IRIMAS UNIVERSITÉ HAUTE-ALSACE

Institut de Recherche en Informatique, Mathématiques, Automatique et Signal

AUTOMATISATION DU DÉNOMBREMENT DES ABERRATIONS CHROMOSOMIQUES ET DES ALTÉRATIONS DES TÉLOMÈRES

Master informatique mobile et répartie
Année universitaire 2017/2018

Étudiant : Stéphane Schwenker
Maître de stage : Bruno Colicchio
Enseignant référent : Yvan Maillot

Période de stage du 11 mars au 22 juillet et du 16 au 24 août 2018

Table des matières

Introduction

Dans le cadre de ma formation master informatique mobile et répartie, j'ai accompli mon stage de fin d'études au sein de l'institut IRIMAS dans le département ASI. J'ai été motivé pour accomplir ce stage d'une part par le fait de pouvoir évoluer dans un laboratoire de recherche, et ainsi d'en apprendre et comprendre son fonctionnement, et d'autre part par la nature intéressante du sujet, qui comprend la participation à un projet plus global avec différents acteurs aussi bien du monde de la recherche que du monde professionnel, qu'est le projet Juxta Rhénum.

Le stage s'est déroulé sur une période de quatre mois. J'ai été encadré par Monsieur Alain Dieterlen, enseignant-chercheur, ainsi que plus directement pour la partie applicative par monsieur Bruno Colicchio enseignant-chercheur à IRIMAS.

Monsieur Yvan Maillot enseignant-chercheur de la faculté des sciences est technique, était mon tuteur.

Remerciements

Je remercie les membres de l'institut IRIMAS pour m'avoir accueilli dans les meilleures conditions pendant ce stage, en particulier je remercie monsieur Alain Dieterlen et monsieur Bruno Colicchio pour m'avoir accompagné dans ce travail tout en me laissant une large part d'autonomie.

Dans une plus large mesure j'adresse mes remerciements à Monsieur François Chabaux de L'IPHC (Institut Pluridisciplimaire Hubert Curien) du CNRS (Centre Nationnal de la recherche Scientifique)de Srasbourg qui a proposé à l'institut IRIMAS de participer au projet Juxta Rhrénum, permettant ainsi le financement de ce stage. Je le remercie également que l'équipe des enseignants-chercheurs de l''HyGeS (Laboratoire d'Hydrologie et de Géochimie de Strasbourg) pour leur accueil et leur soutien dans ce projet.

Un remerciement aussi de manière générale à tout le personnel qui veille au bon fonctionnement de l'Institut Universitaire de Technologie de Mulhouse.

1. Présentation de l'Institut IRIMAS

L'Institut de recherche informatique mathématique automatisme et signal, regroupe depuis cette année le laboratoire MIPS (modélisation intelligence processus et systèmes) et le laboratoire LMIA (laboratoire mathématique informatique et applications).

Sous la direction d'Olivier Haeberlé, l'institut regroupe 75 enseignants-chercheurs et 60 doctorants ainsi que des stagiaires de master et de licence qui viennent augmenter la taille de l'équipe. L'institut est extrêmement actif dans le domaine de la recherche aussi bien que dans le domaine applicatif.

Ils participent à des projets nationaux et internationaux, en particulier des projets européens.

Les domaines de recherche du laboratoire MIPS sont l'électronique, l'électrotechnique, l'automatisme, l'informatique industrielle.

En plus de la recherche l'institut participe à des projets collaboratifs avec les industries, ainsi que le soutien à la création de start-ups et à leurs développements.

L'institut se divise en trois départements :

Le département « mathématiques » :

Les thématiques de recherche portent sur l'algèbre (associative et non associative, n-aires, de Hopf), la géométrie (de contact, structures invariantes), la physique mathématique (quantification, géométrie de Poisson, théorie de la renormalisation), les systèmes dynamiques (vecteurs, équations différentielles et aux différences, systèmes discrets), les équations aux dérivées partielles (mécanique des milieux continus, problèmes inverses).

Le département « informatique » :

Les thématiques de recherche portent sur les méta-heuristiques et l'optimisation, la modélisation des données, la modélisation et reconstruction 3D, l'analyse de données visuelles, la qualité de service des réseaux sans fil, les systèmes autonomes communicants, la simulation et le calcul massivement parallèle.

Le département « automatique, signal, image (ASI) » :

Les thématiques de recherche portent sur la caractérisation de systèmes mécaniques et matériaux souples, le traitement du signal et de l'image, l'imagerie nD, l'imagerie computationnelle et non-conventionnelle, la vision industrielle, la métrologie et l'apprentissage neuronal.

2. Origine du projet Juxta Rhénum

Dans le cadre du démantèlement de la centrale de Fessenheim, un vaste projet d'observation des incidences de la centrale, de sa création jusqu'à sa fermeture, sur les différents écosystèmes, est porté par le CNRS, avec comme nom de projet Juxta-Rhénum.

Plusieurs laboratoires y participent, le laboratoire IRIMAS de l'Université de Haute Alsace, le LHyGeS .

En ce qui concerne le laboratoire IRIMAS, le projet intègre plusieurs partenaires dans un même objectif global d'automatisation des tâches et d'amélioration de la diffusion de l'information dans le cadre du travail de recherche précité, c'est-à-dire la quantification des aberrations chromosomiques, ainsi que la caractérisation de la dimension des télomères et de leurs relations avec l'apparition ou la présence de différentes pathologies.

2.1. Les acteurs :

Figure 1 : partenaires impliqués dans le projet

Cell Environement :

C’est une start-up qui a été créé par Madame Dr. Radiah M‘Kacher, cytogénéticienne, dans le but de développer ses travaux de recherche en cytogénétique, notamment son travail sur les aberrations chromosomiques ainsi que sur l'altération des chromosomes.

GHR MSA :

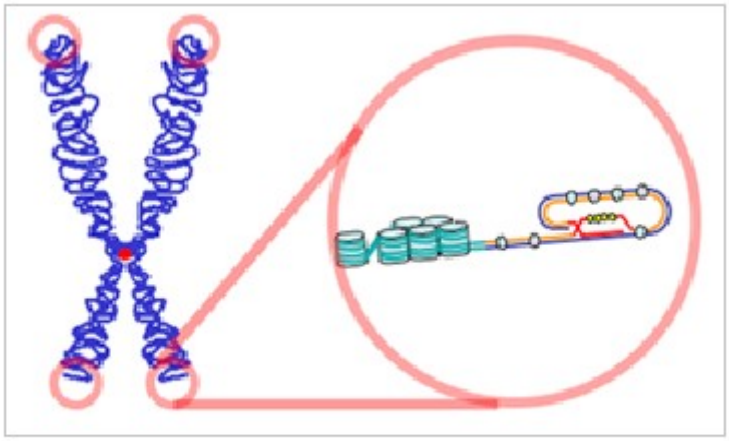
Groupement Hospitalier Régional de Mulhouse Sud Alsace, plus précisément avec le laboratoire d'analyse génétique dirigé par le Dr. Eric Jeandidier.

MetaSystems :

Société spécialisée dans la production de microscope optique dernière génération incluant l’acquisition et le traitement d'images par logiciel informatique. MetaSystems est leader dans son domaine de l'application en cytogénétique et de son traitement informatique.

2.2. L’origine du projet de Cell Environement

Le docteur M‘Kacher a déjà mis en évidence à plusieurs reprises lors de ses travaux, la relation qui existe entre certaines pathologies, et l'altération des télomères ou bien avec des aberrations chromosomiques.



Source : https://upload.wikimedia.org

Figure 2 : image d’un chromosome et du télomère de sa branche droite

Plusieurs publications de ses travaux de recherche ont d’ores et déjà été publiées, et d'autres sont en voie d'être finalisées.

Le point de départ du travail consiste en la mise en œuvre de solutions adéquates pour rendre fluorescentes les parties que le microscope va prendre en image (figure 3). C’est-à-dire que la solution doit marquer spécifiquement certaines zones des chromosomes avec une coloration afin de pouvoir plus facilement détecter, et délimiter ses contours, permettant ainsi la quantification des informations

biologiques qui intéressent les cytogénéticiens (la longueur des télomères, et la présence ou l'absence d'aberrations chromosomiques).

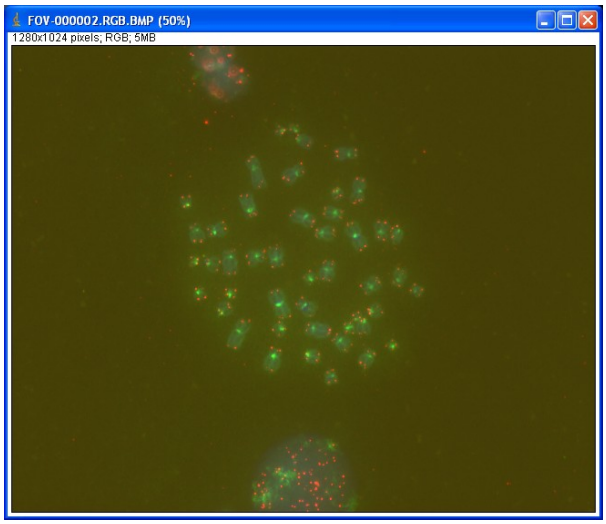


Figure 3 : Noyau cellulaire en métaphase

Ensuite le logiciel d'acquisition et de traitement d'images, développé par MetaSystemes va aller quantifier ces zones en deux étapes : une période d'apprentissage supervisé, c'est-à-dire que le programme doit apprendre à reconnaître ce que l'on recherche. Pour cela on fournit au programme des échantillons ainsi que l'information sur ce que l'on recherche. En effet le logiciel d'acquisition essaye de trouver des objets dans les images, les présente à l'opérateur qui indique si ce sont des objets cibles.

Une seconde étape de test avec des images qui ne font pas partie de l'ensemble de départ, le programme saura reconnaître alors si ce sont des objets d'intérêt ou non. C'est-à-dire que le programme d'acquisition peut ensuite automatiquement parcourir l'échantillon et enregistrer de très nombreux objets

La qualité des zones de mise en fluorescence est donc primordiale. Ainsi pour pouvoir comparer différents échantillons il apparaît évident que la solution de traitement doit être la même. Il faut donc qu'il y est l'établissement de protocoles standardisés.

La qualité des zones de mise en fluorescence est donc primordiale.

Le logiciel permet également d'effectuer des générations de mesures et de traitements d'images sur les objets détectés. Il est alors possible d'exporter sous forme de fichier texte les résultats (figure 4 ci-dessous).

Exemple de fichier de sortie pour un échantillon

ici nombre d'information : 200 000

MetaCyte Exported Cell Data V3.8.6 08/01/17 12:52:50

File P3-PL2.TXT N Cells : 10132 Scanned : 08/01/17 12:48:44

ColNo	ChanNo	FeatNo	Param1	Param2	Param3	FeatDesc
1	1	1				Total Area within Contour in µm²
2	1	29				Capture Integration Time (seconds)
3	1	2				Circumference of Contour in µm
4	1	3				Irregularity of Contour (0..1)
5	1	4				Roundness = 4 Pi Area / Circumference² (0..1)
6	1	5				Maximum Relative Concavity Depth (0..1)
7	1	6				Relative Area of Deepest Concavity (0..1)
8	1	7				Total Relative Concavity Area (0..1)
9	1	8				Integrated Intensity
10	1	9				Mean Intensity
11	1	10				Standard Deviation of Intensity
12	1	111				Horizontal Object Position on the Slide (µm)
13	1	112				Vertical Object Position on the Slide (µm)
14	2	8				Integrated Intensity
15	2	13	20.000			Maximum Intensity, Absolute Spot Area X/100 µm²
16	2	14	20.000			Minimum Intensity, Absolute Spot Area X/100 µm²
17	2	10				Standard Deviation of Intensity
18	2	61	0.000	300.000	0.000	Number of Objects at X% Intensity (Maximum Gain Y%, Upper Thr.
19	2	66				Mean of Object Intensity
20	2	29				Capture Integration Time (seconds)
21	2	21	14.000	9.000	1000.000	Ratio Value of Feature Variables X and Y (Scale Factor Z = 1.0

Figure 4 : exemple de fichier texte de sortie pour un échantillon

Une fois ce travail réalisé, pour pouvoir travailler sur des volumes de données plus importants et pouvoir en interpréter les résultats, l'utilisation d'un langage informatique analytique devient essentielle, ce langage est R. R fait partie des langages informatiques à vocation scientifique les plus utilisés au monde. Ses commandes de création de graphique ne nécessitent pas de connaissance de programmation poussée. C'est un langage très puissant.

Le travail effectué par l'équipe ASI a permis d'aller plus loin dans l'analyse des données et a largement contribué à aider Madame M'Kacher dans ses travaux de recherche et de réalisation de publications.

Il y a eu cinq publications sur le sujet depuis 2014 dont voici un exemple qui se corrèle parfaitement avec le projet Juxta Rhénum :

Telomere shortening: a new prognostic factor for cardiovascular disease post-radiation exposure

R M'kacher et al.

Radiation protection dosimetry 164 (1-2), 134-137, 2014

En croisant des données complémentaires aux mesures génétiques (ici des données sur les maladies cardio-vasculaires), des hypothèses ont pu être vérifiées sur un grand nombre de mesures.

2.3. Le besoin :

Le grand volume de données et de métadonnées constitue un ensemble hétérogène difficile à partager.

Avoir un site en ligne permettant de charger des fichiers textes (figure 4) de résultats d'analyses microscopiques d'échantillons de cellules, provenant de prélèvements sanguins humains permettrait de partager des données et de les comparer à l'avenir avec de nouvelle données.

Il faudra donc mettre en place une architecture adaptée à la construction d'une base de données respectant un certain schéma (métadonnées) et étant souple quant à des éventuelles modifications.

Il faudra donc mettre en place une architecture adaptée à la construction d'une base de données respectant un certain schéma (métadonnées) et étant souple quant à des éventuelles modifications de ces derniers.

Les interfaces doivent être simples et intuitives, éventuellement la mise en place d'un chat box peut être un plus.

Une interface de requête pour accéder aux données recherchées pour téléchargement, afin de pouvoir les traiter en locale. Les résultats de ces traitements pourront être rechargés pour ne pas les rééditer sans cesse. Les résultats seront alors disponibles en ligne.

Une partie sera consacrée à la gestion des droits des utilisateurs, lecture, écriture, chargement, téléchargement….

Enfin il sera étudié l'éventuelle possibilité d'installer un environnement de développement sur le serveur pour pouvoir utiliser directement R par le biais d'interface ou la possibilité d'appliquer des traitements via des menus.

2.4. Déroulement du projet

2.4.1. Planning prévisionnel :

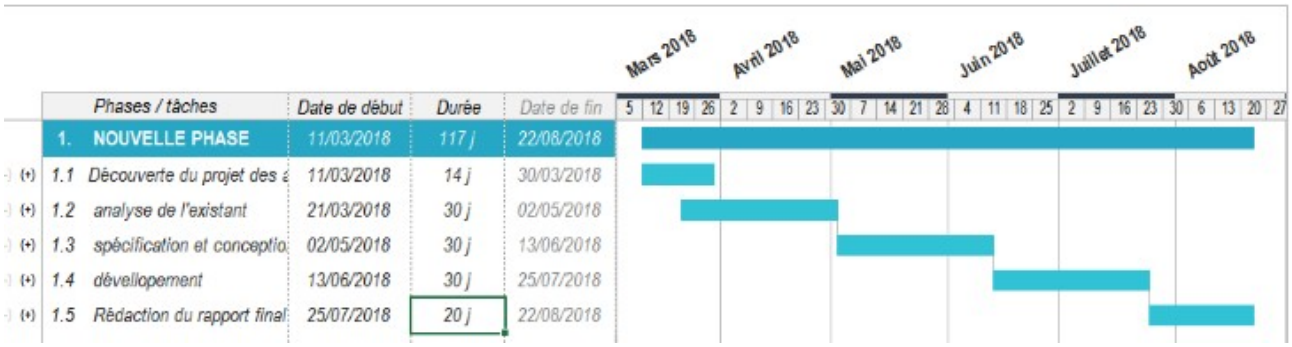


Figure 5 : diagramme de Gantt, planning prévisionnel

Il est assez difficile de prévoir un planning prévisionnel (figure 5) lorsque de nouvelles notions sont

à intégrer, étant donné que le périmètre de travail n'est pas fini, et que les courbes d'apprentissages ne sont pas connues au départ. Malgré tout il est essentiel de fixer des jalons avec l'obligation de passer à l'étape suivante à partir du travail réalisé précédemment quitte à y revenir si besoin.
Nous nous sommes donc astreint à ce planning en connaissance de cause, en sachant dès le départ que dans le temps imparti, inclure la mise en production était utopique, en particulier en prenant en compte que l'IUT est fermé de 22 juillet aux 16 août, ainsi que le fait que je sois seul pour la partie développement du projet.

2.4.2. Déroulement global

Le projet a été rythmé par des réunions d'échanges des résultats de l'avancée des différents projets de chaque master tous les mois à LHyGeS (Laboratoire d'hydrologie et de Géologie de Strasbourg). Ces échanges furent très intéressants sur le plan des connaissances générales ainsi que sur la manière d'approcher des problématiques par des étudiants de formations diverses. Cela a nécessité la préparation de présentations conçues pour être la plus compréhensible de tous, tout en restant technique, afin que chacun reparte enrichi du savoir des autres.

2.4.3. Déroulement local

Une première phase de découverte et d'analyse de ce qui était déjà fait, notamment des différents environnements de développements pour le langage R. S'en est suivi l'étude de Shiny qui est un service proposé par Microsoft permettant d'aller plus loin avec le logiciel R. Le but avec Shiny est de créer des interfaces de gestion de données avec un très bon rendu visuel, ainsi que la possibilité de se servir de Shiny comme serveur, et d'utiliser les services de base de données fournis. Il y a deux inconvénients majeurs, le premier est que lorsque vous voulez passer à un système sur le réseau et que vous voulez stocker des données, la solution devient payante avec un coup très élevé, le deuxième inconvénient majeur est de perdre la main sur les données, car on ne sait pas vraiment où elles sont stockées.

Au cours des réunions avec Monsieur Colicchio, le choix de construire une base de données est apparu comme une première pierre essentielle avant d'aller plus loin.

Une étude des différents SGBD (système de gestion de base de données) fut un passage obligé.
A l'issue de ce travail il en ressorti que Mongodb est une solution adaptée à notre problème. Mongodb est un SGBD noSql (cela veut exprimer que le système ne fonctionne pas sous le modèle relationnel avec des requête de type sql (structured query langage), bien que dans la logique de construction on y retrouve les grands axes d'un SGBD.
L'avantage de Mongodb est sa souplesse d'évolution des schémas. En particulier le fait que tout est écrit en fichier de type JSON (JavaScript Objet Notation : format que l'on retrouve dans le reste de l'application) facilite l'implémentation de l'application.
Ensuite il a fallu choisir le langage et le type de serveur, classiquement nous aurions pu partir sur un serveur apache mais les qualités annoncées par node.js ont fait que nous nous sommes dirigés vers lui :

-rapidité des requêtes
-interactivité

Lorsque nous nous sommes rendu compte qu'il était possible de se libérer de PHP, nous nous sommes directement orientés vers cette solution que j'avais d'ailleurs vue en cour avec Monsieur Stéphane Rivère ("Node.Js + JavaScript + HTML + CSS").

HTML : Hypertex Markup Langage
Langage balisé pour le web

CSS : Cacading style sheets
Feuille de style pour la mise en forme du code HTML

Ensuite je suis passé à la rédaction d'un premier dossier de spécification et de conception, bien que ce travail ne me fût pas imposé, j'en avais besoin pour structurer le projet.

Il était donc temps de rentrer dans la phase de développement en structurant le projet avec ses vues ses modèles, et ses contrôleurs, ainsi qu'un fichier pour le serveur et un autre pour le routeur.
Après avoir développé les parties fonctionnelles du projet et d'avoir discuté avec un collègue, j'ai souhaité rajouter une couche d'Angular qui est un framework Javascript développé par Google. Suite à une période de recherche j'ai découvert le projet Angular-cli qui regroupe tout ce dont on avait besoin :

-un compilateur pour le TypeScript (TypeScript est un langage qui a été créé pour palier au fait que JavaScript est un langage interprété et non pas compilé, TypeScript facilite donc le développement et la solidité du code).
-la possibilité de créer ses propres éléments HTML sous la forme d‘un composant (le terme de composant sera développé plus en détail dans la suite du rapport).
-l‘auto génération de test unitaire.
-la création d'une application dynamique dans une unique page HTML.

Je pensais qu'il allait être simple de migrer vers cette solution. En particulier je pensais que le code du back end de Node.js devait rester le même.

Ce fut une erreur de jugement car le changement de logique et l'apprentissage d'un nouveau langage furent difficiles dans le peu de temps que j'avais. Malgré tous, nous ne regrettons pas ce choix car ce type d'architecture rend vraiment cohérente l'application. Une fois la prise en main effectuée, un développement dans de bonnes conditions est facilité.

L'application étant chargée en entier dans le navigateur lors de son appelle, elle peut continuer de fonctionner sans connexion.

Les intérêts de ce type de projet sont la cohérence des langages, que du JSON et du JavaScript, ainsi que la facilité de développer un module par une tierce personne et l'intégrer à l'application ensuite.

Plusieurs mois sont encore nécessaires pour atteindre la finalité du projet en production, mais la faisabilité et les bases pour parvenir à la mise en production ont bien été réalisées.

3. Choix des technologies

3.1. Choix du système d'exploitation :

Linux, Ubuntu version 16.04.LTS dans une machine virtuelle.

Nous avons choisi de développer le projet dans une machine virtuelle pour pouvoir le déplacer facilement avec une clé USB et continuer à développer sur une autre machine sans refaire toute la configuration et l'installation des environnements.

Le choix de Linux :

- c'est un logiciel libre qui fonctionne très bien avec les différents IDE (environnement de développement), la version a été choisie pour sa grande stabilité.

-pour sa souplesse de travail en ligne de commande.

-pour son caractère non intrusif.

Il y a de nombreuses autres raisons, je n'ai listé ici que celles qui nous concernent directement pour ce projet.

3.2. Choix de la base de données :

3.2.1. Type de données :

Les données en entrées sont des fichiers de trois types, soit texte soit Excel, éventuellement après transformation, des fichiers du type JSON ou XML. Ces fichiers sont liés à des métadonnées qui sont pour l'heure incluent pour partie dans le nom même du fichier, pour d'autres dans les premières lignes du fichier texte, enfin dans des fichiers Excel joints. Il faudra donc les lire et en extraire les informations utiles.

3.2.2. Évolution du schéma des données :

Les bases de données relationnelles sont assez rigides en matière d'évolution. Un modèle relationnel se réfléchit et se pose avant toute insertion de données car la modification d'une table liée à une autre ne peut se faire que si elles sont vides.

Ainsi la suppression d'une table rend le système obsolète du fait qu'elle est attachée à d'autres tables par le biais de liens.

La caractéristique principale des bases de données relationnelles est d'éviter la redondance et ainsi minimiser la place de stockage. Cette problématique qui était primordiale est devenue de second ordre aux vues de l'augmentation exponentielle des capacités de stockages et de calculs.

Les espaces de stockages ont vraiment énormément augmentés, ainsi la redondance passe en deuxième plan par rapport à la demande de vitesse d'exécution, la montée en charge, et la possibilité d'évolution des données.

Malgré tout le modèle relationnel est toujours celui qui est le plus utilisé.

Le type NoSql MongoDb (avec schéma) correspond bien à notre problématique.

Il y a aussi le type NoSql colonne, pas de relation, les liens se font uniquement par des relations clefs valeurs.

Les valeurs peuvent être de toutes sortes, comme un tableau de clefs, par exemple une salle de classe pourrait avoir un tableau d'identifiant élève, un identifiant couleur, ect …

3.3. Les langages de développement :

Javascript, en particulier NodeJs, qui est écrit en Javascript, s'accorde très bien avec MongoDb. NodeJs est un langage qui s'exécute côté serveur (interactivité), ainsi le client n'a pas besoin de faire de requête comme en PHP (côté client) pour voir sa page être modifiée, c'est exactement le principe de service de chat… .

Nous avons donc des pages HTMLs qui sont modifiées dynamiquement. On peut remarquer un gain de temps car les pages ne sont pas rechargées entièrement dans le cas d'une modification partielle.

Nous utiliserons Express (bibliothèque JavaScript) pour nous connecter à la base, Bluebird (bibliothèque JavaScript) pour utiliser ses fonctionnalités avantageuses avec ses promesses (programmation asynchrone), bien plus simple que la gestion des callbacks. Le principe des callbacks est de passer une ou plusieurs fonctions de rappel en paramètre de la fonction appelante. Les promesses ont nativement deux fonctions de rappels, « resolve » et « reject». En clair si vous tentez d'exécuter du code asynchrone, lorsque tout va bien le callback « resolve » est appelé, « reject » dans le cas inverse (il faut implémenter l'intérieur de ces callbacks en fonction du besoin).Nous utiliserons Mongoose pour la persistance des données en base.

Pour pouvoir utiliser un compilateur pour créer le code JavaScript nous avons utilisé TypeScript. Après compilation nous avons les fichiers exécutables JavaScript. TypeScript permet aussi d'avoir une

représentation sous forme de Classe ce qui facilite sa compréhension, ainsi qu'une meilleure gestion de la portée des variables, en utilisant "let" la portée est réduite au bloc courant, alors que précédemment avec "var" la portée minimale était la fonction.

Le compilateur se charge de la gestion des conflits de nommage. En ce qui concerne le front-end nous allons utiliser Angular-Cli qui est un Framework JavaScript.

Ce type d'application a comme acronyme MEAN soit : MongoDb Express Angular Node.js. Un premier intérêt de ce type d'application est la constance des formats, que ce soit en base ou en requête c'est du JSON, quant au langage c'est du JavaScript ou des bibliothèques JavaScript.

Enfin il existe un moteur Angular-Cli qui regroupe tous ces outils avec en plus la création du composant sous forme de balise HTML insérable comme on le désire dans l'application. Cette partie (le composant) sera développée plus loin dans le rapport.

3.4. Outils de développement :

3.4.1. Visual Studio

Visual Studio est parfait pour ce type de projet, nous pouvons inclure la console dans l'IDE et il est très efficace dans la complétion automatique ainsi que dans la correction et la recherche d'erreur automatique.

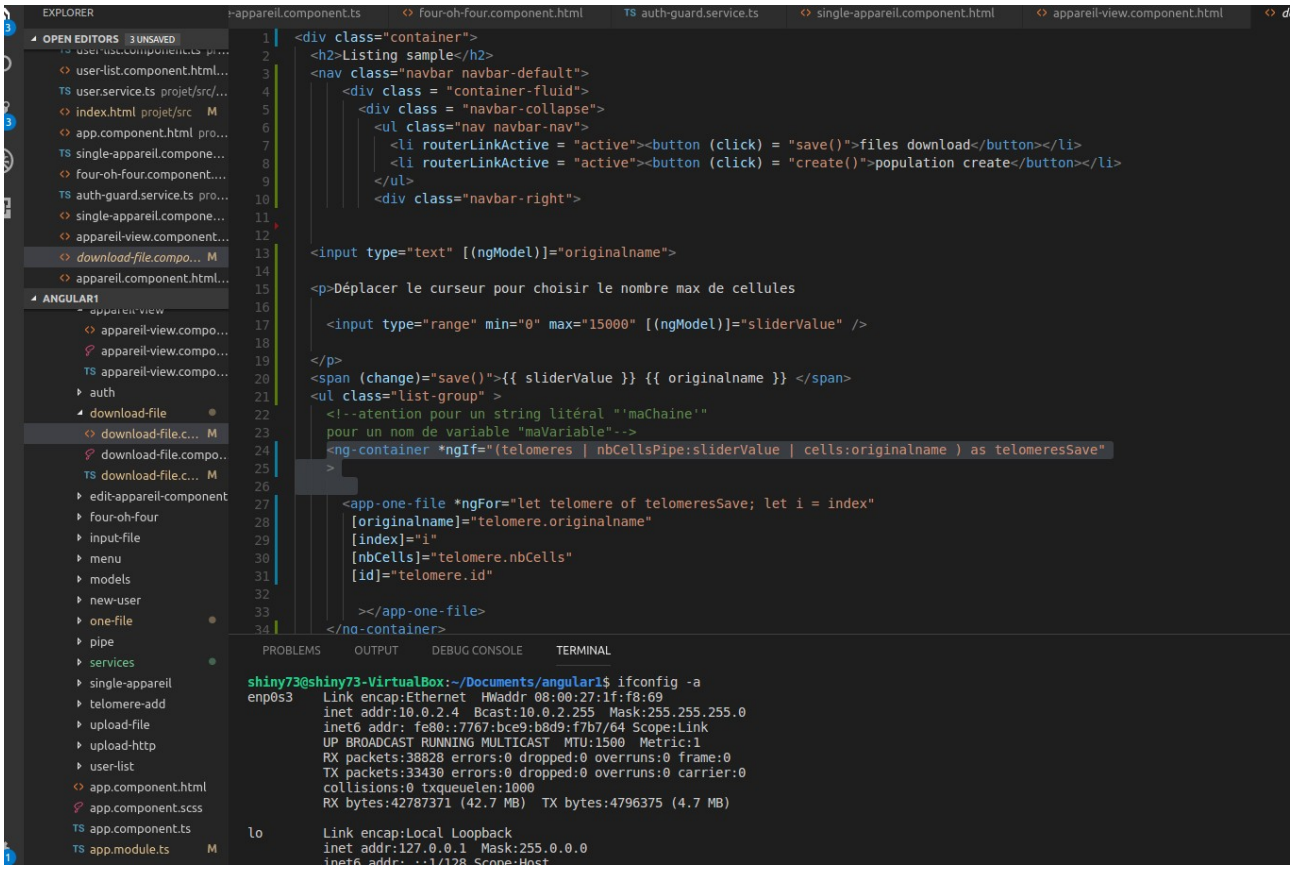


Figure 6 : capture d'écran de Visual Studio avec console intégrée

On voit l'arborescence du projet (figure 6) sur la gauche, sur la droite la page, le code en cours, en bas à droite la console. Pour insérer une console, aller dans le menu à l'onglet « view », pour faire apparaître le menu placer la flèche dans la barre noire en haut, puis descendre jusqu'à "integrate terminal", ensuite en haut de la console, à droite, une touche avec un +, qui vous permet d'insérer autant de console que de besoin (pour la base de donnée, pour le serveur, pour entrer des commandes en ligne de commande).

3.4.2. Robo3t

Robot3t est un logiciel pour effectuer manuellement les requêtes CRUDs (Create, Read, Update, Delete) il est libre et possède une bonne documentation en ligne.

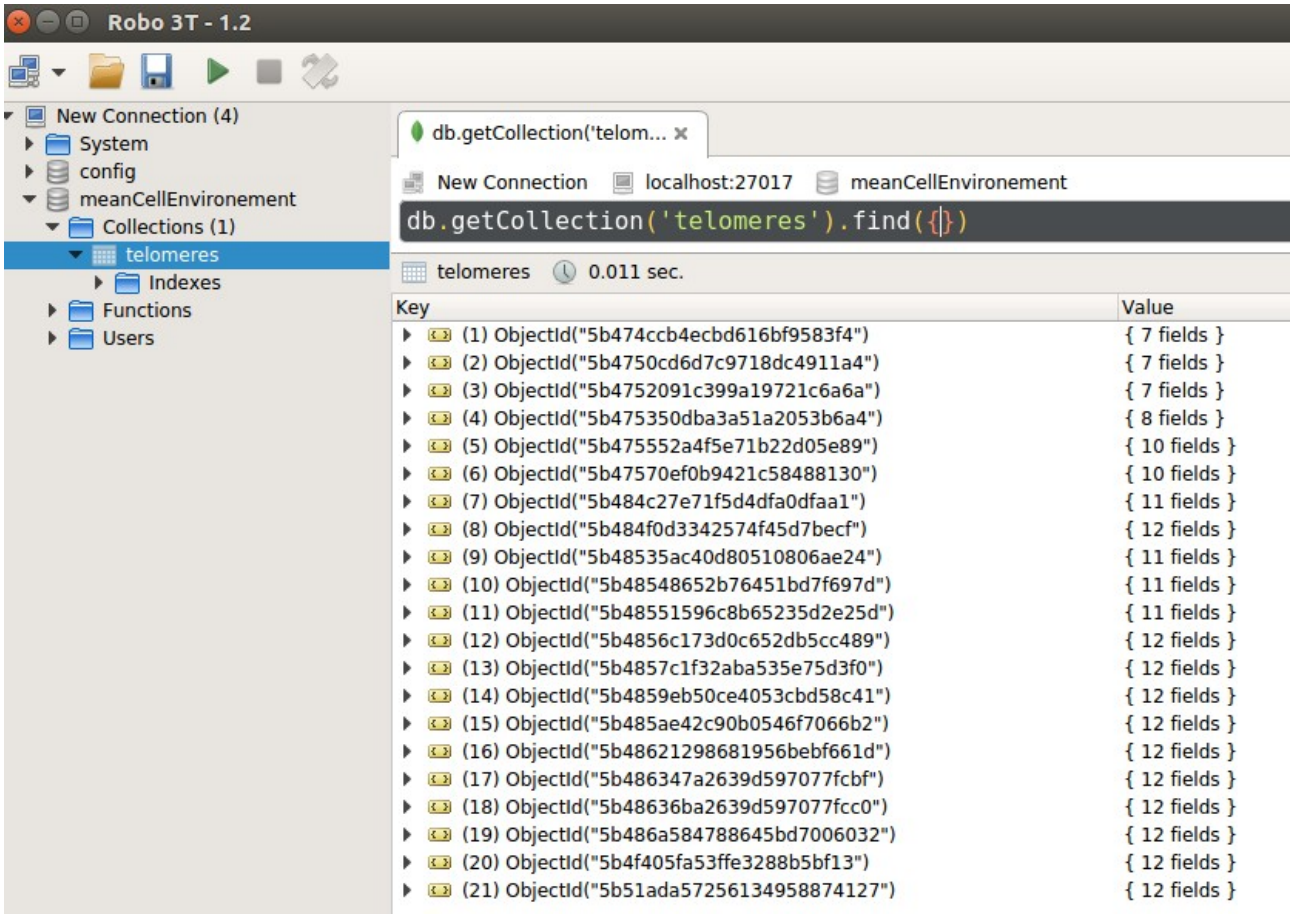


Figure 7 : capture d'écran Robo 3T avec affichage d'une requête « get »

Sur cette capture (Figure 7) nous voyons en haut à gauche la base de données connectée "meanCellEnivronnement" à l'intérieur de celle-ci il y a des collections dans lesquelles sont insérées des documents respectant le schéma des données.

Dans la barre du haut à droite, on y voit une requête et en dessous son résultat.

Pour connecter la base sur le port d'écoute 27017 tapez dans la console cette commande :

"Mongos-dbpath cheminDuDossierDeStcockageData", il faut au préalable avoir créé un dossier "data" à la racine du projet.

3.4.3 Gituhb

Github est un outil de versioning en ligne pour héberger le code source, il permet la sauvegarde des différentes versions du code source, ainsi que des commentaires associés à la version.

En pratique à chaque nouvelle fonctionnalité fonctionnelle on y crée une nouvelle version avec une courte description.

3.5. L’hébergement, plusieurs choix :

-Un serveur dédié et propriétaire, avec l’avantage d’avoir la main sur les données mais l’inconvénient majeur de devoir gérer les problèmes de sécurité, de panne, de duplication des données.

-Un espace disque loué sur un serveur qui propose ses services dédiés ou partagés, vous louez un espace disque avec un environnement de déploiement pour votre application incluant des services plus ou moins étendus comme par exemple un service de mailing.

-le cloud est de plus en plus utilisé, proposant d’ailleurs, de manière native des solutions de duplication de données permettant ainsi de garantir le maintien de la disponibilité des ressources, en clair si un serveur est inaccessible, l’application ira chercher sur un autre serveur les données qui auront été auparavant dupliquées.

Dans le choix du type de duplication, il faut hiérarchiser nos impératifs, c’est-à-dire, ce que voulons-nous, une disponibilité des ressources en facteur un, ou une garantie que les données sont vraiment à jour (l'intégrité des données).

Il est évident qu’il y a un temps de mises à jour des données à prendre en compte, même si ce temps est très court.

Le temps de duplication est aussi à prendre en compte dans le choix du nombre de duplications.

Dans notre cas il n’y a pas de contrainte de mises à jour en temps réel, nous partirons donc sur la disponibilité des données en premier points, les mises à jour passeront après.

On comprend aisément que sur un site de trading les contraintes ne seront pas les mêmes. La rapidité de la réponse est essentielle, par exemple si les données sont en mémoire RAM, il semble, que l’accélération des entrées sorties peut atteindre un facteur 1000, par rapport à un travail sur le disque. Les données étant régulièrement sauvegardées sur le disque, entre chaque sauvegarde il y a un risque de perte d’information en cas de coupure de courant.

Les fournisseurs de services Clouds, comme Amazones sont très attrayants, même si les doutes subsistent dans les esprits quant à la gestion de nos données. De très grandes entreprises ont adopté leurs services, nous voyons leurs systèmes être robustes, et la crainte de perte de données, ou de faille de sécurité est devenue moins élevée.

Dans notre cas nous n’avons pas de données confidentielles, en tous cas pour l’heure.

Finalement le projet sera hébergé sur un espace virtualisé des serveurs de l'UHA pour un premier déploiement.

3.6. Langage d’analyse des données:

R est le langage déjà choisi par l’équipe pour des raisons que nous n’étendrons pas ici, l’étude ayant déjà été faite en amont.

R est un langage riche destiné aux scientifiques, il permet avec peu de commandes d’interpréter graphiquement de très grands volumes de données.

3.7. Compétences supplémentaires nécessaires

Certaines connaissances ont été vues en cours cette année avec Monsieur Stéphane Rivière dans le cours de JavaScript avancé avec en plus un cours sur Node.js, TypeScript, Express et BlueBird.

Pour avoir une plus large ouverture d'horizon nous avons pris connaissance des trois ouvrages disponibles à la bibliothèque universitaire sur ce sujet.

En ce qui concerne les bases de données nous avons eu de nombreux cours et fait plusieurs projets avancés en langage PHP.

En ce qui concerne les bases de données NoSql, j'ai fait des recherches sur le web et ai suivi plusieurs tutoriels pour comprendre leurs fonctionnements. J'ai aussi lu de nombreux comparatifs.

La difficulté est que le web est une source de publicité autant que d'information, et qu’il est difficile d'en tirer le bon grain. Il faut donc passer par une phase de test qui est longue et fastidieuse.

Pour Angular-Cli j'ai suivi plusieurs cours en ligne avant d'adopter une approche qui me convenait. Ensuite des recherches spécifiques à mes besoins de développement m'ont emmené à de multiples sources plus ou moins opportunes.

Une problématique est que de nombreuse sources que l'on tente d'implémenter ne fonctionne pas dans notre environnement, et qu'il est fastidieux d'essayer de les faire fonctionner avant d'y arriver, ou de conclure qu'elle soit dépréciée ou peut être erronée.

3.8. Sécurité :

Le protocole d’échange des données sera https. Le type de certificat sera TLS (Transport layer Security), pour faire simple les données sont protégées pendant leurs transferts par un système de cryptographie, dont les clés (une publique, une privée) sont garanties par un organisme de certification au travers de l’émission d’un certificat dont la validité est systématiquement contrôlée avant la mise en place d’un canal d’échange de données.

4. Construction du projet :

Dans cette partie les explications sont animées des morceaux du code l'application, cela afin de rendre plus facile la lecture du code et la compréhension général de l'application.

4.1. Comment fonctionne un projet Angular-Cli :

4.1.1. Une SPI ou SPA c'est quoi ?

Single Page Interface ou Single Page Application, le principe est de charger toute l'application au démarrage, il n'y a qu'une seule page qui est modifiée dynamiquement, soit par l'action de l'utilisateur, soit par l'action du serveur, c'est-à-dire qu'il n'a pas forcément besoin d'une requête client, comme exemple pour comprendre cela le "chat" en est un très bon, vous écrivez à un utilisateur distant, il vous répond et vous voyez le message s'afficher car le serveur vous l'a envoyé sans que vous lui en fassiez la demande.

Le grand gain de temps est fait par la modification partielle de la page.

Pour l'utilisateur le confort de navigation est amélioré par le fait de ne pas voir des pages lui sauter aux yeux à chaque clic, il conserve aussi son menu principal pour qu'il ne soit jamais perdu dans "l'arborescence" de l'application.

Le seul moyen de naviguer est par le menu, il est inutile d'essayer de passer par la barre d'adresse pour passer d'un état à un autre de l'application, ce qui règle un point de sécurité.

4.2. Démarrer un projet

Une fois que tout est installé, écrire dans la console cette commande : ng new monProjet.

Le moteur Angular-Cli va générer le projet avec tous les fichiers nécessaires à sa configuration.

En ligne de commande il faut installer le gestionnaire de paquets "npm" ensuite installer en ligne de commande tous les paquets nécessaires "npm install –save nomDuPaquet". Les paquets sont mis dans le dossier node_module généré par angular-cli.

Il faut ensuite créer un dossier "bin" à la racine du projet dans lequel il faut mettre un fichier "www" sans extension, dans ce fichier sera indiqué comment le projet démarre avec comme indication les ports d'écoutes (un pour le serveur, un pour la base de données), ainsi que le fichier qui fait office de serveur, ici app.js.

4.3. Comment fonctionne un projet MEAN avec Angular-Cli :

Le fichier app.js permet de faire tourner le serveur Node.Js, il inclut les modules nécessaires ainsi que le, ou les fichiers qui servent de routeur pour les requêtes. Lorsqu'une requête arrive elle est exécutée et la réponse est renvoyée.

Nous retrouvons le même principe que dans l'arborescence du DOM d'une page html, c'est-à-dire qu'il y a un élément racine et des éléments enfants, (donc des parents) et des collatéraux. Ainsi les évènements se propagent de proche en proche jusqu'à la racine.

Maintenant c'est le moteur d'Angular-Cli qui récupère les données et les traites, soit directement par le composant appelant soit en passant par le routeur d'Angular-Cli. Ce routeur se trouve dans le fichier app-module.ts (.ts pour TypeScript).

Tous les modules sont inclus dans app-module.ts.

Pour comprendre le projet il faut partir du fichier index.html qui est le Template générale de la SPI.
Une SPI est une Simple Page Interface, c'est à dire qu'il n'y a qu'une seule page dans l'application et qu'elle est juste modifiée par l'évolution ou le changement de type de ses composants.

Ainsi dans la page index.html, il est inséré la balise racine <app-root> (composant principal) dans laquelle serra insérée toutes les autres balises.
Un composant Angular-Cli se créé avec la commande "ng generate composant mon-composant", un dossier mon-composant est créé. A l'intérieur on y trouve le fichier Template HTML, le fichier .ts du traitement du Template (TypeScript) qui sera compilé au lancement du serveur, son fichier css, et éventuellement son fichier de tests unitaires.
On inclut donc mon-composant en utilisant une balise de ce type "<app.mon-composant> " code … …code …code "</app.mon-composant> code .....

Le code compilé ce trouve dans le dossier "dist/projet"

Lors du développement du projet il faut bien garder en tête que Angular-Cli est du côté navigateur et que Node.js est du côté serveur, il n'y a pas de code Angular dans le serveur.

4.3.1. Architecture d'une SPI avec CLI :

4.3.1.1. Voici l'arborescence du projet nommé angular1 :

(Voir capture page suivante figure 8)

Description :

à la racine, le dossier data, qui contient la base de donnée,

puis le projet généré,

dans le répertoire bin, le fichier www, qui permet de démarrer avec Angular-CLI et NodeJs,

dans le répertoir "dist/projet", le résultat du projet compilé, en particulier les fichiers JavaScript issus des fichiers TypeScripts,

dans le répertoire models, les schémas des données,

dans le répertoire node_modules, les modules installés avec npm,

dans le répertoire route, les routeurs de NodeJs,

dans le répertoire uploads, les fichiers chargés en base.

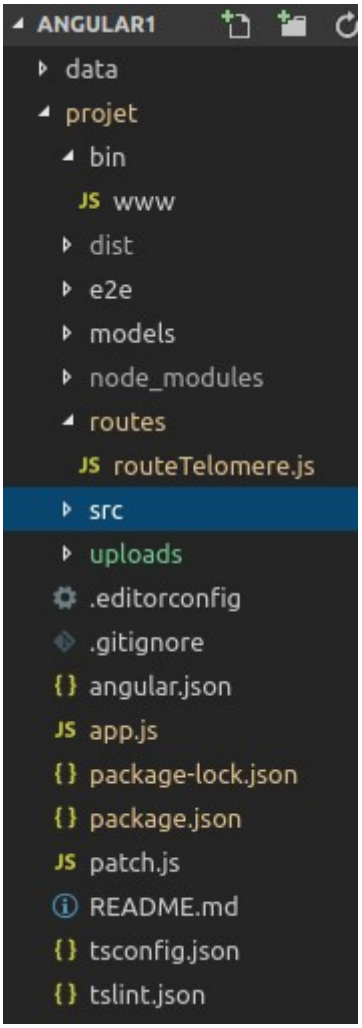


Figure 8 : capture d'écran de l'arborescence du projet

app.js le fichier du serveur node.js,

les fichiers JSON des dépendances du projet,

le fichier .gitignior est le fichier qui indique à Github les fichiers qui ne sont pas à charger et à mettre à jour, car il sont très volumineux et n'ont pas d'intérêt (ce sont principalement les bibliothèques installées avec npm).

4.3.1.2. Le fonctionnement en clair

On voit sur ce schéma que le point d'entrée est le fichier index.html qui est à la racine du projet, le composant minimum de l'application est <app-root> il se place dans le fichier "racine" index.html, cette balise en fait l'appel au composant app-component, généré de base à la création du projet.

Dans <app-component> une balise <router-outlet> réceptionne les résultats du routeur qui sont dans

```
const appRoutes: Routes = [
  { path: 'add-sample', canActivate: [AuthGuard], component: AddSampleComponent },
  { path: 'download', canActivate: [AuthGuard], component: DownloadFileComponent },
  { path: 'upload', canActivate: [AuthGuard], component: UploadFileComponent },
  { path: 'input-file', canActivate: [AuthGuard], component: InputFileComponent },
  { path: 'appareils', canActivate: [AuthGuard], component: AppareilViewComponent },
  { path: 'appareils/:id', canActivate: [AuthGuard], component: SingleAppareilComponent },
  { path: 'auth', component: AuthComponent },
  { path: 'edit', canActivate: [AuthGuard], component: EditAppareilComponentComponent },
  { path: 'users', canActivate: [AuthGuard], component: UserListComponent},
  { path: 'user', component: NewUserComponent },
  { path: 'telomere-add', canActivate: [AuthGuard], component: TelomereAddComponent},
  { path: '',canActivate: [AuthGuard], component: AppareilViewComponent },
  { path: '**', component: FourOhFourComponent }

];
```

Figure 9 : capture de la partie route du fichier app-modules.ts

Attention l'ordre des « path23s » (figure 9) est très important car l'adressage est fait dans l'ordre, ainsi si le « path » '**' est mis en premier, la page d'erreur 404 sera systématiquement renvoyé.

Le module **app-component** (figure 10 à la suite) contient le menu principal, un clic sur un élément du menu envoie l'information au routeur qui effectue les mises à jour dans le navigateur, avec éventuellement un envoi de requête vers le serveur NodeJs dans le cas où des données doivent être échangées avec la base de données MongoDb :

```
<nav class= navbar navbar-default >
  <div class = "container-fluid">
    <div class = "navbar-collapse">
      <ul class="nav navbar-nav">
        <li routerLinkActive = "active"><a routerLink="auth">authentification</a></li>
        <li routerLinkActive = "active"><a routerLink = "appareils">Population</a></li>
        <li routerLinkActive = "active"><a routerLink = "edit">Edit</a></li>
        <li routerLinkActive = "active"><a routerLink = "users">Users</a></li>
        <li routerLinkActive = "active"><a routerLink = "user">New User</a></li>
        <li routerLinkActive = "active"><a routerLink = "telomere-add">New sample</a></li>
        <li routerLinkActive = "active"><a routerLink = "input-file">New full sample</a></li>
        <li routerLinkActive = "active"><a routerLink = "download">Download sample</a></li>
        <li routerLinkActive = "active"><a routerLink = "add-sample">add sample</a></li>
      </ul>
      <div class="navbar-right">

        <p>Vous êtes connecté depuis {{ secondes }} secondes !</p>

      </div>
    </div>
  </div>
</nav>
<div class="container">
    <div class="row">
      <div class = "cols-xs-12"></div>
     <router-outlet></router-outlet>
      </div>
 </div>
```

Figure 10 : capture du fichier app-module.ts

Il s’en sans suit l'insertion d'éléments enfants, qui sont des composants qui s'ajoutent ou se retirent en fonction des demandes de l’utilisateur, qui sont redirigés par le routeur de Angular-Cli qui se trouve dans le ficher app-module.ts (figure 9):

```
const appRoutes: Routes = [
  { path: 'add-sample', canActivate: [AuthGuard], component: AddSampleComponent },
  { path: 'download', canActivate: [AuthGuard], component: DownloadFileComponent },
  { path: 'upload', canActivate: [AuthGuard], component: UploadFileComponent },
  { path: 'input-file', canActivate: [AuthGuard], component: InputFileComponent },
  { path: 'appareils', canActivate: [AuthGuard], component: AppareilViewComponent },
  { path: 'appareils/:id', canActivate: [AuthGuard], component: SingleAppareilComponent },
  { path: 'auth', component: AuthComponent },
  { path: 'edit', canActivate: [AuthGuard], component: EditAppareilComponentComponent },
  { path: 'users', canActivate: [AuthGuard], component: UserListComponent},
  { path: 'user', component: NewUserComponent },
  { path: 'telomere-add', canActivate: [AuthGuard], component: TelomereAddComponent},
  { path: '',canActivate: [AuthGuard], component: AppareilViewComponent },
  { path: '**', component: FourOhFourComponent }

];
```

Figure 9 : capture de la partie route du fichier app-modules.ts

Ici nous avons les routes dans le moteur Angular-Cli, nous sommes du côté du navigateur, dans une route nous appelons donc un composant, qui va être inséré avec ses composants enfants et récursivement les enfants de ses enfants.

A l‘instanciation d'un composant la méthode onInit() ou ngOnInit() (ancienne version de Angular-cli) est appelée avec les éventuelles requêtes à faire cette fois ci du côté du serveur :

```
ngOnInit() {
  this.download();

  this.subscription = this.oneFileService.telomereSubject.subscribe(
    (telomeresSave: any[]) => {
      this.telomeresSave = telomeresSave;
    });

    this.oneFileService.emitTelomereSubject();
    this.nbPipe = new nbCellsPipe();
    this.cellsPipe = new CellsPipe(this.oneFileService);

}

download() {

  this.http.get('/route/sample/download')
    .subscribe(res => {
      console.log('Array avant res? ' + typeof (this.telomeres));
      this.telomeres = <Telomere[]>res;

    }, (err) => {
      console.log('error dans le .ts ', err);
    });
}
```

Figure 11 : capture d'écran de l'instanciation d'un composant.

Ici (figure 11) la méthode download() qui est appelée par onInit() fait appel à la méthode « get » du module HttpClient. HttpClient a été injecté dans le constructeur en propriété privé. Il envoie une requête au serveur avec le chemin passé en paramètre (la route), enfin il souscrit à la réponse.
La réponse "res" est récupérée et passée en paramètre de la méthode flèche "=>" qui retourne l'exécution du résultat dans son corps, qui est ici le remplissage du tableau de telomeres que l'on va pouvoir enfin exploiter à l'affichage du composant.

4.3.1.3. Schéma de l'architecture des composants d'un projet Angular-Cli :

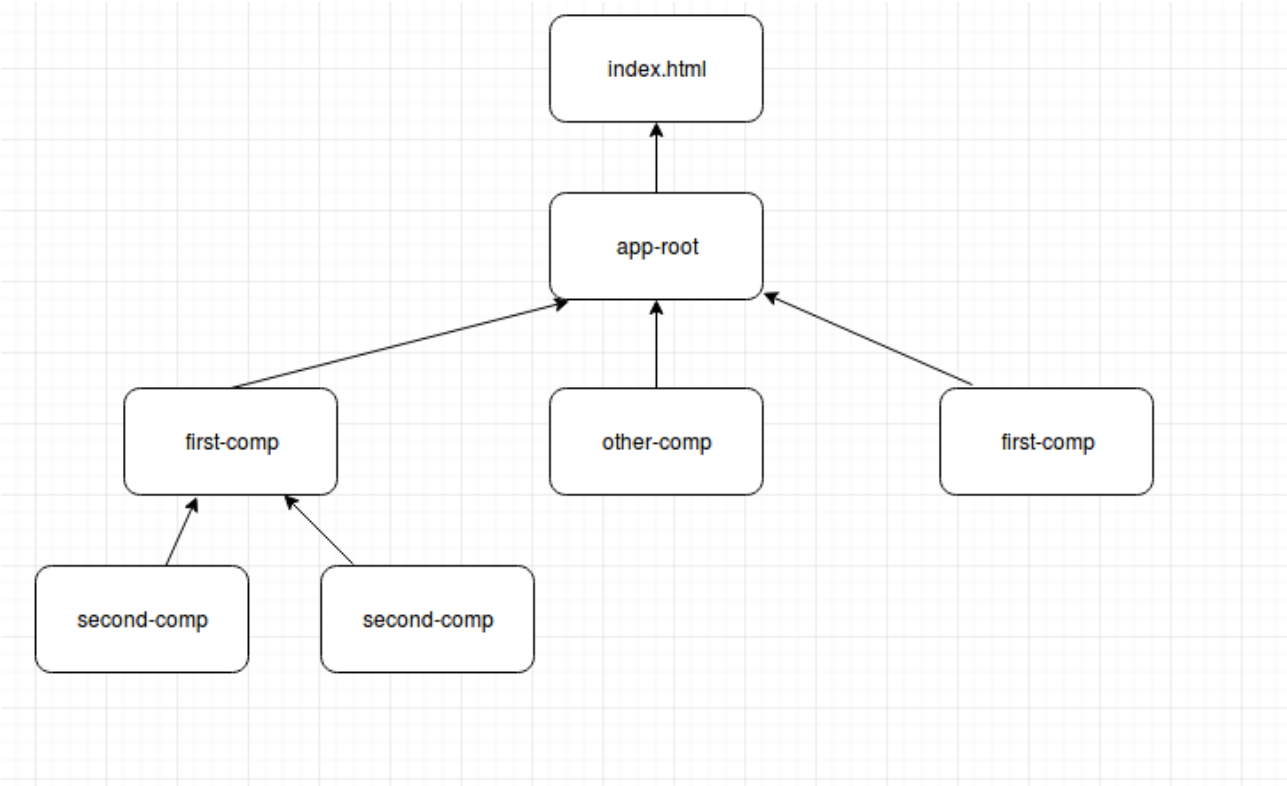


Figure 12 : architecture des composants

Nous voyons sur ce schéma (figure 12) comment sont positionner les composant les uns dans les autres de manière hiérarchisée. Les flèches indiquent dans qui est inséré le composant enfant.
Cette architecture varie au cours de s évènements client ou serveur de la vie de l'application.

4.4. Fonctionnement Node.js et l'application Angular-CLI

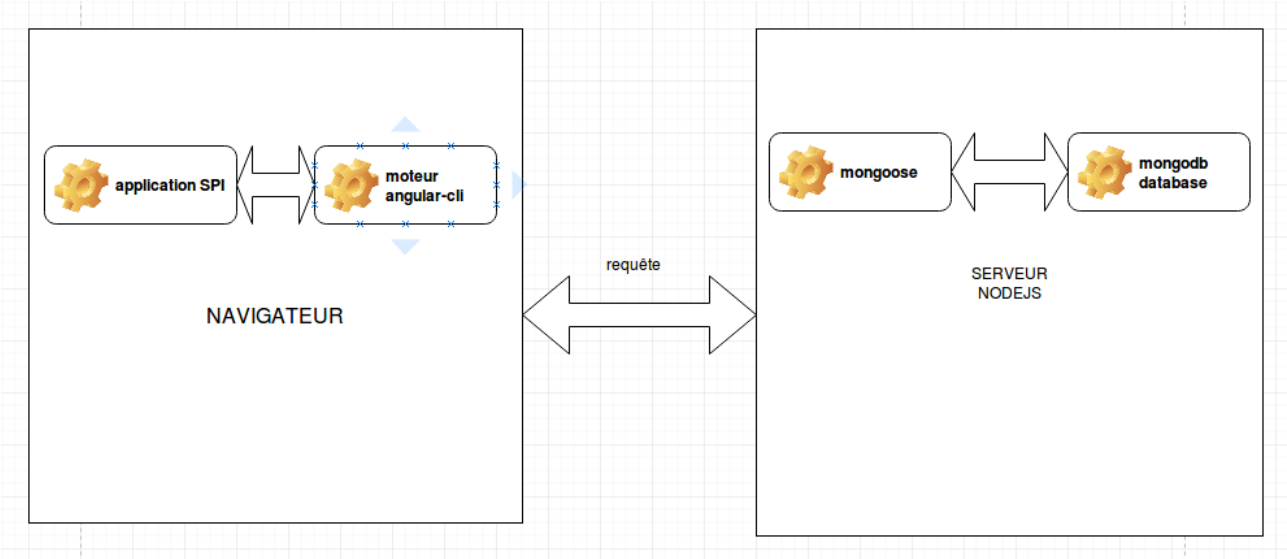


Figure 13 : schéma de la relation client serveur Angular et Node.js

Sur ce schéma on voit l'application qui fait ses demandes au moteur Angular-Cli, si Angular-cli dispose des ressources nécessaires la SPI sera modifiée sans communication avec le serveur NodeJs, dans le cas contraire il y aura une requête vers le serveur Node.js qui arrivera dans le fichier router.js de NodeJs, enfin les données seront renvoyées. Il n'y pas de fichier HTML de renvoyé mais que des codes de retour de requête (code 200 sucess, 304 redirect ou dans le pire des cas 404 not found, ou 500 internal serval error, éventuellement un message pour décrire l'erreur), ainsi que les éventuelles données de la base MongoDb au format JSON.
Le code HTML étant quant à lui chargé au démarrage de l'application au travers de ses composants.

4.5. Un Composant en pratique :

Angular-CLI fonctionne avec des décorateurs pour savoir de quel type sont ses objets :
@Component pour un composant
@Injectable pour un injectable : un service
@Pipe Pour un Pipe
...
Ce sont à chaque fois des classes à instancier.

Un composant a donc son Template HTML et sa classe associée. La classe a son constructeur et sa méthode onInit(), cette méthode sera appelée après le constructeur lors de l'instanciation du composant. On peut donc avoir autant de composant d'un type dans l'application que de besoin. Dans notre cas par exemple le composant single-file représentera un fichier texte, il aura donc des attributs comme son nom original, son id en base, la date d'insertion, ...
 On créera donc autant d'instance de single-file que de fichier dont on a besoin. Ces composants incluront une courte description de son fichier associé.

4.5.1. Exemple de création d'un composant :

Voici un composant (figure 14) new-user généré avec la commande "ng g c new-user":
g pour generat, c pour composant.

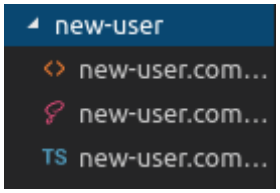


Figure 14 : capture d'écran du dossier d'un composant « new-user »

4.5.2. Les décorateurs :

Lorsque le composant a des besoins spécifiques qui peuvent être réutilisables par d'autres composants la bonne pratique est de déléguer ces fonctionnalités à un fichier de service dans un dossier service.
Pour ce faire on utilise le décorateur "@injectable" pour que Angular-Cli sache que c'est un service, et qu'il va être injecté dans un composant.

En pratique il est passé en argument du constructeur de la class du composant utilisateur, il faut avant le passer en paramètre des "providers" d'app-modules pour qu'Angular-Cli puisse le trouver.

Attention si il n'est pas précisé que le paramètre est public ou privé (dans le constructeur de la classe du composant utilisateur), le service ne sera pas activé sans pour autant qu'un message d'erreur puisse vous aider à comprendre d'où cela provient.

Comme c'est un objet il faudra aussi l' instancier dans la méthode onInit().

4.5.3. Les directives :

Les composants enfants seront affichés dans un composant parent au moyen d'une boucle for (figure 15) "*ngFor" appelée directive, les directives permettent d'écrire des conditionnels dans le code du Template HTML.

```
<app-one-file *ngFor="let telomere of telomeresSave; let i = index"
 [originalname]="telomere.originalname"
 [index]="i"
 [nbCells]="telomere.nbCells"
 [id]="telomere.id"

 ></app-one-file>
```

Figure 15 : capture d'écran d'une directive ngFor

On retrouve l'équivalent de ce que l'on a besoin comme "*ngIf".

D'autres directives sont disponibles dans la documentation officielle. Les liens sont dans la partie bibliographie à la fin du rapport.

4.5.4. Les Pipes :

Nous voyons ici une boucle for, mais comment faire des traitements facilement sur cette boucle avant affichage : avec les Pipes de Angular-Cli. Les données dans la boucle peuvent passer dans un Pipe, décorateur "@Pipe", puis dans un autre, …, et subir ainsi autant de traitement que l'on désire avant d'être affichées :

```
<ng-container *ngIf="(telomeres | nbCellsPipe:sliderValue | cells:originalname ) as telomeresSave"
>
```

Figure 16 : capture d'écran d'une directive ngIf

Dans cet exemple (figure 16) nous avons une propriété "telomeres" qui est un tableau d'échantillons de fichiers. Le pipe est représenté par un "|". Le tableau va donc être passé en argument à nbCellsPipe qui prendra en plus l'argument sliderValue qui est un entier pour effectuer un filtre sur ce tableau, le résultat ira en paramètre du filtre suivant pour enfin être mis dans la variable telomereSave pour l'affichage. Ici les données filtrées serviront à instancier les enfants "app-one-file". « app-one-file » représente un composant fichier texte.

4.5.6. Les services:

@Injectable

Les services doivent être décorés avec @Injectable ce sont des classes à qui on délègue des services pour ensuite les injecter dans le constructeur, attention à penser à les instancier dans la méthode onInit() pour pouvoir se servir de l'objet. Le service est un objet qui est fait pour être réutilisable, et dans une moindre mesure pour aérer le code JavaScript du composant.

4.5.7. Les observables :

Comme son nom l'indique c'est un objet que l'on va observer, on appelle cela la programmation réactionnelle, c'est à dire que le programme va réagir à chaque modification de l'observable auquel on aura souscrit :

Figure 17 : capture d'un observable

```
download() {

  this.http.get('/route/sample/download')
    .subscribe(res => {
      console.log('Array avant res? ' + typeof (this.telomeres));
      this.telomeres = <Telomere[]>res;

    }, (err) => {
      console.log('error dans le .ts ', err);
    });
}
```

"http" est un observable (figure 17), il permet d'envoyer une requête du côté de Angular (navigateur) vers node.js (côté serveur). Ici c'est une méthode "get", on demande donc d'aller au chemin passé en paramètre : 'route/sample/download' dans le routeur de NodeJs (fichier app.js), de prendre les données qui y sont proposées.
Dans la méthode subscribe la souscription à l'observable permet de recevoir la réponse "res". Ensuite une fonction flèche qui prend donc en paramètre "res" et retourne l'exécution le corps de la fonction.

Subject :

Plus complet le Subject (figure 18), le Subject peut émettre et recevoir des données :

```
telomereSubject = new Subject<any[]>();
```

Figure 18 : capture d'écran d'un Subject

Ici on a un Subject qui est un tableau de n'importe quel type, on va pouvoir y "subscribe" :

```
ngOnInit(){

this.subscription = this.telomereSubject.subscribe(
(telomeresSave: any[]) => {
this.telomeres = telomeresSave;
});
this.emitTelomereSubject();
}
```

Figure 20 : capture d'écran : Subject

Ici à chaque modification du Subject (figure 20), le subscribe sera exécuté.
On récupère cette souscription dans un objet de type Subscription pour la conserver.

Et on peut émettre grâce à la méthode next() :

```
emitTelomereSubject(){
this.telomereSubject.next(this.telomeres.slice());
console.log(this.telomeres);
}
```

Figure 21 : capture d'écran : émission d'une modification.

La méthode emitTelomereSubject() (figure 21) va mettre à jour le Subject qui est un tableau, l'ancienne valeur est perdue, this.telomeres.slice() représente un tableau, la méthode slice le découpe en élément simple pour pouvoir être reconstruit selon le modèle du tableau Subject. Maintenant tous ceux qui ont Subscribe vont être mis à jour.

Il existe d'autres variantes comme BehaviorSubject qui prend une valeur par défaut ou AsyncSubject qui est utile pour un Subject ne servant qu'une seule fois.

Lorsque l'on subrscribe, il faut ne pas oublier d'arrêter la souscription en le précisant à l'obervable :

this.telomereSubject.complet();

Cet exemple de programmation réactionnelle se trouve dans le service on-file-service.ts, qui est un injectable "@injectable", c'est-à-dire que l'on va pouvoir l'injecter au constructeur de la class d'un de nos composants. Dans notre cas ce service sert à charger les données en base de données. En fonction des actions du client ce service sert aussi à effectuer des filtres sur le tableau en mémoire afin de faire soit, des sauvegardes sous la forme de population, soit des téléchargements.

5. Fonctionnalités et interfaces

5.1 Le menu :

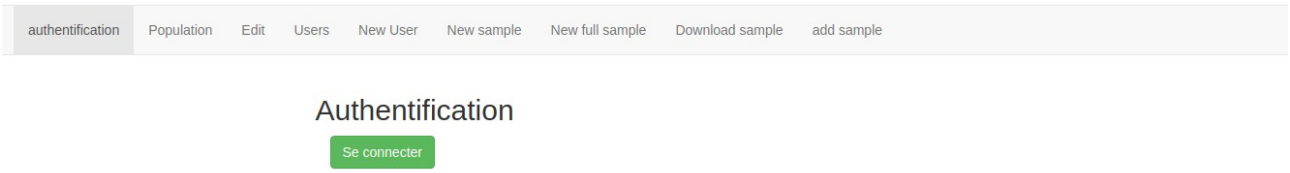


Figure 22 : capture d'écran page d'accueil

Ici on arrive au point d'entrée de l'application (figure 22), seul l'onglet authentification est disponible, vous devez vous authentifier pour activer, en fonction de vos droits, les autres champs du menu.

Explications :

```
const appRoutes: Routes = [
  { path: 'add-sample', canActivate: [AuthGuard], component: AddSampleComponent },
  { path: 'download', canActivate: [AuthGuard], component: DownloadFileComponent },
  { path: 'upload', canActivate: [AuthGuard], component: UploadFileComponent },
  { path: 'input-file', canActivate: [AuthGuard], component: InputFileComponent },
  { path: 'appareils', canActivate: [AuthGuard], component: AppareilViewComponent },
  { path: 'appareils/:id', canActivate: [AuthGuard], component: SingleAppareilComponent },
  { path: 'auth', component: AuthComponent },
  { path: 'edit', canActivate: [AuthGuard], component: EditAppareilComponentComponent },
  { path: 'users', canActivate: [AuthGuard], component: UserListComponent},
  { path: 'user', component: NewUserComponent },
  { path: 'telomere-add', canActivate: [AuthGuard], component: TelomereAddComponent},
  { path: '',canActivate: [AuthGuard], component: AppareilViewComponent },
  { path: '**', component: FourOhFourComponent }

];
```

Figure 9 : route de Angular dans le fichier app-module.ts

Ici chaque chemin est protégé par le service AuthGard.ts, qui est in injectable (@injectable), qui implémente la super classe CanActivate. En fonction de vos droits il vous ouvre le chemin d'accès ou non.

5.2. Gestion des droits utilisateurs :

5.2.1. Interface de création d'utilisateur :

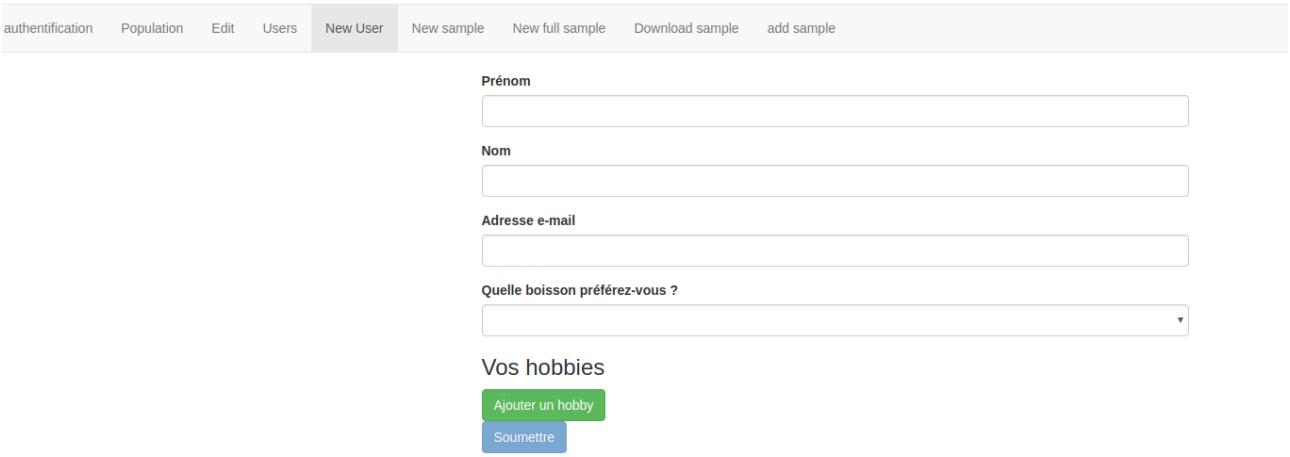


Figure 23 : capture formulaire d'inscription

Voici (figure 23) un formulaire classique dont les champs seront modifiez après consultation des utilisateurs.

5.2.2. Le schéma utilisateur :

```
var mongoose = require('mongoose');
var bcrypt = require('bcrypt');

var UserSchema = new mongoose.Schema({
  email: {
    type: String,
    unique: true,
    required: true,
    trim: true
  },
  username: {
    type: String,
    unique: true,
    required: true,
    trim: true
  },
  password: {
    type: String,
    required: true,
  },
  passwordConf: {
    type: String,
    required: true,
  },
  admin: {
        type: Boolean,
         default: false}
});
```

Figure 24 : capture d'écran du schéma utilisateur

Ici une capture du modèle d'un utilisateur (figure 24), cette partie qui a été implémenté dans le première version du projet avec NodeJs uniquement en "modèle vue contrôleur "avec la gestion de l'encryptage et des cookies“ est à faire migrer ou à recoder.

Droit en lecture, écriture, téléchargement, chargement :

L'administrateur sera, dans un premier temps du projet, le seul à pouvoir octroyer des droits, étant donné que le projet est au point de départ et que nous ne connaissons pas encore tous les utilisateurs finaux ainsi que les droits que nous leurs devons octroyer.

Dans un premier temps donc, seul Radia M'Cacher et Bruno Colicchio seront autorisés à insérer ou télécharger des données.

5.3. Terminologie :

5.3.1. Définitions et schémas de données :

5.3.1.1. Les Individus :

Ce terme va être repris dans l'ensemble du rapport :
"Individu" :
il correspond à un échantillon donné d'une personne, à un moment donné, traité selon un protocole de mise en fluorescence donné. Enfin l'image est traitée par un programme de traitement d'images qui est déterminé en fonction du type de besoins d'informations.

5.3.1.1.1Schéma des données :

```
var mongoose = require('mongoose');
var teloSchema = new mongoose.Schema({
    name : String,
    params : [String],
    date : { type : Date, default: Date.now },
    description: String,
    fileName : String,
    protocole : String,
    date_edition : String,
    nbCells : number,

    populations: [
        {
            type : mongoose.Schema.Types.ObjectId,
            ref: 'Population'
        }
            ]
});
```

Figure 25 : capture du schéma d'un Individu (ici nommé telomere)

Ici en tête require("mongoose") est l'utilitaire qui s'occupe de la persistance des données en base Mongodb, ensuite est créé notre schéma avec nos propriétés. Nous pouvons rajouter toutes sortes de spécifications comme on le voit pour la date. La liste est disponible dans la documentation officielle. Les liens pour les documentations officielles sont donnés en fin de rapport à la rubrique bibliographie.
Enfin on peut lire comment est fait le lien entre l'individu et ses populations, c'est fait au travers d'un tableau d'identifiants.

Il ne reste plus qu'à nommer notre schéma (Figure 26) et à l'exporter pour qu'il puisse être disponible :

```
var Telomere = mongoose.model('Telomere', teloSchema);

module.exports = Telomere;
```

Figure 26 : export schéma Telomere

5.3.1.2. Les populations :

Ce terme va être repris dans l'ensemble du rapport :

"population" :
Une population est le regroupement d'individus selon des critères communs déterminés, afin de pouvoir leurs appliquer un traitement informatique analytique de comparaison et de mise en évidence de particularités globales ou individuelles un individu par rapport à la population).

Schéma des données :

```
var mongoose = require('mongoose');
var telomere = require('./Telomere.js');

var populationSchema = new mongoose.Schema({
    name: String,
    date: { type : Date, default :Date.now },
    description : String,


telomeres: [
    {
        type : mongoose.Schema.Types.ObjectId,
        ref: 'Telomere'
    }
        ],
```

Figure 27 : schéma des données Population

Ici la même construction (figure 27), la présence de `var telomere = require('./Telomere.js');` n'est pas utile dans l'immédiat et peut être supprimé car on ne s'en sert pas. C'est le même principe que pour les Individus, des propriétés simples et un tableau d'identifiants pour faire le lien avec ses individus.

5.4. Chargement des données :

Une interface de sélection de fichiers avec remplissage automatique des champs afin d'éviter les erreurs, et ajout de métadonnées dont la liste reste à définir (nom de la personne qui charge, date lieu ...).

Ces données sont enregistrées en base selon un schéma de données :

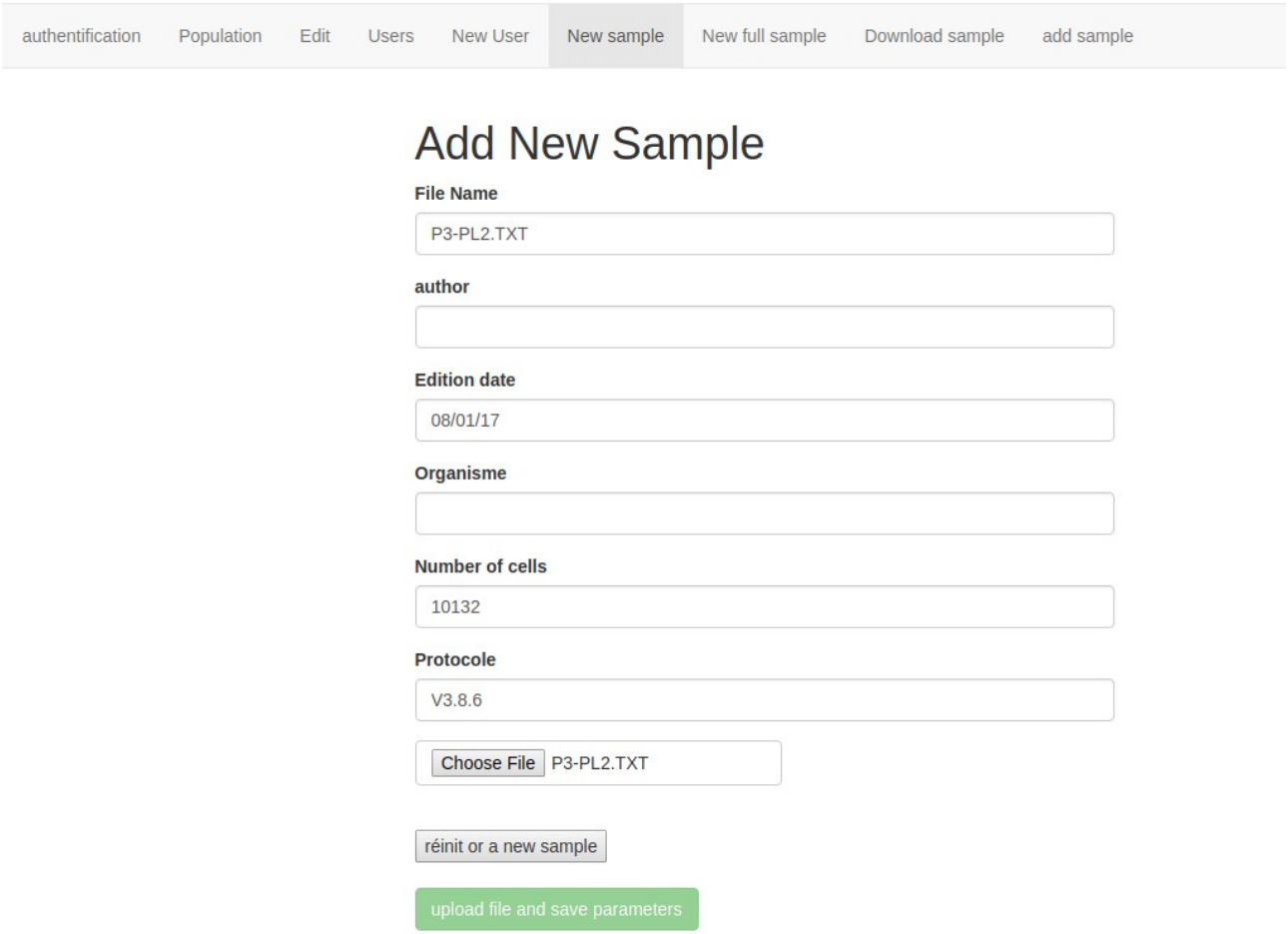


Figure 28 : capture d'écran de l'interface de chargement d'un fichier

Lors de la sélection du fichier (figure 28), le service de lecture de fichier lit et recueille les informations utiles pour ensuite les afficher automatiquement dans les champs concernés. Il ne reste plus que les champs à remplir à la main. On pourrait mettre des valeurs par défaut, ainsi il n'y aurait plus qu'à faire un contrôle visuel et valider.

Pour peupler la base nous avons étudié la mise en place de scripts, ainsi il suffirait de ce placer dans un dossier pour que les fichiers soient chargés en base automatiquement en un simple clic, cette partie reste à implémenter.

5.5. Filtrages des données

5.5.1. Filtrage des individus :

Interface permettant de filtrer les échantillons pour, soit les télécharger sous forme d'archive, soit de créer une population.

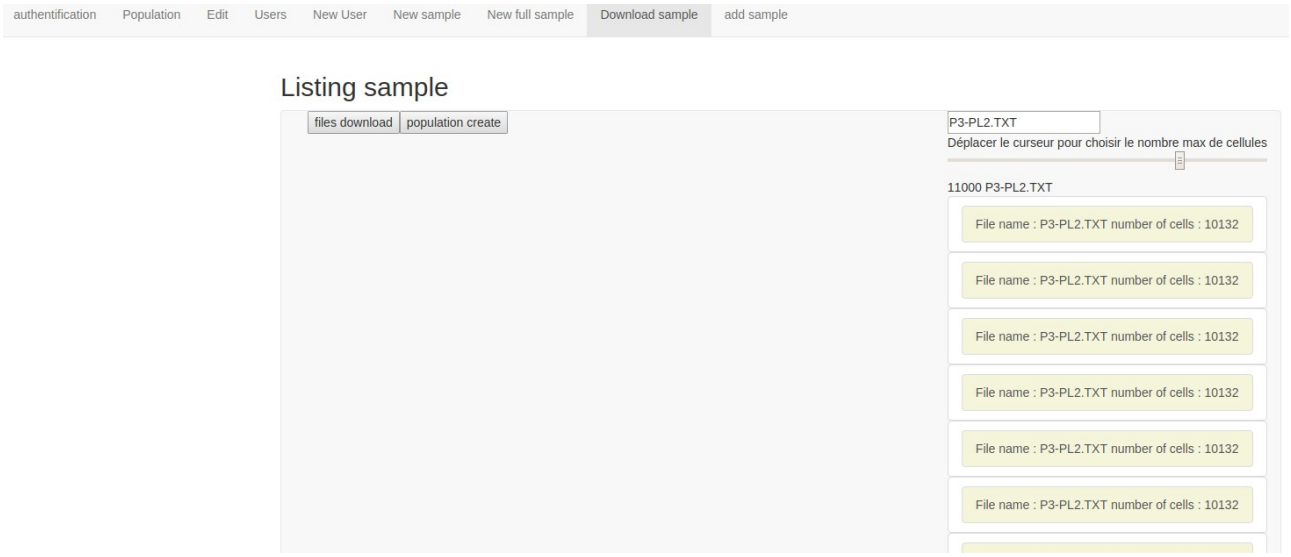


Figure 29 : capture d’écran de l’interface de filtrage des fichiers textes.

5.5.2. Filtrage des données interface

Sur la droite deux types de filtres (figure 29), les filtres restent à définir, ici un filtre sur le nombre de cellules du prélèvement à partir d'un curseur, et un filtre à partir d'un nom ou d'un morceau de nom de fichier.
La liste des prélèvements est mise à jour à chaque variation d'un des deux filtres, les deux filtres sont pris en compte. Cela est fait aux moyens des Pipes "|" (décorateur : @Pipe) :

```
<ng-container *ngIf="(telomeres | nbCellsPipe:sliderValue | cells:originalname ) as telomeresSave"
>
```

Ici "telomeres" représentent le tableau d'échantillons stocké dans la classe associée aux composants. Ce tableau est envoyé à la class "nbCellsPipe" qui est une Pipe, il reçoit en paramètre la valeur du curseur ainsi que le tableau "telomeres", ensuite le résultat est envoyé au Pipe cells qui prend en paramètre la valeur du champ texte de référence ainsi que le tableau résultat du précédent Pipe. Enfin ce résultat est mis dans la variable telomeresSave pour sauvegarde sous la forme d'une population ou pour être téléchargé.

5.5.2.1 Filtrage des populations :

Interface à implémenter permettant de filtrer les populations dans le but de les télécharger sous forme d'archive.

5.6. DataFrame :

Un DataFrame est comme un dictionnaire dont les noms sont les noms des colonnes, et les valeurs sont des tableaux. Les données sont généralement hétérogènes.

Cette partie reste à implémenter.

Il faut créer une fonctionnalité qui permette l'export au format JSON des informations contenus dans les Populations.

5.7. Charger des rapports et les lier aux populations

Cette partie est à implémenter.
Ajouter un schéma pour les documents (pdf) et créer les tableaux de liens avec les populations.
Faire une interface.

5.8. Liste des Populations

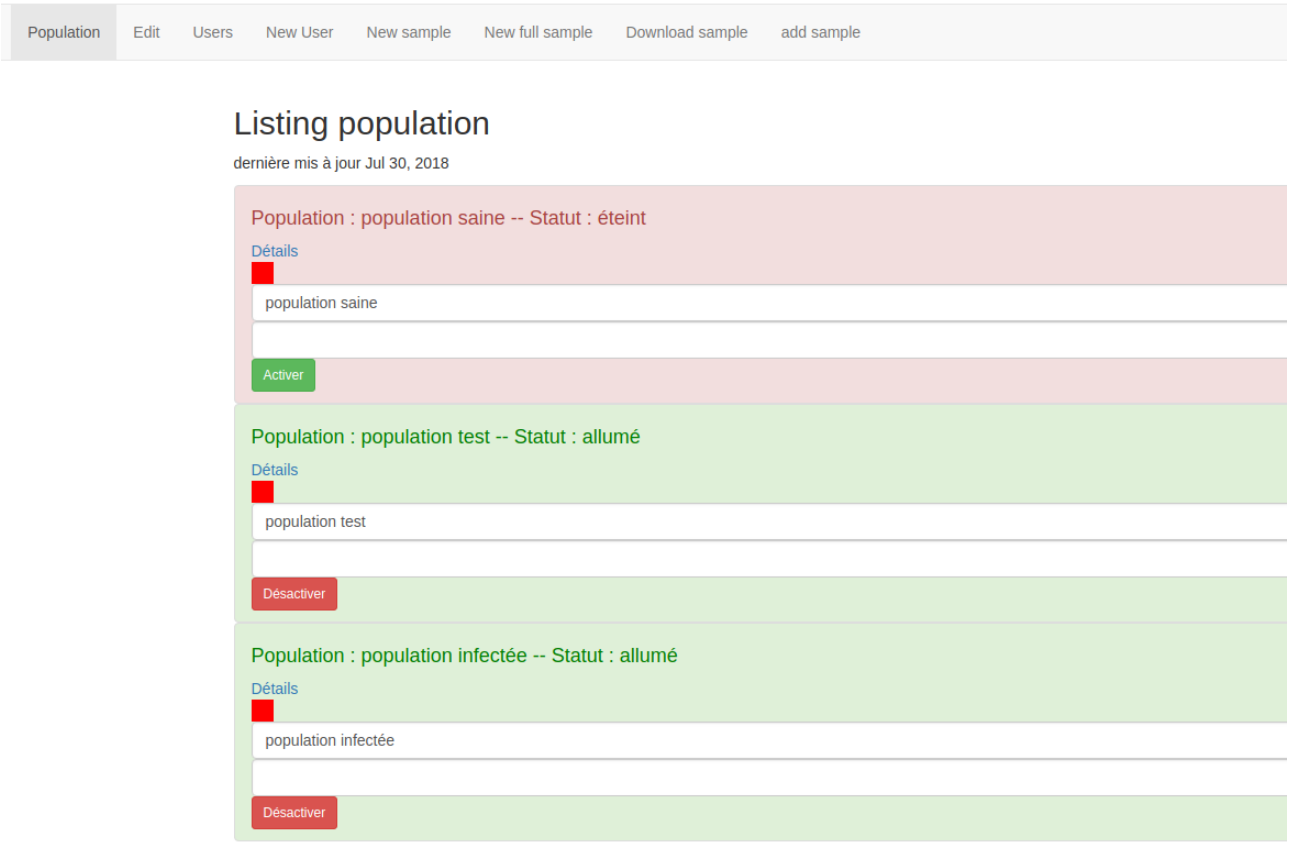


Figure 30 capture d'écran : interface de liste de populations avec sélection des populations saines ou nons

Ici une liste des populations (figure 30) avec un bouton pour activer la population si elle est saine ou l'inverse. Le choix des paramètres reste à définir ainsi que les sécurités à mettre en place pour ne pas avoir de modifications hasardeuses en base de données.

Conclusion :

Les résultats obtenus sont en corrélations avec le besoin, en particulier dans le choix des schémas des données pour la structuration de la base de données MongoDb.

Néanmoins même si les fondations du projet sont en place, il reste plusieurs interfaces à implémenter avant de mettre le projet en production, ainsi que la réalisation des tests unitaires.

Un temps important a été passé à l'étude du projet et du choix et test des outils, la durée de quatre mois pour finaliser un tel projet n'est pas suffisante.

Malgré la rédaction de document de spécification et de conception, les objectifs changent en cours de projet par la découverte de technologie plus adaptée à nos besoins. Les objectifs changent aussi avec l'émergence de nouveaux besoins ou contraintes, ce qui implique de faire des choix stratégiques. En particulier il faut faire le choix entre les délais, ce que l'on veut produire, en termes de technicité, d'efficacité et de modernisme.

Ici le projet n'étant pas figé, nous avons pu reconstruire le projet en fonction de nouvelles technologies rencontrées au travers de recherche sur le web pour résoudre certaines problématiques. Aussi en programmant certaines interfaces, nous nous sommes rendu compte qu'elles ne correspondaient pas forcément aux besoins du document de spécification, et qu'il fallait modifier celle-ci pour être plus proche du besoin réel.

En outre la pensée ne s'arrêtant pas à la clôture d'un document de conception ou de spécification, certaines idées sont apparues en cours de développement.

J'ai beaucoup appris sur le fonctionnement d'un laboratoire de recherche et l'articulation de projet multi-partenaire.

J'ai aussi découvert ce qu'était le travail dans un bureau et trouver un nouvel équilibre dans cet environnement.

Sur le plan technologique, j'ai pu utiliser une grande partie des connaissances étudiées au cours de la formation, ces derniers m'ont permis d'en acquérir de nouvelles comme l'utilisation de nouveaux langages, et l'utilisation de nouvelles structurations d'application web.

<u>Critiques de la formation :</u>

Une question revient souvent tout au long de la formation : faut il être un spécialiste dans un domaine, ou faut il avoir des connaissances plus générales dans tous les domaines.

Dans ce projet la réponse est qu'il faut des connaissances dans tous les domaines (informatique), ainsi ce projet inclue :

1. - une partie traitement d'image (distinguer des objets dans une image)

2. -une partie apprentissage et acquisition de données (reconnaître le type des objets trouver)
3. -une partie analytique (extraire des informations significatives à partir d'une grande masse de données)
4. -des connaissances en réseau (échange de données entre les postes clients et les postes serveurs)
5. - des connaissances en sécurité pour application web
6. - en base de données (pour stocker les données et les rendre disponibles de manière efficace)
7. -des connaissances en développement web (côté serveur et côté client)
8. -une bonne maîtrise de la communication et de ses outils

Perspectives :

Enfin dans un avenir proche je souhaite approfondir cette technologie jusqu'à la maîtriser, dans le but de valoriser le temps passé, et par le vif intérêt qu'elle suscite, tant dans la qualité de son architecture que dans son efficacité.

Une réunion est prévue le cinq septembre entre les différents acteurs pour voir comment il serait envisageable de poursuivre le projet.

Bibliographie :

Sources manuscrites :

Apprendre à créer des applications Web avec JavaScript, Auteur Romain Willmann.

Cours Master 1 et 2 JavaScript et Node.js Faculté des Sciences et Technique, Université de Haute Alsace Mulhouse
Auteur Stéphane Rivière.

Développez efficacement avec JavaScript
Auteur : Alexandre Brillant

Node.js : Exploitez la puissance de JavaScript côté serveur
Auteur : Julien Fontanet, Olivier Lambert.

Sources webs :

démarrer un projet mongodb mongoose et nodejs
https://atinux.developpez.com/tutoriels/javascript/mongodb-nodejs-mongoose/

documentation officielle angular-cli
https://cli.angular.io/

documentation officielle mongodb
http://mongodb.github.io/node-mongodb-native/

Lire un fichier texte mot à mot ou ligne par ligne
https://www.scriptol.fr/javascript/node-readline.php

tutoriel pour créer une app avec angular-cli et google pour le stockage des données
https://openclassrooms.com/fr/courses/4668271-developpez-des-applications-web-avec-angular

Pour continuer :

electron et angular
https://www.maximegris.fr/posts/angular-electron

Webpack
https://www.alsacreations.com/tuto/lire/1754-debuter-avec-webpack.html

Résumé

Rapport de stage de fin d'études master informatique.

L'objectif est d'automatiser le traitement de la quantification des aberrations chromosomiques et l'altération des télomères.

Ce projet se place dans un contexte plus global qu'est le projet Juxta Rhénum. Ce dernier visant à faire ressortir l'impact de la création, jusqu'au démantèlement, de la centrale de Fessenheim, sur son environnement.

La mesure des altérations chromosomiques se fait au moyen de prélèvement sanguin, puis par la prise d'images par un microscope et le traitement de ces dernières par le programme informatique associé. En sortie nous obtenons un fichier texte contenant de 100 000 à 200 000 valeurs numériques à analyser.

Ces échantillons sont regroupés en ensemble pour ensuite y appliquer une analyse avec un langage informatique scientifique analytique, R en l'occurrence, et obtenir ainsi des résultats interprétables.

Tous ces fichiers sont actuellement stockés en locale sur les postes de travail des différents acteurs. Les regroupements des différents échantillons se font à la main. Lorsqu'une personne a besoin des données elle doit prendre contact avec la détentrice des données ou bien les chercher dans son poste de travail.

Ainsi la création d'une base de données est le point de départ de ce projet.

S'en suit la création d'une application à vocation d'être hébergée sur un serveur avec des interfaces de chargement, de téléchargement et de filtrage des échantillons. Les données seront rangées selon un protocole et disponible en ligne pour recherche, tri, regroupement, analyse, comparaison.

Les choix des outils informatiques, de leurs langages de programmation, sont primordiaux pour obtenir une application moderne et évolutive.

Le projet étant au démarrage, ses limites ne sont pas encore connues, il faut donc choisir des outils qui offrent une grande souplesse pour supporter facilement les modifications.

Mots clefs :

Un Individu est un fichier texte représentant un échantillon de sang d'un individu après coloration puis numération par le microscope de ses caractéristiques à un moment donné.

Une Population (une cohorte) est le regroupement d'Individus selon différents critères communs à des fins de comparaison et de mise en évidence de caractéristiques communes ou de différences.