



---

## Mémoire du Projet de Fin d'Etudes

Papaya Volbrain

Extension d'un logiciel open-source de visualisation d'images médicales

---

### **Auteurs :**

Coquilleau Eliot, Declercq Lucas, Flegon Valentin,  
Grimal Jeremie, Le Flohic Antoine, Silari Charles

### **Clients :**

Coupe Pierrick et Mansencal Boris

### **Responsable d'UE :**

Narbel Philippe et Auber David

01 Février 2022 - 30 Mars 2022

# Table des matières

<b>Introduction</b>	<b>3</b>
<b>1 Présentation du projet</b>	<b>3</b>
1.1 VolBrain	3
1.2 Papaya	3
1.3 Existant	3
1.4 Besoins	5
1.4.1 Besoins fonctionnels	5
1.4.2 Besoins non fonctionnels	6
<b>2 Développement</b>	<b>6</b>
2.1 Travail réalisé	6
2.1.1 Onglet Informations	6
2.1.2 Onglet QC	7
2.1.3 Onglet Edits	8
2.1.4 Viewer Papaya	9
2.1.5 Autres tâches	10
2.2 Architecture	10
2.3 Fonctionnement	11
2.3.1 Technologies utilisées	11
2.3.2 Routes	11
2.3.3 Comportements	12
2.4 Tests	15
2.5 Difficultés	17
<b>3 Gestion de projet</b>	<b>18</b>
3.1 Organisation	18
3.1.1 Semaine type	18
3.1.2 Daily meeting	18
3.1.3 Sprint Review	19
3.1.4 Sprint Retrospective	19
3.1.5 Sprint Planning	20
3.2 Jira	20
3.2.1 Découpage du projet	20
3.2.2 User Story	20
3.2.3 Definition of done	21
3.2.4 Planning poker	22
3.2.5 Tableau Scrum	23
3.2.6 Burnup	24
3.3 Git	25
3.3.1 Git flow	25
3.3.2 Convention de nommage	26
3.4 Outils de test	26
3.4.1 Selenium WebDriver	26
3.4.2 Selenium IDE	27
3.4.3 Cahier de recette	28
<b>4 Bilan</b>	<b>28</b>
4.1 Dette technique	28
4.2 Erreurs	28
<b>5 Conclusion</b>	<b>29</b>
5.1 Résumé	29
5.2 Apprentissages	29
5.3 Perspectives	29

# Introduction

Dans le cadre de l’UE ”Projets de fin d’études”, nous avons travaillé sur l’extension de la partie visualisation de la plateforme VolBrain. L’objectif était de s’approprier la méthode agile SCRUM afin de répondre aux attentes de nos clients.

Dans ce mémoire, nous commencerons par une présentation du projet en exposant notamment l’existant et les besoins qui en découlent. Nous parlerons ensuite de notre développement en abordant le travail réalisé, l’architecture du projet ou encore les difficultés rencontrées. Nous détaillerons également notre gestion de projet en évoquant notre organisation et les outils utilisés. Enfin, nous terminerons par un bilan au cours duquel nous traiterons de la dette techniques, de nos erreurs et de nos apprentissages.

## 1 Présentation du projet

### 1.1 VolBrain

Volbrain est une plateforme web de traitement d’images médicales, plus particulièrement d’IRM cérébrales, développé par le LaBRI et l’université de Valence (Espagne). La plateforme actuelle [3] est utilisée par plus de 5000 utilisateurs uniques et a déjà traité plus de 280000 images. Une nouvelle version de la plateforme, s’appuyant sur des micro-services, est en cours de développement pour permettre une plus grande extensibilité. [2].

Les utilisateurs inscrits sur la nouvelle plateforme ont la possibilité de téléverser des images et d’exécuter différents pipelines de traitement sur celles-ci. Ces pipelines produisent en particulier un rapport, un fichier CSV avec différentes mesures et le plus souvent des segmentations, c’est-à-dire un découpage en régions, des images fournies. Une fois les traitements terminés, les utilisateurs peuvent ensuite télécharger les segmentations et les rapports produits. Il est aussi possible de visualiser les images et les segmentations produites directement sur la plateforme. C’est sur cette dernière partie qu’a porté notre travail.

### 1.2 Papaya

Dans l’application, le module de visualisation s’appuie sur Papaya : un visionneur d’images de recherche médicale en JavaScript pur, prenant en charge les formats DICOM (Digital imaging and communications in medicine) et NIFTI (Neuroimaging Informatics Technology Initiative). Il est développé par le groupe Rii-Mango. Celui-ci permet de visualiser plusieurs images ou segmentations ainsi que de décrire les régions de celles-ci.

### 1.3 Existant

Initialement, la page de visualisation de VolBrain était la suivante [1] :

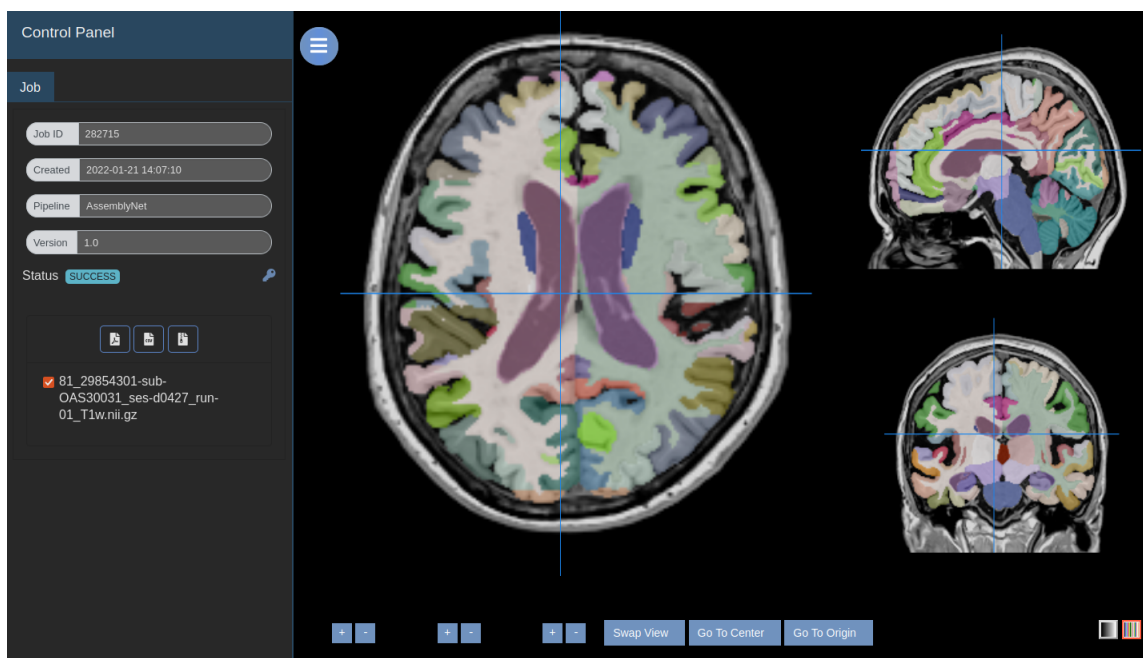


FIGURE 1 – Apparence de l'existant

En résumé, l'existant comportait deux parties. À gauche, un menu très limité détaillant des informations liées à l'analyse courante (aussi appelé job), dont certaines ne sont pas pertinentes. De plus, des boutons inutiles et un onglet inutilisé sont présents. À droite, le viewer Papaya affichant les images de l'archive permet difficilement le choix d'une région précise du cerveau, car par défaut, tous les calques sont affichés simultanément. Nous appelons un calque, une image ou une segmentation visualisée sur le viewer.

Au-delà de l'apparence de l'existant, plusieurs éléments de celui-ci nous ont été fournis par les clients afin d'entamer le développement. Tout d'abord, un fichier `preview.html` qui détaille le squelette HTML de la page et utilise des variables Jinja. Ces variables Jinja sont vouées à être remplacées par des données propres à l'analyse en cours. Ces données sont stockées dans un fichier d'entrée appelé `input.json`, fourni par les clients. D'autres fichiers de données ont également été fournis en relation avec les différentes régions du cerveau :

- des images `.nii` : visualisées par le viewer de Papaya
- des fichiers `bounds_XX.CSV` : stockent les courbes de normalité en fonction du sexe
- des fichiers `mapping_XX.json` : stockent l'identifiant et le nom des labels
- un `report.CSV` : stockent les informations propres au patient

Pour accéder à ces fichiers de données, l'`input.json` fait office d'intermédiaire [2] :

GET https://volbrain.net/user/preview?hash=625e4.....

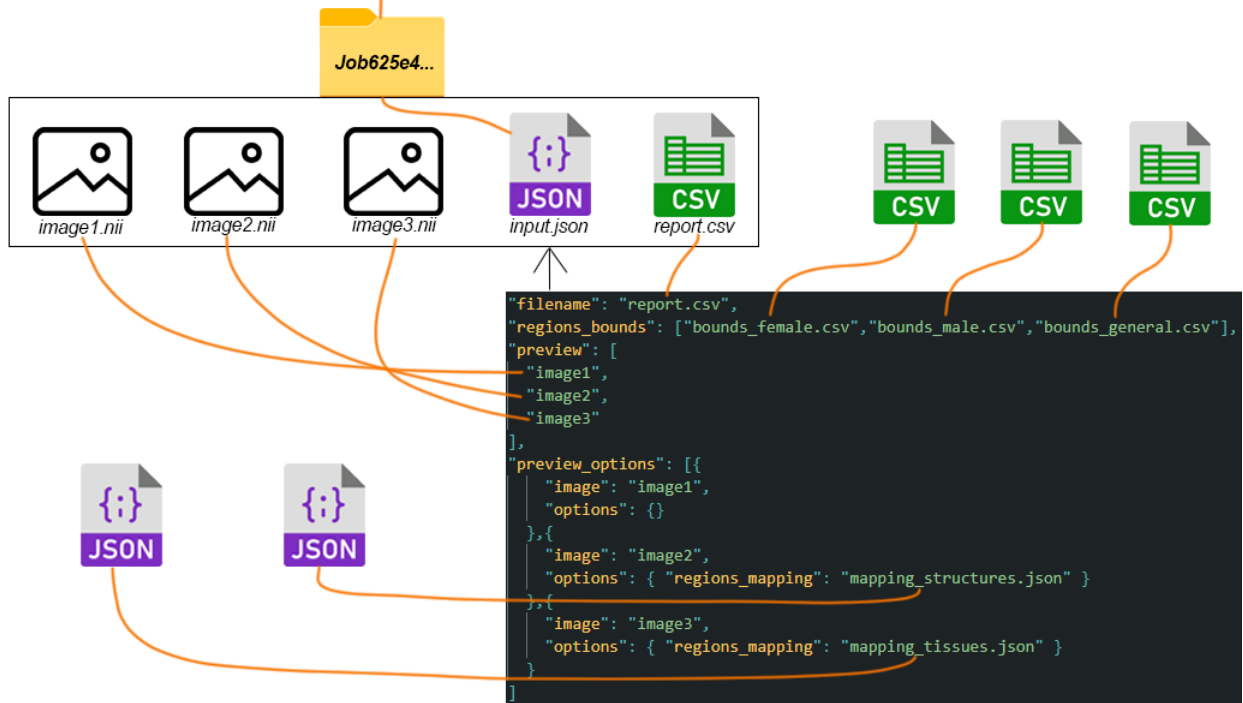


FIGURE 2 – Rôle de l'input dans l'accès aux données

À partir de ces fichiers, les clients souhaitent étendre le panneau de gauche pour afficher des informations relatives aux différentes régions calculées.

## 1.4 Besoins

Au cours des rendez-vous successifs avec nos clients, les besoins et leur priorité ont évolué. Par conséquent, nous listerons uniquement les besoins finaux.

### 1.4.1 Besoins fonctionnels

Dans cette partie, les besoins fonctionnels seront listés par ordre de priorité. La priorité est indiquée de façon numérique où [1] indique un **besoin essentiel** et [3] indique un **besoin peu important**. Les besoins fonctionnels de ce projet sont :

- [1] Afficher les informations relatives à une région sélectionnée : en tant qu'utilisateur, je souhaite connaître le nom, la description et la courbe de normalité de la région sur laquelle je clique.
- [2] Gérer les calques : en tant qu'utilisateur, je souhaite pouvoir afficher, masquer et changer le calque actif sur le viewer Papaya. De plus, des raccourcis clavier doivent permettre de réaliser ces actions. Enfin, seul un calque par catégorie (Image et Segmentation) peut être affiché à la fois.
- [2] Créer un système de retour utilisateur sur la qualité des segmentations et des images : en tant qu'utilisateur, je souhaite pouvoir commenter et noter les éventuelles erreurs de chaque calque puis envoyer ce retour aux administrateurs de VolBrain. De plus, ce retour doit être téléchargeable au format CSV. Ce service doit être disponible que si l'utilisateur accepte de partager ses images.
- [3] Éditer l'analyse : en tant qu'utilisateur, je souhaite pouvoir ajouter une image de format .nii et la visualiser temporairement dans le viewer Papaya.

### 1.4.2 Besoins non fonctionnels

Trois besoins non-fonctionnels sont également à noter :

- Application ergonomique et intuitive pour l'utilisateur : notamment par la mise en place de raccourcis similaires à d'autres logiciels de visualisation d'images médicales comme *ItkSnap* [8].
- Conserver la fluidité d'affichage : notamment en conservant un temps de réponse similaire à l'application existante et un affichage quasi-instantané des informations d'une région.
- Conserver un affichage cohérent pour tous les supports : notamment en respectant la charte graphique de l'application et en conservant le design responsive.

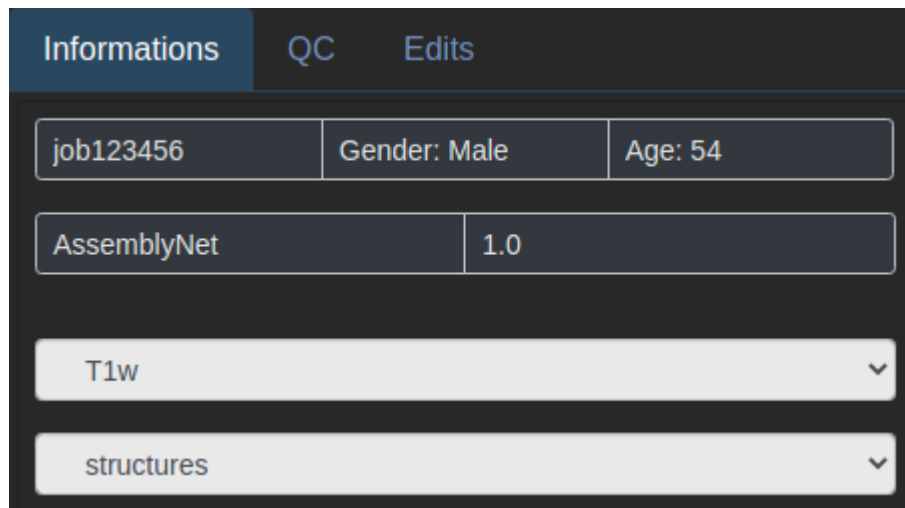
## 2 Développement

### 2.1 Travail réalisé

Dans cette partie, nous allons passer en revue l'ensemble des tâches réalisées. En accord avec les clients, nous avons fait le choix de séparer les fonctionnalités de l'application dans 3 onglets. De ce fait, le plan de cette partie reprend cette logique.

#### 2.1.1 Onglet Informations

L'onglet Information est l'onglet principal de l'application qui est ouvert par défaut lors du chargement de la page. Comme le montre la figure 3, nous affichons des informations telles que l'identifiant de l'analyse, le sexe et l'âge du patient, le nom de la pipeline et la version. Deux listes déroulantes sont également présentes afin de choisir le calque "Image" et le calque "Segmentation" à afficher sur le viewer.



Informations			QC	Edits
job123456	Gender: Male	Age: 54		
AssemblyNet	1.0			
T1w				
structures				

FIGURE 3 – Onglet Informations : détails de l'analyse

Aussi, lors du clique sur une région dans le viewer Papaya, d'autres informations apparaissent (Figure 4) :

- Le nom de la région sélectionnée
- La courbe de normalité de la région et le point du patient placé sur celle-ci : la couleur du point dépend de sa position par rapport à la courbe (vert si le point est à l'intérieur, rouge si le point est à l'extérieur).
- La description de la région et le lien de la page Wikipédia : le bouton description permet de masquer/afficher la description
- La question de satisfaction : si un commentaire existe, les boutons YES et NO sont remplacés par un bouton EDIT

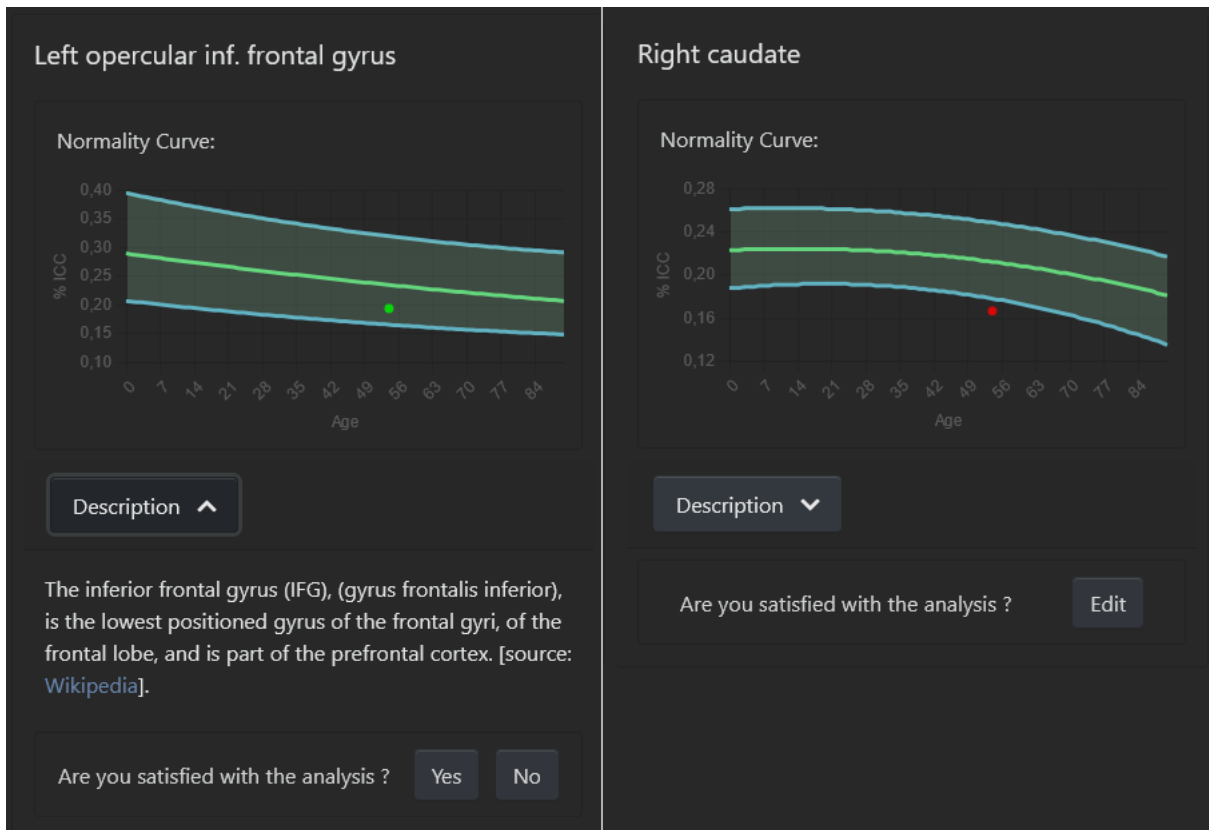


FIGURE 4 – Onglet Informations : informations propres à une région sélectionnée

### 2.1.2 Onglet QC

L'onglet QC (Quality Control) détaille le formulaire de retour. Ce formulaire permet de donner un avis et de laisser un commentaire pour chaque calque et ce, en sélectionnant le calque à commenter via les listes déroulantes. De plus, en fonction de l'avis de l'utilisateur, une liste de problèmes est affichée afin de standardiser les retours pour faciliter leurs analyses.

Si l'utilisateur choisit "Good" alors seule une zone de commentaires est affichée (à gauche sur la figure 5). Si l'utilisateur choisit "Medium" ou "Bad", plusieurs choix sont proposées (à droite sur la figure 5). Les choix de problèmes proposés dépendent de la catégorie du calque.

FIGURE 5 – Onglet QC : Rôle des radio buttons

Enfin, en bas du formulaire, un bouton Send permet d’envoyer le formulaire. Une fois envoyé, un bouton permet de télécharger le contenu de son retour au format CSV.

FIGURE 6 – Onglet QC : Boutons Send et Download CSV

### 2.1.3 Onglet Edits

L’onglet Edit [7] permet l’ajout temporaire de calque .nii ou .nii.gz. Une fois le calque chargé, celui-ci remplace la segmentation active et est visible sur le viewer Papaya. Les nouveaux calques chargés appartiennent à une nouvelle liste déroulante accessible dans les onglets Edit et Informations.



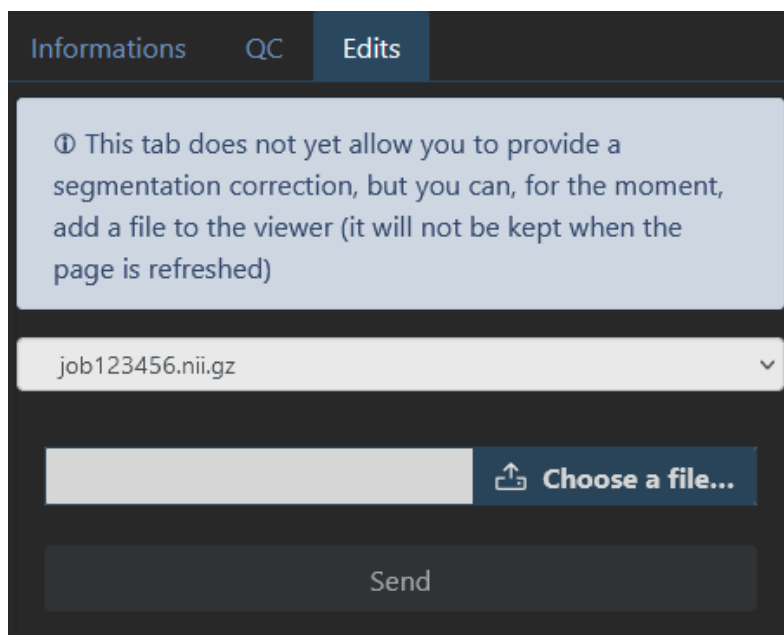


FIGURE 7 – Onglet Edit

#### 2.1.4 Viewer Papaya

Nous avons également modifié et ajouté le comportement et l’affichage du viewer Papaya. La réalisation de ces tâches se traduit par la réécriture de certaines méthodes issues de la bibliothèque Papaya.

Premièrement, nous avons ajouté des raccourcis afin de masquer, afficher et sélectionner un calque. Les touches 1 à 9 permettent de sélectionner les calques respectifs et la touche 's' permet de masquer et afficher le calque actif.

De plus, nous avons ré-affiché certaines fonctionnalités proposées par la bibliothèque Papaya. C’est le cas de la barre de menus, de l’affichage de l’orientation de l’image et des options pour les calques. Nous avons modifié le comportement initial de ces derniers afin que le calque sélectionné soit affiché.

Nous avons également généré de nouvelles tables de couleurs à partir d’un script Python fourni par les clients. Ici, l’objectif était de s’approcher le plus possible des couleurs générées dans le rapport PDF sur le site de VolBrain.

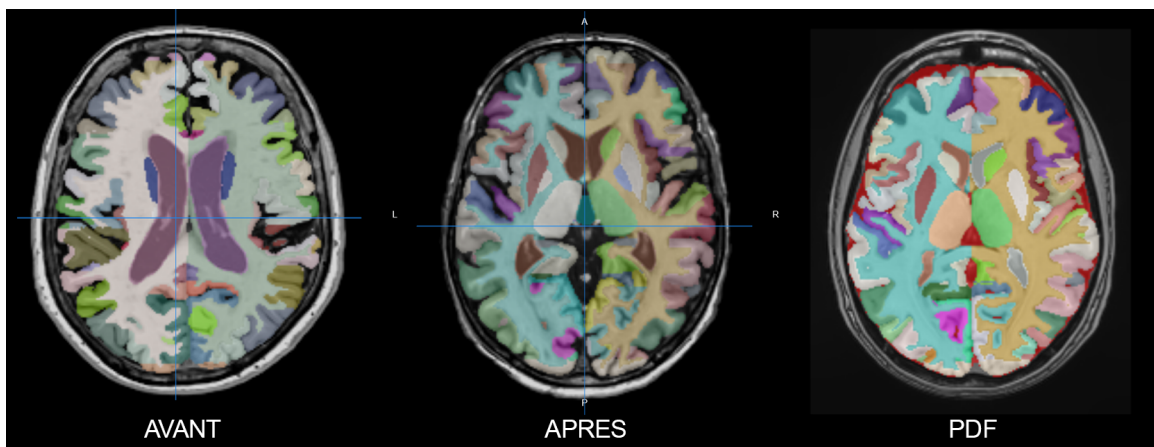


FIGURE 8 – Comparaison des tables de couleurs pour les structures

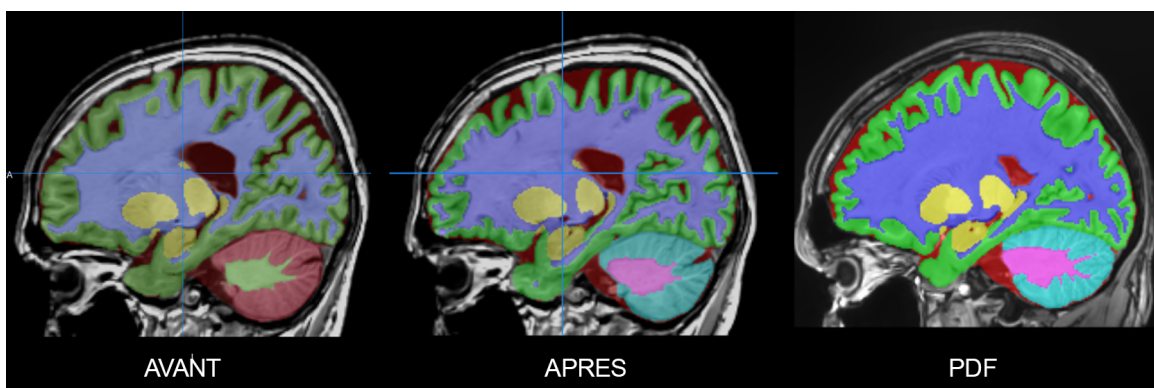


FIGURE 9 – Comparaison des tables de couleurs pour les tissus

### 2.1.5 Autres tâches

Dans cette partie, nous souhaitons évoquer certaines tâches, un peu à part, que nous avons réalisé. C'est le cas des descriptions Wikipédia de chaque région. Initialement, les clients n'avaient pas de description pour les différentes régions du cerveau. Ils nous ont donc demandé de les créer. Pour réaliser cela, nous avons créé un script Javascript récupérant les premiers lignes de la page Wikipédia d'une région si celle-ci existe.

Nous avons également mis en place une convention de nommage pour les fichiers de regions\_mapping et de bounds. Cela a permis de limiter les erreurs lors du chargement des informations d'une région.

Aussi, à la demande des clients, nous avons récupéré la dernière version de Papaya afin de ne pas en utiliser une obsolète.

Enfin, nous avons développé l'application en s'assurant de son bon fonctionnement pour deux pipelines différents : AssemblyNet et DeepLesionBrain.

## 2.2 Architecture

Dans un premier temps, nous devions reprendre l'existant et le mettre en place. Cette condition nous a contraint à créer une architecture autour de l'existant et non l'inverse. En effet, les clients ne nous ont pas fourni l'entièreté du code de la plateforme mais uniquement le code relatif à cette partie visualisation.

De ce fait, l'objectif premier était d'afficher la page de visualisation de VolBrain à l'identique sur nos machines. Pour cela, nous avons créé un serveur Node.js et utilisé le module Nunjucks. Ce dernier est un moteur de template pour JavaScript, et permet à l'aide d'un système de "bloc" d'insérer du contenu dans une page. De plus, il offre également une couche logique permettant par exemple d'ajouter des conditions.

Les données utilisées pour la visualisation, sont celles contenues dans les fichiers CSV et JSON que l'on a cité dans l'étude de l'existant (cf section 1.3). À terme, ces derniers seront transmis au front-end par le back-end. Nous n'avons pas utilisé de base de données et n'avons donc pas de contrôleurs et modèles comme nous comptons le faire à l'origine, nous parlerons d'ailleurs de cela dans la section Erreur (cf section 4.2).

La figure 10 illustre notre architecture et notamment les deux routes que nous détaillerons dans la partie 2.3.2

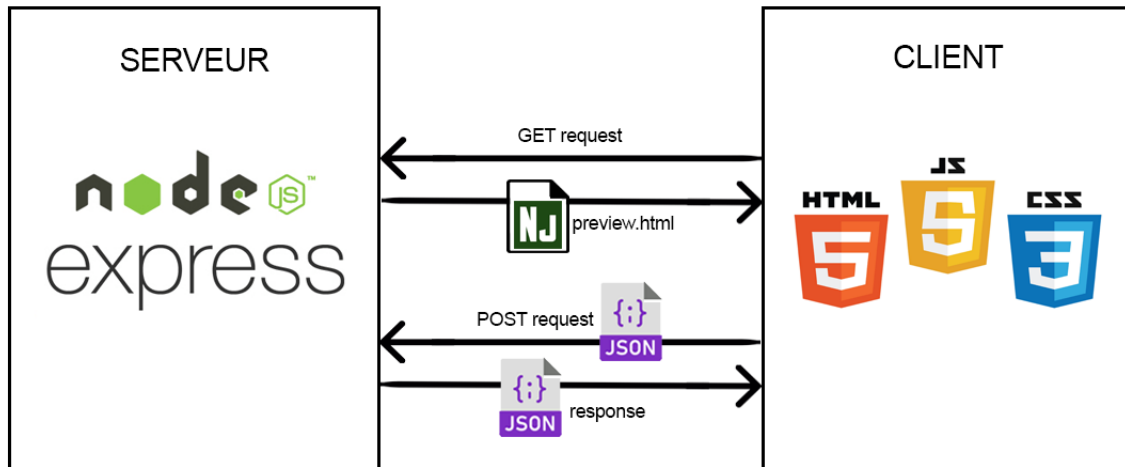


FIGURE 10 – Architecture du projet

## 2.3 Fonctionnement

### 2.3.1 Technologies utilisées

Concernant les technologies, le langage principalement utilisé sur l'ensemble du projet est JavaScript. Concernant le front, nous avons dû nous adapter au `preview.html` existant qui lui était écrit suivant une syntaxe de templating Jinja.

Pour notre serveur Back nous avons mis en place un serveur Rest Node JS avec Express, ainsi que différents modules comme Nunjucks pour la gestion des templates. Nous avons également utilisé des bibliothèques publiques comme :

- Axios pour les appels vers notre serveur (méthode POST détaillé dans la partie suivante).
- Chart.js pour l'affichage des courbes de normalité, puisqu'elle permettait de réaliser l'ensemble des fonctionnalités que les clients souhaitaient (Détails lorsque nous passons notre souris dessus, couleurs, etc...).
- Font-awesome pour différentes icônes.

Enfin, pour réaliser nos tests nous avons utilisé Selenium :

- IDE pour enregistrer des scénarios d'utilisations sous formes de macros.
- Web Driver où cette fois-ci nous codions directement des cas d'utilisations.

### 2.3.2 Routes

Comme indiqué plus tôt, nous utilisons un serveur NodeJs. Celui-ci comporte deux routes :

- GET `/preview?hash=...` : permet d'afficher la page HTML de notre application.  
Après avoir récupéré le hash du job (ou analyse) à afficher dans l'url, nous cherchons l'`input.json` correspondant à ce dernier pour avoir accès aux paramètres nécessaires au "render" de la vue, qui est sous format Jinja.  
Il nous faut par conséquent, toutes les variables propres au module papaya, qui sont regroupés dans la variable "job" créée à partir de l'`input`, ainsi que celle responsable du formulaire retour-client, cette fois-ci nommée "job\_qc".  
Une fois récupérées, le module Nunjucks, cité plus haut, permet à notre template "preview" d'y avoir accès depuis son contexte, ce qui permet de donner cet aspect dynamique à nos vues.
- POST `/save-input` : permet de sauvegarder, dans l'`input.json`, les réponses du formulaire de retour.

Cette fois-ci, il s'agit d'une méthode POST, ce qui implique un envoi de donnée vers notre serveur. Tout comme notre première route, nous avons besoin du hash du job courant pour pouvoir remplacer le champ correspondant aux réponses du formulaire dans le bon input.json. Nous avons fait en sorte, pour cette variable représentant le formulaire, de garder la même structure tout au long de nos modifications provenant d'actions directes sur l'UI (checkbox, etc...). De cette manière, nous pouvons la comparer directement avec l'ancienne lors de l'envoi vers notre back, et mettre à jour un champ booléen "commentExists" (voir 12) lors de la première modification, pour pouvoir ajuster les différents éléments du front.

### 2.3.3 Comportements

Nous souhaitons également détailler certains comportements. Tout d'abord, comme le détaille le diagramme d'activité UML suivant, plusieurs conditions sont nécessaires au bon chargement de la page.

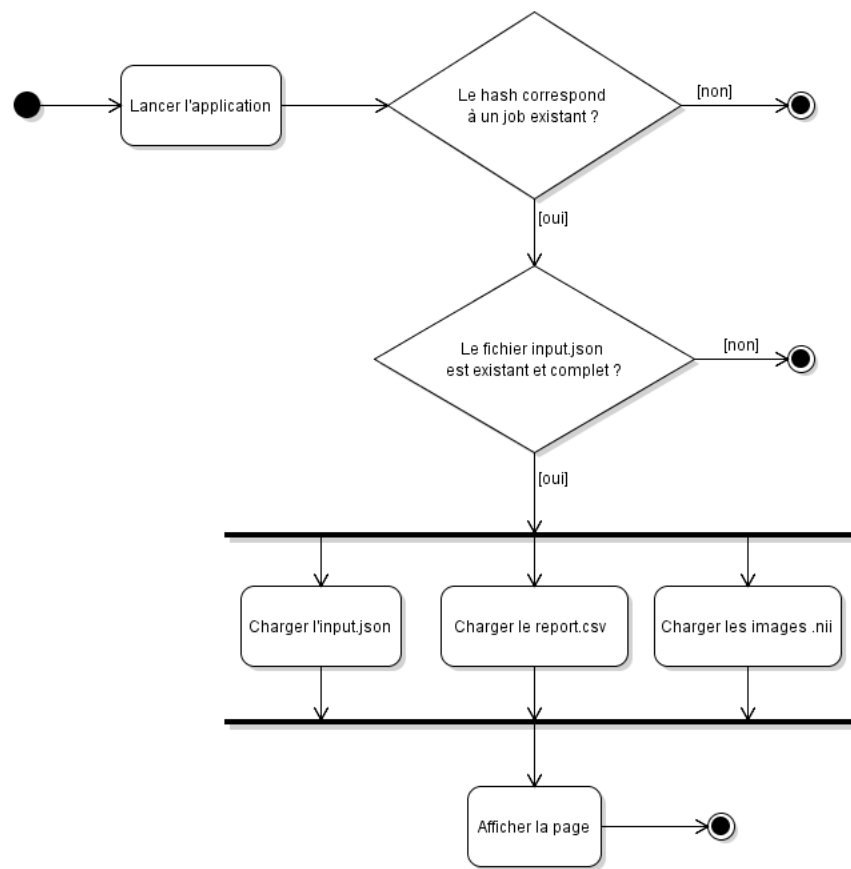


FIGURE 11 – Diagramme d'activité UML : chargement de la page

Pour que la page charge correctement, deux conditions doivent être respectées. D'abord, le hash indiqué dans l'URL doit correspondre à un job connu par l'application. Ensuite, le fichier d'entrée input.json doit être existant et complet c'est à dire qu'il respecte la structure de la figure 12 :

```

"hash": "625e4aeb37529d8a95120c7891620dd8cd0837f5f8a71261f832a2f49105a955",
"pipeline_name": "AssemblyNet",
"pipeline_version": "1.0",
"job_id": "job123456",
"created": "2021-11-24 14:55:39",
"allow_share": true,
"age": "UNKNOWN",
"sex": "UNKNOWN",
"measures": {
  "measure": {
    "filename": "report.csv",
    "name": "volume %",
    "display_name": "volume (in %)",
    "regions_bounds": [
      "bounds_female.csv",
      "bounds_male.csv",
      "bounds_general.csv"
    ]
  }
},
"preview": [
  "image1"
],
"preview_options": [
  {
    "image": "image1",
    "options": {
      "min": 30,
      "max": 300,
      "lut": "Gray Scale",
      "category": "image",
      "name": "T1w",
      "enabled": true
    }
  }
]

```

FIGURE 12 – Squelette de l'input.json

Pour que l'application se lance, tous les champs peuvent être vides mis à part les champs preview et preview\_options.

Une fois la page chargée, certains éléments déterminent ce qui doit être affiché. Par exemple, ce diagramme met en évidence le rôle important que joue l'existence ou non d'un commentaire dans l'affichage de la page.

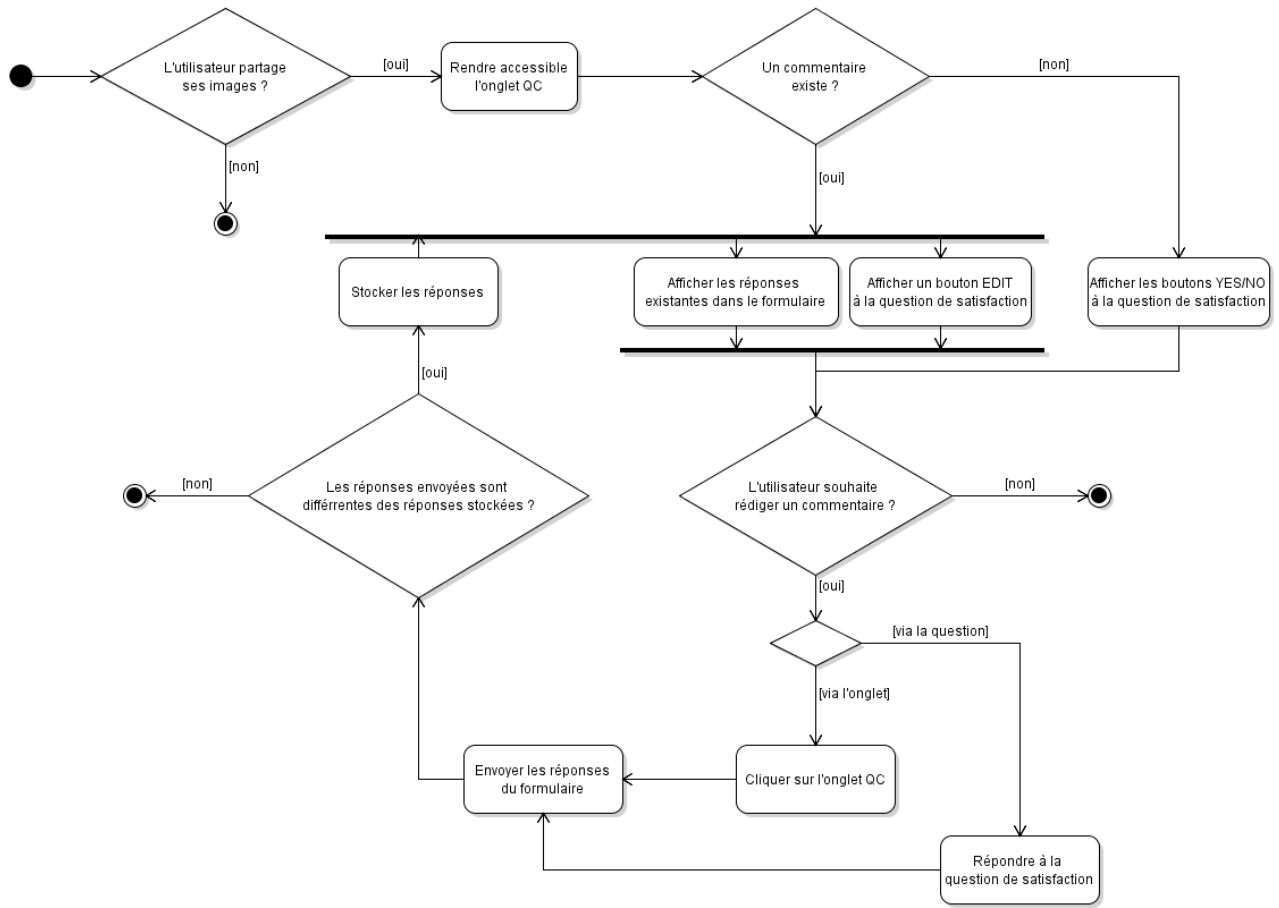


FIGURE 13 – Diagramme d'activité UML : rôle du commentaire dans l'affichage

Le premier critère est le choix de l'utilisateur de partager ses images à volBrain. Si c'est le cas, les éléments de QC sont accessibles. Ensuite, si un commentaire existe, le bouton Edit répond à la question de satisfaction et le commentaire existant est chargé dans le formulaire de retour. Sinon, les boutons "Yes" et "No" répondent à la question de satisfaction. Enfin, lorsqu'un commentaire est rédigé, si celui-ci est différent de celui stocké, il est envoyé et les comportements associés à l'existence d'un commentaire sont répétés.

En résumé, il existe deux types d'utilisateurs : celui qui autorise le partage de ses images et celui qui ne l'autorise pas. Le type d'utilisateur détermine les actions qu'il peut réaliser.



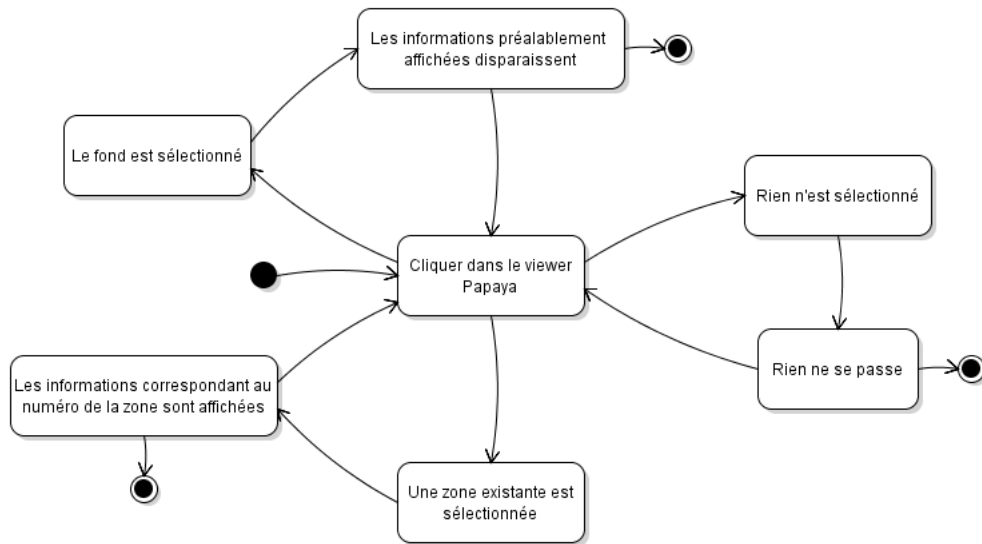


FIGURE 15 – Diagramme d'états UML : Clique sur le viewer Papaya

Le diagramme ci-dessus permet de visualiser l'action attendue lors d'un clique sur le viewer et ce, pour chacun des cas possibles.

Même chose pour le rôle des boutons radios dans l'affichage du formulaire QC :

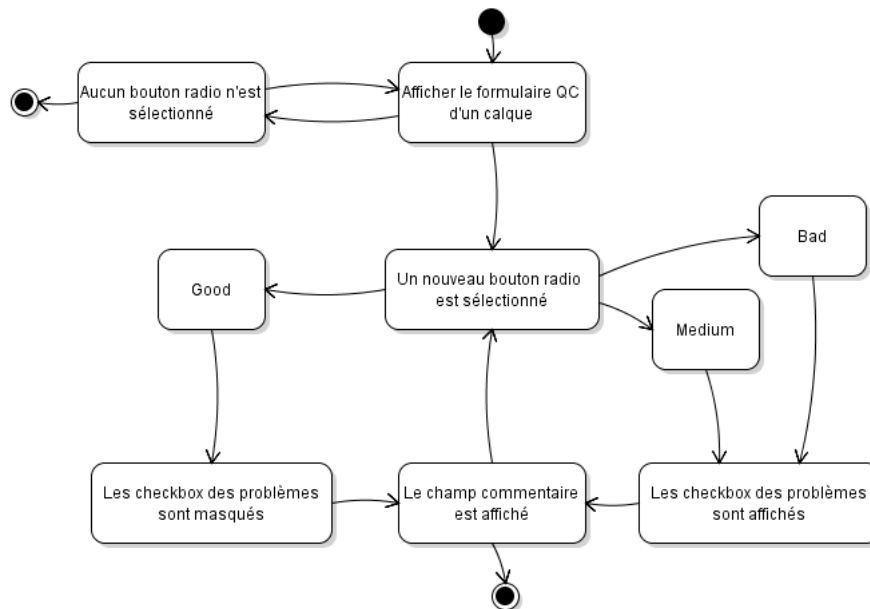


FIGURE 16 – Diagramme d'états UML : Rôle des boutons radios dans le formulaire QC

De ce fait, une fois les différentes cas de figure identifiés, il est beaucoup plus simple de réaliser un test automatique complet.

Voici les tests automatiques que nous avons réalisés :



Catégorie	Intitulé	Technologie
Labels	Les informations affichées correspondent à la région sélectionnée	Selenium Web Driver
Labels	Sélectionner le fond du viewer cache les informations affichées	Selenium Web Driver
Bouton description	Le bouton description cache et affiche le texte de description	Selenium Web Driver
Raccourcis	La touche S cache et affiche le calque actif	Selenium Web Driver
Raccourcis	Les touches 1 à 9 sélectionnent les calques respectifs	Selenium Web Driver
Raccourcis	Les flèches directionnelles changent la position du curseur	Selenium Web Driver
Courbe de normalité	La courbe s'affiche lorsqu'une région est sélectionnée	Selenium Web Driver
QC	Cliquer sur l'onglet QC affiche le formulaire	Selenium Web Driver
QC	Les boutons radios Good affichent un champ commentaire	Selenium Web Driver
QC	Les boutons radios Medium et Bad affichent les checkboxes des problèmes et un champ commentaire	Selenium Web Driver
QC	Le bouton YES redirige vers l'onglet QC et coche les boutons radios Good	Selenium Web Driver
QC	Le bouton NO redirige vers l'onglet QC et coche les boutons radios Medium	Selenium Web Driver
QC	Le bouton EDIT redirige vers l'onglet QC	Selenium Web Driver
QC	Le champ commentaire a une taille maximale de 280 caractères	Selenium Web Driver
Edition	L'image chargée est affichée dans le viewer	Selenium Web Driver
Affichage	La barre menus est affichée	Selenium Web Driver
Affichage	Le bouton pour télécharger le retour au format CSV apparaît	Selenium IDE
Affichage	Les informations sont mises à jour quand le calque change	Selenium IDE
Affichage	Cliquer sur les options d'un calque le rend actif	Selenium IDE

## 2.5 Difficultés

Durant ce projet, nous avons fait face à plusieurs difficultés. Premièrement, nous avons dû mettre en place l'existant qui nous a été fourni par les clients. Ayant très peu d'informations et de fichiers, nous avons dû comprendre comment celui-ci fonctionnait, c'est à dire comprendre comment le fichier HTML était structuré.

Une fois que nous avons compris qu'il s'agissait de templates Jinja, nous avons mis en place Nunjucks sur notre serveur nodeJS, ce qui a permis de les lire et de les transformer en un HTML lisible par un navigateur.

Dans un deuxième temps, une fois que nous avons mis en place notre serveur, nous avons dû comprendre le fonctionnement du module Papaya Viewer. Nous sommes donc allés récupérer le code source de celui-ci afin de pouvoir comprendre comment fonctionnaient les éléments qui nous intéressaient. Par exemple, comment est récupéré l'ID de la région survolée avec le curseur pour pouvoir faire en sorte de le récupérer lors d'un clique.

Enfin, pour la partie sur le retour utilisateur, nous avons rencontré plusieurs autres difficultés. Le but étant de rendre complètement dynamique le formulaire côté front.

Il fallait donc, dans un premier temps, afficher dans les champs correspondants, les données sauvegardées au format JSON dans l'input. Puis mettre à jour cette représentation JSON en fonction des actions de l'utilisateur sur l'interface dédiée.

La principale difficulté provenait des technologies utilisées. En effet rendre dynamique un formulaire HTML pur, uniquement en JavaScript, tout en conservant les différentes fonctionnalités mises en places précédemment, n'était pas évident. Comme par exemple, garder une synchronisation entre les différents

onglets et les calques actifs sur le viewer Papaya.

### 3 Gestion de projet

Pour arriver au résultat global que nous venons de présenter, il a été nécessaire de mettre en place une gestion de projet rigoureuse. Cette gestion de projet est basé sur le manifeste Agile, et plus particulièrement sur la méthode Scrum [5], découpant la période de travail totale en plusieurs périodes appelées sprints. L'ensemble des éléments constituant notre gestion de projet a évolué tout au long de notre travail.

#### 3.1 Organisation

Tout d'abord nous nous sommes mis d'accord sur un plan de travail clair et sur certains rendez-vous clés que nous voulions mettre en place dès le sprint préparatoire permettant une bonne organisation du projet.

##### 3.1.1 Semaine type

La première étape a été d'organiser le déroulement de nos futurs sprints.

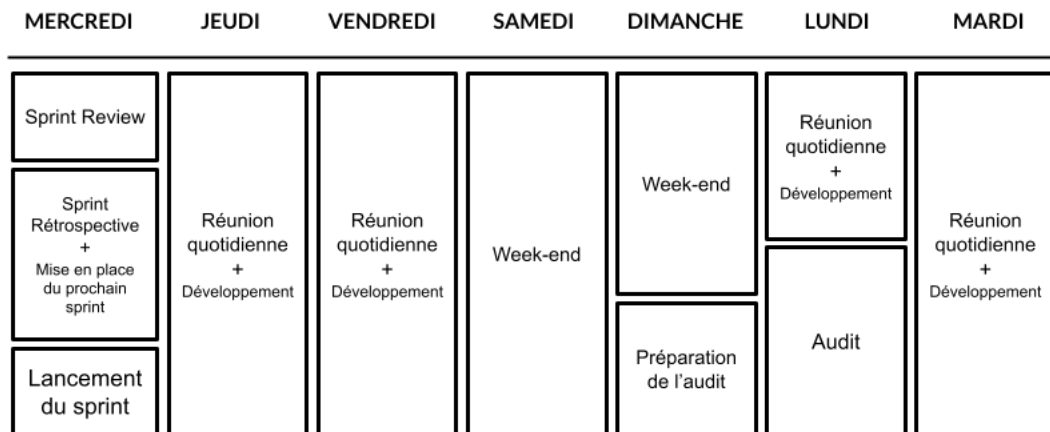


FIGURE 17 – Planning type d'un sprint

Tout d'abord nous lançons les sprints le mercredi en fin d'après midi, et ce pour une durée d'une semaine. Après ce lancement, nous développons le jeudi et le vendredi mais aussi le samedi suivant les disponibilités des membres.

Ensuite, nous préparons la présentation de l'audit de la semaine le dimanche après-midi tous ensemble malgré le fait que seulement deux personnes prenaient la parole. Cela nous permettait de faire un point sur ce qui avait été accompli pendant le sprint et ce qu'il restait à faire. Nous changions chaque semaine d'orateurs. Nous avons ensuite cet audit le lundi après midi. Suite à cela, nous finissons de développer les tâches du sprint le mardi.

Après cela, nous avons rendez-vous avec les clients, généralement le mercredi matin. Après ce rendez-vous, nous mettons au propre les notes faites au cours de la réunion et faisons une rétrospective du sprint.

Enfin, nous prenons une grande partie du mercredi après midi pour préparer l'ensemble du prochain sprint avec la totalité des membres du groupe. Une fois le prochain sprint prêt, nous le lançons et recommençons notre semaine type.

##### 3.1.2 Daily meeting

Lors des phases de développement, nous communiquons beaucoup à l'oral et à l'écrit sur tout le travail que nous réalisons.

En plus d'être rigoureux avec l'utilisation de Jira ou de Git, afin de nous assurer que l'ensemble du groupe savait exactement, et de façon claire, ce sur quoi tous les autres membres travaillaient, où ils en étaient ou encore les problèmes qu'ils rencontraient, nous écrivions l'ensemble des informations importantes à communiquer et nous faisions des appels dès que possible avec un maximum des différents membres pendant la journée.

Nous n'avions pas d'heure précise pour ces appels, nous invitions les autres membres à parler lorsque c'était nécessaire, et suivant les disponibilités de chacun, mais parfois aussi tout simplement afin de travailler en groupe. Ce processus permettait de travailler à plusieurs sur une seule fonctionnalité lorsque c'était nécessaire ou lorsqu'une personne voulait apporter son aide, ou encore d'être plus efficace et motivé que si chaque membre travaillait seul.

### 3.1.3 Sprint Review

Nous avons décidé de rencontrer les clients chaque semaine. Pour ce faire, nous envoyions un planning de nos disponibilités afin d'avoir un créneau qui convenait, mais rapidement, nous nous sommes mis d'accord sur un rendez-vous fixe le Mercredi à 10h, ce qui s'adaptait très bien à l'organisation de notre semaine.

Durant la review avec les clients, nous leur présentions notre travail réalisé durant le sprint. Par la suite, ils nous faisaient un retour sur les possibilités d'améliorations ou d'ajustements. Enfin, nous en profitions pour leur poser plusieurs questions afin d'avoir un maximum d'informations sur les besoins pour pouvoir les atomiser au mieux.

### 3.1.4 Sprint Retrospective

Lors de la fin de nos sprints, nous effectuons une rétrospective. Pour ce faire, nous utilisons des post-its et la méthode "Starfish".

Nous commençons toujours par une "météo" de l'humeur par rapport au sprint. Celle-ci nous permettait de savoir si le sprint s'était bien déroulé ou non pour les différents membres du groupe, et de connaître leur état d'esprit vis à vis du projet. Par la suite, chaque membre listait ce qu'il fallait continuer de faire pour les prochains sprints. De même pour ce qu'il fallait arrêter ou alors approfondir. Enfin la dernière catégorie correspondait à ce qui pourrait être ajouté.

On finissait la rétrospective par la noter elle même afin de savoir si elle était bénéfique ou non.

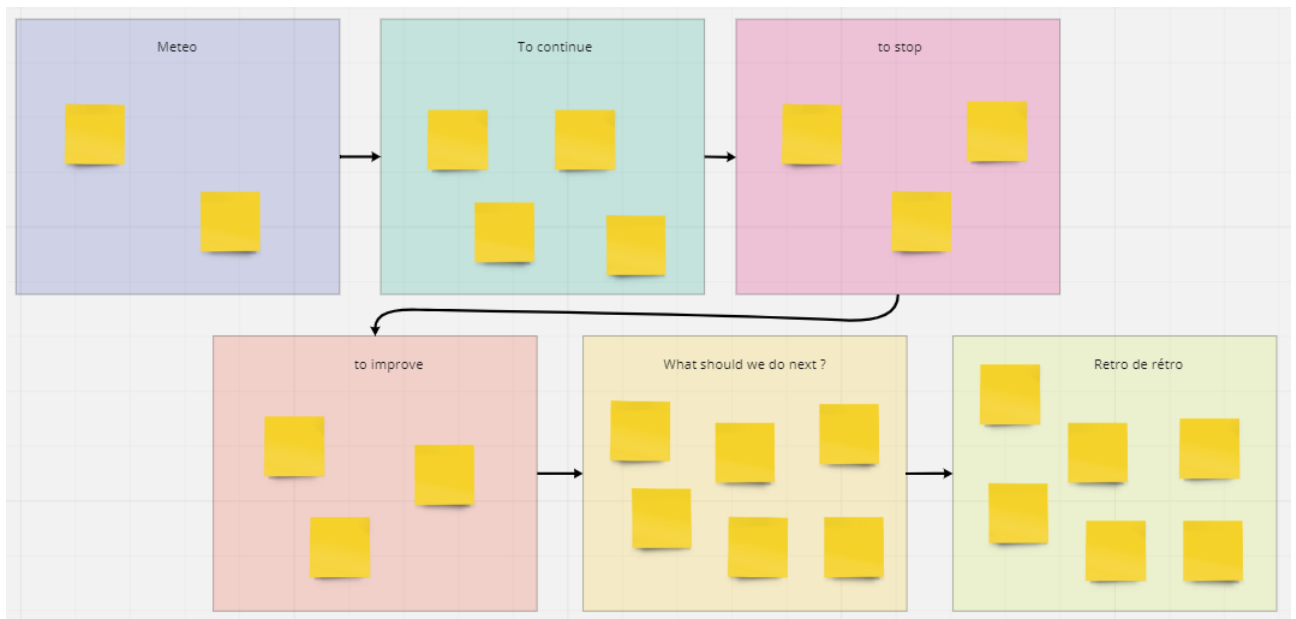


FIGURE 18 – Template utilisé pour nos rétrospectives

### 3.1.5 Sprint Planning

Enfin, après le sprint review le matin même suivi de la rétrospective et la fin du sprint, nous organisons tous ensemble le prochain sprint dans l'après midi.

Nous avons choisi de faire cet exercice ensemble afin que tous les membres puissent améliorer leurs compétences à ce sujet, mais aussi afin d'éviter au maximum les erreurs ou encore pour répartir la charge de travail pour cette préparation qui est nécessaire et utile mais conséquente. Cette préparation de sprint consistait à reprendre nos notes du sprint review (mise au propre avant ce rendez-vous préparatoire) avec l'ensemble des besoins exprimés par les clients pour créer une liste de tâches à faire pendant le projet sur l'outil Jira dont nous parlons juste après dans ce rapport.

Nous listions d'abord l'ensemble des futures tâches à effectuer, puis nous les attribuons au maximum. Lorsque cette étape a été effectuée, nous écrivions l'ensemble des descriptions, les critères d'acceptations, les définition of done (DoD) et choissions un ordre de priorité pour chacune des tâches. Après cela, nous mettions en place une évaluation des différentes tâches grâce à un système appelé planning poker. Enfin, après avoir évalué les différentes tâches à effectuer, et connaissant la vitesse approximative de l'équipe, nous choissions la liste des tâches que nous allons effectuer lors du prochain sprint, puis démarrons directement ce sprint en fin de journée en lui attribuant un objectif précis.

## 3.2 Jira

Jira est un outil de gestion de projet permettant de travailler sur des projets en méthode agile. L'outil permet aux équipes de s'organiser et de visualiser le projet grâce à un suivi visuel des tâches avec un backlog, un tableau de bord, un tableau agile et bien d'autres fonctionnalités. Jira propose des analyses de projet et/ou de sprint avec des burnups par exemple.

### 3.2.1 Découpage du projet

Il est fortement conseillé, de diviser un projet en plusieurs objectifs accessibles et réalisables en un temps raisonnable.

L'outil Jira, permet de découper un projet en plusieurs grandes tâches nommées epics, dans notre projet, nous avons des epics tel que "Mettre en place le front", "Gérer un système de retour pour une analyse", "Améliorer la partie QC" et bien d'autres.

Ces epics peuvent être divisées en tâches plus petites, appelées stories ou user stories.

Les user stories sont des tâches plus spécifiques. Nous avons choisi de les composer d'une description, de tests d'acceptations, d'une définition of done, d'une priorité, d'un responsable, possiblement de participants et de story point.

Les user stories peuvent si besoin être subdivisées en sous-tâches. L'objectif est de découper le projet en tâches simples et facilement exprimables par une phrase, qui pourront être compréhensibles par les personnes qui les effectueront.

### 3.2.2 User Story

Une user story est une tâche simple ou un regroupement de sous-tâches simples, qui peuvent être exprimé par une phrase claire, de la forme "En tant que ... je souhaite ... afin de ...", elle permet de décrire la tâche de façon suffisamment précise afin de la comprendre et de pouvoir la réaliser.

Par exemple "En tant qu'utilisateur, je souhaite pouvoir télécharger un rapport CSV du formulaire que j'ai rempli pour noter une segmentation".

Les user stories sont composées de critères d'acceptations, ils permettent d'exprimer les limites de la tâche. Cela aide à comprendre ce qu'elle doit faire et ce qui n'est pas de sa portée.

Ils peuvent être par exemple :

- Respecter le design (couleur, format texte, background)
- Le bouton permet de télécharger un CSV
- Le CSV téléchargé contient les informations saisies dans le formulaire
- Le bouton est visible seulement si une évaluation a été saisie

La priorité d'une tâche se calcule en fonction de l'importance de la livrer, si elle a de la valeur pour les clients alors elle va avoir une importance élevée.

Il est possible de mettre un ordre de priorité avec la méthode MoSCow

- Must Have, doit être réalisé
- Should Have, doit être réalisé si possible
- Could Have, pourrait être réalisé
- Won't Have, ne sera pas réalisé et ne rentrera pas dans le backlog, pour le moment

Par défaut une user story peut être attribuée à une personne, mais lors de notre utilisation, nous nous sommes aperçu qu'il était possible de travailler à plusieurs sur une même tâche. Nous avons alors, ajoutés un attribut "Participants" permettant de rajouter les personnes qui ont contribué à sa réalisation.

### 3.2.3 Definition of done

La Definition of done (DoD) est une liste de critères, mise en place par l'équipe Scrum, qui permet de définir à partir de quand une story est traitée. Une DoD rajoute de la cohérence et de la qualité au projet.

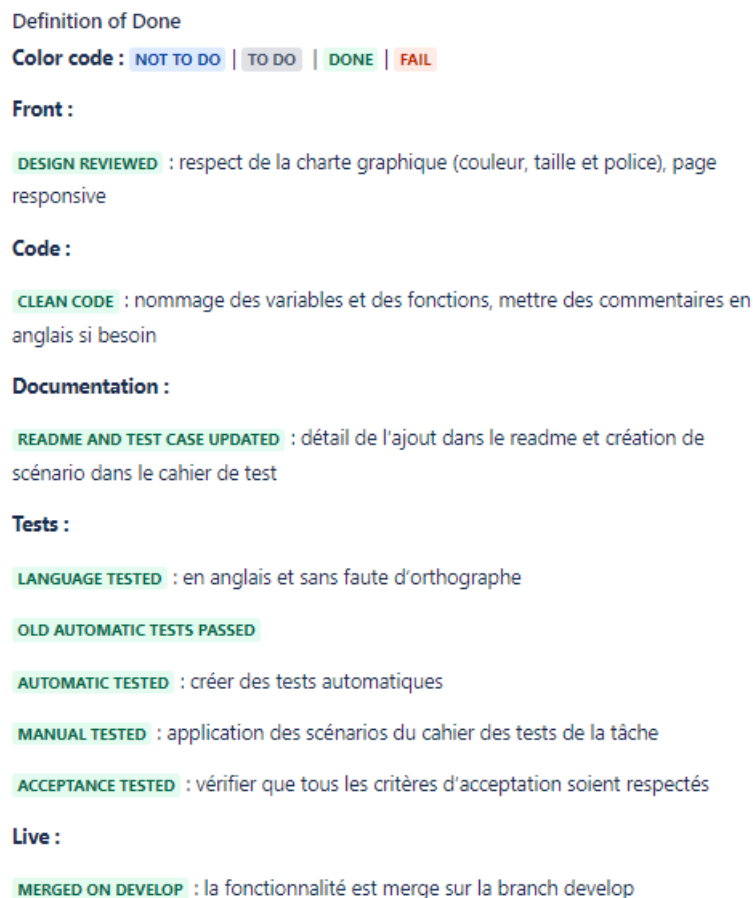


FIGURE 19 – Capture d'écran de notre Definition of Done

Comme nous pouvons le voir dans la figure 19, nous avons mis en place une DoD avec un code couleur et une liste de critères. Ces règles sont propres à notre équipe Scrum sur le projet Papaya. Ces règles de gestion seront différentes selon les projets et les équipes.

Nous avons mis en place un code couleur, permettant de savoir dans quel état étaient les critères. Ils peuvent être à "NOT TO DO" si le critère n'est pas applicable à la tâche. Par exemple, une tâche

du back-end n'aura pas besoin d'une revue du design. Lors du lancement du sprint, tous les critères sont à l'état "TO DO", ils n'ont pas encore été validés. Afin de passer la tâche à l'état fini, il faut qu'elle respecte tous les critères de la DoD. Si elle respecte le critère alors nous pouvons le passer à "DONE", sinon nous le passons à "FAIL" et la tâche va devoir être reprise et modifiée afin de respecter la DoD.

- Une tâche qui modifie le front-end doit s'assurer que le nouveau design respecte la charte graphique au niveau des couleurs, de la taille, de la police et que la page reste responsive.
- Dans le code, il a été mis en place une convention de nommage pour le nom des variables et des fonctions (respecte le clean code[7]). Il est également demandé de mettre des commentaires en anglais.
- Nous avons mis en place un critère qui nous demande de rédiger de la documentation. Dans le README, pour suivre ce qui a été fait au cours du sprint courant, mais également dans un cahier de tests en faisant des scénarios pour la tâche réalisée.
- Plusieurs critères de tests existent dans notre DoD, ce qui renforce la qualité et confiance dans le projet. Nous avons un test sur le langage, il veille à ce que tout soit en anglais et sans fautes d'orthographe. Tous les tests automatiques, des tâches réalisées précédemment doivent être valides, cela correspond à nos tests de non-régression. Ensuite, nous devons mettre en place des tests automatiques avec Selenium. Des tests manuels ont également été faits. Ils reprenaient les scénarios du cahier de recette. Il fallait également que les critères d'acceptations soient respectés.
- En dernier critère, quand tous les points ci-dessus étaient respectés, il fallait fusionner la branche sur laquelle nous avons ajouté la fonctionnalité à develop.

### 3.2.4 Planning poker

Juste après le découpage des tâches, et avant le commencement de chaque sprint, nous faisons un planning poker pour estimer la difficulté (i.e. le nombre de story points) des "User Stories". Nous utilisons notamment un outil en ligne [6], comme nous pouvons le voir sur la figure 20, pour faciliter l'échange ainsi que la suite de Fibonacci comme échelle de notation. De cette manière, nous pouvions aisément débattre, en cas de désaccord sur l'attribution de point à une tâche et trouver un terrain d'entente après coup.

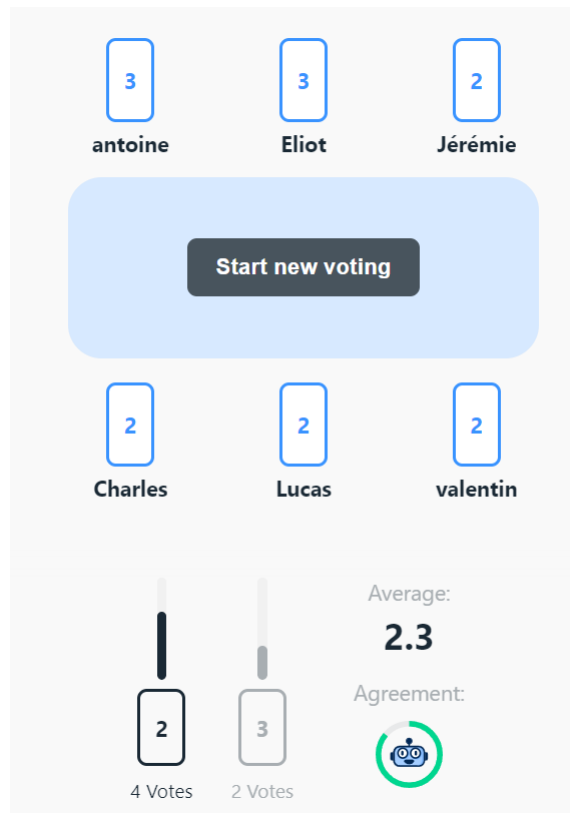


FIGURE 20 – Planning poker

### 3.2.5 Tableau Scrum

Jira offre un tableau Scrum modulable et paramétrable (cf figure 21). Nous avons mis en place un workflow (cf figure 22), il a permis de mettre en place un processus de gestion qui va permettre de mener à bien les tâches.

Les tickets vont passer d'un état à un autre au moyen de transitions entre les colonnes. De notre côté, nous avons fait le choix, lors du démarrage d'un sprint de mettre les tâches dans la colonne "A FAIRE", lorsqu'une personne se met responsable d'une tâche et veut commencer sa réalisation alors la tâche doit passer dans l'état "EN COURS".

Une fois que la tâche est faite, il va falloir la tester, le ticket va aller dans la colonne "A TESTER" jusqu'au moment où nous réalisons les tests, ce qui va permettre de la déplacer dans la colonne "EN COURS DE TEST". Après avoir été testée et que la DoD est respectée, nous pouvons et seulement à ce moment considérer que le ticket est "FINI". Mais si un ticket ne passe pas les tests alors il va être remis dans "EN COURS" afin d'être revu et amélioré pour passer les tests.

Sur les colonnes, nous avons également mis en place un système de tâche maximal, une colonne était limitée à 6 tickets en simultané. Il était en réalité possible de rajouter des tickets, mais la colonne était mis en sur-brillance. Cela nous a forcés à faire nos tests après qu'une tâche soit réalisée. En effet lors du premier sprint, nous avons créé des embouteillages dans la colonne "A TESTER" c'était dû au fait que nous n'avions pas mis complètement en place les tests.

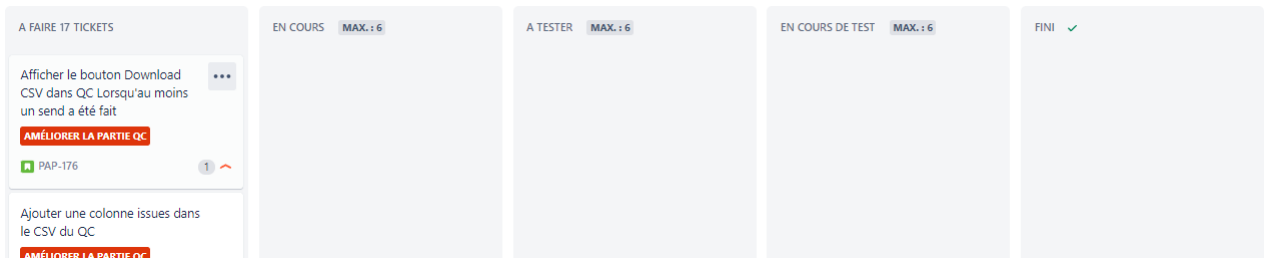


FIGURE 21 – Tableau Scrum

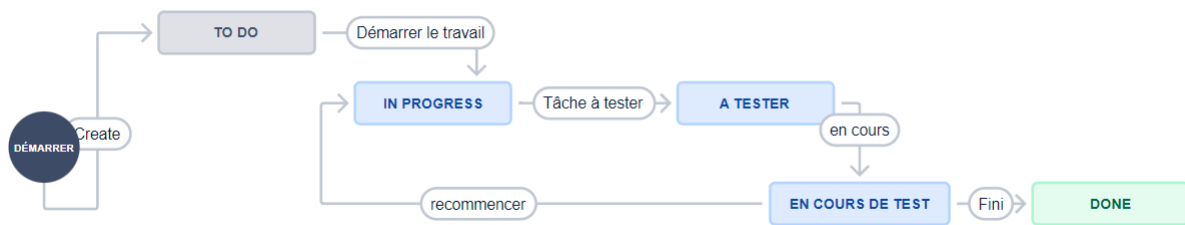


FIGURE 22 – Work flow jira

### 3.2.6 Burnup

Le Burnup est un outil graphique, qui montre l'avancement de l'équipe sur une itération. Il met en évidence l'avance ou le retard dans le développement et permet de faire de la prédiction sur l'évolution du projet. C'est un outil de communication, il va montrer l'avancement et les variations à l'équipe, mais également aux parties prenantes.

Lors de notre premier sprint, qui prenait en compte le sprint préparatoire et l'implémentation du MVP. Comme nous pouvons le voir sur la figure 23, notre burnup n'était pas bon. Il y avait plusieurs raisons à cela, la première était que de nombreuses tâches avaient déjà été réalisées avant le lancement du sprint, nous pouvons le voir sur le burnup qui augmente, dès le lancement, de près de 35 story points. Il y avait également le découpage et l'attribution de story points qui auraient pu être améliorés. On peut également voir que le sprint a été coupé, car il ne correspondait plus à l'objectif du sprint.

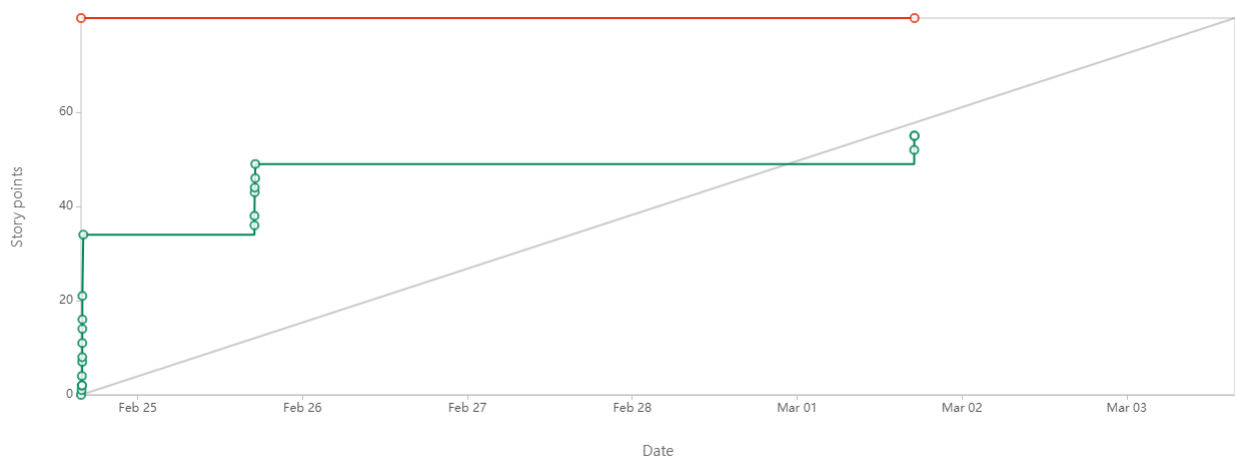


FIGURE 23 – Burnup sprint 1



Les tâches de notre sprint 4 étaient mieux découpées et il y avait une meilleure cohérence dans l'attribution des story points (cf figure 24). Au cours des itérations, l'équipe était plus en mesure de connaître la charge de travail qu'elle pouvait supporter et réaliser. On peut tout de même critiquer le burnup du sprint 4, il y a un plat durant deux jours, il correspond au week-end. Si les sprints avaient été plus longs, par exemple 1 mois, les plats des week-ends n'auraient pas été autant visibles.

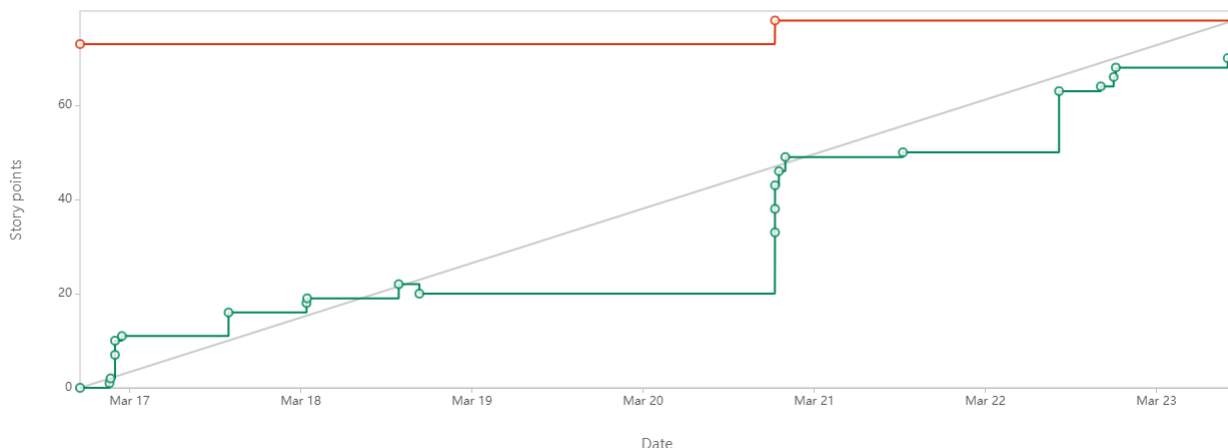


FIGURE 24 – Burnup sprint 4

### 3.3 Git

Git est un logiciel de gestion de versions décentralisé très populaire depuis quelques années permettant le développement et la mise en commun de fichier pour des projets informatiques en groupe. Pour ce projet, nous avons travaillé avec l'outil GitLab nous permettant l'utilisation de Git de façon privée, car le GitLab utilisé est hébergé directement par le centre de Ressources pour l'Enseignement des Mathématiques et de l'Informatique (CREMI) de l'université de Bordeaux.

#### 3.3.1 Git flow

Nous avons basé notre gestion de Git sur le workflow Gitflow. Pour cela, nous avons mis en place deux branches principales, une branche *main* contenant une version stable pour le client, et une branche *develop*, contenant les éléments en cours de développement. Ensuite, nous avons créé une branche pour chaque nouvelle fonctionnalité, que nous avons merge sur *develop* une fois cette dernière est terminée et testée. La branche *develop* a, quant à elle, été merge sur la *main* à chaque fin de sprint.

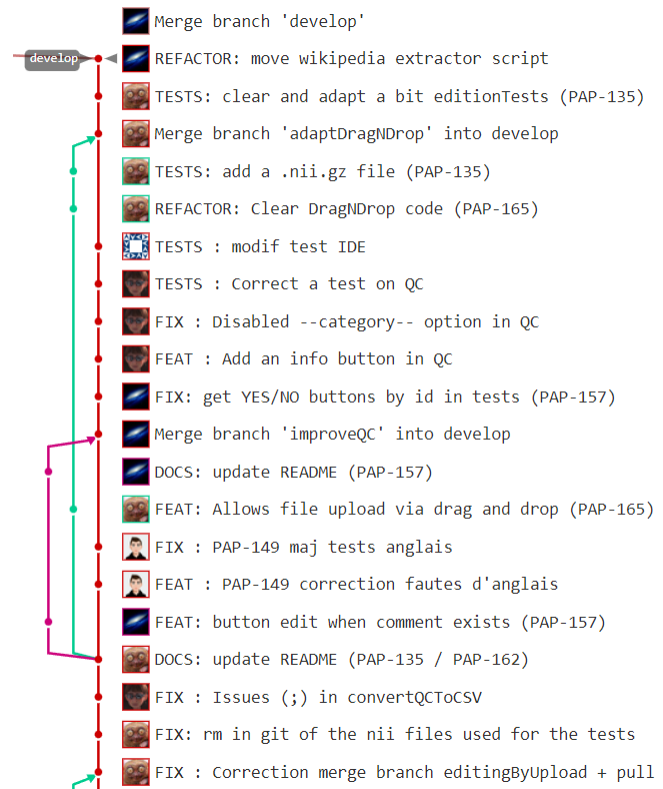


FIGURE 25 – Git flow

### 3.3.2 Convention de nommage

À partir du sprint 3 nous avons mis en place une convention concernant le nommage des commits pour y voir plus clair. Ainsi, nous faisons en sorte que chaque commit soit écrit sous la forme :

— <type> : <sujet> (PAP-X)

Où les types disponibles sont les suivants :

- DOCS : Lié à la documentation (ex : README ...)
- FEAT : Ajout d'une fonctionnalité
- FIX : Correction de bug
- REFACTOR : Changement du style du code (sans changement de la logique)
- TESTS : Ajout ou Modification de tests

Il va sans dire qu'après avoir mis tout cela en place, la relecture de commits et la compréhension de ces derniers était beaucoup plus simple et rapide.

## 3.4 Outils de test

### 3.4.1 Selenium WebDriver

À partir du deuxième sprint, nous avons initié la mise en place de tests automatiques. Pour cela, nous avons commencé par utiliser Selenium WebDriver, qui permet via l'écriture de code, de piloter un navigateur, et donc de simuler une suite d'interactions sur une application web. Ainsi, nous avons réparti les tests par fonctionnalités, dans plusieurs fichiers. Le principal *mainTest.js* fait appel à tous les autres lors du lancement des tests. Ceux-ci peuvent être exécutés à l'aide de la commande :

```
npm run tests x
```

(où x est l'identifiant de la pipeline à tester). Et ils produisent une sortie lisible et facilement interprétable :

```

✓ tests click on Go to center passed !
✓ tests click on arrow down passed !
✓ tests Tabs passed !
✓ tests QC Question Top passed !
✓ tests QC - Image - Good button passed !
✓ tests QC - Image - Medium button passed !
✓ tests QC - Image - Bad button passed !
✓ tests QC - Image - Bad and Good button passed !
✓ tests QC - Image - question text passed !
✓ tests QC - Image - checkbox passed !
✓ tests QC - Image - text area passed !
✓ tests QC Segmentation - Good button passed !
✓ tests QC Segmentation - Medium button passed !
✓ tests QC Segmentation - Bad button passed !

Description button tests passed !
=> shortcut "s" passed
=> shortcut "1" passed
=> shortcut "2" passed
=> shortcut "3" passed
=> shortcut "4" passed
=> shortcut "5" passed
=> shortcut "0" passed
=> shortcut "0" passed
=> shortcut "0" passed
=> shortcut "0" passed
=> shortcut "0" passed
=> shortcut "0" passed
Overlays tests passed !
=> background hide passed

```

FIGURE 26 – Affichage du terminal après le lancement des tests webdriver selenium

### 3.4.2 Selenium IDE

Par la suite, nous avons aussi réalisé certains tests à l'aide de Selenium IDE. Celui-ci se présente sous la forme d'une extension de navigateur et permet de mettre en place des tests plus facilement et plus rapidement. Pour cela, il suffit, depuis l'extension, de lancer un enregistrement et de faire les différentes manipulations souhaitées directement sur l'application. Suite à cela, les tests peuvent être sauvegardés dans un fichier .side et lancés directement depuis l'IDE. Nous ne l'avons pas mis en place mais il est possible de lancer les tests du .side directement en ligne de commande grâce à un package node.

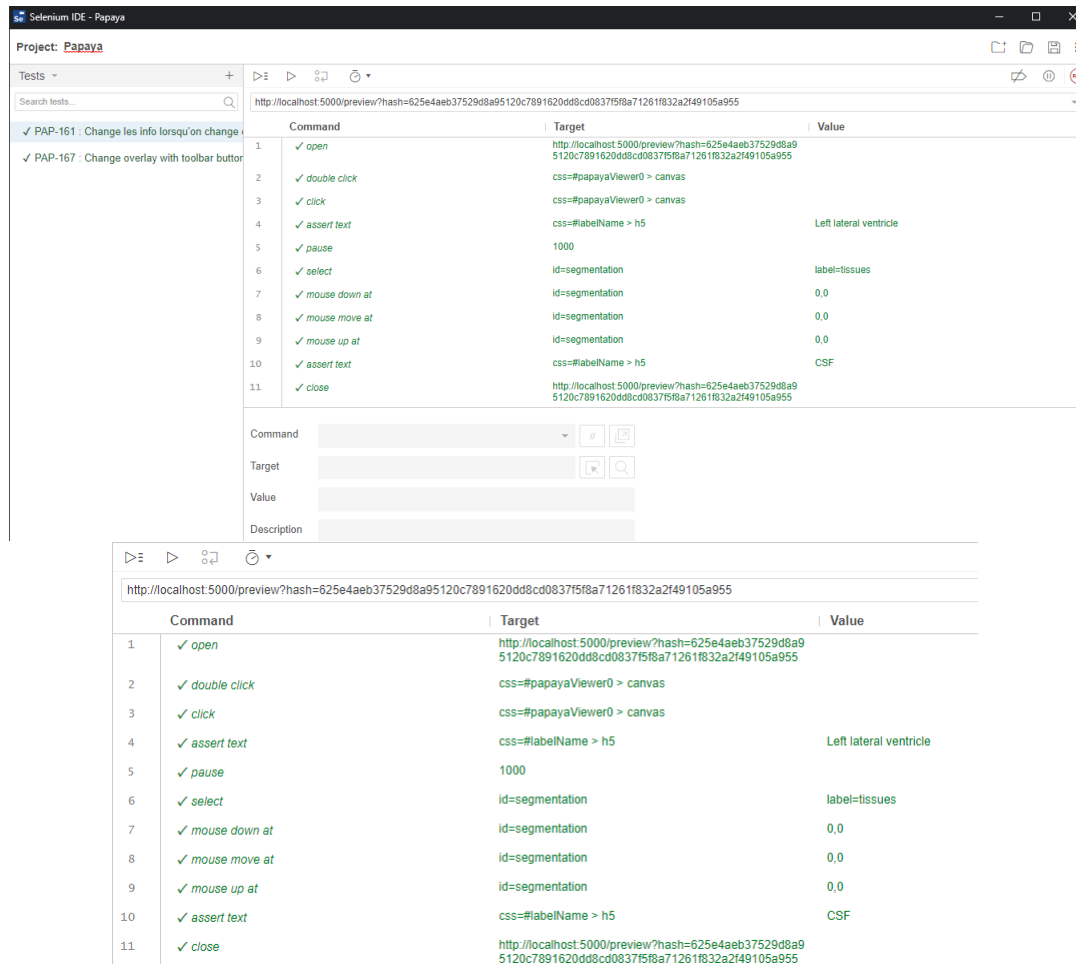


FIGURE 27 – Interface des tests selenium IDE

### 3.4.3 Cahier de recette

Afin de compléter les tests automatiques mis en place avec Selenium, nous avons décidé de faire un cahier de recette recensant une suite de tests manuels. Et ce, pour confirmer les tests effectués et afin de lever toutes ambiguïtés sur la phase de test. Cependant, une partie de ces tests était seulement manuelle, car nous ne pouvions pas les contrôler avec Selenium, mais nous aurions dû utiliser un autre outil afin de les automatiser.

Le fichier se présentait de cette manière :

Id task Jira	Action	Attendu	Résultat	Date	Testeur

La première colonne, permettait d'identifier la tâche mise en cause, et la deuxième à décrire l'action à réaliser pour le test. Ensuite, dans la troisième colonne, nous notions le résultat attendu suite à l'exécution de l'action, et si celle-ci se réalisait sans défauts, nous écrivions "OK" dans la colonne qui suit. Cependant, si un problème était relevé, nous l'inscrivions dans cette même colonne et le test devait être refait, après correction, jusqu'à ce qu'il passe correctement. Enfin, les deux dernières colonnes correspondaient à la date et à l'exécutant du test.

## 4 Bilan

### 4.1 Dette technique

Tout au long du projet, nous avons essayé, au maximum, de limiter la dette technique. Pour cela, nous avons commencé par nous imposer des règles de codage afin de produire un code le plus propre et clair possible. Celles-ci ont été notamment définies dans la Definition of Done, afin de proscrire les mauvaises habitudes. Nous avons fait attention, par exemple, à nommer les méthodes et variables correctement et à ajouter des commentaires aux fonctions qui en nécessitaient, dans un anglais compréhensible. Ensuite, lors du choix des technologies à utiliser, nous nous sommes basés sur l'existant et sur les différents besoins émis par les clients, afin d'opter pour celles qui ajoutaient le moins de dépendances, ou qui étaient les plus faciles à maintenir. Par exemple, pour tracer les courbes de normalité, nous avons utilisé la bibliothèque Chart.js, qui est une des plus populaires. Et enfin, nous avons mis en place des tests à l'aide de Selenium ou encore d'un cahier de recette, dont nous avons parlé précédemment, afin d'éviter les bugs et de ne pas régresser durant le développement.

Cependant, bien que nous ayons mis cela en place, une dette s'est installée, notamment à cause du fait que nous n'avons pas l'exacte connaissance du fonctionnement de l'existant. Par exemple, nous ne connaissons pas l'arborescence de l'application originelle, donc certains de nos développements pourraient poser problème lors de la fusion, comme les chemins utilisés pour les fichiers d'entrées.

De plus, nous avons relevé une dette déjà présente dans l'existant. Les fichiers CSV fournis et utilisés manquent de cohérence, car ils n'utilisent pas les mêmes conventions de nommage. Par exemple, certains utilisent des acronymes alors que d'autres emploient les mots complets. Mais cela n'étant pas de notre fait, nous avons fourni toutes les différences relevées aux clients.

### 4.2 Erreurs

Lors du projet, notre première erreur principale était la réalisation de tâches en dehors du sprint courant.

Par exemple, au début du projet, nous avons mis en place tout notre back (serveur Node JS) ainsi qu'une base de données MySQL externe pour gérer le stockage d'informations du projet (les retours clients Volbrain de la page Preview).

Tout ceci a donc été fait avant même de rencontrer les clients et d'échanger sur les différents besoins du projet, et par conséquent, avant même le sprint préparatoire.

En revanche, après avoir discuté avec eux, nous nous sommes vite rendu compte qu'une base de données externe n'était pas nécessaire, et qu'ils préféreraient utiliser des fichiers JSON, pour rester conforme avec leur architecture.

Bien que cette perte de temps ne nous ait pas vraiment impactés étant donné que cela a été fait avant même que le projet ne soit mis en place, il faut reconnaître que, dans un milieu professionnel, l'aspect

financier et la perte associée ne sont pas à négliger.

Suite à cette erreur, nous avons fait en sorte de ne pas réaliser de nouvelles tâches conséquentes avant de consulter les clients ou en dehors de sprints.

Un autre point qui nous a fait défaut, est le fait d’attribuer toutes les tâches aux membres du groupe dès le début d’un sprint.

Nous avons essayé de mettre ce processus en place lors du sprint 2, et nous nous sommes très rapidement rendu compte que ce n’était plus à refaire puisque cela va à l’encontre de la méthodologie Agile. En effet, nos estimations de difficultés des U.S n’étaient pas forcément toutes correctes. Plus de flexibilité au niveau de la prise de tâche aurait aussi été adaptée, puisque certaines d’entre elles étaient ”verrouillées” par des membres du groupe.

Nous avons donc bien évidemment arrêté cette méthode lors des sprints suivant, et sommes retournés à une répartition dynamique des tâches.

## 5 Conclusion

### 5.1 Résumé

Le travail réalisé avait pour but d’améliorer une partie de la plate-forme Volbrain, outil de segmentation, d’analyse et de visualisation d’images médicales en ligne. La partie sur laquelle nous avons travaillé était la partie ”Preview NII files” correspondant à la visualisation d’images médicales après segmentation et analyse par l’intelligence artificielle de Volbrain. Cette visualisation des images au format nii s’est faite grâce au module Papaya. Les objectifs principaux que nous avons mis en place étaient de récupérer cette page ayant déjà le module Papaya intégré, permettre à l’utilisateur d’avoir un maximum d’informations correspondantes à une zone du cerveau sélectionnée, via le module Papaya et de façon claire, et de créer un formulaire de satisfaction complet de retour utilisateur. Pour arriver au résultat final obtenu, nous avons dû adopter une gestion de projet précise et utiliser certains outils rigoureusement. Il a aussi été très important de mettre en place un maximum de tests pour assurer les différents ajouts apportés. Le produit final est donc un produit fini, livrable et, d’après eux, satisfaisant les besoins de nos clients en plus des bonnes interactions entre nous et eux pendant ce projet.

Ce projet s’est déroulé sur 2 mois, avec un groupe de six personnes toutes motivées, ce qui nous a permis d’avancer rapidement et de mettre en place l’ensemble des besoins attendu par les clients, tout en mettant en place une gestion de projet rigoureuse. D’après nous, ce projet de fin d’étude est la consécration de notre apprentissage de l’informatique et a été extrêmement instructif.

### 5.2 Apprentissages

Ce projet nous a donc permis de nous améliorer sur certains points techniques, mais surtout, sur un grand nombre de méthodes et d’outils liés à la gestion de projet.

Nous avons appris à mettre en place l’ensemble des différents rendez-vous du contexte Scrum, c’est-à-dire la planification de sprint, les daily meeting, les sprint review et les rétrospectives. Nous nous sommes grandement améliorés pour la préparation de sprint, principalement sur l’exercice de découpage des différentes tâches à effectuer. Nous avons aussi pu développer nos compétences avec différents outils. Nous avons pu mettre en place comme dit précédemment un git suivant le workflow gitFlow, ce qui nous a permis d’accroître nos compétences Git. Nous avons également appris à utiliser en profondeur l’outil Jira, comme nous venons de le présenter dans ce document. Un autre aspect que nous avons développé est la récupération des besoins d’un client. Comme nous venons de le dire, nous avons un rendez-vous avec nos clients une fois par semaine, dans lequel nous avons appris à récupérer des besoins client en profondeur en apprenant à poser un maximum de question afin que les besoins et les attentes soient compris au mieux. Un dernier point sur lequel nous avons pu en apprendre davantage grâce à ce projet est la différence et les devoirs des différents rôles du manifeste agile, c’est-à-dire les développeurs, le scrum master et le product owner.

### 5.3 Perspectives

Enfin, plusieurs perspectives pourraient être envisagées sur ce projet de fin d’étude. Nous avons testé sur deux pipelines, il pourrait cependant être intéressant de tester sur d’autres

pipelines. Il faudrait que notre projet soit intégré à la plateforme volbrain. Également, le mode édition devrait être approfondi, il faudrait pouvoir corriger les segmentations en direct.

## Références

- [1] <https://github.com/rii-mango/Papaya>, (Page consultée le 30 mars 2022)
- [2] <https://volbrain.net/>, (Page consultée le 30 mars 2022)
- [3] <https://www.volbrain.upv.es>, (Page consultée le 30 mars 2022)
- [4] Grafikart.fr, Comprendre Git (13/18), <https://youtu.be/AlHohDBBAMY?t=179>, (Page consultée le 30 mars 2022)
- [5] Ken Schwaber, Jeff Sutherland, Le guide Scrum, 2020, <https://scrumguides.org/docs/scrumguide/v2020/2020-Scrum-Guide-French.pdf>, (Page consultée le 30 mars 2022)
- [6] <https://planningpokeronline.com/>, (Page consultée le 30 mars 2022)
- [7] Martin, Robert C. Clean Code : A Handbook of Agile Software Craftsmanship. Upper Saddle River, NJ : Prentice Hall, ISBN : 978-0-13-235088-4 ,2009.
- [8] <http://www.itksnap.org/pmwiki/pmwiki.php>, (Page consultée le 30 mars 2022)