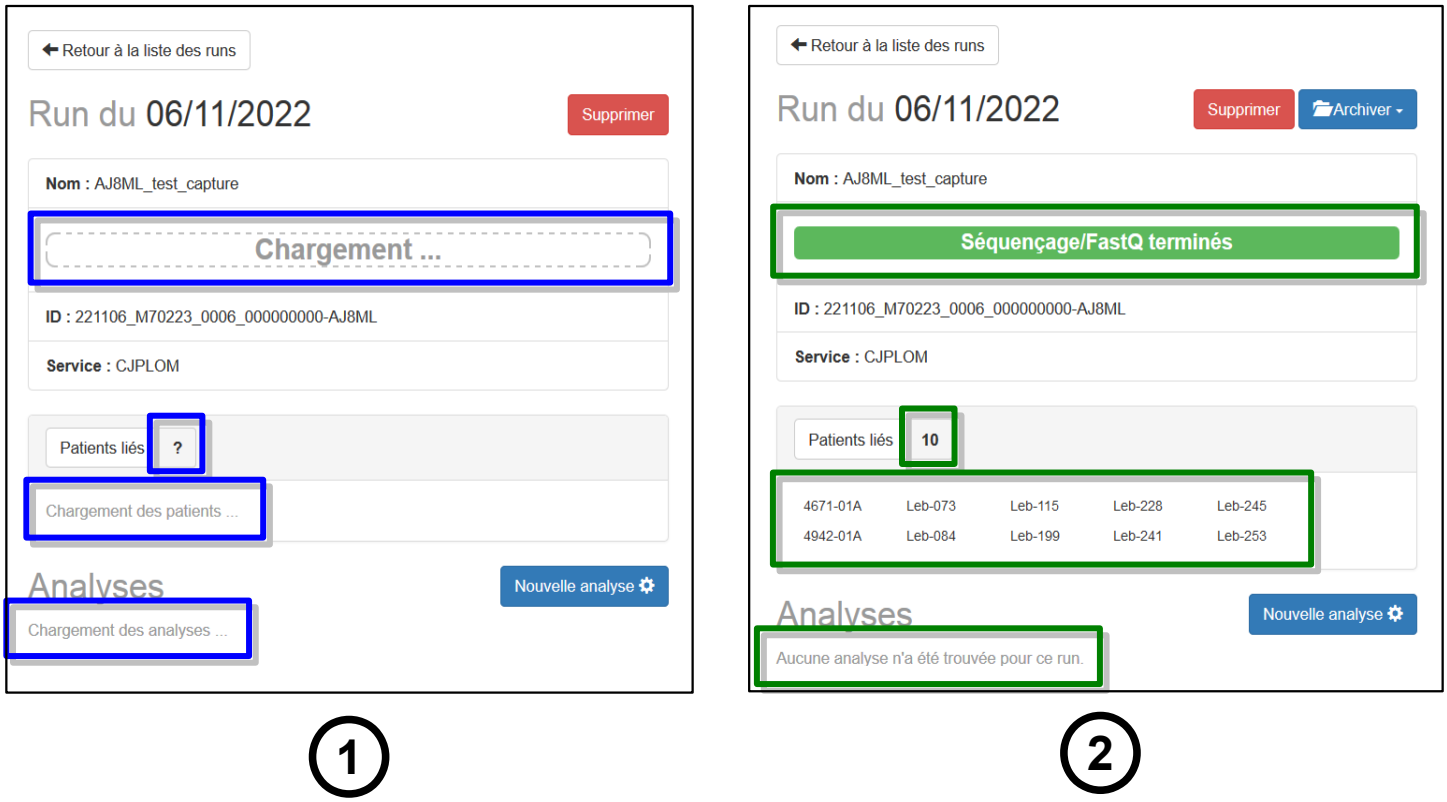


Figure 17 : Captures d'écran partielles de la page de détail d'un run montrant les deux étapes d'affichage

Ce procédé a deux intérêts :

- permettre de commencer l'envoi du squelette de la page avant d'avoir effectué tous les accès à la base de données,
- ne pas avoir à implémenter deux logiques d'affichage, une en PHP et une en Javascript. Celle en Javascript étant nécessaire pour la mise à jour de la page en temps réel, cette technique permet de s'affranchir de devoir générer la page complète en PHP.

e) Diagramme récapitulatif de l'architecture

Le diagramme suivant permet de récapituler l'architecture de l'application :

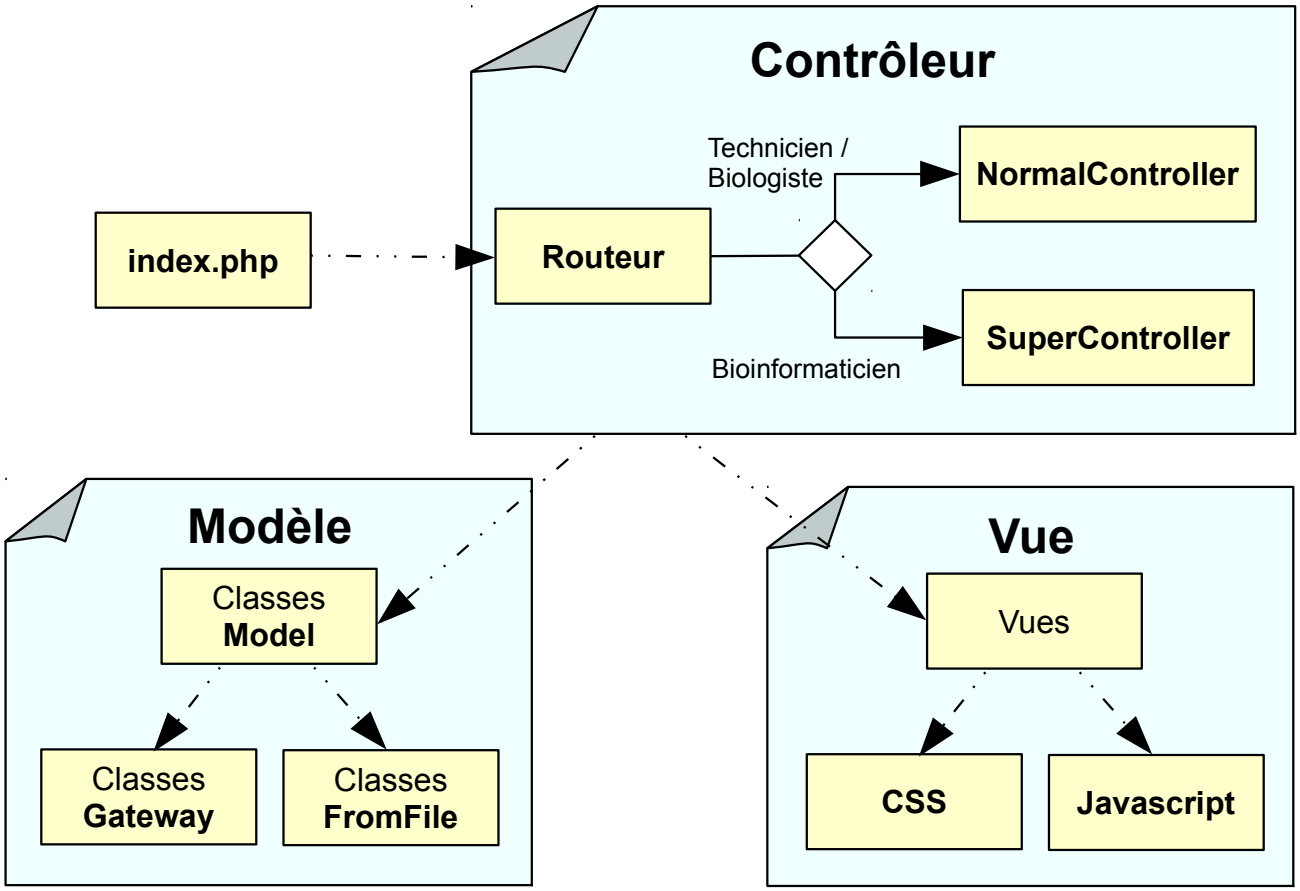


Figure 18: Diagramme répresentant l'architecture du modèle MVC appliqué à l'application

IV. Interface

Afin d'illustrer la réalisation de l'interface de l'application sans en présenter toutes les pages, je vais détailler dans cette partie deux fonctionnalités clés de l'application : le formulaire de lancement des analyses et le résultat de l'analyse complémentaire de création du fichier des variants.

1) Formulaire de lancement d'analyse

Ce formulaire, accessible depuis la page de détail d'un run, permet à l'utilisateur d'exécuter un script d'analyse en spécifiant différents paramètres :

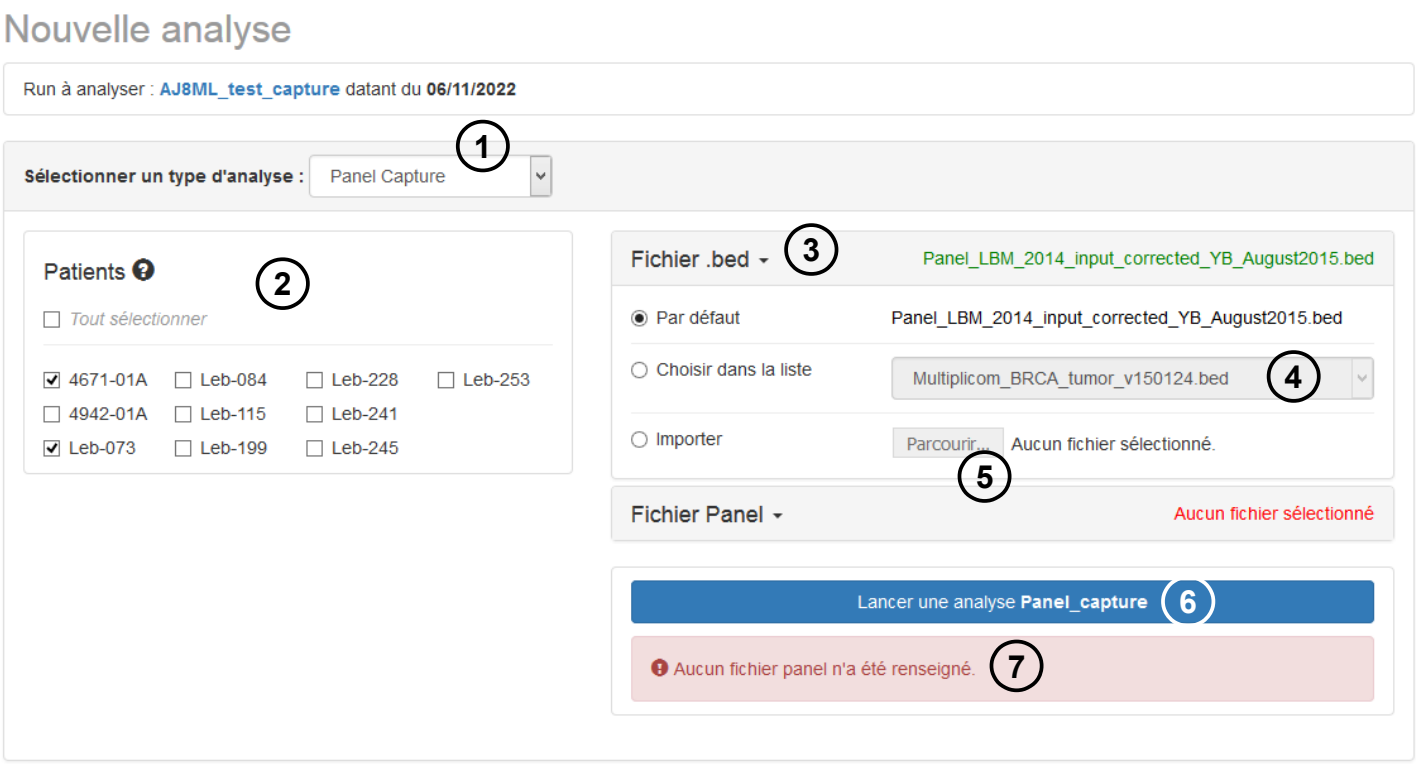


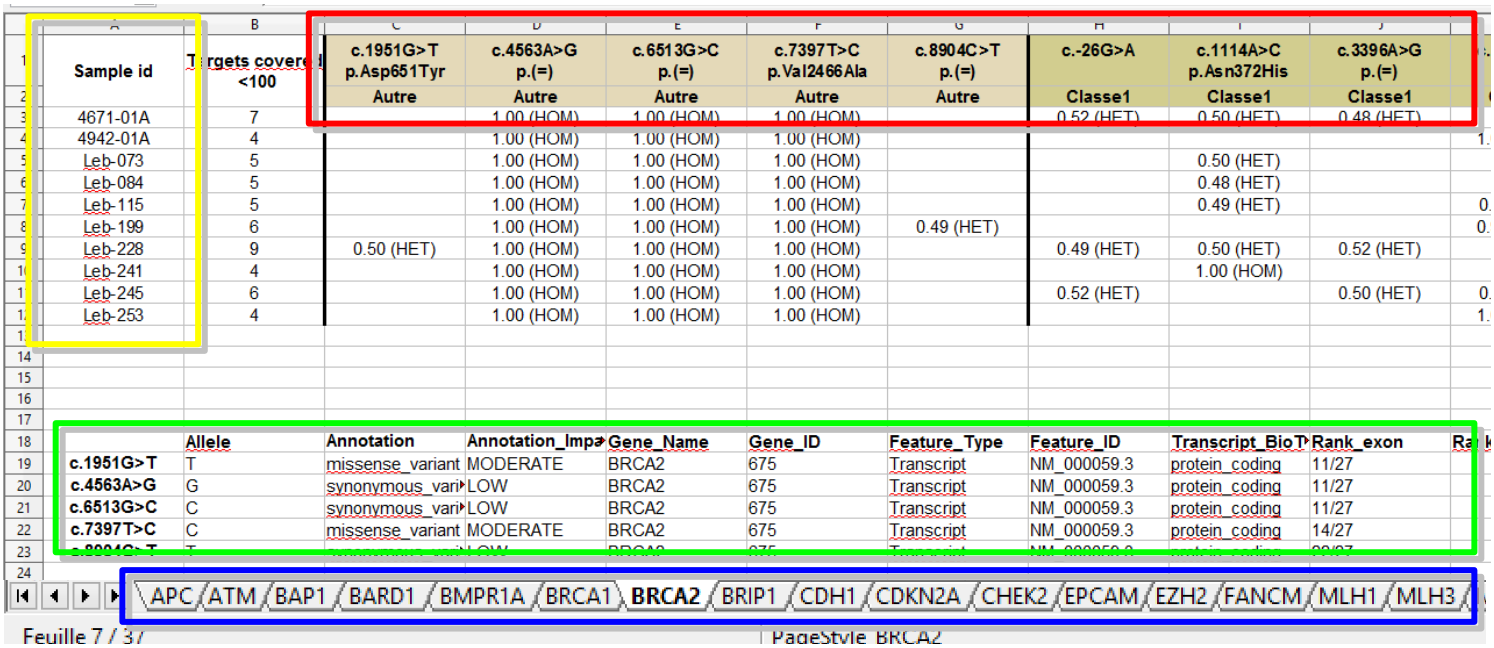
Figure 19: Capture d'écran partielle de la page de lancement d'une analyse

- le type d'analyse : tous les types sont récupérés depuis la base de données grâce à une requête AJAX puis affichés dans une liste déroulante (1). Lorsque l'utilisateur sélectionne un type, la partie inférieure de l'interface apparait.
- les patients : l'utilisateur choisit les patients à analyser dans la liste des patients du run (2). Celle-ci est également récupérée en AJAX.
- les fichiers : chaque type d'analyse requiert un ou plusieurs fichiers. Les fichiers requis pour chaque type sont spécifiés dans la base de données et le Javascript met à jour le formulaire selon le type d'analyse choisi. Chaque type de fichier est réprésenté par un selecteur de fichier (3). Les trois boutons radio permettent de choisir entre le fichier par défaut, un fichier parmi ceux présents dans le dossier du type (4 : /mnt/Analyses_Secondaires/Fichiers_input/Bed) ou un fichier importé (5). Les fichiers importés par l'utilisateur sont stockés sous un nom temporaire dans le dossier *tmp* du serveur et supprimés après une semaine.
- le nom : dans le cas des analyses complémentaires uniquement, un champ "Nom" doit être rempli. Il permet de nommer le fichier de résultat de l'analyse complémentaire. Les éventuels caractères spéciaux de ce champ sont supprimés afin de protéger l'application contre les attaques XSS (Cross-Site Scripting) qui permettent d'injecter du code Javascript dans une page web.

Lors de l'envoi du formulaire (6), une double vérification des paramètres est effectuée. Dans un premier temps en Javascript, chaque champ est vérifié et un message d'erreur (7) informe l'utilisateur en cas d'oubli ou de données invalides. Mais puisque le Javascript peut être modifié ou désactivé par le client, une seconde vérification, cette fois en PHP, est effectuée dans un second temps. En cas de validation par le PHP, la ligne de commande est créée avec les paramètres renseignés par l'utilisateur et le script est exécuté.

2) Résultat d'une analyse complémentaire

Les scripts d'analyse complémentaire peuvent être exécutés une fois que l'analyse est terminée. L'un d'eux génère le fichier Excel des variants*. Ce fichier est interprété par les biologistes et permet de savoir parmi les patients du run, quel patient porte quel variant* (rappel : un variant est une différence entre la portion du gène d'un patient et cette même portion du gène de référence. Un variant peut être à l'origine d'une mutation délétère, elle-même pouvant favoriser un cancer).



Sample id	Targets covered <100	c.1951G>T p.Asp651Tyr Autre	c.4563A>G p.(=) Autre	c.6513G>C p.(=) Autre	c.7397T>C p.Val2466Ala Autre	c.8904C>T p.(=) Autre	c.-26G>A Classe1	c.1114A>C p.Asn372His Classe1	c.3396A>G p.(=) Classe1
4671-01A	7		1.00 (HOM)	1.00 (HOM)	1.00 (HOM)		0.52 (HET)	0.50 (HET)	0.48 (HET)
4942-01A	4		1.00 (HOM)	1.00 (HOM)	1.00 (HOM)				
Leb-073	5		1.00 (HOM)	1.00 (HOM)	1.00 (HOM)			0.50 (HET)	
Leb-084	5		1.00 (HOM)	1.00 (HOM)	1.00 (HOM)			0.48 (HET)	
Leb-115	5		1.00 (HOM)	1.00 (HOM)	1.00 (HOM)			0.49 (HET)	
Leb-199	6		1.00 (HOM)	1.00 (HOM)	1.00 (HOM)	0.49 (HET)			
Leb-228	9	0.50 (HET)	1.00 (HOM)	1.00 (HOM)	1.00 (HOM)		0.49 (HET)	0.50 (HET)	0.52 (HET)
Leb-241	4		1.00 (HOM)	1.00 (HOM)	1.00 (HOM)			1.00 (HOM)	
Leb-245	6		1.00 (HOM)	1.00 (HOM)	1.00 (HOM)		0.52 (HET)		0.50 (HET)
Leb-253	4		1.00 (HOM)	1.00 (HOM)	1.00 (HOM)				

	Allele	Annotation	Annotation_Impa	Gene_Name	Gene_ID	Feature_Type	Feature_ID	Transcript_BioT	Rank_exon
c.1951G>T	T	missense_variant	MODERATE	BRCA2	675	Transcript	NM_000059.3	protein_coding	11/27
c.4563A>G	G	synonymous_vari	LOW	BRCA2	675	Transcript	NM_000059.3	protein_coding	11/27
c.6513G>C	C	synonymous_vari	LOW	BRCA2	675	Transcript	NM_000059.3	protein_coding	11/27
c.7397T>C	C	missense_variant	MODERATE	BRCA2	675	Transcript	NM_000059.3	protein_coding	14/27

APC ATM BAP1 BARD1 BMPR1A BRCA1 **BRCA2** BRIP1 CDH1 CDKN2A CHEK2 EPCAM EZH2 FANCM MLH1 MLH3

Figure 20: Exemple de fichier Excel des variants créé par un script d'analyse complémentaire

Le tableau dans la partie supérieure contient les patients dans le cadre jaune et les noms des variants dans le cadre rouge. Si la case à l'intersection est remplie, cela signifie que le patient porte ce variant. Dans la partie inférieure (cadre vert), on trouve des informations concernant chaque variant. Ces informations sont appelés des "annotations". Enfin, ce fichier Excel contient un onglet par gène, comme présenté dans le cadre bleu (ici, c'est la page du gène BRCA2 qui est affichée).

Une partie de mon stage a consisté en la réalisation d'une interface pouvant afficher les informations de ce fichier de manière plus ergonomique. J'ai donc demandé à mon maître de stage de modifier le script de création du fichier Excel afin qu'il génère en plus des fichiers JSON contenant les informations à afficher. Ces fichiers sont lus par le serveur et les objets JSON sont ensuite transferés en AJAX au navigateur du client puis affiché dans la page ci-dessous :

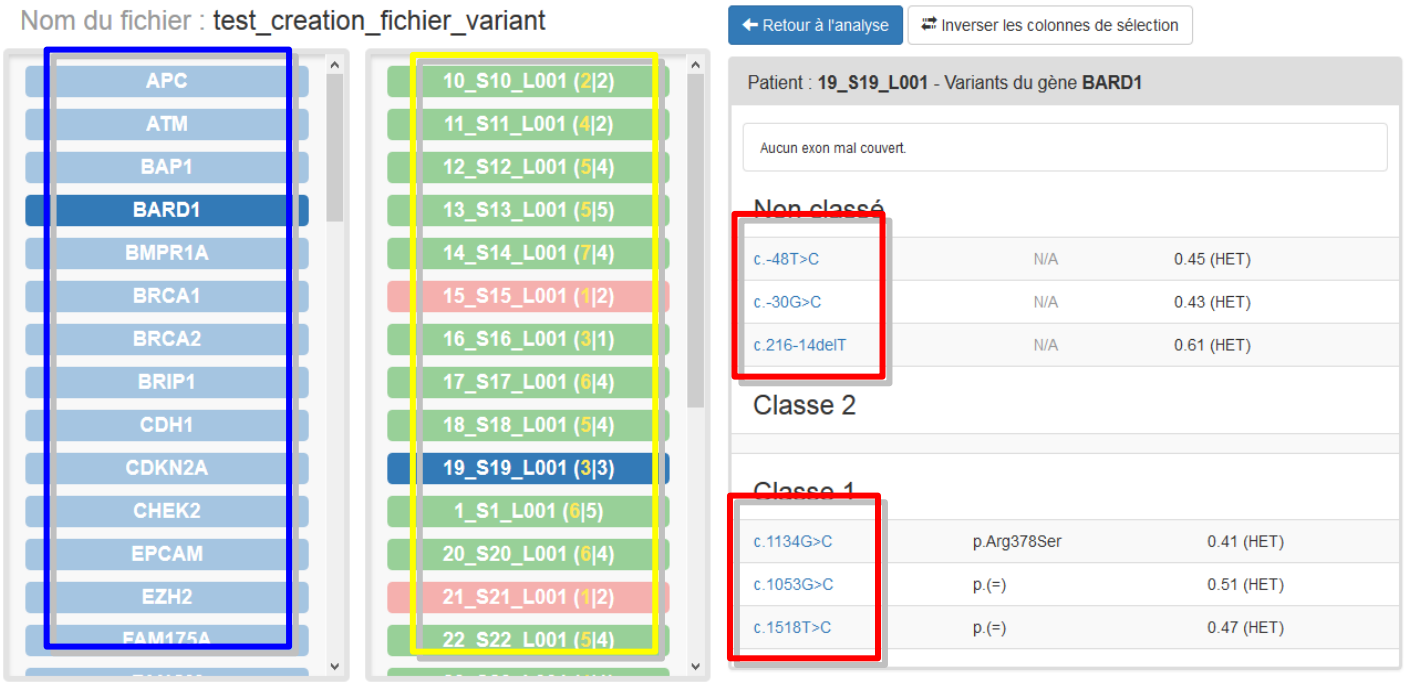


Figure 21: Page de résultat d'analyse complémentaire affichant le fichier des variants

On retrouve la sélection par gène dans la colonne de gauche, puis celle par patient dans la colonne du milieu. Lorsque l'utilisateur sélectionne un gène, le nombre de variants de ce patient pour ce gène (affiché à droite de l'ID du patient) se met à jour. Par défaut, on obtient les informations des patients pour chaque gène : on regarde ainsi un gène en particulier. A la demande des biologistes, j'ai intégré la possibilité inverse, c'est à dire obtenir les informations des gènes pour chaque patient, afin de concentrer l'affichage sur un seul patient. Quand un gène et un patient sont sélectionnés, le Javascript met à jour le tableau des variants dans la partie droite de la page. L'utilisateur peut alors afficher les annotations d'un variant en cliquant sur son nom, ce qui fera apparaître la fenêtre suivante :

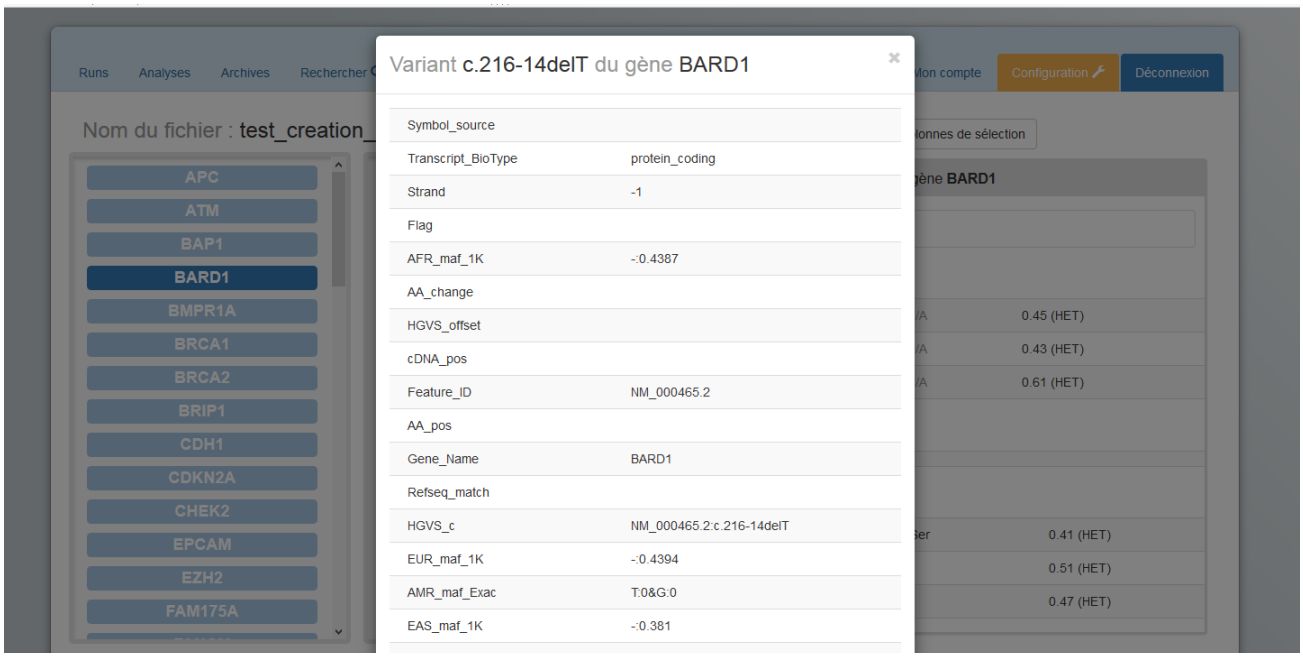


Figure 22: Fenêtre affichant le tableau des annotations d'un variant

V. Mise à jour en temps réel

Une des fonctionnalités les plus importantes de l'application est sa capacité à mettre à jour ses données en temps réel. En effet, l'interface web doit réfléter l'état réel des données présentes dans le système de fichier. J'ai donc dû développer un système de synchronisation entre l'interface, la base de données et les serveurs de fichiers. Cette mise à jour est effectuée en deux temps : la base de données est d'abord mise à jour à partir du système de fichier puis l'interface web évolue en fonction des nouvelles données de la base de données. Chacune de ces deux étapes est détaillée ci-après.

1) Mise à jour de la base de données

Comme expliqué dans la section Détails des données existantes de la partie Analyse, les données que l'on doit exploiter sont présentes sous différentes formes dans les serveurs *Donnees_Brutes* et *Analyses_Secondaires*. Pour que la base de données soit synchronisée avec le système de fichiers, il faut que l'application vérifie régulièrement le contenu des serveurs de fichiers afin d'y détecter des changements. Pour cela, il a fallu développer un script de mise à jour. Ce script est réalisé en PHP et utilise les différentes classes *Gateway* et *FromFile* afin d'accéder respectivement à la base de données pour la mettre à jour et au système de fichier pour y récupérer les données. Toutefois, ce script est indépendant de l'exécution du serveur : il n'est pas exécuté suite à une requête d'un client et ne passe donc pas par le fichier index.php. Il s'agit d'un autre fichier nommé update.php exécutable via la ligne de commande suivante :

```
php /var/www/html/script/update.php 5
```

Ce script va exécuter une boucle de parcours du système de fichier contenant les données des runs et des analyses. Il prend en argument un entier définissant le temps d'attente en seconde entre deux boucles (par défaut, 5 secondes). La fonction PHP **`sleep`** permet d'effectuer cette action afin d'éviter que le programme ne réalise une attente active qui consommerait trop de ressources.

Le script va commencer par réaliser la synchronisation entre les runs du serveur *Donnees_Brutes* et ceux contenus dans la base de données. Tout d'abord, il doit détecter l'apparation des runs nouvellement créés par le séquenceur. Pour cela, il doit surveiller la création de nouveaux dossiers de run dans les dossiers des années contenus dans *Donnees_Brutes*. Lorsqu'un nouveau dossier de run apparait, cela signifie que le séquenceur a démarré l'étape de séquençage d'un nouveau run. Une première technique de détection consistait à lister le contenu de chaque dossier d'année (grâce à la fonction PHP **`scan_dir`**), puis de vérifier un à un chaque dossier de run afin de savoir s'il n'existe pas déjà dans la base de données. C'est un procédé couteux que j'ai amélioré en ne regardant que le dossier de l'année en cours et en ne vérifiant que les dossiers de run dont la date est postérieure à la date de la dernière exécution du script.

Lorsque qu'un run a été ajouté dans la base de données, le script doit alors maintenir son état à jour. C'est la deuxième phase du script d'exécution. Il s'agit de récupérer les chemins vers les dossiers de tous les runs non-archivés et de vérifier l'existence des fichiers témoins de leur état à savoir RTAComplete.txt et CompletedJobInfo.xml (cf. le diagramme d'états-transitions d'un run, figure 9 de la partie Analyse). L'omission des runs archivés permet de limiter les accès au système de fichiers en ne mettant pas à jour

l'état des anciens runs. La mise à jour des états des runs est moins coûteuse que leur détection puisqu'il n'est plus nécessaire de lister le contenu des dossiers mais de vérifier simplement si le fichier témoin existe grâce à la fonction **file_exists** de PHP. Si le run a changé d'état, l'information est communiquée à la base de données. Le procédé est identique pour la mise à jour des états des analyses et des analyses complémentaires : le script va vérifier la présence de fichiers témoins. La différence avec les runs réside dans le fait que les analyses et analyses complémentaires sont lancées depuis l'interface web. Ainsi la base de données est informée de leur création avant même le lancement des scripts et la détection de nouvelle analyse dans le système de fichier n'est donc pas nécessaire.

Le processus qui fait fonctionner le script de mise à jour est prévu pour s'exécuter en continu. Toutefois, on doit pouvoir l'arrêter en cas de besoin. Une méthode radicale consisterait à tuer le processus grâce à la commande unix **kill**. Toutefois, cela pourrait causer des erreurs dans le cas où le script est en train de modifier les informations de la base de données et qu'il est interrompu entre deux requêtes SQL. Pour pallier ce problème, le script gère l'arrêt grâce à un signal posix personnalisé. Un signal est un évènement qui peut être envoyé à un processus. Généralement, le signal met fin au processus mais dans notre cas, le script va gérer la réception d'un signal particulier (nommé *SIGUSR1*) afin de s'arrêter à la fin de son tour de boucle.

Pour gérer ce signal, le script va utiliser la commande PHP suivante :

```
pcntl_signal(SIGUSR1, "handleSignalShouldStop");
```

Après l'exécution de cette commande, si le processus reçoit le signal *SIGUSR1*, la fonction **handleSignalShouldStop** est appelée. Cette fonction inscrit la valeur "true" dans une variable qui sera testée à chaque tour de boucle et qui fera éventuellement terminer le script.

La gestion de ce script est possible à travers l'interface web dans la page *Configuration* réservée aux bioinformaticiens. La partie de l'écran présentée ci-dessous permet ainsi de lancer, d'arrêter (grâce au signal) et de connaître l'état d'exécution du script.

Figure 23: Panneau de contrôle permettant de gérer le script de mise à jour

2) Mise à jour de l'interface

Une fois que la base de données a été mise à jour avec les données des serveurs de fichiers, c'est au tour de l'interface web d'afficher en temps réel les données ajoutées ou modifiées dans la base de

données. Cependant, ce type de fonctionnalité pose un problème dans le cas du développement web car il va à l'encontre du paradigme client-serveur. En effet, c'est d'ordinaire le client qui est à l'origine de la requête et le serveur qui répond. Or, ici, on cherche à ce que le serveur informe de lui-même le client du changement des données.

Mes recherches m'ont orienté vers plusieurs technologies permettant d'intégrer cette fonctionnalité. Toutefois, la plupart nécessitait la mise en place d'environnements complexes ou posait des problèmes de compatibilité envers certains navigateurs comme Internet Explorer (pour rappel, l'application doit fonctionner sous ce navigateur). Finalement, une solution consistait à envoyer depuis le navigateur du client une requête AJAX en boucle afin de demander régulièrement au serveur si des changements ont eu lieu. Bien que cette solution aurait été difficilement envisageable dans le cas d'un site public puisque risquant de surcharger les serveurs, elle m'a paru la plus simple à mettre en oeuvre pour une application intranet.

a) En Javascript côté client :

La mise à jour en temps réel de l'interface doit fonctionner sur 4 pages : la liste des runs, la page de détail d'un run, la liste des analyses et la page de détail d'une analyse. Bien que chacune de ces pages doive implémenter la logique de la mise à jour en temps réel, toutes ne seront pas concernées par les mêmes changements de données. Par exemple, la page listant les runs ne devra recevoir que les modifications liées aux runs mais pas celles liées aux analyses.

Pour mettre en place cette solution, j'ai commencé par créer un objet Javascript nommé RTU (Real-Time Update). Cet objet possède 15 évènements de mise à jour possible et permet à chaque page de s'abonner aux évènements qui la concernent. Le tableau ci-dessous détaille ces évènements :

Code	Nom	Déclenché par ...	Page concernée
0	onAddRunAny	Ajout d'un run	Liste des runs
1	onDeleteRunAny	Suppression d'un run	Liste des runs
2	onDeleteRun	Suppression du run d'ID spécifié	Détail d'un run
3	onChangeRunAny	Changement d'état d'un run	Liste des runs
4	onChangeRun	Changement d'état du run d'ID spécifié	Détail d'un run
5	onAddAnalyseAny	Ajout d'une analyse	Liste des analyses
6	onAddAnalyseRun	Ajout d'une analyse liée au run d'ID spécifié	Détail d'un run
7	onDeleteAnalyseAny	Suppression d'une analyse	Liste des analyses
8	onDeleteAnalyseRun	Suppression d'une analyse liée au run d'ID spécifié	Détail d'un run
9	onDeleteAnalyse	Suppression de l'analyse d'ID spécifié	Détail d'une analyse
10	onChangeAnalyseAny	Changement d'état d'une analyse	Liste des analyses
11	onChangeAnalyseRun	Changement d'état d'une analyse liée au run d'ID spécifié	Détail d'un run
12	onChangeAnalyse	Changement d'état de l'analyse d'ID spécifié	Détail d'une analyse
13	onChangePatientAnalyse	Changement d'état d'un patient lié à l'analyse d'ID spécifié	Détail d'une analyse
14	onChangeAnacompAnalyse	Changement d'état de l'analyse complémentaire lié à l'analyse d'ID spécifié	Détail d'une analyse

Figure 24: Tableau détaillant tous les évènements pouvant être déclenchés lors d'une mise à jour

L'objet RTU possède une méthode setEvent qui va permettre à une page de s'abonner à un évènement. La ligne suivante appelera la méthode **`updateRunView()`** lorsqu'un changement sera effectué sur le run d'id '160408_M70223_G07DV' dans la base de données.

```
RTU.setEvent(RTU.ev.onChangeRun,{id_run:'160408_M70223_G07DV'},updateRunView);
```

L'objet RTU enverra alors au serveur une requête AJAX en boucle (par défaut, toutes les 2 secondes) en spécifiant de quels changements la page doit être informée (grâce aux évènements). Si des changements ont eu lieu, le serveur renverra le détail des modifications afin que la vue puisse afficher les nouvelles informations.

b) En PHP côté serveur :

Afin de garder en mémoire les changements effectués dans la base de données, le système gérant la mise à jour en temps réel doit être informé manuellement de chaque modification. La classe *ModelRTU* possède 8 méthodes **`notify`** permettant de notifier chaque changement (ex : **`notifyAddRun`**, **`notifyChangeAnalyse`**). Cette classe appartient au modèle car les changements sont stockés dans une table spéciale de la base de données nommée *t_log_rtu*. Cette table contient la date, le type et le détail de chaque changement notifié effectué sur la base de données (voir le dictionnaire des ressources en annexe pour le détail des attributs). Cette table pouvant contenir beaucoup de données, les lignes concernant les changements les plus anciens sont automatiquements supprimés.

Lorsque le serveur reçoit la requête AJAX envoyé en Javascript par l'objet RTU, il va d'abord récupérer le détail des changements effectués depuis la dernière requête du client dans la table *t_log_rtu* (en passant par la classe *RTUGateway*). Ensuite, s'il y a eu des changements, il va déterminer pour chacun d'eux s'il correspond à l'un des évènements auquel s'est abonné la page. Cette étape est réalisée par la classe *RTUMask* et utilise les évènements contenus dans la requête AJAX. Enfin, si un ou plusieurs changements correspondent aux évènements, le serveur envoie au client le détail du/des changement(s) afin que l'interface se mette à jour.

Bilan technique

Au terme des dix semaines de stage, l'application fonctionne en local et a été testée sur plusieurs machines. Par manque de temps, je n'ai pas pu participer à son déploiement sur la Plateforme de Génétique Médicale. Toutefois, à l'exception de la partie "Analyse de couverture", toutes les fonctionnalités demandées au début du stage ont été réalisées. Ainsi, l'application permet actuellement à mon maître de stage de gérer les runs, de lancer et de suivre les analyses et d'afficher les résultats des analyses complémentaires.

Cependant, l'application n'est pas encore prête à être utilisée à grande échelle. Comme j'ai pu le constater lors de ma présentation de l'application aux biologistes, ceux-ci avaient de nombreuses suggestions et idées d'évolution de l'interface. Ces nouvelles fonctionnalités auraient sans doute nécessité plusieurs semaines de stage supplémentaires pour être développées, mais ce n'était pas l'objectif. Ma réalisation est fonctionnelle et pourra servir de base pour permettre, par la suite, d'étendre ses possibilités et de couvrir plus de besoins chez les biologistes et les techniciens. Ainsi, l'application va continuer d'être développée par les bioinformaticiens afin de répondre au mieux à ces besoins. Pour permettre à d'autres développeurs de reprendre l'application, celle-ci se devait d'être clairement structurée, maintenable et facilement extensible ; des contraintes que j'estime avoir respectées.

Conclusion

Ce stage a constitué pour moi une première expérience professionnelle dans le monde de l'informatique. Tout d'abord, il m'a permis de me faire une idée de l'organisation d'un laboratoire et d'une entreprise en général. J'ai été très intéressé par l'exigence de qualité du LBM et par les moyens mis en oeuvre pour satisfaire cette exigence à tous les niveaux. Bien que très techniques, les réunions hebdomadaires de service auxquelles j'ai pu assister furent très informatives sur les enjeux et les difficultés rencontrés par la recherche médicale actuelle.

Ensuite, il s'agissait pour moi de la première fois où je travaillais en équipe ou en collaboration avec des personnes ayant des compétences et des connaissances différentes des miennes dans le cadre d'un projet informatique. L'appréhension des termes techniques liés à la biologie et au milieu médical ainsi que la compréhension du travail et des besoins des biologistes ont nécessité une importante capacité d'adaptation. Pour réaliser certaines fonctionnalités de l'interface, j'ai pu discuter avec un biologiste afin de comprendre son travail et de déterminer comment mon interface pourrait répondre à ses besoins. Ce travail de communication a été très utile puisqu'il a permis de faire modifier et d'améliorer l'interface suite aux suggestions d'un des futurs utilisateurs de l'application.

Le travail réalisé conjointement avec Stéphanie, la stagiaire bioinformaticienne, a également été enrichissant même s'il n'a pas pu être terminé. Il s'agissait là aussi d'avoir une bonne communication afin d'intégrer son script Perl à mon interface en PHP, alors que nous n'avions chacun que très peu de connaissances dans le langage maîtrisé par l'autre. Ce fut également le cas vis-à-vis de mon maître de stage : c'est en discutant avec elle et en comprenant son travail que j'ai pu lui proposer des solutions à mettre en place de son côté (notamment des modifications de son script) pour faciliter le lien avec ma partie.

Enfin, ce stage m'a permis, en autonomie, d'acquérir des compétences en conception et en architecture d'application. C'est un exercice tout à fait différent de ce que j'ai pu réalisé auparavant : le fait que mon programme soit repris par d'autres développeurs m'a incité à être d'autant plus rigoureux et attentif quant aux exigences de conception. J'ai pu également développer mes connaissances techniques en approfondissant la programmation en PHP et en découvrant le langage Javascript et la technologie Ajax, le tout sur un projet réel et utile.

English summary

From April 4th until June 10th, I did an internship in the medical laboratory of the Cancer Care and Research Center "Centre Jean Perrin" in Clermont-Ferrand. This facility owns an Illumina Next-Generation DNA sequencer, a device able to determine the order of nucleotides within a DNA molecule. The staff uses this machine to detect deleterious mutations inherited by patients or present in their tumor to inform their genetic susceptibility to cancer or help the choice of their treatment.

The sequencing process is called a "run". Throughout the run, raw data are created and stored on dedicated servers, then analyzed and processed by computer programs in order to be usable by biologists.

The purpose of my internship was to create a user-friendly web application allowing biologists to manage their runs and their analyses. The application should allow them to see the state of a run in real-time, to launch an analysis through a form when the sequencing process is done, and to display the results of the analysis. It must take into account the department of the user in the facility so he can only access the runs and the analysis related to his department.

First, I had to install a web server on a computer which uses CentOS operating system. Then, I wrote the application in PHP on the server side and in HTML, CSS and Javascript on the client side. I used the CSS framework Bootstrap to design the graphical user interface. The application was built upon the Model-View-Controller architectural pattern, that allows the code to be organized around three components :

- the Model, which manages the data,
- the View, which displays the graphical interface to the client,
- the Controller, which commands the application and calls upon the model to get or modify data and the view to display it.

A MySQL database has been created to store the run and analysis data coming from the file system. I also created a script that updates the database in real-time with the data contained on the file servers.

Though I developed all the required features, the application isn't fully completed. The bioinformatics team is going to continue the development and implement multiple new features suggested by the biologists. For that purpose, I took great care in designing the application so it can be easy to understand, to maintain and to improve.

Lexique

ADN : Molécule biologique présente dans toutes les cellules et contenant toute l'information génétique permettant le développement et le fonctionnement des êtres vivants.

Analyse de couverture : Désigne une analyse complémentaire servant à déterminer la fiabilité du séquençage sur une portion de gène donnée.

Bioinformaticien : Personne formée à la résolution de problèmes scientifiques posés par la biologique grâce à l'outil informatique.

Injection SQL : Méthode visant à exploiter une faille de sécurité dans une base de données en injecter du code SQL dans une requête SQL existante afin d'en tourner la fonction.

Mutation délétère : Modification rare, accidentelle ou provoquée, de l'information génétique pouvant être la cause de l'apparition d'un cancer chez un individu.

MVC : (Voir la partie Présentation générale du modèle MVC à la page 31)

Oncogénétique : Branche de la génétique qui se rapporte aux cancers. Elle étudie les phénomènes qui engendrent la reproduction excessive des cellules, à l'origine des cancers ainsi que les facteurs génétiques de risque de contracter un cancer.

Première forme normale : Les formes normales sont un ensemble de règles visant à s'assurer qu'une base de données est correctement architecturée. La première forme normale exige que les données soient atomiques et non-répétitives afin d'éviter les redondances.

Run : Un run désigne le processus de séquençage réalisé par le séquenceur. Un dossier de run désigne l'ensemble des données brutes générées par le séquenceur pour un run donné.

Script : Programme écrit dans un langage interprété.

Séquence nucléotidique : Succession des nucléotides constituant l'ADN et contenant l'information génétique de chaque individu.

Séquençeur : Appareil capable d'automatiser l'opération de séquençage de l'ADN. Un séquenceur sert à déterminer l'ordre des bases nucléiques d'un échantillon d'ADN et à le présenter, après traitement, sous forme d'une suite de lettres, représentant des nucléotides.

Variant génétique : Chez un individu, portion du génome qui diffère du génome de référence. Synonyme de mutation.

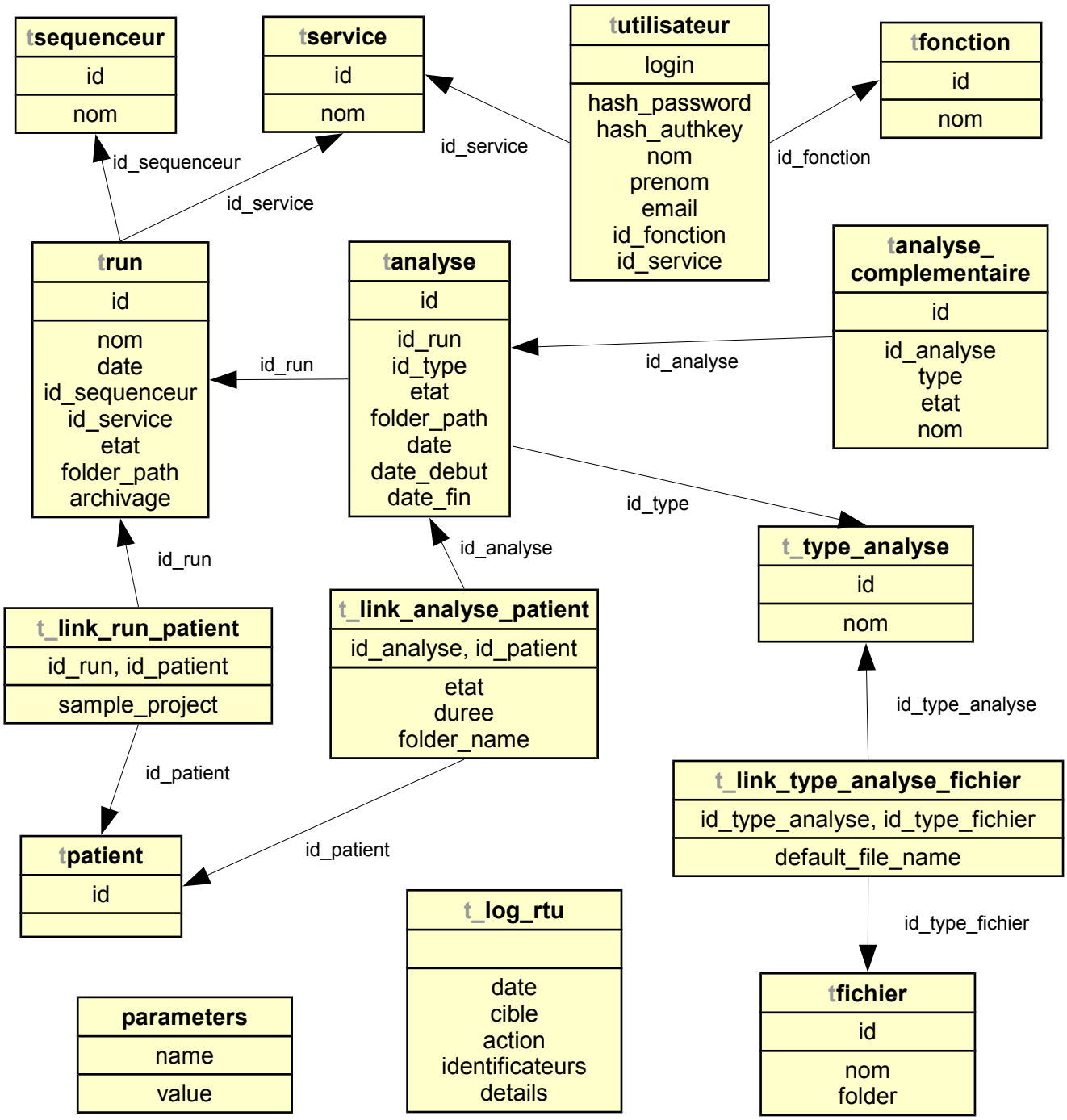
Annexes

Annexe 1 : Un fichier SampleSheet.csv

Le fichier SampleSheet.csv est présent à la racine des dossiers de runs et contient le service et les identifiants des patients d'un run.

```
[Header],,,,,,,,,
IEMFileVersion,4,,,,,,,,
Date,08/04/2016,,,,,,,,
Workflow,GenerateFASTQ,,,,,,,,
Application,FASTQ Only,,,,,,,,
Assay,Multiplicom MASTR (MID 1-48),,,,,,,,
Description,CJPLOM_multiplicom_run4_BB,,,,,,,,
Chemistry,amplicon,,,,,,,,
,,,,,,,,,
[Manifests],,,,,,,,,
,,,,,,,,,
[Reads],,,,,,,,,
151,,,,,,,,,
151,,,,,,,,,
,,,,,,,,,
[Settings],,,,,,,,,
ReverseComplement,0,,,,,,,,
Adapter,TGGAGAACAGTGACGATCGC,,,,,,,,
AdapterRead2,TGGAGATGCTGCCGAGTCTT,,,,,,,,
,,,,,,,,,
[Data],,,,,,,,,
Sample_ID,Sample_Name,Sample_Plate,Sample_Well,I7_Index_ID,index,I5_Index_ID,index2,Sample_Proje
ct,Description,GenomeFolder
13H2961,,,,p701,TAATGGAG,p501,GTCAATAC,,Homo_sapiens\UCSC\hg19\Sequence\WholeGenomeFASTA,
13H0414,,,,p702,TGATAGTG,p501,GTCAATAC,,Homo_sapiens\UCSC\hg19\Sequence\WholeGenomeFASTA,
15H1189,,,,p703,ATCAAAGG,p501,GTCAATAC,,Homo_sapiens\UCSC\hg19\Sequence\WholeGenomeFASTA,
14H3845,,,,p704,CACCGGGA,p501,GTCAATAC,,Homo_sapiens\UCSC\hg19\Sequence\WholeGenomeFASTA,
16H0600,,,,p705,CCGTAGTT,p501,GTCAATAC,,Homo_sapiens\UCSC\hg19\Sequence\WholeGenomeFASTA,
14H4035,,,,p706,TGTCGAAA,p501,GTCAATAC,,Homo_sapiens\UCSC\hg19\Sequence\WholeGenomeFASTA,
12H3718,,,,p707,ACTGCCAT,p501,GTCAATAC,,Homo_sapiens\UCSC\hg19\Sequence\WholeGenomeFASTA,
16H0515,,,,p708,TTGGCCAC,p501,GTCAATAC,,Homo_sapiens\UCSC\hg19\Sequence\WholeGenomeFASTA,
15H0844,,,,p701,TAATGGAG,p502,ACTTGAGT,,Homo_sapiens\UCSC\hg19\Sequence\WholeGenomeFASTA,
13H0219,,,,p702,TGATAGTG,p502,ACTTGAGT,,Homo_sapiens\UCSC\hg19\Sequence\WholeGenomeFASTA,
15H0670,,,,p703,ATCAAAGG,p502,ACTTGAGT,,Homo_sapiens\UCSC\hg19\Sequence\WholeGenomeFASTA,
11H2264,,,,p704,CACCGGGA,p502,ACTTGAGT,,Homo_sapiens\UCSC\hg19\Sequence\WholeGenomeFASTA,
08H2221,,,,p705,CCGTAGTT,p502,ACTTGAGT,,Homo_sapiens\UCSC\hg19\Sequence\WholeGenomeFASTA,
14H1718,,,,p706,TGTCGAAA,p502,ACTTGAGT,,Homo_sapiens\UCSC\hg19\Sequence\WholeGenomeFASTA,
13H3817,,,,p707,ACTGCCAT,p502,ACTTGAGT,,Homo_sapiens\UCSC\hg19\Sequence\WholeGenomeFASTA,
15H0169,,,,p708,TTGGCCAC,p502,ACTTGAGT,,Homo_sapiens\UCSC\hg19\Sequence\WholeGenomeFASTA,
13H2291,,,,p701,TAATGGAG,p503,ATCATGAG,,Homo_sapiens\UCSC\hg19\Sequence\WholeGenomeFASTA,
16R0791,,,,p703,ATCAAAGG,p503,ATCATGAG,,Homo_sapiens\UCSC\hg19\Sequence\WholeGenomeFASTA,
```

Annexe 2 : Modèle relationnel complet



Annexe 3 : Dictionnaire des données

Le fond jaune signifie que l'attribut est une clé primaire.
Le fond bleu signifie que l'attribut est une clé étrangère.

Table **trun** : contient l'ensemble des runs de séquençage

Attribut	Description	Type	Exemple
id	Identifiant unique du run attribué par le séquenceur au lancement du run	Chaine de caractères (100 car. maximum)	160224_M70223_0013_000000000-G06HV
nom	Nom du run constitué de la fin de l'identifiant suivi d'un nom personnalisé	Chaine de caractères (100 car. maximum)	G06HV_multiplicom_run2
id_sequenceur	Identifiant du séquenceur (référence l'attribut **id** de la table **tsequenceur**)	Chaine de caractères (50 car. maximum)	M70223
date	Date de la création du run par le séquenceur	Date au format AAAA-MM-JJ	2016-02-24
id_service	Identifiant du service pouvant accéder à ce run (référence l'attribut **id** de la table **tservice**). Peut être NULL dans le cas où le service de ce run n'a pas été spécifié.	Chaine de caractères (50 car. maximum)	CJPLOM
etat	Code de l'état du run pouvant prendre les valeurs suivantes -2 : Erreur incomplet -1 : Erreur introuvable 0 : En cours de séquençage 1 : En cours de génération des fichiers FastQ 2 : Séquençage terminé	Entier sur 1 octet (-128 à 127)	2
folder_path	Chemin du dossier de run dans le système de fichier	Chaine de caractères (256 car. maximum)	/mnt/Donnees_Brutes/2016/160224_M70223_0013_000000000-G06HV_multiplicom_run2
archivage	Code de l'état d'archivage du run 0 : Non archivé 1 : Archivé en tant que run réussi 2 : Archivé en tant que run échoué	Entier sur 1 octet (-128 à 127)	0

Table **tanalyse** : contient l'ensemble des analyses effectuées sur les runs

Attribut	Description	Type	Exemple
id	Identifiant unique de l'analyse généré aléatoirement	Entier sur 4 octet (-2.147.483.648 à 2.147.483.647)	968450880
id_type	Identifiant du type d'analyse (référence l'attribut **id** de la table **t_type_analyse**)	Chaine de caractères (50 car. maximum)	Multiplicom_paired
etat	Code de l'état de l'analyse pouvant prendre les valeurs suivantes 0 : En attente de démarrage 1 : En cours 2 : Terminée 3 : Introuvable ou inaccessible dans le système de fichier	Entier sur 1 octet (-128 à 127)	0
folder_path	Chemin du dossier de l'analyse dans le système de fichier	Chaine de caractères (256 car. maximum)	/mnt/Analyses_Secondaires/2016/160224_M70223_0013_000000000-G06HV_multiplicom_run2
id_run	Identifiant du run (référence l'attribut **id** de la table **trun**)	Chaine de caractères (100 car. maximum)	160224_M70223_0013_000000000-G06HV
date	Date du lancement de l'analyse	Date au format AAAA-MM-JJ	2016-05-25

Table **t_type_analyse** : contient l'ensemble des types d'analyses

Attribut	Description	Type	Exemple
id	Identifiant unique du type d'analyse utilisé en paramètre du script d'analyse	Chaine de caractères (50 car. maximum)	Multiplicom_paired
nom	Nom du type d'analyse	Chaine de caractères (100 car. maximum)	Multiplicom Paired

Table **tpatient** : contient les identifiants des patients des runs

Attribut	Description	Type	Exemple
id	Identifiant d'un patient	Chaine de caractères (50 car. maximum)	02H2354

Table **tanalyse_complementaire** : contient les analyses complémentaires

Attribut	Description	Type	Exemple
id	Identifiant unique de l'analyse complémentaire généré aléatoirement	Entier sur 4 octet (-2.147.483.648 à 2.147.483.647)	1711169229
id_analyse	Identifiant unique de l'analyse (référence l'attribut **id** de la table **tanalyse**)	Entier sur 4 octet (-2.147.483.648 à 2.147.483.647)	968450880
type	Type de l'analyse complémentaire (actuellement "GRAPH" ou "EXCEL")	Chaine de caractères (50 car. maximum)	GRAPH
etat	Code de l'état de l'analyse complémentaire pouvant prendre les valeurs suivantes 0 En attente de démarrage 1 En cours 2 Terminé 3 En état d'erreur 4 Introuvable ou inaccessible dans le système de fichier	Entier sur 1 octet (-128 à 127)	2
nom	Le nom d'analyse complémentaire affiché dans l'interface	Chaine de caractères (256 car. maximum)	generation_graph_DI44

Table **tfichier** : contient les types des fichiers requis en paramètres des arguments

Attribut	Description	Type	Exemple
id	Identifiant (parfois extension) du type de fichier	Chaine de caractères (20 car. maximum)	bed
nom	Nom du type de fichier à afficher aux utilisateurs	Chaine de caractères (50 car. maximum)	Fichier .bed
folder	Chemin vers le dossier qui contient tous les fichiers du type concerné utilisables par les utilisateurs	Chaine de caractères (256 car. maximum)	/mnt/Analyses_Secondaires/Fichiers_input/Bed

Table **tfonction** : contient les fonctions des utilisateurs

Attribut	Description	Type	Exemple
id	Numéro de la fonction	Entier sur 1 octet (-128 à 127)	1
nom	Intitulé de la fonction	Chaine de caractères (50 car. maximum)	Biologiste

Table **tutilisateur** : contient les informations des utilisateurs de l'application

Attribut	Description	Type	Exemple
login	Identifiant de l'utilisateur lui permettant de se connecter	Chaine de caractères (50 car. maximum)	pdupont
hash_passwd	Mot de passe hashé de l'utilisateur (peut être NULL si le mot de passe n'est pas initialisé)	Chaine de caractères (256 car. maximum)	$2y$10$6kTgB0giQmMnQ/R2y9MlV.rWn1QnOUqQrkZ0gmufCwBACaoHBI.X.
hash_authkey	Clé d'identification de la session de l'utilisateur (peut être NULL si celui-ci est déconnecté)	Chaine de caractères (256 car. maximum)	$2y$10$TmOh8Nx.8Mvsec.TBXesCuZWE21E0nxJ80rmTk2hX7O0h6j44HJ1e
nom	Nom de l'utilisateur	Chaine de caractères (50 car. maximum)	Pierre
prenom	Prénom de l'utilisateur	Chaine de caractères (50 car. maximum)	Dupont
email	E-mail de l'utilisateur	Chaine de caractères (50 car. maximum)	p.dupont@exemple.com
id_fonction	Numéro de la fonction de l'utilisateur (référence l'attribut **id** de **tfonction**)	Entier sur 1 octet (-128 à 127)	1
id_service	Identifiant du service de l'utilisateur (référence l'attribut **id** de **tservice**)	Chaine de caractères (50 car. maximum)	CJPLOM

Table **tsequenceur** : contient les identifiants et les noms des séquenceurs

Attribut	Description	Type	Exemple
id	Identifiant du séquenceur	Chaine de caractères (20 car. maximum)	M70223
nom	Nom du séquenceur à afficher aux utilisateurs	Chaine de caractères (50 car. maximum)	MiSeq CJP
folder	Description du séquenceur (attribut actuellement inutilisé, peut être NULL)	Chaine de caractères (100 car. maximum)	*NULL*

Table **tservice**: contient les services des utilisateurs

Attribut	Description	Type	Exemple
id	Identifiant du service	Chaine de caractères (50 car. maximum)	CJPLOM
nom	Intitulé du service	Chaine de caractères (50 car. maximum)	LOM Oncogénétique

Table t_link_run_patient: : table d'association entre trun et tpatient

Attribut	Description	Type	Exemple
id_run	Identifiant du run (référence l'attribut id de la table trun)	Chaine de caractères (100 car. maximum)	160224_M70223_0013_000000000-G06HV
id_patient	Identifiant du patient contenu dans le run (référence l'attribut id de tpatient)	Chaine de caractères (50 car. maximum)	02H2354
sample_project	Nom du dossier contenant les fichiers FastQ du patient (peut être NULL)	Chaine de caractères (100 car. maximum)	Capture_nimblegen

Table t_link_run_patient: : table d'association entre tanalyse et tpatient

Attribut	Description	Type	Exemple
id_analyse	Identifiant de l'analyse (référence l'attribut id de la table tanalyse)	Entier sur 4 octet (-2.147.483.648 à 2.147.483.647)	147251184
id_patient	Identifiant du patient contenu dans l'analyse (référence l'attribut id de la table tpatient)	Chaine de caractères (50 car. maximum)	02H2354
etat	Code de l'état de l'analyse pour ce patient : 0 En attente de démarrage 1 En cours 2 Terminé	Entier sur 1 octet (-128 à 127)	2
duree	Durée de l'analyse du patient (peut être NULL si l'analyse n'est pas terminée)	Chaine de caractères (50 car. maximum)	47 min 37 sec
folder_name	Nom du dossier d'analyse du patient	Chaine de caractères (100 car. maximum)	02H2354_S2_L001

Table parameters: contient les paramètres de l'application

Attribut	Description	Type	Exemple
name	Nom du paramètre	Chaine de caractères (50 car. maximum)	scriptpath_analyse
vallue	Valeur du paramètre	Chaine de caractères (200 car. maximum)	/mnt/msourdeix/Pipelines/parallel_global_pipeline.pl

Table t_link_type_analyse_fichier: : table d'association entre tfichier et t_type_analyse

Attribut	Description	Type	Exemple
id_type_analyse	Identifiant du type d'analyse (référence l'attribut id de la table t_type_analyse)	Chaine de caractères (50 car. maximum)	Multiplicom_tumorOnly
id_type_fichier	Identifiant du type de fichier requis pour cette analyse (référence l'attribut id de la table tfichier)	Chaine de caractères (20 car. maximum)	bed
default_file_name	Nom du fichier par défaut de ce type pour cette analyse (peut être NULL dans le cas où le fichier est requis mais qu'aucun fichier par défaut n'est spécifié)	Chaine de caractères (256 car. maximum)	Multiplicom_BRCA_tumor_v150124.bed

Table t_log_rtu: : contient les modifications de la base de données notifiées au système RTU (*Real-Time Update*)

Attribut	Description	Type	Exemple
date	Date de la modification	Date et heure au format AAAA-MM-JJ HH:MM:SS	2016-06-08 15:14:13
cible	Type de données modifiées (vaut 'run', 'analyse', 'anacomp' ou 'patient')	Chaine de caractères (50 car. maximum)	analyse
action	Type de modification effectuée (vaut 'change', 'add' ou 'delete'	Chaine de caractères (50 car. maximum)	change
identificateurs	Identificateurs de l'objet modifié au format JSON	Chaine de caractères (300 car. maximum)	{"id_analyse":2088672643,"id_run":"160429_M70223_0018_000000000-ANH16"}
details	Information de l'objet modifié au format JSON		{"id":"2088672643","id_type":"Panel_amplicon","id_run":"160429_M70223_0018_000000000-ANH16","etat":"0","folder_...