École Polytechnique de l'Université de Tours
64, Avenue Jean Portalis
37200 TOURS, FRANCE
Tél. +33 (0)2 47 36 14 14
www.polytech.univ-tours.fr

Département Informatique
5e année
2013 - 2014

Rapport de PFE

VizAssist - Assistant Web pour la visualisation de données

Encadrants
Gilles VENTURINI
gilles.venturini@univ-tours.fr

Université François-Rabelais, Tours

Étudiants
Jean de BAROCHEZ
jean.debarochez@etu.univ-tours.fr
Rémy PRADIGNAC
remy.pradignac@etu.univ-tours.fr

DI5 2013 - 2014

Version du 4 mai 2014

Table des matières

Table des figures

Liste des tableaux

Présentation du projet

1.1 Introduction

Ce document est le rapport de deux projets de fin d'étude d'étudiants de Polytech Tours en spécialité informatique. L'objectif est de présenter les différentes méthodes et technologies utilisées pour développer les fonctionnalités du site Web VizAssist. Celui-ci est une application Web pour la visualisation de données que vous pouvez retrouver à l'adresse : http://www.vizassist.fr.

Le projet se base sur la thèse effectuée par Abdelheq Guettala, encadrés par Fatma Bouali et Gilles Venturini. Ce dernier est par ailleurs encadrant de ces projets.

Les auteurs de ce rapport sont Rémy Pradignac et Jean de Barochez, étudiants de cinquième année travaillant en coopération sur l'élaboration du site Web VizAssist. Ces deux projets avaient donc un tronc commun : le développement de l'assistant Web de visualisation de données VizAssist. Puis, une fois la structure principale construite, Rémy a implémenté un algorithme génétique interactif pour permettre d'affiner les visualisations proposées à l'utilisateur et Jean a géré les retours et tests par les utilisateurs du site, tout en ajoutant de nouvelles visualisations, plus complexes à mettre en place que les premières.

Ce rapport a pour but, tout d'abord, de relater et d'expliquer les choix effectués par Rémy et Jean au cours de ce projet de fin d'étude pour créer l'application qu'est VizAssist aujourd'hui. Nous espérons que ce projet aura une suite l'an prochain. Ce rapport doit pouvoir aider nos successeurs à continuer le projet entrepris.

1.2 Contexte de la réalisation

1.2.1 Contexte

Étant donné que le secteur de l'analyse de données et de l'information décisionnelle est en plein essor, il devient important de pouvoir représenter au mieux ses données afin de faire ressortir les tendances, savoir quelles informations extraire de ces données et pouvoir les interpréter.

Des outils de visualisation, il en existe de nombreux : du tableur contenant vos données et vous permettant de visualiser celles-ci selon des outils graphiques disponibles, aux programmes interprétant en brut les algorithmes nécessaires à l'élaboration de votre diagramme.

Cependant, ceux-ci possèdent une importante limite. Quelles que soient les données que vous possédez, ou leur type, votre tableur exécutera simplement les ordres et visualisera vos données comme vous l'avez demandé. Mais si vos données pouvaient être mieux interprétées autrement, votre logiciel ne vous l'indiquerait pas. De même, si l'utilisateur connait son but avec ses données et sait ce qu'il souhaite interpréter, il ne sait peut être pas quelles sont les meilleurs façons de les visualiser : un nuage de point, des diagrammes circulaires, une "heatmap" ? Il existe des multiples manières de visualiser des données, mais ces dernières ne sont pas forcément bien représentées selon certains graphiques.

1.2.1.1 Historique

La recherche dans ce domaine à réussi à faire ressortir plusieurs assistants d'aide à la visualisation.

Tout d'abord, deux premiers assistants qui s'appuient sur la relation attributs visuels / attributs de données :

— VISTA, qui se base sur un ensemble de règle
— ViA, qui propose des jeux de couple donnée/visuel

Cependant, ces deux assistants ne tenait pas compte des objectifs de l'utilisateur. C'est dans ce cadre qu'un novuel assistant est apparu :

— WizWizz, retourne des visualisation selon ce que veut l'utilisateur, en utilisant un vecteur de préférence pour les objectifs.

Néanmoins, aucun de ces assistants ne possèdent d'interactions avec l'utilisateur, sans retours ni critiques. La possibilité d'affiner les résultats est inexistante et l'architecture est statique.

C'est pour cela qu'une thèse a été écrite :

— VizAssist : un assistant utilisateur pour le choix et le paramétrage des méthodes de fouille visuelle de données

Thèse par Abdelheq Guettala, publiée en 01-2007, Encadrés par Fatma Bouali et Gilles Venturini

1.2.1.2 Existant

Comme dit précédemment, un existant en C# existait. Cependant, nous ne nous sommes pas servi de cette base.

En effet, nous avons créé cette application Web depuis le début. Seuls les algorithmes implémentés subsistent tels que :

— le matching des datas
— le calcul des scores des objectifs
— le calcul des scores d'appariement
— l'algorithme génétique interactif

1.2.2 Objectifs

Ce projet reprend donc la thèse "VizAssist : un assistant utilisateur pour le choix et le paramétrage des méthodes de fouille visuelle de données", menée par Abdelheq Guettala. Ses recherches portaient sur les moyens de visualiser les données selon les objectifs de l'utilisateur. Un algorithme génétique interactif a été élaboré pour approximer au mieux les volontés de l'utilisateur. Un logiciel nommé VizAssist a été développé en `Java` suite à ces recherches. L'objectif de celui-ci est de proposer un assistant utilisateur pour choisir les différentes visualisations en fonction des données et des objectifs de ce dernier.

1.2.3 Hypothèses de départ

Nous n'avions que peu d'hypothèses. En effet, comme notre projet reprenait le travail de recherche et de développement de A. Guettala, les besoins étaient déjà définis ainsi que les frontières délimitées.

Les seules hypothèses que nous possédions étaient fondées sur nos faibles connaissance dans les performances de la librairie D3.js utilisée. Si la librairie s'avérait être gourmande en ressource selon certains paramètres (nombre de points, nombre de graphe à dessiner en simultanée, ...), certaines méthodes seraient sujettes à être révisées et à évoluer.

1.2.4 Bases méthodologiques

1.2.4.1 Méthodes de gestion

Ce projet était un travail en binôme. Bien que certains objectifs restaient encore à définir en début de projet, les limites fonctionnelles furent définies à l'aide d'une première maquette et validée par M. Venturini.

Nous avions donc un premier découpage en module de notre site.

Ces raisons nous ont donc poussés à adopter les méthodes agiles pour gérer ce projet. Le développement des modules un à un et les retours de notre client (ici M. Venturini) nous on permis de pouvoir séparer notre période de développement en sprints. Cette méthode est très utilisée dans les entreprises de développement Web, car elle s'adapte facilement aux changements de besoin du client et mettent en avant le contentement de celui-ci. Notre projet s'est organiser ainsi afin de développer un à un les modules du site.

Pour plus d'information sur la gestion du projet, reportez vous au chapitre 2 : Gestion du projet.

1.2.4.2 Langages

L'objectif de ce projet était de développer une application Web visualisant des données. Nous avons utilisé donc du HTML, CSS et JavaScript pour la structure et la charte graphique du site.

Le code métier quant à lui, et la visualisation des données se fera à l'aide de la librairie D3 ("Data Driven Document"), utilisant le langage JavaScript et permettant la génération de visualisation au format SVG.

Côté serveur, nous avions opté pour de nouvelles technologies qui apparaissent utilisant le JavaScript et montrant un réel avantage lors de multiples requêtes asynchrones. La gestion d'événements étant essentielle en JavaScript, de nombreuses requêtes sont générées entre le front-end et le back-end.

Node.js est un serveur créant une révolution dans le monde du Web par son architecture et ses performances pour le développement d'applications légères client-side. En effet, celui ci exécute les instructions données en JavaScript, comme on le ferait avec du PHP, ASP ou encore JavaEE.

Cependant, afin de pouvoir mettre notre projet en production, nous avons opté pour une solution simple avec la mise en place d'un serveur Apache PHP. En restant généraliste, on garantit un système facile à mettre en place et aisément maintenable. Devant la constante présence de sa communauté Web, PHP est un langage performant avec une documentation très accessible. De plus, celui-ci s'interface très bien avec le JavaScript via des requêtes AJAX entre le serveur et le client.

1.2.4.3 Règles de programmation

Comme nous avons travaillé en binôme, il était essentiel que nous utilisions les mêmes noms de variable entre nous afin que nos relectures soient rapides et efficaces. De plus, bien que le JavaScript et le PHP soient très flexibles avec le typage et le contenu des variables, il était important de rester cohérent et strict, sans cela notre application aurait été difficilement maintenable. Enfin notre projet avait, dans l'un de ses objectifs, la possibilité d'ajouter facilement de nouvelles visualisations et ce, de façon à ce que chaque nouvelles méthodes aient la même syntaxe, il est indispensable de rester homogène dans notre programmation.

On avons donc utilisé donc les règles de programmation suivantes :

- Chaque nom de classe commence par une majuscule
- Les noms de fonctions et de variables ont une minuscule comme première lettre et majuscule à chaque mot.
- Chaque méthode possède en en-tête ses spécifications décrivant chaque paramètres d'entrées et de sorties.

1.3 Description générale

1.3.1 Environnement du projet

Notre travail fut une poursuite du travail effectué par A. Guettala sur la visualisation de données. Cependant, bien qu'un existant nous a été présenté, il n'a pas imputé notre travail.

En effet, ce dernier est un applicatif développé en C# alors que nous sommes sur la création d'un nouveau rendu Web.

Nous sommes donc parti de zéro, d'un point de vue développement.

Notre site a une architecture classique, à savoir :
— Le chargement des données par l'utilisateur
— Choix de visualisation de base en fonction des objectifs et du score d'appariement
— La possibilité d'utiliser un algorithme génétique pour de nouvelles visualisations
— Un retour utilisateur disponible à chaque instant

1.3.2 Caractéristiques des utilisateurs

Le public visé n'est pas un public initié aux sciences de l'informatique. Leurs compétences dépendra surtout de types de visualisation qu'ils recherchent. De plus, les utilisateurs ne seront pas obligatoirement réguliers, cela dépendra de leur besoin. C'est pour cela que nous n'avons pas pu nous permettre de créer une interface trop complexe.

Ceci étant posé, nous avons créer une interface, qui de par sa simplicité, est intuitive pour permettre une facilité d'utilisation de l'application.

1.3.3 Fonctionnalités et structure générale

La figure 1.1 représente le workflow d'un utilisateur sur l'application :
A tout moment, l'utilisateur peut retourner à une étape précédente, afin d'appliquer de possibles changements (tels que ces objectifs par exemple).

Comme on peut le voir sur ce schéma, le fonctionnement de notre application est très simple et linéaire, ce qui nous a amené à mettre en place un visuel de type "Wizard" d'installation.

1.3.4 Contraintes de développement, d'exploitation et de maintenance

Notre site est une application sans comptes utilisateurs. De plus, toutes les informations dont nous avons besoin (tel que les données permettant de construire nos graphiques) sont stockés dans des csv, car il nous est interdit de les sauvegarder car elles peuvent être sensible. Il en résulte que nous n'avons donc pas l'utilité d'une base de données.

Nous conservons les standards du Web pour notre développement en utilisant du HTML et du CSS. Afin d'effectuer notre code dit "métier", nous avons utilisé du JavaScript. Pour la visualisation, une librairie nous a été imposée : D3.js. Ceci entraîne une contrainte du point de vue des compatibilités avec les navigateurs Web. D3js en plus de manipuler simplement les éléments du DOM, D3.js (Data-Driven Documents) est une librairie graphique JavaScript, permettant un affichage dynamique de visualisation de données sous le format SVG. Or, ces images sont incompatibles avec d'anciens navigateurs Web comme IE8 et ses versions antérieures, encore utilisés de nos jours (moins de 10% des parts des marchés en Juillet 2013).

Notre projet est un site en ligne, il existe donc une contrainte matérielle, à savoir un ordinateur avec une connexion internet.

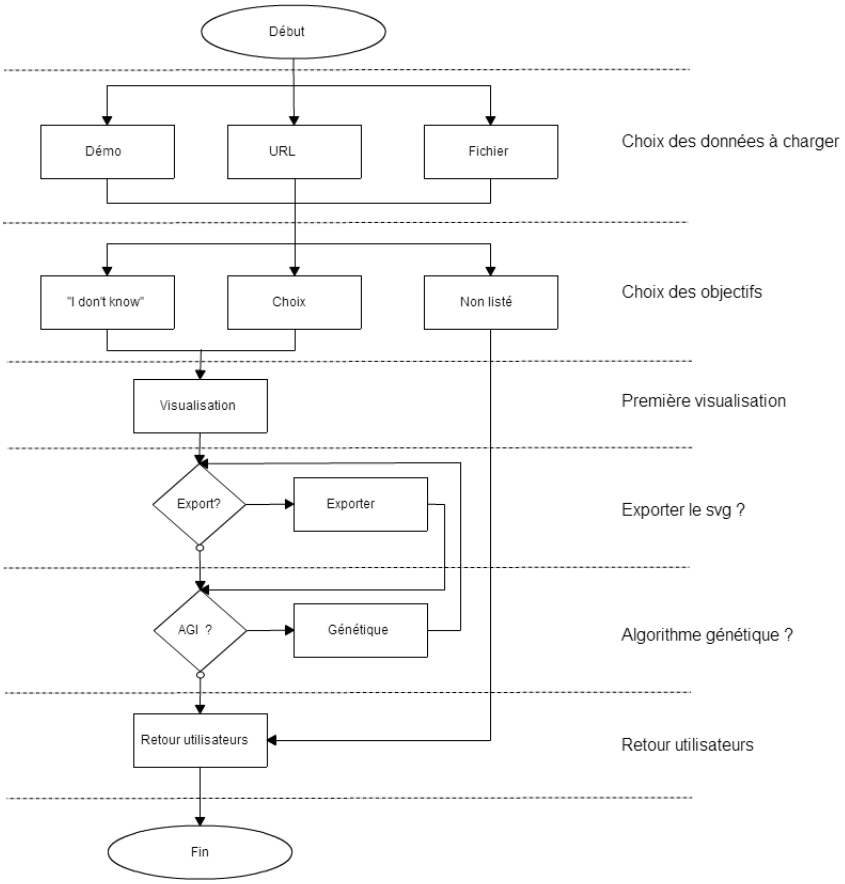


FIGURE 1.1 – Workflow

Nous avons choisi d'utiliser l'IDE NetBeans avec le plugin SVN pour la gestion de notre dépôt. Ce dernier nous a été imposé afin de garantir l'intégrité de nos données sur les serveurs de l'école.

Nos environnements de développement étaient Windows 7 et Windows 8.1, alors que l'environnement de production (OVH) est une machine Unix.

Afin de découvrir de nouvelles technologies lors de nos projets de fin d'étude, nous voulions opté Node.js, un serveur créant une révolution dans le monde du Web par son architecture et ses performances pour le développement d'applications légères client-side. En effet, celui ci traite les requêtes clientes en asynchrones et de manière plus optimisé qu'un Apache Server.

Cependant, nous n'avions que ce dernier de disponible pour déployer notre application en production. Nous avons donc choisi la technologie PHP pour déployer notre petite application côté serveur.

1.3.5 Conditions de fonctionnement

L'avantage de notre application, comme de toutes applications Web, est que le système d'exploitation de la machine utilisateur n'influe en rien sur l'application, contrairement à un programme en C# par exemple.

Cependant, certaines conditions pour le bon fonctionnement de VizAssist subsistent, telles que :
- Une connexion Internet,
- Un navigateur récent :
  - Internet Explorer 10

  - Firefox 27
  - Chrome 31
- Autorisation d'exécution de code Javascript

Sans ces pré-requis, ce projet ne fonctionnera pas de manière optimale. Les technologies employées par notre application (SVG, File API, dernières spécifications JavaScript) empêchent à l'heure actuelle d'être disponible sur une plus grande majorité de navigateur.

1.3.6 Les collaborateurs

De nombreuses personnes ont participé sur ce projet. Tout d'abord, Monsieur Venturini et Madame Bouali en tant que superviseur de tous les projets qui résultaient de cette application :

- Thèse de :
  - Abdelheq Guettala
- Projet de Fin d'étude de :
  - Jean De Barochez
  - Rémy Pradignac
  - Noureddine Saifi
- Projet d'option Web et Multimédia de :
  - Fabien Buda
  - Aili Dong
  - Guillaume LeBihan
  - Zui Zhang

Un jeu de données est même disponible sur l'application en ligne, pour visualiser tous les collaborateurs à VizAssist.

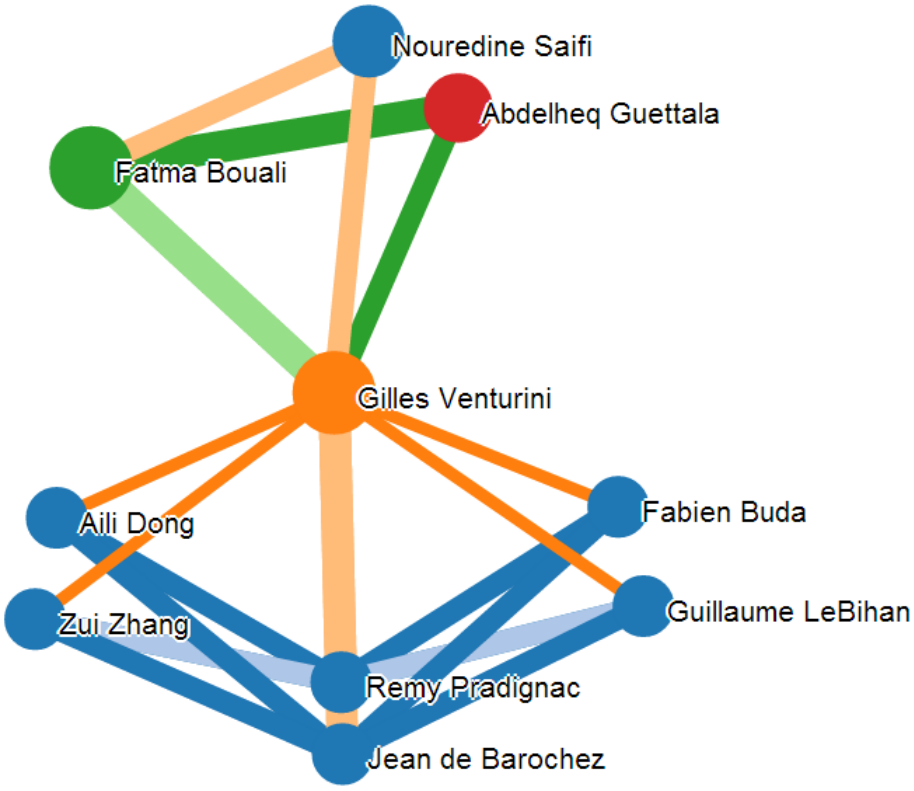


FIGURE 1.2 – Graphique des collaborateurs généré par VizAssist.

Gestion du projet

Ce projet a duré sept mois durant lesquels nous avons dû le gérer afin de mener à bien. Nous sommes parti dans un contexte agile dès le départ. Nous travaillions en binôme et avions Monsieur Venturini comme encadrant, voir parfois même client. En effet, il faisait ses requêtes et nous proposions des solutions à ses besoins sur l'application.

Bien que notre projet reprenait l'idée d'un travail existant, tous les objectifs n'étaient pas fixés dans le détail. Monsieur Venturini n'était pas certain du résultat qu'il souhaitait. Nous n'avions pour dire que les grandes lignes et les principales fonctionnalités à implémenter. C'est pourquoi nous avons adopté un contexte agile rapidement. Ainsi, nous étions capable de modifier notre course vers quelque chose que notre client préférait.

2.1 Méthodes agiles employées

2.1.1 Découpage en sprint

Parmi les méthodes agiles appliquées durant notre projet, on retrouve le découpage en sprint. Ce découpage permet de découper notre projet en plus petites parties et de réaliser des cycles réguliers pour le développer. A chaque fin de sprint, nous avions un retour de Monsieur Venturini sur notre travail et pouvions adapter ou améliorer notre travail en fonction.

2.1.2 Réunions régulières

Tout au long du projet, nous avons eu des réunions pour discuter de l'avancement, des spécificités des fonctionnalités et des retours utilisateurs. Cela a donc demandé de nombreuses rencontres, d'une part entre nous deux, d'autre part entre Monsieur Venturini et nous.

Les réunions avec notre encadrant se déroulaient au moins une fois par semaine, où on se penchait sur l'état actuelle des choses, sur ce qui venait d'être effectué et ce qui allait être fait. Ces réunions permettaient à Monsieur Venturini d'avoir un suivi constant sur le projet.

Suite à l'arrivée des nouveaux collaborateurs, nous avons eu de nouvelles réunions avec ceux-ci. Comme ils s'intégraient à notre projet, nous devions tout d'abord les former sur l'application, puis sur les technologies employées. Sur ces dernières, nous étions seulement entre étudiants pour discuter techniques, où nous faisions remonter notre expérience après six mois passés à utiliser la librairie D3.

2.1.3 Travailler en binôme

Effectuer un projet à deux personnes oblige à prendre certaines habitudes pour pouvoir s'organiser et les méthodes agiles interviennent même à cette échelle. Nous n'avions pas de Scrum master, car nous n'étions que deux et donc avons choisi que nous devions travailler de pair pour le projet. Pour chaque problème, nous discutions de l'avantage d'un ou de l'autre avis pour choisir la meilleur solution.

Cela nous a obligés à avoir une communication constante entre nous deux. Si l'un de nous effectuait quelque chose, l'autre devait être au courant. Soit par les outils de gestion utilisés (voir 2.2), soit par voix orale, lorsque l'un de nous avait une rencontre avec Monsieur Venturini, l'autre savait quel avait été le résultat de la réunion.

Au niveau du partage des tâches, nous avons travaillé d'octobre à mars en totale collaboration avant de nous séparer. A partir de janvier, nous avions finit la phase de spécification et nous avons attaqué les sprints de développement. Chaque tâche qu'on se fixait était développée par l'un et relue et testée par l'autre. Cela offrait une compétitivité dans le travail effectué et a permis de relever de nombreuses erreurs. On se forçait à avoir un regard critique sur les actions de l'autre et à justifier chaque décision prise. Une fois validée par le deuxième développeur, le ticket était résolu sur Redmine et on passait à la suite.

Chaque journée commençait par une discussion similaire au daily des méthodes agiles. Cela augmentait la communication dans l'équipe et permettait de savoir quels problèmes de la veille, l'autre avait, quel était son plan pour la journée, que pensait-il faire après avoir finit sa tâche en cours. Enfin, cette discussion permettait de partager les connaissances du langage (principalement en JavaScript) appris la veille.

2.1.4 Plan d'un sprint type

Nos sprints duraient en moyenne deux semaines. Selon les tâches à effectuer, cela fluctuait d'un ou deux jours. Cependant, on a conservé le même déroulement au cours de chacun d'entre eux.

2.1.4.1 Définition des tâches

Un sprint commençait toujours par une réunion de notre binôme. Cette réunion avait pour but de préciser les tâches prévues pour ce sprint. Chaque sprint avait son thème ou son module à implémenter et chaque sprint pouvait être découpé en plusieurs tâches à effectuer.

Nous définissions donc le déroulement du sprint : Qu'avait-on en entrée ? Qu'était-il nécessaire d'ajouter ? Que fallait-il tester ? Que devait-il y avoir en sortie ?

Cette phase durait le plus souvent deux jours, durant lesquels nous rédigions un cahier de spécification pour ce sprint où nous définissions chaque tâche. Si des questions persistaient sur la demande, nous allions voir M. Venturini pour connaître ses besoins. On avait ainsi un va et vient entre nos réunions et M. Venturini pour programmer notre sprint. Chaque création de tâche entraînait l'ouverture d'un ticket Redmine de développement.

Au final, nous arrivons à des périodes de 6 jours sur le projet qui correspondaient à deux semaines de cours, selon notre emploi du temps du second semestre.

2.1.4.2 Exécution des tâches

Une fois le cahier rédigé, nous développions les ajouts nécessaires à chaque tâche. On se fixait deux jours de développement par sprint.

Chaque tâche entraînait une rédaction des tests si cela était possible (voir Environnement de Test 2.4.1). Elles étaient relues par le deuxième qui réalisaient ensuite les structure de test nécessaires.

2.1.4.3 Retroplanning

Lors de cette phase, nous essayions de prendre du recul par rapport au début du projet. On effectuait une maintenance sur les précédents sprints en prenant en compte les retours utilisateurs et critiques de M. Venturini.

Plus on avançait dans le projet et plus notre connaissance des langages et des librairies utilisés augmentaient. On repassait ainsi systématiquement sur les premiers modules, pour garder une cohérence dans notre code tout au long du projet. De même, les tests qu'on effectuait sur les utilisateurs montraient la plupart du temps de nouvelles modifications à effectuer.

Si les estimations avaient été correctes lors de la définition des tâches, on avait passé deux semaines sur ce module. Cependant, devant les retours et le recul qu'on prenait sur notre projet, les sprints pouvaient être allongés

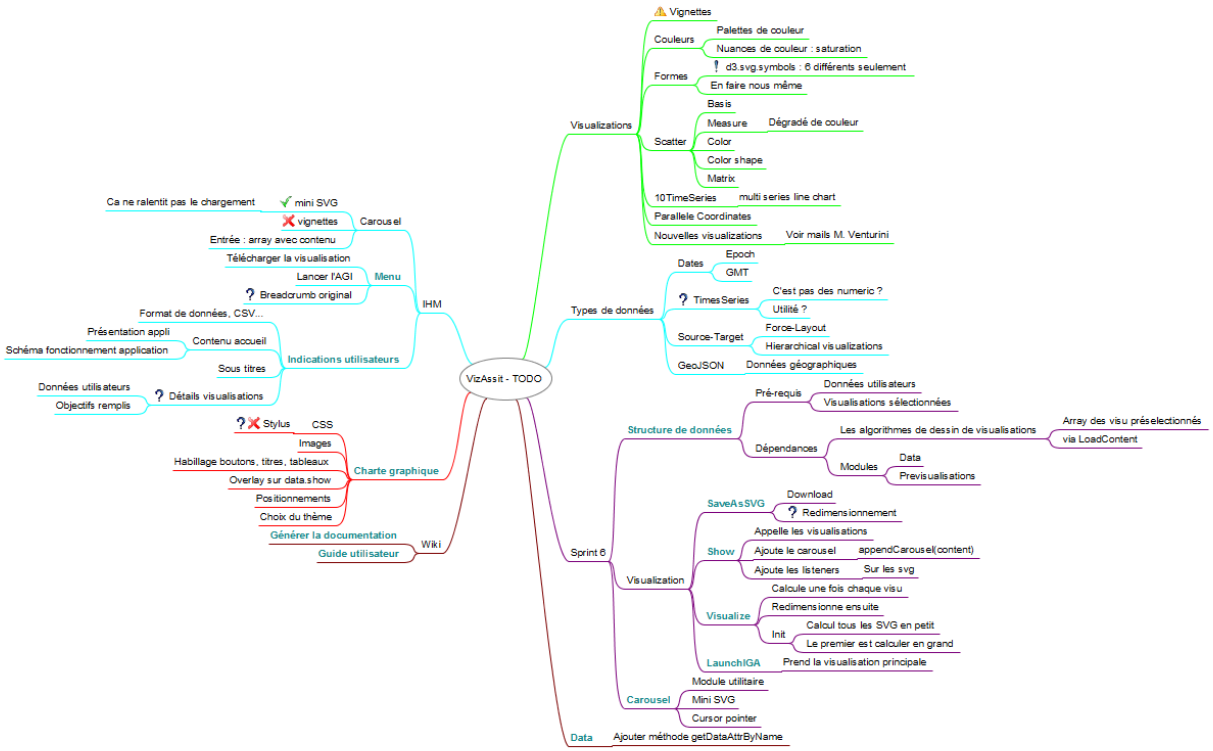


FIGURE 2.1 – Exemple d'une mindmap avant le sprint 6

2.2 Outils de gestion utilisés

Au cours de notre projet, nous avons utilisé plusieurs outils pour assurer une bonne organisation, un suivi des tâches et un contrôle des sources de l'application.

2.2.1 Les outils de gestion de projet utilisés

2.2.1.1 Redmine

Pour gérer notre projet, nous avons principalement utilisé le Redmine de l'école. Celui-ci offre un suivi des tâches qui permet de savoir qui fait quoi à un instant t. Nous avons essayé de le maintenir le plus souvent à jour. Une fois une tâche terminée, nous fermions son ticket et passions à une autre tâche.

Le Redmine a été très utile aussi pour notifier à l'équipe les erreurs qu'on nous signalait. Chaque fois qu'une erreur a été repérée par Monsieur Venturini, un contributeur de Web et Multimédia ou un de nous deux, nous avons créé un ticket de type Anomalie, décrivant le bug et comment il avait été obtenu.

Enfin, cet outil permet aussi à notre client de vérifier l'avancée de notre travail et de suivre nos travaux.

2.2.1.2 Freeplane

Ce logiciel est un outil de mind mapping gratuit. Nous avons utilisé celui-ci au cours du projet pour organiser nos idées avant un sprint (exemple figure 2.1). Ce simple outil permet de mettre à plat des idées tout en gardant des relation entre eux.

2.2.1.3 Tous moyens de communication

Tout au long du projet, nous avons constamment gardé une forte communication entre notre encadrant et client Monsieur Venturini, ainsi qu'entre nous deux. Principalement des mails ont été échangés entre étudiants et encadrant. Mais entre étudiants, tous moyens étaient utilisés. Dès qu'une idée émergeait, qu'un bug était relevé, le deuxième était rapidement au courant.

Lors du démarrage des projets de Web et Multimédia, nous avons immédiatement envoyé un mail aux deux autres binômes pour tenir de forts échanges afin que la collaboration se fasse au mieux.

Les moyens de communication permettent ainsi de tenir une cohésion et améliore l'implication entre les différents acteurs du projet.

2.2.2 Outils de versionning

2.2.2.1 Subversion

Nous avons commencé dès le début à utiliser le dépôt SVN offert par l'école. Travailler avec SVN seul ne pose pas de problème. Le développeur est maître de son dépôt et évite facilement les conflits lors des commits. A deux, cela se complique un peu. Il faut pouvoir s'organiser pour ne pas travailler sur les mêmes fichiers en même temps. SVN a une gestion difficile et lourde des branches. C'est pourquoi nous avons préféré s'organiser pour éviter les conflits plutôt que laisser SVN gérer ça pour nous. Cela aurait ajouté du travail en plus et ainsi entraîner des pertes de temps. Nous n'étions que deux à travailler sur le projet. Chaque utilisation de SVN était signalé à l'autre avant de commit ses changements.

2.2.2.2 Git

Git a été utile sur la fin du projet pour la mise en production de l'application. Avec le recul, nous aurions dû l'utiliser plus tôt en parallèle de Subversion. Git est un outil de versionning décentralisé : chaque contributeur possède son propre dépôt et son propre historique local. Cela permet ainsi de pouvoir travailler hors connexion ce qui est arrivé plus d'une fois à l'école malheureusement. Une fois la connexion remise, on envoie ses modifications à l'autre développeur et le versionning est conservé.

En plus de la mise en production, cet outil a été très utile pour intégrer le travail des groupes de l'option Web et Multimédia. Pour chaque test d'intégration, nous créions une branche sur notre principale et nous effectuions les modifications nécessaires à l'ajout de leurs visualisations. Notre travail et le leur étaient ainsi séparés et nous pouvions comparer leur travail au nôtre et remonter nos avis.

2.3 Environnement de travail

2.3.1 Outils de développement

Pour fonctionner, notre application a besoin d'un conteneur d'application PHP. Parmi les nombreuses applications disponibles, nous avons opté pour XAMPP [1], une distribution Apache contenant MySQL, PHP 5.4 et Perl. Accompagné de l'outil XDebug [2], XAMPP offre tous les outils nécessaires pour contenir une application PHP et interagir avec (debug, error log, log Apache...).

FIGURE 2.2 – XAMPP est une distribution Apache contenant MySQL, PHP et Perl

Ensuite, pour développer nos programmes, nous avions juste l'embarras du choix du point de vu des éditeurs PHP, HTML et JavaScript. Le but d'un IDE est d'accélérer le travail du développeur en lui faisant remonter les erreurs et en lui conseillant les bonnes pratiques logicielles. Accélérer le travail du développeur est très souvent associé à la connaissance que le développeur a pour son logiciel favori. Un bon IDE apporte aussi avec lui de nombreux outils pour travailler en collaboration avec d'autres personnes, en intégrant des plugins pour les outils de versionning que nous utilisions.

FIGURE 2.3 – NetBeans, un IDE puissant pour développer une application PHP

Ainsi, on a utilisé NetBeans [3] pour développer en PHP, qui permet de communiquer avec le serveur Apache PHP et éditer, exécuter et débugger son programme PHP. Pour le JavaScript, Sublime Text 3 [4] est un outils puissant, facilement personnalisable qui offre de nombreux outils pour assurer la qualité de code JavaScript. Ce langage étant très permissif, il est donc important d'avoir de nombreux contrôles derrière soit.

Enfin, une application comme celle-ci nécessite un navigateur Web avec un moteur JavaScript. Le Chrome Devtools offre une console et plusieurs outils pour étudier le comportement de l'application, qui à l'heure actuelle est dans les meilleures du marché. On peut ainsi à tout moment débugger son programme, faire des modifications en live et ré-exécuter le code. On pourrait le prendre pour un IDE à part-entière si le code édité était ensuite sauvegardé localement.

1. https ://www.apachefriends.org
2. http ://www.xdebug.org
3. https ://netbeans.org/
4. http ://www.sublimetext.com

2.3.2 Côté production

En production, le serveur hébergé chez OVH est un serveur mutualisé, dans un environnement Unix. Il offre un conteur Apache PHP 5.4, de même version donc que notre environnement de développement. OVH offre avec ce serveur des accès SSH et FTP pour déployer son site Web.

Selon les besoins, nous avons donc utilisé FileZilla et Putty pour nous connecter au serveur. Le premier est un puissant client FTP, que nous avons utilisé pour les premiers déploiementset pour l'ajout de composant, par exemple le Wiki (voir 5.1.1.1). Le second est, entre autre, un client SSH qui a permis ensuite de gérer les mises à jour du serveur via Git.

Les mises à jour de l'application ont suivi un processus régulier, pour permettre une amélioration constante de l'application, tout en conservant une stabilité permanente (voir 5.1).

2.4 Assurance qualité

Développer une application, c'est bien. La développer correctement, c'est mieux. C'est pourquoi chacun de nos codes métiers ont subit des contrôles et de tests pour être validés avant de passer en production. Pour cela, on avait donc deux phases : le contrôle de la qualité de code, selon de bonnes pratiques de développement inculqués par les géants du Web ; et les tests du bon fonctionnement de notre application par des frameworks de tests et par les utilisateurs.

2.4.1 Environnements de tests

Dans notre projet, on peut distinguer plusieurs milieux au sein de notre environnement de test. Nous avons une première partie PHP, où se situe le serveur d'application. Puis nous avons la partie JavaScript métier, qui interagit avec le serveur et effectue les calculs au bon fonctionnement de l'application. Enfin, une deuxième partie JavaScript qui interagit avec le client (voir Architecture du système 3.3).

2.4.1.1 PHPUnit

PHPUnit[5] est un framework de test que nous avons utilisé pour nos tests serveurs. Équivalent à jUnit pour Java, il a permis de rédiger les tests unitaires de notre code PHP.

Notre partie PHP est la base de notre application. Il était donc primordial de s'assurer du résultat de nos méthodes PHP pour que tout le reste du projet fonctionne comme prévu.

Cependant, notre application PHP se base sur des fichiers de données en entrée, fournis par Monsieur Venturini. Si une erreur est introduite dans ces fichiers, alors l'application peut avoir des résultats non attendus.

5. http ://PHPunit.de

2.4.1.2 Mocha

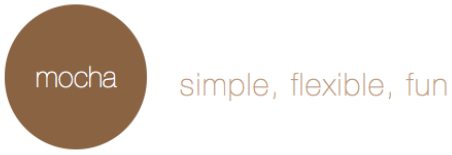


FIGURE 2.4 – Mocha, notre framework de test JavaScript

Mocha [6] est un riche framework de test JavaScript, originellement basé sur NodeJS. Il existe cependant une version pour tester le code uniquement client-side et donc adapté à notre architecture. Accompagné de la librairie ChaiJS [7], on se obtient un framework complet pour tester les assertions, les requêtes AJAX et ainsi notre application JavaScript.

Mocha permet ainsi de tester notre code métier JavaScript, mais pas toutes les interactions que nous créions sur le DOM. Nous avons donc dû rédiger notre code de façon à ce qu'il soit testable et cela en séparant tout code métier des interactions visuelles. Ainsi, nous avons été forcé de développer correctement notre code en modules, en séparant bien les contrôleurs de la vue, des méthodes métiers de l'application.

FIGURE 2.5 – Extrait des tests executés par Mocha

De nombreux autres outils auraient pu être utilisés pour tester notre application et nous retourner des statistiques sur notre qualité de code (couverture de code, robustesse de l'interface graphique...). Cependant, ceux-ci ne sont pas nativement adapté pour fonctionner juste côté client et requiert NodeJS pour fonctionner. Les mettre en place aurait entraîner des retards sur le développement de l'application, mais aurait renforcer notre environnement de développement.

6. http ://visionmedia.github.io/mocha/
7. http ://chaijs.com

2.4.1.3 Tests utilisateurs

La dernière partie restante à tester est donc la partie interface utilisateur, la plus importante pour un assistant utilisateur. Et pour tester un assistant utilisateur, rien de mieux que d'avoir les retours des utilisateurs.

Avant que l'application soit mise en ligne, seuls Monsieur Venturini, et notre binôme testions l'interface. Notre interface est une suite de fenêtre (voir Interface Homme/Machine 3.3.2), si bien que notre binôme, au fur et à mesure du développement, repassions systématiquement sur les premières étapes développées.

L'interface utilisateur suit un même fil conducteur, mais selon la navigation de l'utilisateur, des fonctionnalités doivent ou non apparaître (différence de navigation néophyte ou experte, voir les cas d'utilisation 1.1). Selon les spécificités que voulait Monsieur Venturini, nous nous appliquions à obtenir les résultats voulus.

L'application a donc été de nombreuses fois testées par nos soins et les personnes curieuses de l'avancée de notre projet, avant d'être mise en production. Cette étape achevée, l'application était plus facilement accessible au public, ce qui a amené de nouveaux et plus riches retours utilisateurs.

Voilà comment nous procédions aux tests par les utilisateurs :

1. Nous leur présentions tout d'abord le but de l'application et résumions les différentes étapes qu'ils allaient devoir remplir.
2. Puis, nous les mettions face à l'application et les laissions découvrir les différentes étapes. Pendant ce temps, nous observions leur comportement
3. Enfin, nous retenions leur retour et leurs avis sur l'utilisation de l'application.

.

Nous avons utilisé ce processus pour les personnes que nous pouvions observer, à l'école ou domiciles. Cela a permis d'améliorer grandement l'interface et de choisir le design à utiliser. Nous avions un retour direct avec la personne sur pourquoi telle étape l'avait gênée, telle réaction de l'application elle n'avait pas comprise et telle fonctionnalité valait il mieux présenter autrement.

Cependant, après l'ouverture au public de VizAssist, Monsieur Venturini a créé un formulaire Google de sondage, pour collecter les différentes opinions des utilisateurs.

L'ouverture au public s'est fait en plusieurs étapes et a ciblé différents groupes de contacts de Monsieur Venturini, toujours plus larges. Plusieurs retours positifs ont été reçu à ce jour.

2.4.2 Contrôle du code

Beaucoup d'outils existent pour obtenir des statistiques sur la qualité de code en JavaScript, cependant comme pour nos tests, nous n'avons pas eu le temps de mettre en place un environnement de développement avec intégrations continues et contrôles qualitatifs du travail effectué comme une forge logicielle telle que Jenkins l'aurait fait.

Cependant, nous avons tout de même utilisé des outils comme JSHint pour nous guider dans la rédaction de code JavaScript. Le JavaScript est un langage très faiblement typé et très permissif, où un objet peut être une fonction, puis un string, puis un tableau pour redevenir ensuite un objet. On peut se retrouver ainsi avec un code très difficilement abordable et maintenable, et cela en juste quelques lignes de code. L'outil JSHint apporte ainsi une certaine cohérence dans notre code JavaScript, en indiquant les bonnes pratiques de développement JavaScript. Un autre outil que nous avons pu utiliser est JSLint, qui est beaucoup plus stricte, puisqu'il indique aussi les erreurs d'indentation et d'espacement dans le code.

C'est aussi pourquoi, lors de notre premier sprint, nous avons pris le soin d'adopter diverses méthologies de développement JavaScript, parmi celles enseignées par le Mozilla Developper Network [8] ou celles de GitHub [9].

8. https ://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide
9. https ://github.com/styleguide/javascript

2.5 Déroulement du projet

2.5.1 Estimations initiales

Les diagrammes de Gantt et montrent l'estimation des tâches que nous avions imaginé au sprint initial. Chaque module avait été pensé, et comptions créer chaque module sur une période de deux semaines à chaque fois (6 jours de projet lors du second semestre).

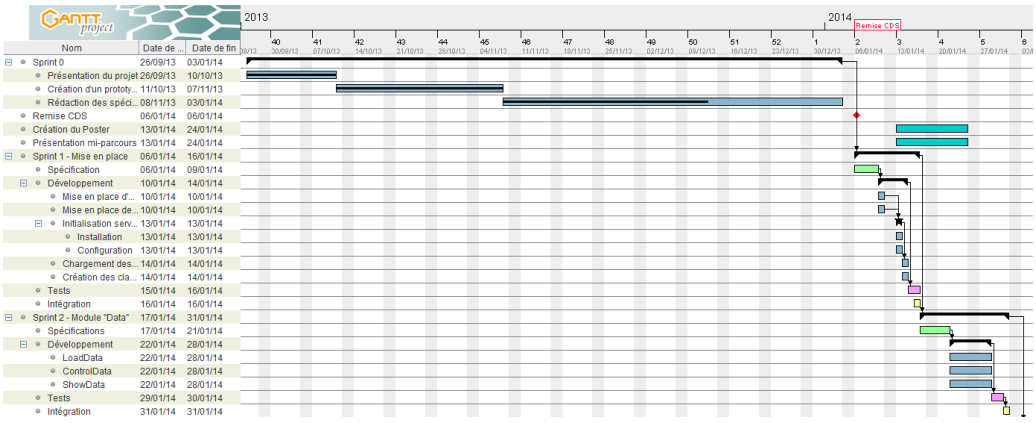


FIGURE 2.6 – Planning prévisionnel effectué au sprint 0 (début)

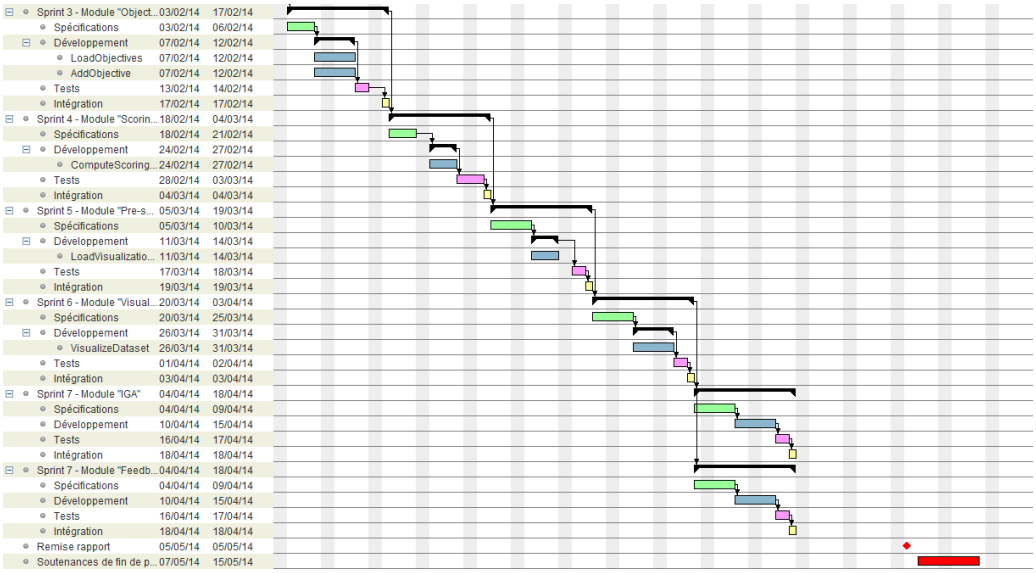


FIGURE 2.7 – Planning prévisionnel effectué au sprint 0 (fin)

Nous prévoyions donc pour chaque sprint de développer un module, qui serait une étape de l'assistant utilisateur et un objectif du projet à remplir parmi :

— Pouvoir charger des données CSV sur l'application,
— Choisir ses objectifs,
— Effectuer l'appariement données/objectifs et visualisations,
— Pouvoir choisir parmi les visualisations résultantes,
— Visualiser ses données sur les graphiques choisis,
— Explorer d'autres solutions à l'aide de l'algorithme génétique interactif.

2.5.2 Planning réel

Cependant, nos prévisions ont légèrement fluctué. Lorsque nous sommes arrivés au sprint 4, qui devait effectuer le matching, nous nous sommes rendus compte que cette étape pouvait très facilement être assimilée au sprint 5 qui devait effectuer l'affichage du résultat du sprint 4.

Vous trouverez donc le diagramme de Gantt effectif 2.8 qui compare notre déroulement prévu et celui réel. Les dates réelles résultent de notre suivi des tâches sur Redmine que nous avons exportées pour créer ce graphique.

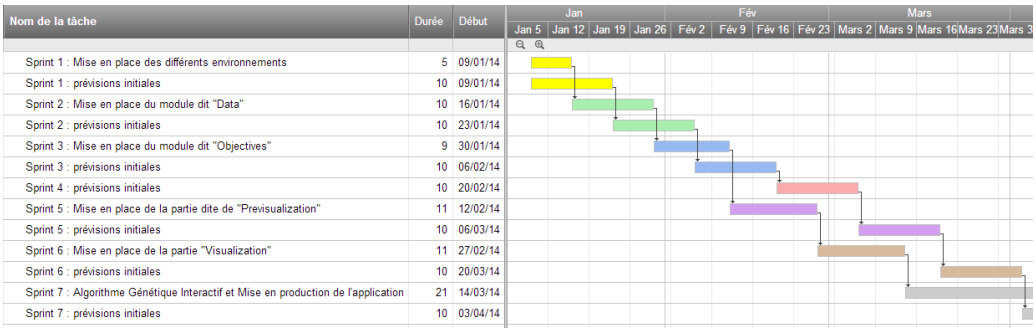


FIGURE 2.8 – Déroulement effectif des sprints, comparé à leurs prévisions

2.5.3 Étude des différences

Au final, la mise en place de la partie serveur et des différents environnements a été plus rapide que prévue et a permis de commencer la partie de chargement des données plus tôt. Les différentes études de documentation que nous avions effectué au sprint initial, accompagné de la création de la maquette, a entraîné un développement plus rapide et plus concis de nos sprints. Nous n'avons pas eu à nous pencher sur l'étude de ce sprint, tant il nous paraissait simple arrivé en face.

Cette mauvaise estimation de notre sprint 4 nous a au final fait gagné deux semaines sur notre planning initial à la mi-février (tâche rose sur le diagramme 2.8).

Enfin, notre sprint 7 a pris plus de temps que prévu, pour plusieurs raisons. Nous avons tout d'abord pris le temps d'assister les projets de Web et Multimédia dans leur découverte des technologies. D'un autre côté, malgré tous les efforts que nous avons mis en place, nous avons souffert de quelques incompatibilités inter-navigateur à ce moment là, notamment vers Firefox (voir la section Problèmes rencontrés 3.7.1).

Pour conclure ces estimations, nous pouvons tout de même relever que, malgré les erreurs d'estimations effectuées sur deux sprint (surtout le premier et le quatrième), notre découpage en sprint de deux semaines s'est relevé correct. Nous avons su tenir les délais que nous nous étions imposés.

Solution mise en place

3.1 Présentation

VizAssist est une application Web qui assiste l'utilisateur à choisir la visualisation la plus adaptée à ses données, selon ses objectifs. Elle accompagne donc l'utilisateur au cours de 5 étapes pour jouer son rôle d'assistant.

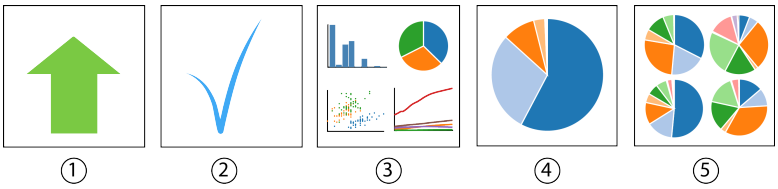


FIGURE 3.1 – Schéma de fonctionnement de l'application

La plus grosse contrainte imposée pour le développement de l'application concerne les données de l'utilisateur, qui doivent restées privées à tous prix. C'est pourquoi VizAssist est une application entièrement JavaScript, exécutée uniquement sur le navigateur de l'utilisateur. Tous les calculs, les chargements de fichiers et la navigation de l'utilisateur sont effectués en local pour garantir la sécurité des données. Les données ne sont jamais stockées en session, ni en cache dans le navigateur. Une fois la page rafraîchit, toutes données chargées sont perdues.

Le serveur PHP mis en place n'a alors qu'un rôle d'entrepôts de ressources et le client effectue les requêtes nécessaires pour que l'application puisse fonctionner correctement :

— Les jeux de données de démonstration, si l'utilisateur les demande
— La liste des objectifs
— La liste des visualisations

Ces jeux de données et listes sont différents fichiers CSV hébergés sur le serveur, auquel on peut accéder et mettre à jour.

Un fichiers CSV ("Comma Separated Values") est un fichier sous un format ouvert, contenant des données tabulaires séparées par une virgule. Ce type de fichier ne possédant pas une norme officielle, nous nous sommes aperçus que dans certains pays, tel que la France, ce format ne suivait pas le standard, et que les tableurs Excel français mettaient d'eux même un point-virgule en guise de caractère de séparation. Notre application ayant un but international, rédigé en anglais, nous avons décidé (avec avis de notre encadrant) de garder le standard du format et de ne lire que les formats CSV utilisant une virgule en guise de séparateur de valeurs.

L'ajout et la modification de nouvelles données, objectifs et attributs visuels, est grandement simplifié. Ceci se répercute ensuite sur l'application, qui charge dynamiquement les différentes informations.

L'application propose de nombreuses visualisations, du simple nuage de points aux graphes dirigés, en passant par les séries temporelles. La liste complète se trouve en annexe de ce rapport A et peut évoluer facilement. Nous avons pensé et développé notre application pour que l'ajout de nouveaux graphiques soit la plus simple possible pour que quiconque participe au projet puisse ajouter facilement sa visualisation.

Le point du contenu dynamique était la deuxième contrainte forte de notre projet, sur lequel Monsieur Venturini comptait beaucoup. Il lui est ainsi aisé d'ajouter de nouveaux objectifs, de nouvelles visualisations

et nouveaux jeux de données, sans avoir à modifier l'application. Le contenu que propose VizAssist peut ainsi très rapidement évolué selon les besoins de son propriétaire.

3.2 Description des sprints

3.2.1 Sprint 0 : prise de connaissance du sujet

Le sprint initial a permis à notre binôme de s'organiser pour commencer dans les meilleurs conditions la réalisation de VizAssist. Au cours de celui ci, nous avons pris connaissance du sujet, de son contexte (les contraintes existantes), les hypothèses à avoir, étudier les technologies à employer et créer un plan d'organisation pour découper notre projet en plusieurs sprints.

Monsieur Venturini avait une bonne idée générale de ce qu'il voulait. Afin de comprendre ses besoins et de les délimiter, nous avons créer une maquette de l'application. Cette réalisation nous a permis d'approcher les technologies du Web comme le HTML5 et le CSS3, mais surtout le JavaScript que nous connaissions peu au final. Cette prise de connaissance du langage s'est accompagné de la découverte de la librairie D3.js que monsieur Venturini comptait utiliser pour l'application.

Ce qui a été produit à la fin de ce sprint est le cahier des spécification du sprint 0. Placé en annexe de ce rapport, il décrit les différentes spécificités déclarées au sprint initial (structure générale de l'application, son environnement...), ainsi que le découpage en sprint mis en place pour la suite du développement.

Au départ, nous avions analysé 8 fonctionnalités que notre application devait comporter :

— Mettre en place une structure côté serveur pour lire les fichiers de données CSV
— Charger des données utilisateurs de plusieurs façons différentes
— Choisir des objectifs
— Effectuer un appariement entre les visualisations en fonction des données et des objectifs
— Pouvoir sélectionner les visualisations en résultant
— Visualiser les données à l'aide de ce qui avait été sélectionné
— Implémenter un algorithme génétique interactif
— Être déployable en ligne pour subir des tests par les premiers utilisateurs

Nous avons donc découpé notre projet en un nombre équivalent de sprint, chacun ayant pour but de réaliser la fonctionnalité correspondante.

Ce sprint s'est terminé le 8 janvier. La remise du cahier des spécifications s'était fait le 5 auprès de notre encadrant. Les derniers jours de ce sprint ont été mis au profit du poster de présentation de VizAssist et à la préparation au début du développement.

3.2.2 Sprint 1 : Mise en place des différents environnements

Le premier objectif de ce sprint fut de mettre en place la structure et l'environnement nécessaire au bon déploiement du projet jusqu'à sa livraison : les environnements de développements, de déploiement et de tests.

Le second objectif fut de développer le module serveur. Celui-ci doit charger les fichiers de données de M. Venturini sur les différentes visualisations, les objectifs ainsi que la table des scores objectifs-visualisations. Nous avons donc mis en place des méthodes génériques en PHP permettant de lire les CSV.

Le troisième et dernier objectif est de créer la page d'accueil du site pour tester le bon fonctionnement de l'application sur le navigateur Web.

On a donc eu pour but d'effectuer dans cette période la mise en place du front-end et du back-end de l'application. A la fin de cette période :

— Un serveur Apache était installé
— Nos différents IDE (Sublime Text pour Jean et NetBeans pour Rémy) étaient configurés
— La structure de test Mocha et PHPUnit étaient configurées

— La page d'accueil était terminé, vérifiant si le JavaScript et les SVG étaient autorisés sur le navigateur client.

Ce sprint s'est terminé le 15 Janvier, avec la série de test PHPUnit.

3.2.3 Sprint 2 : Mise en place du module dit "Data"

Ce sprint avait pour but de mettre en place la gestion complète des différents jeux de données,ainsi que des différents moyen d'importation, à savoir par le biais de :

— Fichier de démonstration
— URL
— Fichier en local sur la machine Cliente

C'est durant cette période que nous avons mis en place le chargement, le contrôle et le visuel des données. Nous vérifions que le fichier est bien un fichier .csv, et qu'il est formaté de manière à ce que l'application puisse le traiter convenablement. Dans le cas contraire, un message d'erreur lui est spécifié. Si tout se passe bien, l'utilisateur peut passer à l'étape suivante, que nous avons développé dans le sprint 3 : Les objectifs.

Selon le modèle de VizAssist, un jeu de données est composé de plusieurs attributs de données, caractérisés par :

— Le nom de l'attribut
— Le type de cet attribut
— Leur importance
— Le nombre de valeurs différentes de cet attribut
— Les valeurs en elle même

Data sample from the selected dataset

Attribute Name	Petal width	Petal length	Sepal length	Sepal width	Class
Attribute Type	numeric	numeric	numeric	numeric	nominal
Importance	50	50	50	50	50
Max nb. of values	150	150	150	150	3
	5.1	3.5	1.4	0.2	Iris-setosa
	4.9	3.0	1.4	0.2	Iris-setosa
	4.7	3.2	1.3	0.2	Iris-setosa
	4.6	3.1	1.5	0.2	Iris-setosa
	5.0	3.6	1.4	0.2	Iris-setosa

FIGURE 3.2 – Exemple de visualisation de data

De plus, en nous basant sur notre prototype, nous avons pu en ressortir une première IHM basique. Ce sprint a donc eut pour but de fixé une première interface validé par Monsieur Venturini.

Pour finir, une première version de notre module Navigation fut créé à ce moment là de notre projet.

Lors de notre première réunion, nous avons défini que l'utilisateur pouvait avoir l'idée de télécharger un CSV corrompu et/ou non-valide. C'est pour cela que la création d'une fonction ControlData était nécessaire.

Nous avons pensé aussi que l'utilisateur pouvait avoir envie de vérifier ces données avant de rentrer dans le coeur de l'application, c'est pour cela que la méthode ShowData a été mise en place.

Les sécurités au niveau de la navigation partent de l'hypothèse que l'utilisateur peut avoir un oubli quant au remplissage des différents champs.

Ce sprint s'est terminé le 29 Janvier, avec les tests sur toute la partie Data, et une vérification par Monsieur Venturini, notre encadrant.

3.2.4 Sprint 3 : Mise en place du module dit "Objectives"

Le but de ce sprint fut de mettre en place tout le module que nous avons nommé "Objectives".

Celui-ci se réfère à la troisième étape de notre workflow : le choix des objectifs de l'utilisateur.

L'utilisateur en se rendant à cette étape doit avoir trois possibilités :

— Choisir les objectifs qu'il souhaite dans la liste proposée,
— Choisir de ne pas sélectionner d'objectif (au cas où il ne sait pas),
— Nous suggérer de nouveaux objectifs via email

Ce module devait donc effectuer un chargement de la liste des objectifs. Celle-ci se trouve sur le serveur dans un fichier CSV. Une fois chargé, ce fichier est parsé pour enfin être affiché sous forme de liste, où l'utilisateur peut effectuer ses choix. Si l'utilisateur trouve que les objectifs affichés ne permettent pas d'exprimer son besoin, on propose l'envoie d'un email pour expliquer ce qu'il lui manque.

Nous avons voulu effectuer un code qui permet de simplement rajouter un objectif dans le fichier CSV correspondant pour que cela soit effectif sur le site, en le plaçant au bon endroit. Un objectif est caractérisé ainsi par notre application :

— Un type : "Data mining tasks" ou "Properties of visualizations"
— Un nom, par exemple : "Analyse", "Cluster", "Zoom"
— Une cible, par exemple : "Data", "Attribute"

C'est au vu de cet envoi de mail, que nous avons du mettre en place toute la structure PHP pour cette fonctionnalité et c'est aussi à partir de là, que nous avons mis en place le menu, afin que l'utilisateur puisse envoyer un retour ou un report de bug. En utilisant la même fonction d'email, mais en modifiant légèrement l'apparence de la fenêtre de dialogue, on a ainsi pu mettre rapidement en place cette fonctionnalité.

En sortie de ce sprint, on obtient un vecteur de booléen, représentant les objectifs sélectionnés par l'utilisateur.

Ce sprint s'est terminé le 11 Février, avec les tests sur toute la partie Objectif.

3.2.5 Sprint 4 : Mise en place de la partie Scoring et Matching

L'objectif du quatrième sprint était de mettre en place tout le module pour effectuer l'appariement entre les données, les objectifs et les visualisations. Celui ci se réfère à l'étape intermédiaire entre les objectifs et la pré-sélection de notre workflow. En effet, le résultat de ce score et de cet appariement est primordial, puisque seules les visualisations valides (qui ont été appariées) sont triées par ordre décroissant du score résultant des objectifs.

Selon le modèle de VizAssist, une visualisation est caractérisée par des attributs visuels. Chaque attribut visuel est de la forme :

— Le nom de la visualisation dont il fait parti, par exemple : "2DSCATTER", "2DSCATTERCOLOR-NOMINAL"...
— La description de la visualisation, par exemple : "2D scatter plot", "2D scatter plot with colors for class"...
— Le nom du fichier d'une miniature de cette visualisation : "2dscatterplot.jpg"
— Le nom de l'attribut visuel : "X", "Y"

— Le type de l'attribut visuel : "position", "color"
— Le type de l'attribut de donnée correspondant : "numeric", "nominal"
— L'importance de l'attribut visuel : une valeur comprise entre 0 et 100
— La capacité de cet attribut visuel : une valeur limitant le nombre de valeur que cet attribut visuel peut accepter

Par exemple, la visualisation effectuant un simple nuage de points sera rédigé ainsi dans la liste :

```
2DSCATTER,2D scatter plot,2dscatterplot.jpg,X,position,numeric,99,2000
2DSCATTER,2D scatter plot,2dscatterplot.jpg,Y,position,numeric,99,2000
```

Et la visualisation effectuant un nuage de points, avec une couleur différente pour des points de différentes classes, sera ainsi :

```
2DSCATTERCOLORNOMINAL,2D scatter plot with colors for class,
    2dscatterplotwithcolorsforclass.jpg,X,position,numeric,99,2000
2DSCATTERCOLORNOMINAL,2D scatter plot with colors for class,
    2dscatterplotwithcolorsforclass.jpg,Y,position,numeric,99,2000
2DSCATTERCOLORNOMINAL,2D scatter plot with colors for class,
    2dscatterplotwithcolorsforclass.jpg,C,color hue,nominal,100,10
```

Chaque visualisation est ainsi déclarée. Ce modèle force la répétition des informations, mais distingue parfaitement chaque visualisation d'une autre. On aurait pu généraliser cette liste, en faisant une unique catégorie 2DSCATTER et en décrivant tous les attributs visuels disponibles pour ce type de visualisation, par exemple. Mais cela aurait eu une répercussion énorme sur les méthodes de dessin de ces visualisations, où nous aurions dû tester chaque attribut visuel pour savoir s'il était représenté ou non.

3.2.5.1 L'appariement avec les données

L'appariement d'un attribut de donnée avec un attribut visuel se fait selon deux règles :

— Un attribut visuel n'accepte qu'un attribut de donnée de type correspondant, par exemple : "nominal", "numeric"
— Un attribut visuel n'accepte qu'un attribut de donnée ayant un nombre de valeurs différentes inférieur à sa capacité.

Ce dernier point est indispensable pour certains attributs visuels, comme les couleurs. Par exemple, il a été étudié que l'oeil humain ne distinguait pas plus d'une dizaine de nuances de couleurs différentes. Il est donc important de mettre cette limitation en place.

L'algorithme se passe ensuite en deux parties. On trie par importance décroissante, les attributs de données ainsi que les attributs visuels pour chaque visualisation. On parcourt une première fois notre liste

de visualisation, en essayant de matcher nos données à nos attributs visuels selon les deux règles dictées.

```
Data: Partie A du matching entre donnees et visualisations
Result: Apparie des attributs de donnees aux attributs visuels
trierParImportanceDecroissante(attributsDonnees) ;
forall the visualisations do
    trierParImportanceDecroissante(visualisations.attributsVisuels) ;
end
while pas a la fin des visualisations do
    // Parcourt d'une visualisation
    forall the attributsDonnees do
        forall the attributVisuel do
            if attributVisuel.capacite >= attributDonnees.nbValeursDifferentes &&
            attributVisuel.typeCorrespondant == attributDonnees.type then
                attributVisuel.matchedAttribute = attributDonnees.nom ; break ;
            end
        end
    end
end
```

Algorithm 1: Comment sont appariés les attributs de données et attributs visuels

La deuxième partie contrôle le matching effectué et valide une visualisation selon une règle simple : si un attribut visuel d'importance supérieure à 50 n'a pas été apparié, alors la visualisation n'est pas valide. En reprenant l'exemple de notre nuage de points, cela signifierait que notre axe X ou Y n'ait pas pu être apparié. Sans l'un des deux attributs visuels, notre visualisation n'a aucun sens. Elle est donc invalidée.

```
Data: Partie B du matching entre donnees et visualisations
Result: Valide ou non les visualisations
forall the visualisations do
    forall the attributsVisuels do
        if attributVisuel.importance >= 50 && attributVisuel.matchedAttribute == null then
            attributVisuel.valide = true ;
        else
            attributVisuel.valide = false ; invalider(attributsVisuels) ; // On invalide les autres
            attributs visuels de cette visualisation
            break ;
        end
    end
end
```

Algorithm 2: Comment sont validées les visualisations

Ainsi, si l'utilisateur entre un jeu de donnée ne comprenant que des attributs de données numériques et nominaux, alors on ne lui proposera pas de série à base de séries temporelles ou de graphes de noeuds. C'est sur cet algorithme et ces simples règles que se base tout VizAssist.

3.2.5.2 Le calcul des scores

Nous effectuons un calcul de deux scores différents pour chaque visualisation : le score des objectifs ainsi que le score des données.

Pour calculer le score des objectifs, on les a tous pondéré d'une valeur entre 0 et 100 pour chaque visualisation. En sortie du précédent sprint, nous obtenions un vecteur de valeurs booléennes représentant les objectifs sélectionnés par l'utilisateur.

Soient $o_{i,j} \in [0, 100]$, le score de l'objectif i pour la visualisation j, $u_i \in \{0, 1\}$ le booléen associé à l'objectif i et $v_j \in [0, 100]$ le score de la visualisation j, on a donc :

$v_j = \sum_{i=0}^{n} o_{i,j} * u_i$, avec n le nombre d'objectifs

On normalise ensuite par le nombre d'objectif sélectionnés k :

$k = \sum_{i=0}^{n} u_i$

$v_j = v_j / k$

Ensuite on calcule le score de l'appariement effectué entre les données et les attributs visuels. Celui-ci permet de noter le rendu des données sur une visualisation. Un score parfait signifierait pour une visualisation que tous les attributs de données aient été appariés à tous les attributs visuels. On a donc mis en place la formule suivante : soient d_i l'importance de l'attribut de donnée i, $v_{i,j}$ l'importance de l'attribut visuel i de la visualisation j et u_i le booléen à 1 si l'attribut visuel est apparié :

$score_j = \sum_{i=0}^{n} d_i * v_{i,j} * u_i$ avec ici, n le nombre d'attributs de données

On normalise ensuite ce score par le score idéal :

$scoreIdeal = \sum_{i=0}^{n} d_i * 100$

$score_j = score_j * scoreIdeal$

Ce module de scoring et matching permet donc de calculer les différents scores résultant du matching. Ce sprint s'est terminé le 13 Février, avec les tests à partir de nos jeux de données ainsi que des tests unitaires à l'aide de Mocha.

3.2.6 Sprint 5 : Mise en place de la partie dite "Previsualization"

L'objectif du cinquième sprint fut de mettre en place tout le module que nous avons nommé "Previsualization". Celui ci se réfère à l'étape de pré-sélection de notre workflow. L'utilisateur en se rendant à cette étape doit avoir quatre possibilités :

- Sélectionner toutes les visualisations (triées par ordre décroissant de score)
- Pouvoir faire un choix aux niveaux des différentes visualisations
- Désélectionner les différentes visualisations précédemment choisis.
- Visualiser le résultat du matching des données avec les attributs visuels

Ce sprint devait donc mettre en place le module qui effectue un chargement de la liste des visualisations, les trie par ordre décroissant de score, ne propose que les visualisations disponible en fonction des données précédemment choisi (à l'étape 1) et récupérer les choix de l'utilisateur.

Le tableau de récapitulatif des visualisations ne doit proposer que des visualisations correspondant aux données de l'utilisateur. Si un attribut indispensable d'une visualisation n'a pas pu être matché, alors la visualisation n'est pas proposé.

L'intégralité de ces options se situe sur un fichier csv de l'application.

Ce sprint s'est terminé le 22 Février, avec les tests unitaires et l'intégration associée.

3.2.7 Sprint 6 : Mise en place de la partie "Visualization"

Ce sixième sprint avait pour but de mettre en place les différentes visualisations et de n'afficher que celles précédemment sélectionnées par l'utilisateur.

Une IHM spécifique a du être réfléchit afin que toutes les miniatures soient affichées et qu'une visualisation principal soient disponibles.

Un système de carrousel a donc été mis en place pour répondre à ce besoin.

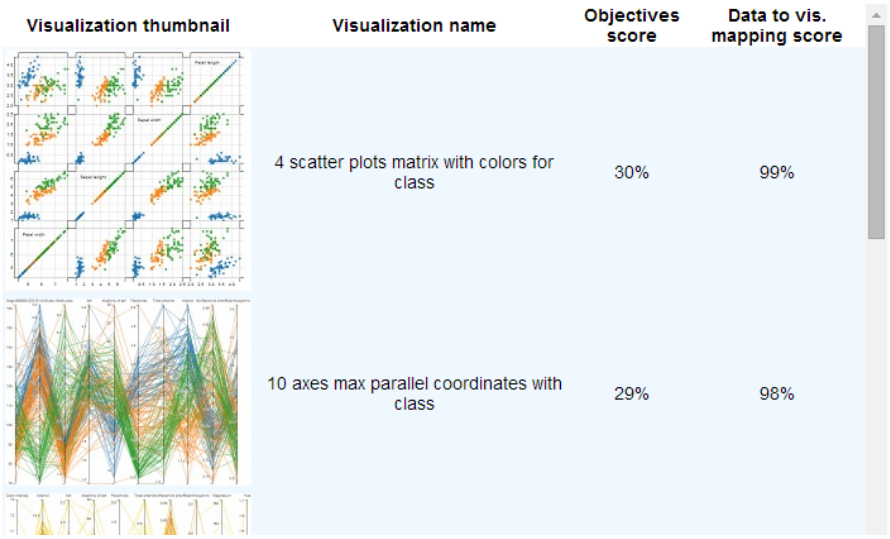


FIGURE 3.3 – Exemple de pré-visualisation

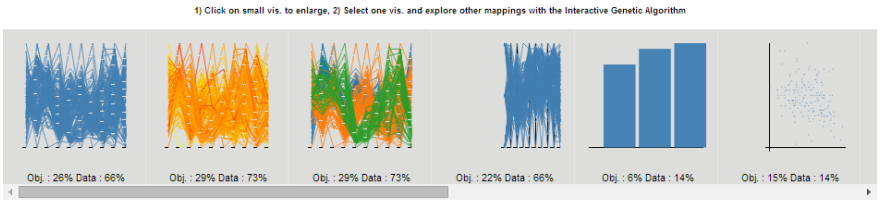


FIGURE 3.4 – Illustration du carrousel mis en place

En effet, la deuxième contrainte, après la sécurité des données, était de pouvoir ajouter simplement de nouvelles visualisations. La liste des visualisation étant chargée dynamiquement depuis le fichier CSV correspondant sur le serveur, en ajoutant de nouveaux attributs visuels à cette liste, on modifie ce que propose l'application. Ensuite, la méthode de dessin de cette visualisation doit être ajoutée aux fichiers sources.

VizAssist gérera ensuite tout seul le chargement des visualisations à l'aide de la librairie RequireJS, qu'on décrit plus tard en partie 3.5.3. VizAssist ne charge que les visualisations précédemment sélectionnées par l'utilisateur. Ceci permet de ne pas alourdir l'application en chargeant l'intégralité de nos fichiers de visualisation. On espère que par le futur, le nombre de visualisation va considérablement augmenté. Il était donc nécessaire d'optimiser ce téléchargement des sources.

3.2.8 Sprint 7 : Algorithme Génétique Interactif et Mise en production de l'application

C'est lors de ce sprint que notre binôme s'est séparé vers des tâches spécifiques. Rémy décrit l'implémentation de son algorithme génétique interactif en partie 4. Jean de son côté explique comment il a mis en production l'application en partie 5

3.3 Architecture du système

L'architecture du système est une architecture MVC. Notre architecture manipule trois objets simples :
— Un ensemble de visualisation

— Un ensemble d'objectifs
— Un jeu de donnée

La vue est gérée par les fichiers HTML, qui sont contrôlés par des classes JavaScript. Notre application est principalement côté client. On ne retrouve pas une architecture proposée par un framework PHP comme Symfony ou CakePHP. Le serveur PHP dans notre cas ne fait que livrer des ressources pour que notre application "in-browser" puisse fonctionner correctement.

3.3.1 Interfaces logiciel/logiciel

Notre système possède deux interfaces internes logiciel/logiciel. La première est l'interface entre les tableaux CSV et notre application. La seconde se situe entre le serveur et le navigateur du client, via des requêtes HTTP et AJAX.

3.3.1.1 Interface avec les fichiers CSV

Il y a deux cas où cet interfaçage a lieu. Une première fois sur le serveur, où sont situés les deux tableaux au format CSV et qui contiennent la liste des objectifs et des visualisations. Ensuite, si l'utilisateur choisi un fichier de démonstration de données, le serveur va lire ces données dans les jeux de données du serveur. PHP contient des méthodes d'accession aux fichiers, ce qui nous permet de récupérer et de parser nos fichiers CSV.

Le second cas où nous avons besoin d'une interface avec les fichiers CSV se trouve directement dans notre application JavaScript. Lors du chargement de donnée de l'utilisateur, on se doit de pouvoir lire ces données qu'en local. C'est l'une des contraintes du projet qui veut ça. Les données de l'utilisateur peuvent être sensibles et nous devons à tous prix les garder en local. L'interface entre les données de l'utilisateur et notre application JavaScript se fait avec l'API File, issue de HTML5 [1].

3.3.1.2 Interface HTTP

La deuxième interface est interne à notre logiciel et se situe entre notre client et notre serveur : front-end to back-end. La discution entre les deux parties est effectuée au travers de requète HTTP et plus précisément à l'aide d'objet XMLHttpRequest pour les requêtes AJAX. Ces objets permettent la communication entre notre serveur Apache et le navigateur Web de l'utilisateur, sans avoir à faire de requête POST.

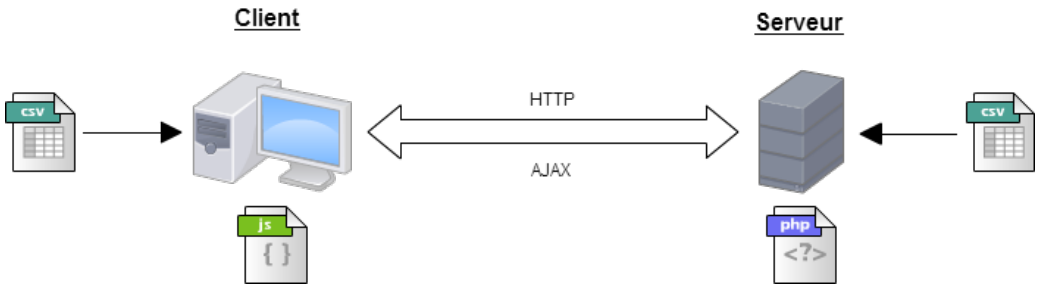


FIGURE 3.5 – Les différentes interfaces de notre logiciel

3.3.2 Interfaces homme/machine

3.3.2.1 Général

Afin de définir les limites de notre application et les besoins de M. Venturini, nous avons effectué un prototype du site. Celui-ci définit dans les grandes lignes la navigation sur le site et l'organisation des

1. Lire la partie décrivant les technologies employées : 3.4.2.4

différentes pages.

VizAssist a pour but d'assister l'utilisateur dans la visualisation de ses données. De notre point de vue, la meilleur façon de représenter visuellement cela est d'effectuer une interface similaire à celle d'un wizard d'installation de logiciel. C'est à dire de proposer plusieurs étapes consécutives les plus simples possibles à l'utilisateur, pour arriver à ses fins.

Comme montré précédemment, nous aurons donc une interface séparée en plusieurs étapes. Chacune de ces étapes représente une fonctionnalité. On a ainsi 6 étapes à illustrer :

— Un accueil
— Choix des données à visualiser
— Choix des objectifs de l'utilisateur
— Choix des visualisations que nous lui proposons, en fonction de ses données et de ses objectifs
— Affichage des visualisations
— Application de l'algorithme génétique interactif

Nous avons donc au final six fenêtres à générer. Un système de notification pour l'utilisateur semble essentiel pour le prévenir de diverses évènements tel qu'un formulaire invalide, la validation d'un champ etc.

3.3.2.2 Mise en Oeuvre

Notre application ne possède pas de base de données, ni de gestion de profils utilisateurs. Elle ne doit pas non plus stocker les jeu de données de l'utilisateur pendant sa session. C'est pourquoi nous avons donc choisi de tout effectuer sur une seule page. Ainsi, l'utilisateur ne voit pas de changement de pages, seul son contenu différera. Cela amène une lisibilité dans le déroulement du processus. Ceci est très facilement mis en place en JavaScript avec l'intégration de l'AJAX à la formule. L'utilisateur a ainsi une suite de "vues" que nous modéliserions par différentes fenêtres sur un wizard classique.

3.3.2.3 Illustration des différentes vues

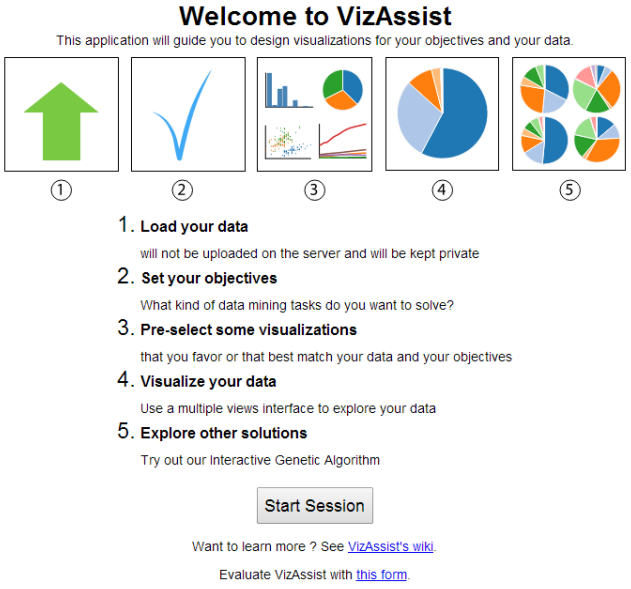


FIGURE 3.6 – Page d'accueil

3.3.2.3.a Etape 0 : Accueil

La page d'accueil sert à plusieurs choses. Tout d'abord, elle doit permettre à l'utilisateur de comprendre le but de l'application et lui présenter les différentes étapes. Dans un second temps, en arrière plan, elle vérifie que le JavaScript est bien activé sur le navigateur du client et que celui est compatible avec les technologies utilisée par VizAssist.

Enfin, on dispose des liens vers le wiki de l'application et vers un formulaire d'évaluation, afin les utilisateurs sachent que leur avis nous intéresse.

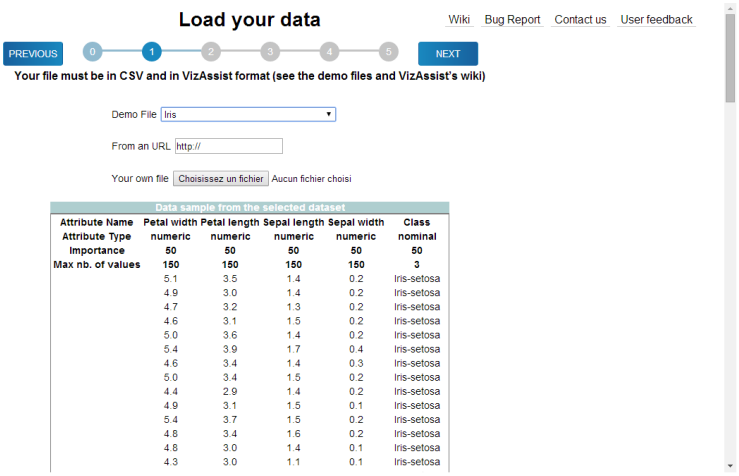
3.3.2.3.b Etape 1 : Chargement des données

Viens ensuite la page de chargement des données. Comme on peut l'apercevoir, il existe deux méthodes différentes pour l'utilisateur d'importer des données, à savoir :

— Par URL

— Par upload de fichier

Afin de pouvoir tester la puissance de VizAssist, des fichiers de démonstration ont été créé avec des visualisations diverses et variées associées à ces fichiers.



Attribute Name	Petal width	Petal length	Sepal length	Sepal width	Class
Attribute Type	numeric	numeric	numeric	numeric	nominal
Importance	50	50	50	50	50
Max nb. of values	150	150	150	150	3
	5.1	3.5	1.4	0.2	Iris-setosa
	4.9	3.0	1.4	0.2	Iris-setosa
	4.7	3.2	1.3	0.2	Iris-setosa
	4.6	3.1	1.5	0.2	Iris-setosa
	5.0	3.6	1.4	0.2	Iris-setosa
	5.4	3.9	1.7	0.4	Iris-setosa
	4.6	3.4	1.4	0.3	Iris-setosa
	5.0	3.4	1.5	0.2	Iris-setosa
	4.4	2.9	1.4	0.2	Iris-setosa
	4.9	3.1	1.5	0.1	Iris-setosa
	5.4	3.7	1.5	0.2	Iris-setosa
	4.8	3.4	1.6	0.2	Iris-setosa
	4.8	3.0	1.4	0.1	Iris-setosa
	4.3	3.0	1.1	0.1	Iris-setosa

FIGURE 3.7 – Page des données

Des sécurités ont été mis en place afin d'éviter l'importation de fichier corrompu. Une notification de succès ou d'erreur s'affiche en fonction du cas rencontré.

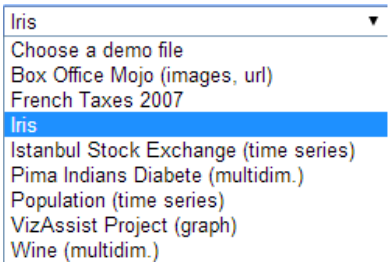


FIGURE 3.8 – Liste des exemples de Data

3.3.2.3.c Etape 2 : Choix des objectifs

Après avoir importé ses données, l'utilisateur a accès à la partie "Objectifs" de l'application. Cette partie comporte trois grandes catégories, en fonction de ces besoins, à savoir :

— Il ne sait pas ce qu'il désire
— Il sait ce qu'il désire mais nous ne lui proposons pas
— Il sait ce qu'il désire et ceci est disponible dans la liste.

Dans le premier cas, l'application s'occupera de tout, en calculant un score "moyen" à chaque visualisation. Lorsqu'il ne sait pas, nous considérons qu'il a tout coché.

Dans le second cas, une fenêtre de dialogue s'ouvre pour qu'il puisse nous envoyer son idée.

Dans le dernier cas, nous ne tenons compte que des objectifs qu'il a coché.

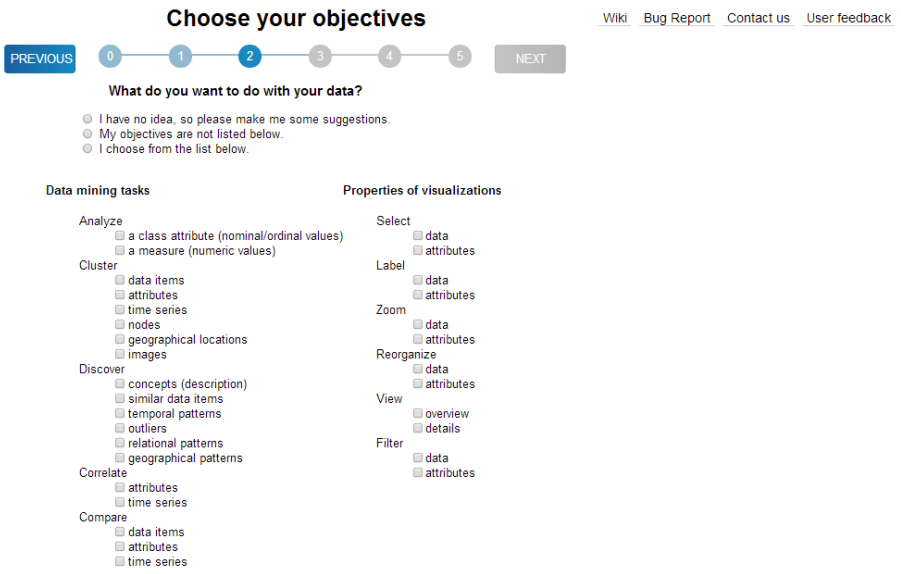


FIGURE 3.9 – Page des objectifs

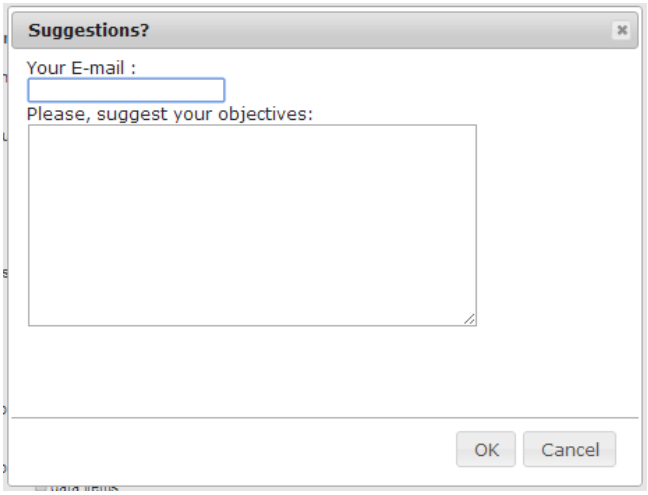


FIGURE 3.10 – Fenêtre de dialogue

3.3.2.3.d Etape 3 : Choix des visualisations

Ses objectifs choisis, l'utilisateur a accès aux prévisualisations. C'est grâce à notre matching de données que nous pouvons proposer des visualisations. Un score est calculé en fonction des objectifs et un second score en rapport avec le nombre de données qu'une visualisation arrive à utiliser.

En effet, un "ScatterPlot" basique n'utilise que deux attribut de données dans un jeu, alors qu'un "ScatterMatrix" en utilise quatre. C'est pour cela que le second score sera plus élevé dans un ScatterMatrix que dans un ScatterPlot.

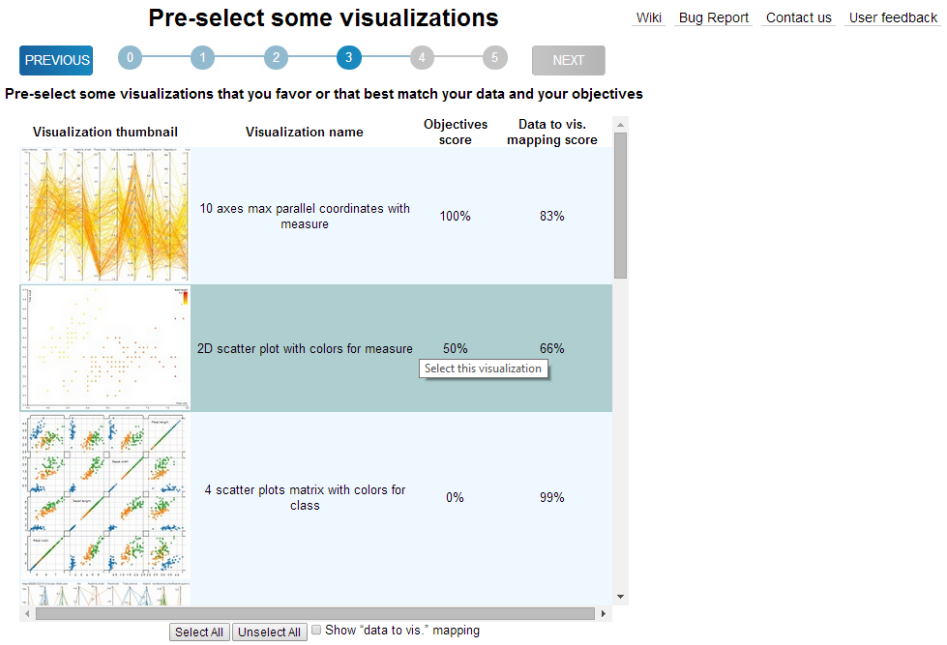
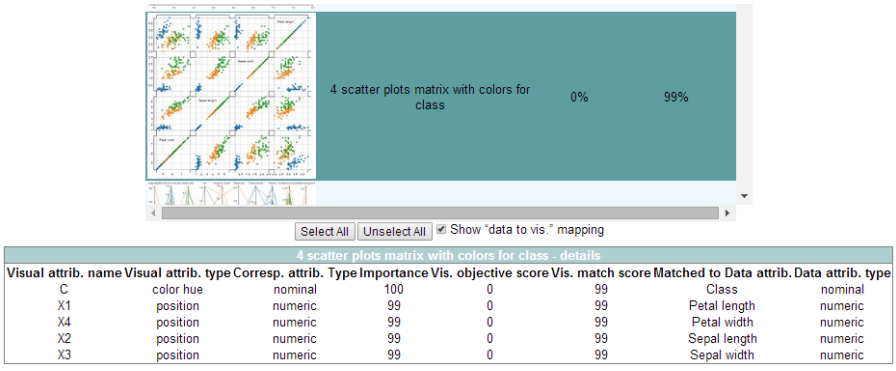


FIGURE 3.11 – Page des previsualisations

L'utilisateur possède trois fonctionnalités sur cette page. En effet, il peut sélectionner l'intégralité des visualisations, afin de vérifier de lui même les rendus à la page suivante ; les dé-sélectionner ; mais surtout, il a la possibilité de voir le matching des données qui a été effectué.



4 scatter plots matrix with colors for class - details

Visual attrib. name	Visual attrib. type	Corresp. attrib. Type	Importance	Vis. objective score	Vis. match score	Matched to Data attrib.	Data attrib. type
C	color hue	nominal	100	0	99	Class	nominal
X1	position	numeric	99	0	99	Petal length	numeric
X4	position	numeric	99	0	99	Petal width	numeric
X2	position	numeric	99	0	99	Sepal length	numeric
X3	position	numeric	99	0	99	Sepal width	numeric

FIGURE 3.12 – Visualisation du matching effectué

3.3.2.3.e Etape 4 : Visualisation

La partie visualisation permet à l'utilisateur de visualiser ces données avec les visualisations qu'il a choisies à l'étape précédente. Un carrousel lui permet de naviguer entre les différentes visualisations. Les légendes sont disponibles sur la droite de la visualisation.

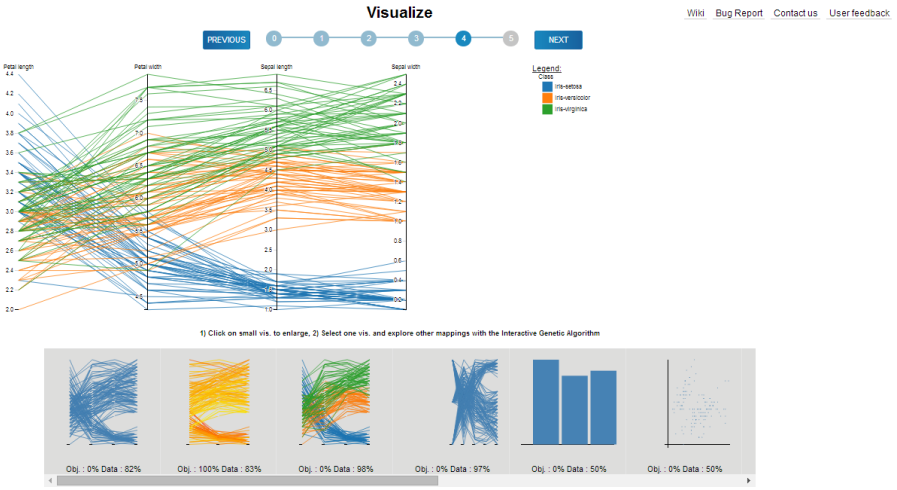


FIGURE 3.13 – Visualisations

3.3.2.3.f Etape 5 : Algorithme Génétique Interactif

Afin de garder le même type de design que pour le reste de l'application, l'étape 5 reste "mono-page". En effet, la visualisation principal de l'étape précédente est celle sélectionnée pour l'algorithme génétique.

Le pannel d'enfants est situé sur la gauche, ce qui permet de tout avoir sur la même page.

Le "panneau de contrôle" est situé au dessus de tout affichage, afin que l'utilisateur trouve directement le contrôle de cette partie. Ce panneau est composé de quatre parties :

— Un bouton Reset, qui permet de réinitialiser l'algorithme
— Un bouton Next Generation, qui permet de poursuivre l'algorithme
— Un range, concernant la diversité des prochaines génération
— Un checkbox, permettant à l'utilisateur de voir, ou non, le détails des importances des nouvelles visualisations

A coté de la légende, se trouve un tableau de détails sur les nouvelles importances d'une visualisation. Ce qui permet à l'utilisateur de voir l'évolution de l'algorithme.

FIGURE 3.14 – Page sur L'Algorithme Génétique Interactif

3.3.2.3.g Menu

Hormis sur la page d'accueil, un menu est accessible à tout moment dans l'application. Ce menu permet de :

— Accéder au wiki
— Envoyer un rapport de bug
— Envoyer un mail
— Répondre à un questionnaire de retour utilisateur

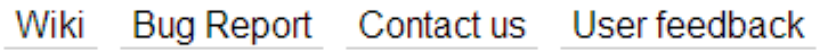


FIGURE 3.15 – Menu de VizAssist

3.3.3 Diagramme de classe

Notre application est séparée en deux parties : la partie PHP et la partie JavaScript, qui communiquent au travers de requêtes AJAX. Le rôle de la partie PHP est de lire dans des fichiers CSV et de mettre ces données à disposition de l'application JavaScript.

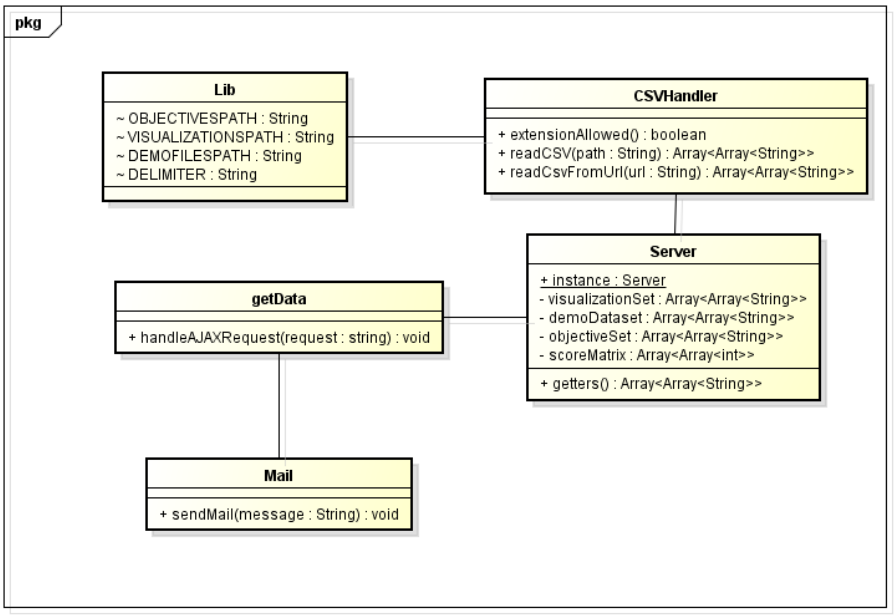


FIGURE 3.16 – Diagramme des classes PHP

La figure 3.16 montre comment est organisé notre serveur PHP. La classe Lib possède les constantes des différents chemins vers nos fichiers de données, ainsi que le délimiteur utilisé dans nos fichiers CSV. La classe CSVHandler lit les fichiers CSV à l'aide des constantes données.

La classe getData va jouer le rôle d'interface avec le JavaScript. Elle reçoit les requètes AJAX et traite ensuite la demande, soit en appelant la classe Serveur pour obtenir une liste de visualisations par exemple, soit pour faire la demande d'envoi d'un email, via la classe Mail.

Côté JavaScript, nous avons au centre de nos contrôleurs, une classe Navigation, qui gère le chargement des différents modules au fur et à mesure de la navigation de l'utilisateur. Sur la figure 3.17, les contrôleurs sont rangés par ordre de chargement. A partir des boutons "Next" et "Previous", montrés à la section

précédente, on indique à Navigation quelle vue charger au travers de son attribut index. Le contrôleur associé est chargé en même temps.

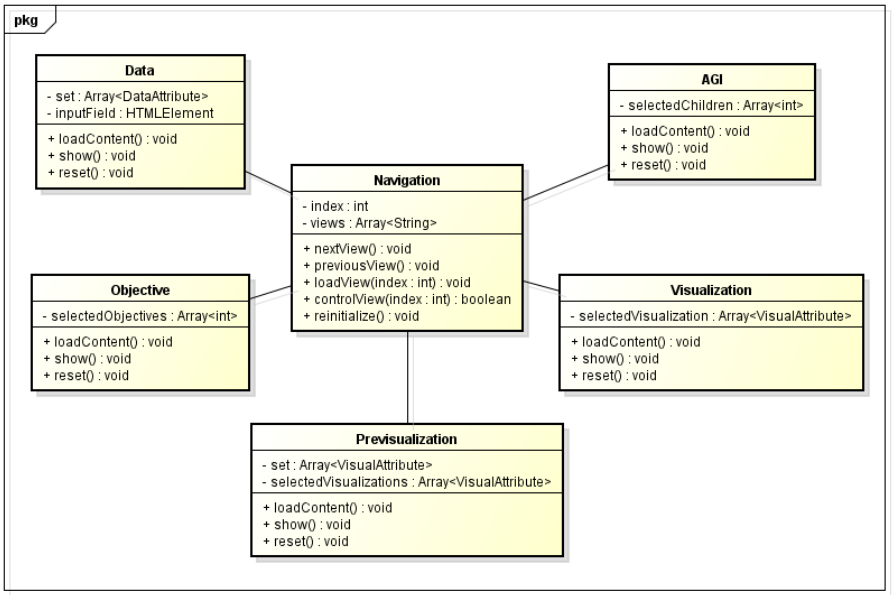


FIGURE 3.17 – Relation de la classe Navigation avec les différents contrôleurs

Le diagramme de classe de notre application 3.17 a été simplifié pour ne montrer que les points intéressants et les interactions qu'effectuent Navigation avec les contrôleurs.

Le chargement d'un contrôleur se fait en deux temps. Tout d'abord, on demande au serveur les ressources nécessaires (liste des objectifs, listes des visualisations...) au travers de la méthode loadContent(), puis show() permet d'afficher les différents contenus dynamiques.

Au fur et à mesure que l'utilisateur avance, ses choix et actions sont sauvegardés dans chaque contrôleur (ses objectifs choisis, ses données chargées...) qui lui permet, en revenant en arrière de modifier ses entrées. Cependant, si les attributs d'un contrôleur sont modifiés, cela entraîne la réinitialisation des contrôleurs suivants. De nombreux paramètres dépendent de ce qui a été effectué auparavant, il est donc nécessaire de réinitialiser la suite de l'application. Chaque contrôleur possède donc une méthode reset() qui est appelé par Navigation lors de sa méthode reinitialize().

Nous sommes conscients que notre diagramme de classes JavaScript 3.17 aurait pu être organisé d'une manière différente, par exemple par un simple objet Controlleur qu'on aurait décliné plusieurs fois pour nos différents modules. Cependant, le JavaScript n'a aucune notion native d'héritage. Nous avons donc restreint notre charte de programmation pour respecter une convention similaire selon nos contrôleurs, pour rester cohérent.

3.4 Technologies utilisées

Comme vu précédemment, on peut séparer notre application en deux parties distincts, à savoir le côté serveur et le côté client.

3.4.1 Côté serveur

Pour la partie serveur, nous avons utilisé deux technologies, qui travaillent conjointement à savoir le serveur Apache et le langage de programmation PHP.

3.4.1.1 Apache HTTP Server

Apache HTTP Server est un serveur HTTP libre, considéré comme le plus populaire par le World Wide Web.

Apache est capable d'interpréter des langages tel que :
— Perl
— PHP
— Python
— Ruby
— ...

La principal particularité d'Apache est sa simplicité de configuration. En effet, chaque fichiers de configuration peut être géré de manière totalement indépendante, ce qui permet aux fournisseurs d'hébergement de n'avoir qu'un serveur HTTP pour plusieurs sites web. Les clients du fournisseur peuvent s'en rendre compte grâce au fichier .htaccess.

3.4.1.2 PHP

PHP, signifiant PHP : Hypertext Preprocessor, est un langage libre de programmation, utilisé principalement pour le web à travers un serveur HTTP.

Créé en 1994, a atteint la version 5 depuis 2004. Cette version a apporté une modélisation objet beaucoup plus performante, le modèle d'exception a été adopté pour la gestion des erreurs. PHP 5 apporte beaucoup de nouveautés, telles que le support de SQLite, une base de données légère.

3.4.2 Côté client

Du côté client, les technologies mise en place sont restées très basiques, en utilisant HTML5, CSS3 et Javascript. Dans le cadre de notre application, nous avons aussi utilisé le SVG, dans sa version 1.1 et File API.

3.4.2.1 Document Object Model

Dans un premier temps, nous allons voir, ce qu'est le DOM. En effet, ce standard du W3C, permet d'accéder ou de mettre à jour , la structure ou le contenu de document de type XML, donc xHTML.

Le DOM n'a pas toujours été standardisé, ce qui laissait les navigateurs Web de posséder le leur. Le problème était, par exemple, que Netscape conseillait de parcourir un tableau indexé nommé document.layers[], tandis qu'Internet Explorer préférait document.all[]. Cela obligeait les développeurs de sortir deux versions de script au minimum afin d'avoir une compatibilité internavigateur.

Chaque nouvelle version du DOM, rajoute de nouvelle fonctionnalité, sans jamais remettre en cause les précédentes, ce qui fait qu'aujourd'hui, le DOM est fourni et empêche la duplication de script comme avant cette standardisation.

3.4.2.2 HTML5 et CSS3

Le HTML (HyperText Markup Language) est un format de données permettant de faire des pages web. La dernière version en date, la version 5 a apporté un vent de fraîcheur dans le monde du Web. En effet, 10 ans après la sortie de HTML 4, de nouvelles fonctionnalités sont apparu telles que :

— De nouvelles balises, permettant une meilleur lisibilité (header, footer, section, article ...)
— Un doctype plus lisible
— Tout ceci permettant un meilleur référencement

Cependant, pour compléter les nouvelles fonctionnalités d'HTML5 de manière optimale, il faut le coupler avec CSS (Cascading Style Sheets) dans sa dernière version, la version 3. Ces nouvelles feuilles de styles apportent :

— L'ajout de fonctionnalité de manière native, tel que les ombrages ou les bords ronds
— Les animations
— ...

Ces deux technologies, récentes, aident à la création de véritables applications Web et ont amené ce que l'on appelle aujourd'hui le Web 2.0.

3.4.2.3 JavaScript

Le JavaScript est un langage, comme son nom l'indique, de script, utilisé dans le monde du Web pour le côté interactif. Mais depuis quelques années, il est aussi utilisé côté serveur avec notamment NodeJS.JavaScript est un langage orienté objet à prototype. Cela signifie que les bases du langage ainsi que ses interfaces sont fournies par des objets qui ne sont pas des instances de classes.

Voici l'exemple classique du « Hello world » en JavaScript :

```
1            window.alert("Hello world");
```

Cependant, comme la classe DOM window est récurrente, on peut se permettre de se passer de l'écrire. On écrira donc :

```
1            alert("Hello world");
```

A une époque, JavaScript avait pour principale utilité le contrôle des données dans les formulaires HTML. Puis sont arrivées les interactions avec le document HTML via le DOM, fourni par le navigateur.

Ensuite, il a été utilisé pour des raisons purement cosmétique comme par exemple les animations jQuery.

Cependant, JavaScript n'est pas limité à la manipulation de documents HTML et peut aussi servir à manipuler des documents SVG, chose que nous faisons avec D3.

3.4.2.4 L'API File

Introduite par HTML5, cette API provient de spécifications du W3C [2] qui permettent à une application Web d'avoir accès à la manipulation de fichiers en local. Elle permet ainsi à l'utilisateur d'accéder à des

2. http://www.w3.org/TR/FileAPI/

données d'une application, hors connexion Internet. Ou alors à la lecture de fichier par l'application sans chargement de données vers le serveur.

Dans notre cas et selon nos contraintes, vous aurez compris l'importance de cette API pour VizAssist. C'est grâce à elle que nous effectuons le chargement des données de l'utilisateur, seulement dans son navigateur.

3.5 Librairies utilisées

3.5.1 D3.js

D3 (Data-Driven Documents) est une librairie JavaScript open source, qui offre des outils pour la visualisation de données. Elle a été créé par Mike Bostock et aide ainsi la manipulation de documents HTML, CSS et SVG, pilotées par les données.

```
1        /*
2        * Position est l'element HTML qui va contenir le rectangle
3        */
4        function appendRectangle(position) {
5            var svg = d3.select(position)
6                        .append('svg');
7
8            svg.append('rect')
9                .attr('id','a-rectangle')
10             .attr("width", 60)
11             .attr("height", 40)
12             .attr("x", 50)
13             .attr("y", 50);
14
15           return svg;
16       }
```

Listing 3.1 – Ajouter un simple rectangle à une position donnée avec D3.js

3.5.1.1 Manipulation du DOM

D3 permet donc d'attacher au Document Object Model des données, pour ensuite les transformer selon les besoins. Il devient ainsi très facile de joindre au DOM ses données extraites d'un tableur, pour les afficher dans un premier tableau HTML, puis de créer un histogramme à côté.

3.5.1.2 Créer une visualisation

Créer une visualisation revient à attacher des données au DOM, et à l'aide des éléments SVG et HTML, de dessiner sa visualisation.

```
1    // On definit le tableau des hauteurs de nos barres
2    var barHeight = [10, 50, 40, 70, 20];
3
4    d3.select('body')
5        .append('svg')
6        .selectAll('rect')
```

```
7         // On attache les donnees au DOM
8         .data(barHeight)
9         // et on itere sur chaque donnees du tableau passee en parametre de la
              methode data()
10        .enter()
11        // on ajoute notre element SVG
12        .append('rect')
13        // on peut ensuite definir les dimensions en dur
14        .attr('width', '20')
15        .attr('height', function(height) {
16            // ou alors en fonction des donnees contenues dans le tableau
17            return height;
18        })
19        .attr('x', function (height, i) {
20            // selon la position de la valeur dans le tableau,
21            // on decale notre forme sur les x
22            return i * 25;
23        })
24        .attr('y', function (height) {
25            // y commence en haut a gauche
26            // il faut donc inverser les mesures
27            return 100 - height;
28        })
29        .style('fill', 'steelblue'); // on definit le style CSS de notre forme
```

Listing 3.2 – Créer un bar chart

Ce snippet de code aura pour résultat la figure 3.18. Le code peut avoir une approche difficile car le passage de paramètre est implicite. Atteindre les données liées se fait par une méthode anonyme et peut prendre jusqu'à deux paramètres en entrée : la donnée actuellement lue et l'index de cette donnée dans le tableau lié (cf ligne 13).

FIGURE 3.18 – Résultat du snippet de code pour créer un bar chart

Cependant, en restant ainsi avec des dimensions statiques (dans l'exemple la position sur x), on va très vite se retrouver bloqué si notre jeu de données grandit. C'est pourquoi D3 apporte des outils de mise à échelle (linéaires, logarithmiques, ...) pour que, quelque soit la taille du jeu de données, le dessin puisse rester cohérent. En définissant un intervalle de mesure, et en définissant les valeurs minimums et maximums de nos données, on peut se retrouver avec des visualisations qui auront toujours le même comportement quelque soit l'échelle des valeurs et leur nombre.

Pour VizAssist, ces outils sont très importants. On doit pouvoir, quelques soient les données de l'utilisateur, lui offrir une visualisation convenable pour ces données. C'est à dire : qui ne dépasse pas, qui reste cohérente et lisible.

3.5.1.3 Introduction à ColorBrewer

ColorBrewer[3] est une librairie de couleur créée par Cynthia Brewer, professeur de géographie américaine qui a notamment travaillé sur la théorie des couleurs en cartographie.

3. http://colorbrewer2.org

Cette librairie a été adaptée pour D3 et permet d'accéder à différentes panoplies de couleur [4]. Le choix des couleurs pour les visualisations est donc assuré. En reprenant notre précédent exemple sur l'histogramme, on peut mettre en place une échelle de couleur selon la hauteur de chaque barre :

```
var barHeight = [10, 50, 40, 70, 20, 60, 35];

// definition de notre echelle de couleur,
// ici 10 couleurs differentes
var colors = d3.scale.category10();

// definition de notre domaine de valeurs
colors.domain(barHeight);

d3.select('body')
    .append('svg')
    .selectAll('rect')
    .data(barHeight)
    .enter()
    .append('rect')
    .attr('width', '20')
    .attr('height', function(height) {
        return height;
    })
          .attr('x', function (height, i) {
        return i * 25;
    })
.attr('y', function (height) {
    return 100 - height;
})
.style('fill', function (height) {
    // on retourne la valeur associee a cette hauteur
    // dans notre echelle de couleur
    return colors(height);
});
```

Listing 3.3 – Créer un bar chart avec des couleurs

FIGURE 3.19 – Résultat du snippet de code créant un bar chart coloré

3.5.1.4 Une librairie puissante

En somme, D3 est une puissante librairie pour créer des visualisations de tous types, avec une forte communauté de développeur et une riche documentation. Accompagné d'outils comme ColorBrewer ou encore TopoJSON pour des données géographiques, elle apporte tous les outils nécessaires à la visualisation de données et possède de nombreux exemples pour s'inspirer [5]. Après avoir créé de simples visualisations (nuages de points, histogramme...), nous avons commencé à intégrer des plus puissantes qu'on vous décrit par la suite dans la partie de Jean 5.3.

4. Toutes les échelles de couleur : http://bl.ocks.org/mbostock/5577023
5. Gallerie d'exemples de D3 : https://github.com/mbostock/d3/wiki/Gallery

3.5.2 jQuery et jQuery-UI

jQuery est une bibliothèque JavaScript libre qui porte sur l'interaction entre JavaScript (comprenant Ajax) et HTML, et a pour but de simplifier des commandes communes de JavaScript.

```
1         $('p').hide('slow');
```

Listing 3.4 – Cacher un paragraphe en 600ms

Bien que jQuery possède de nombreuses méthodes pour gérer le DOM, le CSS, AJAX ainsi que la gestion événementielle, certaine méthode d'interface utilisateur ne sont pas native à ce framework.En effet,

jQuery UI ("user interface") est très souvent couplé avec jQuery. Il est en quelque sorte une bibliothèque de méthodes, accessibles à travers un seul fichier JavaScript, facilitant l'interface utilisateur.

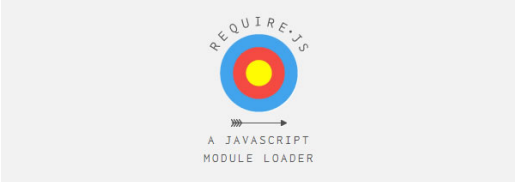
Quant à jQuery en lui même, c'est grâce à ce framework que nous avons pu mettre en place toute notre architecture de dialogue entre nos pages JavaScript (client) et nos pages PHP (serveur) de manière simplifié. En effet, sans cela, nous aurions du écrire l'intégralité de nos requêtes AJAX, ce qui se serait révélé beaucoup plus ardu sans l'aide de jQuery.

Dans ce projet, nous ne l'avons utilisé uniquement pour la méthode pour "dialog". Cette méthode permet de créer très facilement une boîte de dialogue entièrement configurable.

```
1         $(function() {
2             $( "#dialog" ).dialog();
3         });
```

Listing 3.5 – Afficher une boite de dialogue

3.5.3 RequireJS



RequireJS est un gestionnaire de dépendance pour des applications JavaScript "in-browser" comme la nôtre. Basé sur le pattern AMD (Asynchronous Module Definition), il permet de charger les modules nécessaires au fonctionnement d'une application et d'optimiser ainsi les temps de chargement d'un site Web.

Devant la recrudescence des applications Web JavaScript, les développeurs ont cherché un moyen de pouvoir découper leur application en module. Plus un site grandit, plus sa complexité augmente. L'assemblage devient donc de plus en plus difficile. Tout ça, parce que nativement, le JavaScript ne possède pas de gestionnaire de dépendance, l'équivalent d'une méthode "#include", "import" ou "require". Développer une application JavaScript se faisait dans un unique scope, où les variables devenaient quasiment toutes de portées globales.

```
1        <script src="ma_classe1.js"></script>
2        <script src="ma_classe2.js"></script>
3        ...
4        <script src="ma_classe_n.js"></script>
```

Listing 3.6 – Inclusion de script dans un fichier HTML

Gérer une application Web de cette manière devient coûteux au fur et à mesure que l'application évolue. RequireJS apporte une solution à tous ces problèmes. Il permet d'offrir un simple point d'entrée de l'application JavaScript, sur le HTML :

```
1        <script data-main="js/main" src="js/lib/require.js"></script>
```

Ensuite, toute dépendance est gérée par RequireJS au travers de deux méthodes : "require" et "define". Ainsi, le code est encapsulé dans des modules et restreint la portée des variables à leur module.

On définit un module ainsi :

```
1        define(["module"], function(module) {
2            // declarer les methodes et classes du module
3        });
```

Define prend en paramètre les dépendances du module. Ainsi, un module peut être découpé en plusieurs fichiers pour simplifier la maintenance de l'application, mais aussi dépendre d'autres modules :

```
1        define(["module", "d3"], function(module, d3) {
2            // declarer les methodes et classes du module, en utilisant d3
3        });
```

La méthode require de son côté permet de charger un module de manière asynchrone et d'appeler une fonction en callback. Ainsi, on peut requérir un module et exécuter une méthode de celui-ci, une fois qu'il est chargé. VizAssist utilise ce système pour charger l'application au fur et à mesure de la navigation de l'utilisateur.

```
1        require(["module"], function(module) {
2            // effectuer des traitements avec modules
3        });
```

Enfin, RequireJS permet de suffixer les fichiers sources. A partir de cela, nous pouvons déjouer le système de cache du navigateur, pour le forcer à récupérer la dernière version des sources de l'application. A l'heure actuelle, VizAssist est à sa version 1.0.2, voilà pourquoi ses fichiers sources sont de la forme "script.js ?v1.0.2".

3.5.4 Modernizr

FIGURE 3.20 – Outil de build de Modernizr

Lors de la création d'application Web, nous ne développons pas pour une seule plateforme. En effet, il existe une multitude de navigateur, et de version de ces navigateurs. Il devient donc important de gérer la compatibilité des navigateurs avec son application, pour prévoir les comportements que celle-ci doit avoir, ou pas.

C'est là que Modernizr entre. Cette librairie permet d'écrire du JavaScript et CSS conditionnel, en fonction du navigateur du client. On test la compatibilité du navigateur avec telle technologie, et on voit en fonction ce qu'on fait. Cette librairie est très utile pour gérer les polyfills : des méthodes JavaScript pour reproduire du code non natif au navigateur.

Pour VizAssist, cela se traduit par la compatibilité du navigateur à accepter les SVG et la File API. Nous n'avons pas eu à gérer de polyfill parce que le temps nous aurait manqués. On a donc abandonné l'idée de supporter les navigateurs plus anciens que Internet Explorer 10. C'est à dire, ceux qui ne supportent pas la File API.

Dans ce cas, nous avertissons l'utilisateur de la non-compatibilité du navigateur et l'invitons à installer un navigateur plus récent. Autre exemple de JavaScript conditionnel, est le fallback mis en place si l'utilisateur se connecte sur le site, et que son navigateur ne supporte pas le SVG. Le schéma d'accueil est alors chargé au format PNG, et non SVG comme il le serait normalement. Ainsi, on peut toujours présenter l'application à l'utilisateur et le tenter de mettre à jour son navigateur.

Modernizr propose un builder pour générer une version de la librairie n'utilisant que les fonctionnalités à tester, ce qui optimise la taille de la librairie aux seules méthodes nécessaires.

Modernizr s'utilise de la façon suivante, pour tester le support d'une fonctionnalité :

```
1        if (Modernizr.svg) {
2            // faire quelque chose avec les SVG
3        }
4        else {
5            // utiliser un polyfill
6            // ou simplement ne pas inclure le code avec les SVG
7        }
```

Couplé avec RequireJS, on pourrait ainsi charger telle version d'un module en fonction de ce qui est accepté par le client. On arriverait à un site le plus compatible et optimisé pour la plupart des navigateurs.

Notre application se devait de fonctionner avec les SVG, donc nous devions nous assurer le support de celui-ci sur le navigateur du client.

3.6 Performances

3.6.1 Charge supportée

Les différents navigateurs Web supportent plus ou moins l'application, mais surtout l'utilisation du SVG. Des ralentissements peuvent se faire sentir sur nos ordinateurs portables offerts par la Région Centre sur Chrome lors de l'utilisation de graphe se déplaçant, mais pour autant n'a aucun problème sur Internet Explorer 11. La charge supportée dépend ainsi de la façon dont sont supportées les fonctionnalités de VizAssist sur les différents navigateurs.

Nous avons également testé l'application avec de gros volumes de données, notamment un jeu de 800 attributs de données, chacun comportant 1000 valeurs différentes. Nous avons ainsi pu tester les limites de notre application, mais surtout de D3. La librairie avait du mal à afficher les 800,000 valeurs après leur chargement. Le contrôle des données se faisait pourtant en quelques secondes, malgré cela l'affichage était interminable. C'est pourquoi suite à cette découverte, nous n'affichons arbitrairement qu'un extrait des données si le jeu de donnée dépasse les 300 valeurs différentes et les 15 attributs de données.

Enfin, la charge supportée dépend de l'ordinateur du client. Notre serveur n'effectue qu'une lecture de fichiers CSV et sert les requêtes AJAX. C'est l'avantage d'avoir déporté toute l'application en local, dans le navigateur du client. Notre serveur ne fait que répondre aux requêtes et lire dans les fichiers CSV.

3.6.2 Poids de l'application

Grâce à RequireJS, notre application est chargée petit à petit. La première page qui effectue le chargement des librairies entraîne un chargement de 343KB, selon les données collectées par l'outil de développement de Google Chrome.

Ensuite, l'application se charge petit à petit pour atteindre au maximum 1.1MB en l'utilisant dans tous ses recoins (télécharger tous les jeux de données et les essayer tous).

Les temps de chargement de l'application dépendent donc de la navigation de l'utilisateur et de son utilisation, mais pour se charger par tranche de 100KB en moyenne entre chaque page.

3.6.3 Configuration requise

Afin d'utiliser VizAssist correctement, vous devez posséder un navigateur récent. Les fonctionnalités utilisées par l'application nous empêchent de fonctionner sur des versions inférieurs ou égales à Internet Explorer 9. D'une part, le SVG restreint les navigateurs plus ancien que IE 9, mais surtout l'API File empêche d'être supporté sur IE 9. Pour connaître le support de ces technologies par les navigateurs, nous avons principalement utilisé le site CanIUse [6] qui permet de comparer la compatibilité des navigateurs sur les technologies Web.

Nous ne pouvions cependant pas faire autrement. Nous devons laissé la possibilité à l'utilisateur de charger ses propres données, sinon l'application n'aurait aucun sens. Sans cette API, nous serions obligés d'utiliser d'anciennes méthodes, qui consisteraient à envoyer les données sur le serveur, ce qui potentiellement, pourrait les mettre à risque.

6. http://caniuse.com/

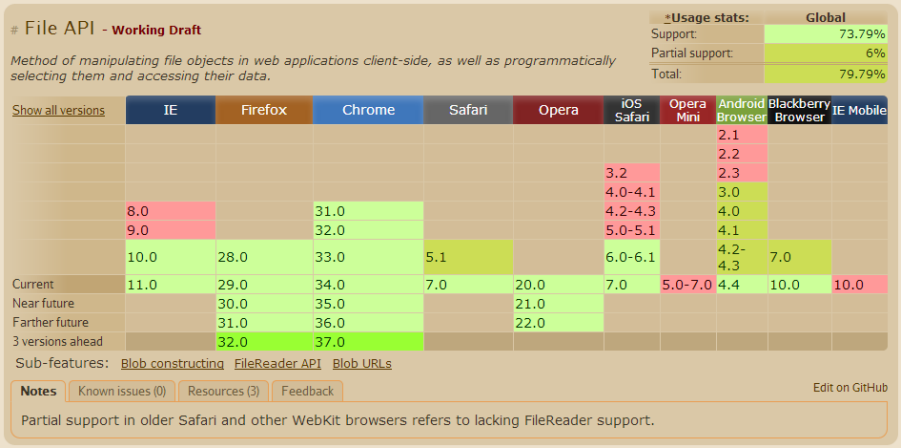


FIGURE 3.21 – Comparaison du support de l'API File par le site CanIUse

Malgré cela, notre application fonctionne aussi sur tablette et smartphone, avec quelques problèmes d'affichages, dûs à la différence des résolutions.

3.7 Problèmes rencontrés

3.7.1 Compatibilité inter-navigateur

On ne peut malheureusement pas parler de développement Web, sans problèmes de compatibilités entre les navigateurs Web. Malgré nos efforts pour prévenir de ces problèmes, plusieurs sont quand mêmes survenus.

Les plus récurrents proviennent de problème d'affichage dues aux règles CSS. Chaque navigateur choisit ou non d'utiliser certaines propriétés, qui accéléreraient la mise en place du design d'un site Web. Créer un site Web, avec un design unique sur tous les navigateurs demande du temps, tant il existe de polyfill, ou de pirouettes en CSS, pour obtenir les résultats voulus. Si bien que nous avons perdu du temps sur ces futilités, pour avoir un résultat pareil aujourd'hui.

La plus grosse erreur de compatibilité qu'on ait eu, est provenu de Firefox, qui était le seul à ne pas supporter nos déclarations d'évènement et d'ajout de listeners. Ce navigateur a une gestion différente de l'objet global "event", si on tente de faire passer des paramètres à notre listener. Lors de la découverte de cette incompatibilité, nous avons dû revoir notre implémentation au travers d'une nouvelle méthode et donc de redéfinir les listeners qu'on avait mis en place jusque là.

3.7.2 Maîtrise partielle des technologies

Avant ce projet, nous avions tous deux qu'une légère approche du JavaScript par la manipulation du DOM. Nous avons découvert par la suite tout la puissance du langage et des nombreuses technologies qui gravitent autour. Cependant, cela demande du temps de se former à l'approche de ce langage.

Par exemple, le JavaScript possède certaines mécaniques qui le distinguent d'autres langages. Le langage passe implicitement les paramètres par valeur si la variable ne fait pas référence à un objet, sinon la variable contient la référence à l'objet. Le mot-clé "this" fait référence au contexte dans lequel on se trouve. Il est très rapidement modifié et peut en quelques lignes de code passer de la référence d'un objet, à celui de la variable global "window", à un élément du DOM. On peut très vite arriver à des résultats étranges si on ne maîtrise pas ces mécaniques, et ce n'est qu'une infime partie de ce que nous avons découvert dans ce langage.

Le langage est beaucoup critiqué pour son côté trop permissif et trop peu typé. Il obéit cependant à de nombreuses règles qui permettent d'être ce qu'il est aujourd'hui, et nous l'avons rapidement remarqué.

Implémentation d'une AGI

Par Rémy Pradignac

La partie personnelle de mon Projet de Fin d'Etude consistait à mettre en place un Algorithme Génétique Interactif (AGI), afin d'affiner les visualisations, pour obtenir le rendu attendu par l'utilisateur.

Bien que cette partie du projet ne dura que deux mois, elle ne se limita pas uniquement au développement d'une AGI.

J'ai ainsi pu :

— Rajouter des visualisations
— Faire de la correction de bug, remonté par les utilisateurs et/ou notre encadrant
— Mettre en place un système de navigation plus simple et moins restrictif pour l'utilisateur
— Développer une AGI

4.1 Algorithme Génétique Interactif

Dans l'éventualité où l'utilisateur souhaite explorer de nouvelle solution, ce dernier devait changer les importances de son fichier de données afin de mettre une priorité plus importante sur de nouveaux attributs de données.

Cette méthode bien qu'efficace aurait été fastidieuse, et très peu pratique. En effet, si le fichier de données comprends une centaine d'attributs, cela faisait modifier de manière trop importante le fichier de données.

C'est dans ce cadre que l'algorithme génétique intervient, afin de permettre une automatisation de ce changement d'importance, tout en suivant les préférences de l'utilisateur.

Je vais tout d'abord vous présenter les algorithmes génétiques, puis vous expliquer pourquoi celui est interactif, ainsi que ces différences avec un algorithme génétique basique, et je terminerai par une explication dans l'implémentation de cette méthode dans VizAssist.

4.1.1 Algorithme Génétique

Un algorithme génétique est une métaheuristique que l'on peut découper en 5 parties, à savoir :

— Création de la première population,
— Évaluation des individus de cette population,
— Sélection des meilleurs individus (suivant différents critères),
— Croisement et mutation des individus sélectionnés,
— On recommence jusqu'à ce que les résultats soient ceux escomptés.

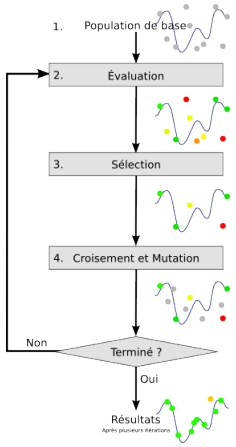


FIGURE 4.1 – Illustration d'un algorithme génétique

On peut s'apercevoir que cet algorithme est donc basé sur le principe de l'évolution de Darwin, d'où son affectation à la catégorie des algorithmes évolutionistes.

4.1.2 Interactif

Notre algorithme est donc un algorithme génétique. Mais pourquoi avoir dit "Interactif" ?

Dans notre cadre, la sélection des différents critères se fait par l'utilisateur. Il n'y a pas de meilleur ou de moins bon individu avec une méthode qualitative, juste l'appréciation de la personne derrière son ordinateur pour définir si l'individu est suffisamment performant pour survivre et engendrer des enfants ou non. C'est donc de là que vient le terme interactif. L'utilisateur peut sauver un individu de manière complètement arbitraire le temps d'une génération.

4.1.3 Algorithme dans VizAssist

Nous allons maintenant voir en détail le fonctionnement de l'algorithme génétique interactif à travers VizAssist.

4.1.3.1 Création de la première population

Tout d'abord la visualisation choisie pour l'algorithme génétique est celle qui était présente dans l'agrandissement de la partie précédente. Dans le cas où l'utilisateur décide de l'appliquer sur une autre visualisation, il devra retourner sur l'écran d'avant, afin de relancer l'AGI.

Sur le panel de 9 individus, lors de la première génération, le premier individu est l'original et ne sera pas modifié.

Les 8 autres individus sont générés à partir d'une mutation de l'individu originel.

Grâce à cette méthode, on arrive a avoir dès la première population une diversité de population conséquente.

4.1.3.2 Croisement

Il existe plusieurs scénarios concernant le croisement.

Dans un premier temps, nous partons du principe que l'utilisateur a sélectionné deux parents ou plus. On ajoute un coefficient de croisement, défini aléatoirement entre 0 et 1. Si ce coefficient est supérieur ou égal à 0.8, l'individu sera le croisement de deux des parents aléatoirement choisis dans la liste de sélection de l'utilisateur.

Il existe deux types de croisement :

— Croisement linéaire ,

qui consiste à faire la moyenne des importances des deux parents pour créer celle du nouvel individu (i.e $\frac{40+60}{2} = 50$).

— Croisement uniforme ,

qui consiste à prendre aléatoirement l'importance d'un parent ou de l'autre et de créer ainsi un nouvel individu.

Bien entendu, ces deux croisements ne sont possibles qu'à partir du moment où la liste de parents est supérieure ou égale à 2.

Si la liste de parents est nulle, on n'effectue aucune méthode de croisement.

Dans la cas où liste ne contient qu'un élément, on sélectionne ce parent pour récupérer ses importances et créer les 8 autres individus à partir de ces derniers.

4.1.3.3 Mutation

Après le croisement, il faut effectuer la mutation. Pour ce faire, il faut récupérer la diversité sélectionnée par l'utilisateur. En effet, cette dernière influera sur la "puissance" de la mutation.

La mutation possède aussi un caractère aléatoire. Cette probabilité vient entre autre de l'index de l'individu dans la liste. En effet, l'individu 0 aura moins de chance de muter que l'individu 8.

Voici la formule sur la probabilité de mutation :

probMutInf $= \frac{diversity}{200}$

probMutSup $= \frac{diversity}{200} + 0.2$

probMut $= ((1 - \frac{indexIndividu}{8}) * probMutInf) + (\frac{indexMutant}{8} * probMutSup)$

Une fois cette probabilité calculé, on peut effectuer la mutation de cette manière :

importance $= importance + ((random - 0.5) * 2 * \frac{diversity}{5})$

Pour terminer, on vérifie que les nouvelles importances ne sortent pas de l'intervalle [0-100]. Dans le cas contraire, on borne le résultat.

4.1.3.4 Nouvelle Génération

Afin d'effectuer une nouvelle génération, il faut donc récupérer les importances de toutes les visualisations sélectionnées.

En effet, si aucune visualisation n'a été sélectionné, on recommence la procédure à son début en effectuant un reset de l'AGI.

Si qu'une seule visualisation a été sélectionné, le croisement ne s'effectuera pas, et nous passerons directement à la mutation dans le cas où il n'y a qu'un parent, décrite ci-dessus.

Dans les autres cas, on effectue un croisement, puis une mutation, avant d'afficher les résultats.

4.1.4 Interface Homme-Machine

Une autre complexité de cette partie fut toute l'interface Homme-Machine.

En effet, plusieurs choix s'offraient à moi, à savoir :

— Un carrousel, comme pour les visualisations
— Un panel de choix
— Un contour autour d'une visualisation principal
— ...

Dans tous les cas, il fallait, pour rester dans la continuité de l'application l'intégralité des informations sans avoir besoin de "scroller". En effet, cela n'aurait pas était correct, au vu du reste de notre application, de ne pas tout laisser sur la même page.

C'est pour cela, que j'ai choisi, avec validation de Monsieur Venturini, le panel à droite, avec la visualisation principal à gauche.

La légende se situe en bas de la visualisation principale, ainsi que les détails de valeur d'importance (si l'utilisateur désire les voir).

Pour finir, l'intégralité du panneau de contrôle se situe au dessus de la page, afin que l'on comprenne d'un simple coup d'oeil que cette partie, permet de contrôler l'AGI.

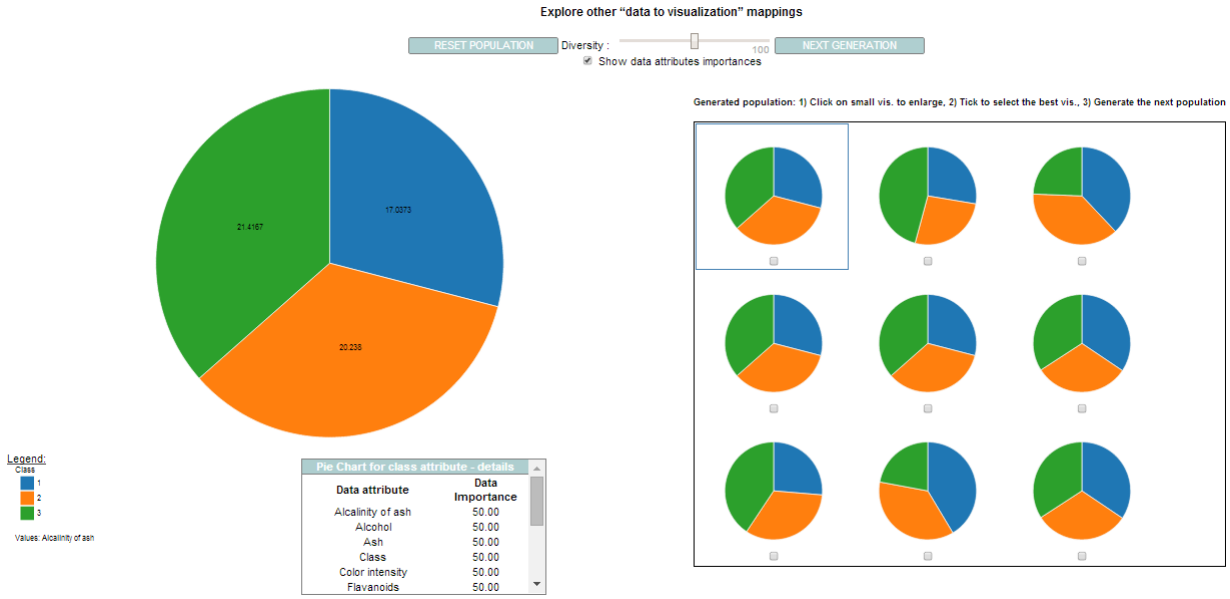


FIGURE 4.2 – Interface Homme Machine de L'AGI

4.2 Ajout de visualisations basiques

Lors du développement de l'algorithme génétique interactif, j'ai eu certain "temps morts", ce qui m'a permis de développer quelques visualisations supplémentaires afin de fournir l'application, pour sa sortie en production, avec un maximum de visualisation disponible.

Cependant, je n'ai pas intégré de visualisation complexe telle qu'un "force layout graphe" comme Jean. Je suis resté sur des visualisations plus basique telle que les Scatter Matrix, ou encore le Scatter color shape.

Cette dernière c'est d'ailleurs révélé, légèrement plus complexe que les autres. En effet, lorsque l'on regarde la librairie d3js, il existe des méthodes déjà enregistrées pour les couleurs :
— category10()

— category20a()
— category20b()
— category20c()
C'est grâce à ces méthodes de couleurs qu'il nous a été simple de colorer les différents points.

Cependant, pour les formes, cela fut compliqué car il n'existait pas de méthode de mapping aussi simple dans la librairie. Après plusieurs recherches dans la documentation de d3js, je me suis aperçu qu'il existait une liste de symbole :
— d3.svg.symbol
Ce tableau contient les différentes formes déjà créées. Avec l'attribut .types, on obtient les index des différentes formes qui sont :
— circle
— cross
— diamond
— square
— triangle-down
— triangle-up
Pour finir, il m'a fallu mapper moi même par un système de boucle pour affecter une forme à un attribut de donnée, plutôt que par un simple mapping afin de ressortir un résultat comme celui ci :

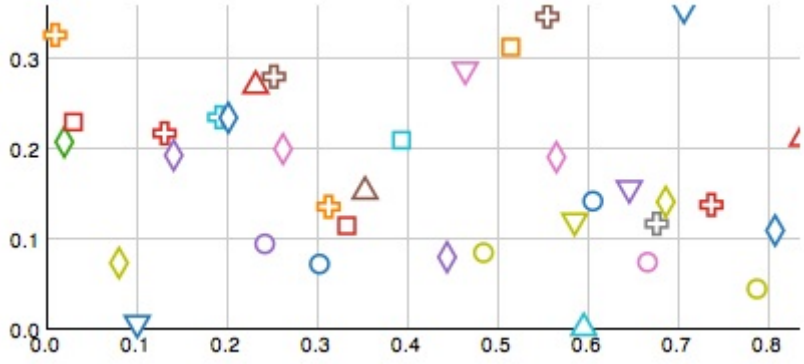


FIGURE 4.3 – Exemple d'un graphique "Scatter Color Shape"

4.3 Correction de bug

Bien que nous le savions déjà, durant le développement de l'AGI, nous nous sommes rendu compte de l'implication du passage par référence des objets en JavaScript.

En effet, durant le développement de ce sprint, j'ai été obligé de dupliquer l'objet Data, afin d'en avoir neuf indépendant. J'avais donc tout simplement créé un nouvel objet en y mettant l'objet data à l'intérieur. Le problème est qu'en faisant cela, l'objet sera passé par référence et non par valeur.

4.3.1 Passage par valeur, passage par référence

Pour mémoire, un argument passé par référence est utilisé directement, la fonction modifie donc l'original, contrairement à un argument passé par valeur. Cette dernière est copié, n'entraînant donc aucune modification sur la valeur original.
Cependant en JavaScript, on ne peut pas choisir le mode de passage des valeurs. On peut faire l'analogie avec le Java : si c'est un type natif (Number, String, etc.) le passage s'effectue par valeur, tandis que

si c'est un objet de classe personnel,le passage se fait par référence.

C'est dans ce cadre que la création d'une fonction de clone s'est avéré nécessaire :

```
1        if (obj instanceof Array) {
2            copy = [];
3            for (var i = 0, len = obj.length; i < len; i++) {
4                copy[i] = clone(obj[i]);
5            }
6            return copy;
7        }
8
9        if (obj instanceof Object) {
10           copy = {};
11           for (var attr in obj) {
12               if (obj.hasOwnProperty(attr)) copy[attr] = clone(obj[attr]);
13           }
14           return copy;
15       }
```

Cela m'a donc permis de pouvoir travailler sur des copies de data, et non plus sur les originaux.

4.4 Système de navigation

Au fur et à mesure de notre développement de l'application, nous nous sommes aperçus que la navigation de type Wizard pouvait être fastidieuse, pour changer le jeu de données par exemple. C'est dans ce cadre que nous avons eut l'idée d'incorporer un système de navigation de style "Fil d'Arianne".

Le design a été choisi comme pour le reste de l'application, avec des teintes de bleu, tout en gardant un maximum de sobriété, pour ne pas perdre l'utilisateur du but principal de l'application. De plus, pour les parties non-visitées nous avons décidé de garder une couleur grise, couleur universelle en informatique du lien non sélectionnable.

Des tooltips sont présents au survol de chaque lien, afin de bien rappeler vers où l'utilisateur se dirige en cliquant dessus.

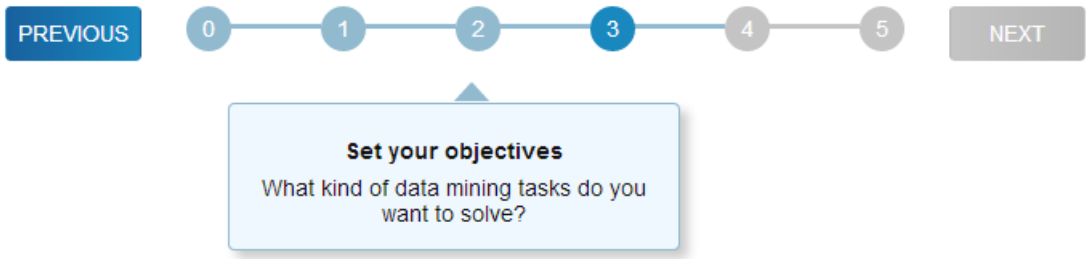


FIGURE 4.4 – Système de navigation

La difficulté de cette partie fut de réfléchir à l'intégralité des cas possibles.

En effet, nous avions décidé que si l'utilisateur effectuait un retour en arrière, mais en ne modifiant rien, l'intégralité de ses choix restaient conservés dans l'application.

A contrario, si l'utilisateur effectuait un retour, afin de modifier quoi que ce soit, l'application se devait d'être réinitialisé, à partir de la modification.

Une série de test utilisateur a été effectué sur cette navigation par "breadcrumb". On s'est aperçu que les personnes l'utilisaient instinctivement. Et lorsque nous posions la question directement sur leur avis de cette fonctionnalité, toutes se sont révélées positives.

Maintient en production de l'application

Par Jean de Barochez

Le sujet de mon projet de fin d'étude a, pour ma part, évolué au cours du développement du projet. A la base, je ne devais que m'occuper d'un système de retour utilisateur à partir d'un algorithme de feedback. Cependant, l'application a eu d'autres besoins entre temps, et monsieur Venturini a préféré abandonner cette partie pour se concentrer sur d'autres tâches.

Après s'être séparés Rémy et moi à la mi-mars, mon travail a été très divers. Il ne restait certes que 2 mois de projet, mais avec le recul, ce fut riche.

J'ai ainsi jonglé entre plusieurs postes :

— Déployer l'application en production
— Mise en place d'un wiki avec DokuWiki
— Faciliter la reprise de l'application au travers d'une documentation développeur
— Ajouter de nouvelles visualisations plus complexes de D3.js
— Ajouter de nouvelles fonctionnalités

5.1 Déploiements en production

La première mise en production de l'application a été effectuée le 18 mars à l'adresse http://www.vizassist.fr. L'application ne contenait pas encore l'AGI, nous venions tout juste de séparer nos projets Rémy et moi. Cependant les premières visualisations étaient disponibles. On pouvait ainsi commencer les tests utilisateurs publics. Jusqu'à présent, les tests étaient effectués sur l'environnement de développement par une dizaine de personne, dont Monsieur Venturini, Rémy et moi. A partir de là, nous avons pu présenter VizAssist à un premier cercle de proches.

Monsieur Venturini voulait un hébergement chez OVH, donc nous avons étudié les différentes offres et l'avons conseillé sur une offre de serveur mutualisé, qui offre pour un budget correct, un espace largement suffisant pour héberger notre application. Notre serveur ne fait presque qu'héberger des fichiers statiques à l'heure actuelle, hormis les requêtes sur la base des fichiers CSV.

On pouvait donc, sans crainte, utiliser un serveur mutualisé, en assurant ainsi un temps de traitement raisonnable pour le service de VizAssist.

5.1.1 Assurer les déploiements

Je n'avais jusqu'à présent aucune expérience en déploiement d'application. Nous avions jusqu'à présent qu'un environnement de développement et nos sources étaient hébergées sur le SVN de Polytech Tours.

Par manque de temps, les premiers déploiements ont été effectué par FTP, en remplaçant les anciennes sources par les nouvelles et en maintenant une archive comprenant les sources à disposition pour tout nouveau collaborateur. SVN a une gestion peu fiable des branches parallèles. A deux, nous pouvions nous arranger Rémy et moi pour éviter les conflits. Intégrer de nouveaux développeurs pouvaient se révéler difficile tant la fusion dans les codes aurait pu être fastidieux.

C'est pourquoi nous avons laissé une archive à disposition en attendant de mettre en place une structure plus souple (pour accepter de nouveaux contributeurs), et plus solide pour assurer les déploiements de l'application.

C'est pourquoi nous avons choisi Git. Il a l'avantage d'offrir tout d'abord des dépôts locaux où chacun peut travailler sans perturber les autres développeurs, à l'instar de SVN où chaque commit s'effectue sur

un unique dépôt pour tout le monde. Dans un second temps, pour un projet d'une telle dimension, nous pouvons nous permettre d'utiliser juste deux différentes branches pour créer deux environnements distincts pour le développement et la production. Enfin, la migration et l'hébergement de nos sources sur le serveur Ovh permet de sauvegarder l'historique du projet pour l'an prochain si le projet est repris.

5.1.1.1 Ajout de DokuWiki

A la demande de M. Venturini, je me suis occupé de chercher à mettre en place un Wiki pour VizAssist. Celui-ci permettrait à M. Venturini de pouvoir documenter les mécaniques de l'application et ainsi d'aider l'utilisateur à pouvoir rapidement comprendre en profondeur comment fonctionne notre assistant.

Le wiki recherché devait être simple d'utilisation, simple d'installation et être compatible avec notre serveur Apache. En effet, notre serveur fonctionne sous PHP 5.4 mais sans base de donnée SQL, ce qui élimine plus de 80% des CMS ("Content Management Systems") du marché. Enfin, il fallait que ce Wiki puisse être accessible par nos utilisateurs, et donc être supporté sur Internet Explorer 10.

DokuWiki est un CMS Wiki Open Source, qui contrairement à ses concurrents utilisent un système de "flat file database" pour gérer ses pages. Chaque fichier est ainsi rédigé au format Markdown [1] et sauvegardé au format .txt (d'où le "flat file"). Il ne demande ainsi que très peu de ressources serveurs. Il est prévu pour les plateformes PHP de version 5.2 ou supérieur et supporte les navigateurs jusqu'à Internet Explorer 7.

Après l'avoir présenté à M. Venturini dans notre environnement de développement, je l'ai ajouté en ligne. DokuWiki se présente sous la forme d'une archive, comprenant une page PHP d'installation et de configuration. Une fois extrait, en suivant les étapes conseillés par la page d'installation, on met en place notre Wiki. Une fois effectué, on supprime la page d'installation, pour éviter que d'autres puissent modifier la configuration.

5.1.1.2 Migrer SVN à Git

Tout d'abord, pour effectuer cette migration, j'ai importé le dépôt SVN vers un nouveau dépôt Git. Pour garder nos identités à Rémy et moi, j'ai fait correspondre nos numéros étudiants à nos noms respectifs pour que Git puisse se retrouver entre les différents messages de révisions. Pour cela, j'ai créé un fichier users.txt dans lequel j'ai ajouté la liste des contributeurs du dépôt SVN, qui sont renseignés par les numéros étudiants :

```
num_etu_remy = Remy Pradignac <remy.pradignac@gmail.com>
num_etu_jean = Jean de Barochez <jean.barochez@gmail.com>
```

On indique ainsi à Git quels utilisateurs correspondaient à quels identifiants sur SVN. On lance ensuite la migration de SVN vers Git :

```
  git svn clone --username num_etu http://redmine.polytech.univ-tours.fr/svn/
      assistweb \
  --authors-file=users.txt my_project
```

Listing 5.1 – Migration de SVN vers Git

Sur le serveur, j'ai ensuite créé le dépôt central :

```
mkdir vizassist.git      # creation du dossier
cd vizassist.git
```

1. http://fr.wikipedia.org/wiki/Markdown

```
git --bare init          # creation du depot Git central
```

Listing 5.2 – Création du dépôt central

Le paramètre bare définit le type de dépôt créé, ici on aura un dépôt central. Chaque push effectué par un contributeur sera effectué sur ce dépôt. Ce dépôt permet ainsi de garder une sauvegarde de toutes les manipulations effectuées par les collaborateurs. Un dépôt Git central a le même rôle qu'un dépôt SVN. Dans un système décentralisé comme Git, ce dépôt permet de sauvegarder un historique cohérent et ainsi de pouvoir laisser à nos successeurs l'historique de notre travail, sans que Rémy ou moi, n'ayons à faire de manipulation pour transmettre nos sources. Revenons à l'initialisation de notre dépôt sur notre machine où sont situées les sources en développement. Ajoutons un dépôt distant et envoyons les données à ce dépôt :

```
  # xxx est votre identifiant Ovh
  # yyy est obtenu a partir de la commande pwd sur le serveur
  git remote add VizAssist ssh://xxx@ftp.xxx.fr/homez.yyy/xxx/vizassist.git
  git push VizAssist master
```

'VizAssist' ici est le nom de notre dépôt distant ajouté. Pour savoir à tout moment quels sont les dépôts enregistrés au dépôt courant, tapez :

```
git remote -v
```

Notre dépôt central possède les sources, mais nous n'avons toujours pas géré notre environnement production, qui concerne notre site en ligne.

```
cd /www
git init
git add .
git commit -m "Initial import of our web site"
```

On ajoute notre dépôt central à la liste des dépôts distant :

```
git remote add VizAssist ../vizassist.git
```

Cependant, à ce point là, on atteint un conflit. Nous avons d'un côté notre site Web possédant sa branche 'master' et notre dépôt possédant la sienne aussi. Ce qu'on souhaite, c'est avoir d'un côté une branche 'master' où on développe en paix, sans déranger le site, et avoir une branche spéciale pour ce dernier.

On créé donc une branche à partir de 'master' qu'on va appeler 'release' :

```
git checkout -b release master
```

On annonce à notre dépôt distant qu'on vient de créer une nouvelle branche :

```
git push -u VizAssist release
```

On supprime enfin notre branche 'master' pour n'obtenir qu'une seule branche 'release' sur ce dépôt.

```
git branch -d master
```

Voilà, nous avons nos deux branches de créées. Le dépôt Git associé à notre site Web travail sur la branche 'release'. Tapez 'git branch' pour le vérifier, une étoile à côté du nom de la branche indique la branche active.

Vérifions maintenant que sur notre dépôt central, nous ayons bien deux branches :

```
cd ../vizassist.git
git branch
```

Maintenant, n'importe qui possédant le code SSH d'identification au serveur peut obtenir les sources avec cette commande :

```
git clone ssh://xxx@ftp.xxx.fr/homez.yyy/xxx/vizassist.git mon_dossier
```

Ce qui clônera le dépôt git vers **mon_dossier**, dans lequel on retrouvera toutes les sources contenues sur le dépôt, ainsi que l'historique depuis le début de notre projet.

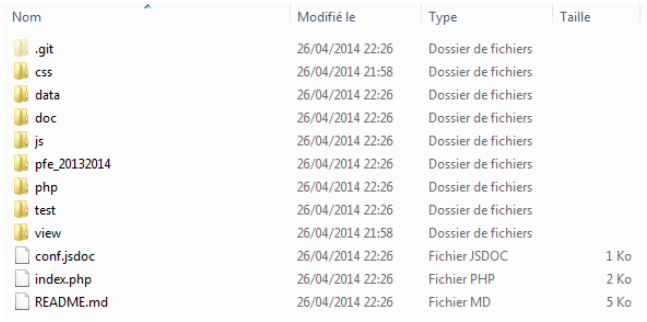


FIGURE 5.1 – Liste des dossiers sur la branche master

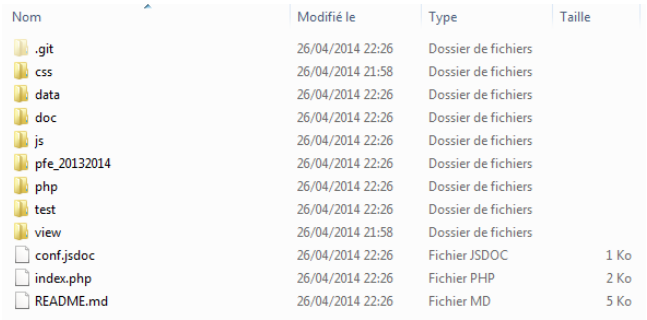


FIGURE 5.2 – Liste des dossiers sur la branche release

5.1.2 Déployer une mise à jour en production

Ceux ayant cloné le dépôt ont accès aux différentes branches. Pour passer d'une branche à une autre, il suffit simplement de taper :

```
git checkout <nom_branche>
```

On peut voir ainsi les deux versions de nos dépôts en modifiant les branches sur lesquels on travail :

Maintenant, pour déployer notre mise à jour, nous créons une nouvelle branche temporaire à partir de 'master' pour faire la transition :

```
git checkout -b new_version master
```

Ainsi, avec cette branche, nous pouvons choisir les fichiers à déployer. Si certains fichiers ne doivent pas être mis en ligne, on peut les supprimer de cette branche sans affecter la branche de développement principale. Cependant, cette méthode peut entraîner des erreurs si un fichier en développement passe entre les mailles sans être fini.

Il faut maintenant fusionner nos branches ensembles pour mettre à jour la branche 'release' :

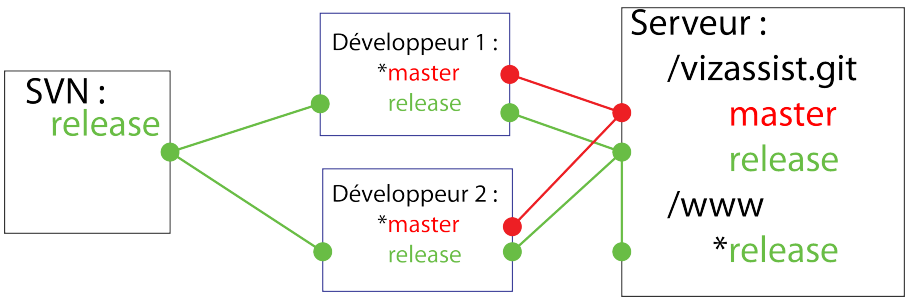


FIGURE 5.3 – Schéma du workflow mis en place

```
git merge release
```

Durant nos déploiements, nous avons essayé d'être les plus vigilants pour mettre à jour l'application. Afin de gérer de tels problèmes, il est conseillé d'utiliser une interface graphique pour Git, qui permet d'avoir un meilleur visuel sur les fichiers modifiés et ajoutés, à chaque branche. Dans ce domaine, le logiciel gratuit Source Tree est très performant.

Bien que les déploiements à l'aide de Git peuvent être effrayants au premier abord, il est beaucoup plus facile de contrôler les erreurs et les fichiers modifiés ainsi, que de passer par un déploiement manuel. Avant de mettre en place ce système, nous avons effectué les mises à jours par FTP qui s'avouent être longues et entraînant souvent un oubli de fichier.

5.1.3 Problèmes rencontrés

La première mise en production a révélé des défauts dans notre environnement de développement. Ce dernier se doit d'être identique à l'environnement de production, ce qui n'était pas le cas : les serveurs Ovh utilisent un système Unix, alors que nous travaillions sous Windows.

Cela a entraîné au premier déploiement une gestion différente de l'indexation des fichiers. Alors que sur Windows, les fichiers sont triés par ordre alphabétiques, dans notre environnement de production, le tri est fait par ordre lexicographique. Cela a ainsi modifié notre organisation des fichiers, notamment pour nos fichiers de démonstration qui sont appelés par index.

Les autres problèmes rencontrés proviennent ensuite d'erreur de manipulation lors des premiers déploiements avant que le Git soit en place et que la mise à jour soit plus contrôlée.

5.2 Faciliter la contribution au projet

Cette mise en production s'est aussi accompagnée du partage de nos sources de l'application avec les contributeurs au projet : nos collègues de l'option Web et Multimédia. Quatre personnes s'ajoutaient au projet et nous devions Rémy et moi les assister dans leur projet pour que l'intégration de leurs travaux se passent au mieux. Agrandir l'équipe de développeur sur le projet demande un effort supplémentaire sur la clarté de son code et de la documentation associée. C'est pourquoi je me suis occupé de générer une documentation à l'aide de JSDoc, ainsi que de ré-écrire certaines méthodes, qui avec le recul, étaient mal rédigées et ne convenaient pas à notre application.

A partir des annotations en en-tête de méthodes, modules ou d'attributs, JSDoc permet de générer une documentation du code, similaire à une JavaDoc ou Doxygen. Cette documentation profitera aussi à nos successeurs.

Pour générer la documentation d'un fichier, on exécute la commande suivante :

```
jsdoc script.js
```

Pour notre projet, après avoir configuré JSDoc, on peut générer toute la documentation de notre application JavaScript comme le montre la figure 5.4.

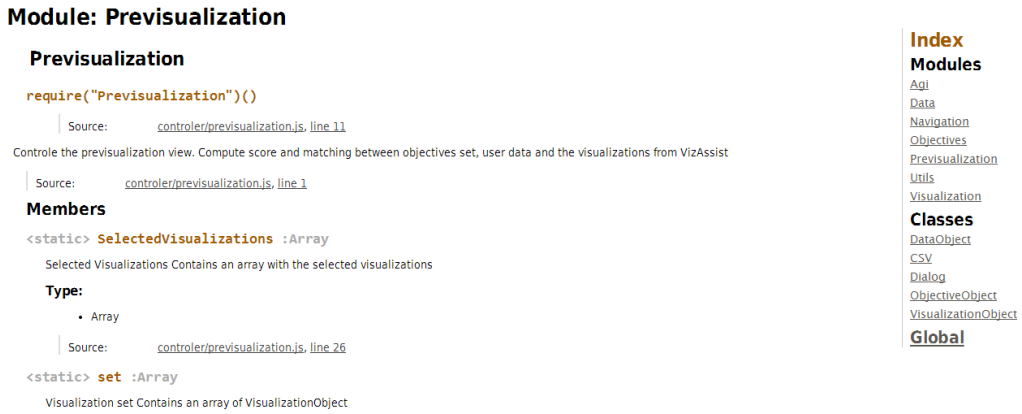


FIGURE 5.4 – Extrait de la documentation développeur générée par JSDoc

5.2.1 Gestion de notes de version

Avec cette documentation, j'ai maintenu un fichier de notes de mises à jour pour décrire ce qui était modifié à chaque nouveau déploiement de l'application. Ces notes sont joins à la documentation et permettent à n'importe qui sur le projet d'être au courant des dernières modifications et ajouts.

Pour rédiger ces notes de version, j'ai repris les derniers commits effectués en développement pour noter quels avaient été les corrections et ajouts effectués.

Comme nous ne travaillions pas sur les mêmes dépôts entre les différents contributeurs, il était important que ceux ci soient au courant si des changements de l'API arrivaient. Notamment, une correction a été effectuée au début de la collaboration et qui aurait pu gêner nos collègues. Elle concernait une méthode pour obtenir un attribut de donnée par son nom et qui retournait l'attribut de donnée par référence et non par valeur, ce qui aurait pu créer des erreurs par la suite. Nos collègues n'étaient pas encore arrivé à l'intégration à VizAssist quand nous avons repéré cette erreur. Mais cela a montré qu'il nous fallait doubler de vigilance pour la suite. L'erreur corrigée, nous avons notifié les autres collaborateurs de notre erreur et indiqué dans les notes de version quelle méthode était plus appropriée.

5.3 Ajout de nouvelles visualisations "complexes"

Mon deuxième rôle après les déploiements et mises en production, fut d'ajouter des visualisations plus complexes que celles mises jusqu'à présent sur VizAssist. On les considère complexes, par le fait qu'elles utilisent une autre approche des visualisations, via les layouts de D3. Pour notre application, elles se révèlent plus complexe car elles demandent une interface différente pour adapter nos données à D3.

Ces layouts permettent ainsi de mettre plus rapidement en place des visualisations, mais demandent que les données à attachées soient sous un certain format.

5.3.1 Force Layout Graphe

Ce layout est celui utilisé pour générer un graphe de noeud. Pouvoir générer un graphe entièrement personnalisable faciliterait le travail de nombreuses personnes à l'école. De nombreux résultats de tests pourraient être ainsi modélisés sous forme de graphe. Très peu d'outils existent actuellement pour pouvoir générer un graphe à partir de données aussi simples que le modèle proposé par VizAssist.

5.3.1.1 Interfaçage des données CSV vers D3

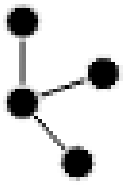
Le layout graphe demande en entrée de spécifier les noeuds sous forme de JSON et possédant un attribut name, et les liens, eux aussi sous forme de JSON, avec deux attributs source et target, contenant

le nom des noeuds au départ et à l'arrivée du lien.

La première problématique a donc été de savoir comment formater nos données CSV pour reconnaître quel attribut de données allait devenir nos noeuds et lequel allait devenir nos liens. D3 proposait un exemple pour répondre à ce problème et proposer une solution partielle :

```
source,target
A1,A2
A2,A3
A2,A4
```

Aurait pour rendu, le graphe de la figure suivante :



En parsant ce CSV et en reconnaissant les différents noeuds cités, on peut créer un premier graphe. Cependant, ce modèle ne permet pas d'associer d'autres attributs de données pour décrire les noeuds ou les arêtes de notre visualisation.

C'est pourquoi nous sommes partis sur ce format qui permet de distinguer quels sont les liens et les noeuds, depuis le fichier CSV et ainsi de pouvoir ajouter de nouveaux attributs de données pour dimensionner chaque noeud et lien :

```
source,target,node size,edge thickness
A1,A2,-,5
A2,A3,-,10
A2,A4,-,15
A1,A1,10,-
A2,A2,20,-
A3,A3,10,-
A4,A4,10,-
```

Lorsque la valeur de l'attribut de donnée "source" est égal à la valeur de l'attribut de donnée "target", alors on décrit un noeud, sinon un lien.

On a donc introduit de nouveaux types d'attribut de données à VizAssist pour correspondre aux graphes : source, target, nodenumeric, nodenominal, edgenumeric et edgenominal. Ces quatre derniers types permettent de spécifier directement un attribut de donnée nominal ou numérique pour nos noeuds et nos liens.

C'est ainsi qu'il est possible de visualiser les différents participants au projet de la figure 5.3.1.1. Celle ci a été obtenue depuis l'application, après l'algorithme génétique interactif pour obtenir la visualisation qui me semblait la plus intéressante à vous proposer sur les relations entre les protagonistes du projet VizAssist.

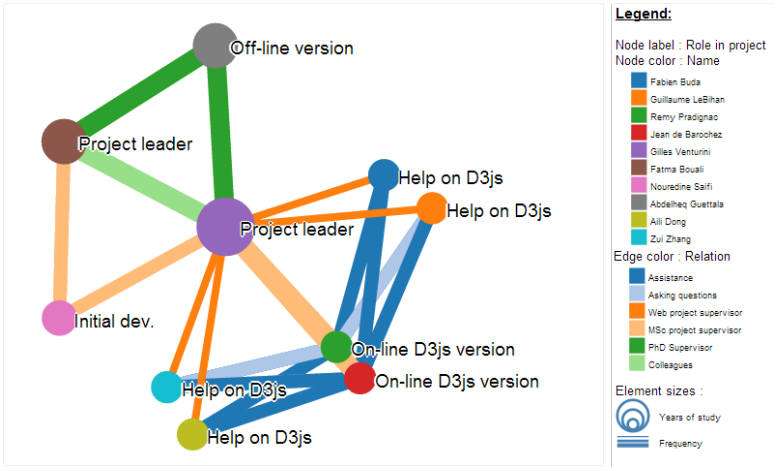


FIGURE 5.5 – Relation entre les différents participants au projet VizAssist, depuis 7 ans

5.3.1.2 Problèmes rencontrés

Dessiner cette visualisation a demandé beaucoup de temps. Comparé aux autres visualisations, celle-ci possède de nombreux attributs visuels et on a voulu être générique sur celle ci. Plus tôt, on vous expliquait pourquoi on avait créé plusieurs types de nuages de point, pour cibler plus facilement ce qu'on allait dessiner.

Cette visualisation a été déclinée en deux versions seulement : les graphes dirigés et non dirigés. Du coup, si un attribut visuel n'est pas matché à un attribut de donnée, il ne faut pas l'afficher, ce qui rend la visualisation complexe avec de nombreuses conditions. En effet, s'il manque un attribut visuel, il ne faut pas l'afficher d'une partie sur le graphe, mais aussi dans la légende. Ce problème ne se rencontre pas sur les autres visualisations, car les attributs visuels ne sont que des copies d'un autre du même type. Par exemple le nombre de séries temporelles parallèles à afficher peut varier, mais on en affichera toujours au moins une.

L'autre soucis rencontré concerne les dimensions de la visualisation. Quelque soit les valeurs, quelque soit le nombre de noeuds, on doit pouvoir visualiser un graphe. Du coup, chaque mesure doit prendre en paramètre le nombre de noeud, la largeur de la zone de dessin, ce qui devient lourd à gérer.

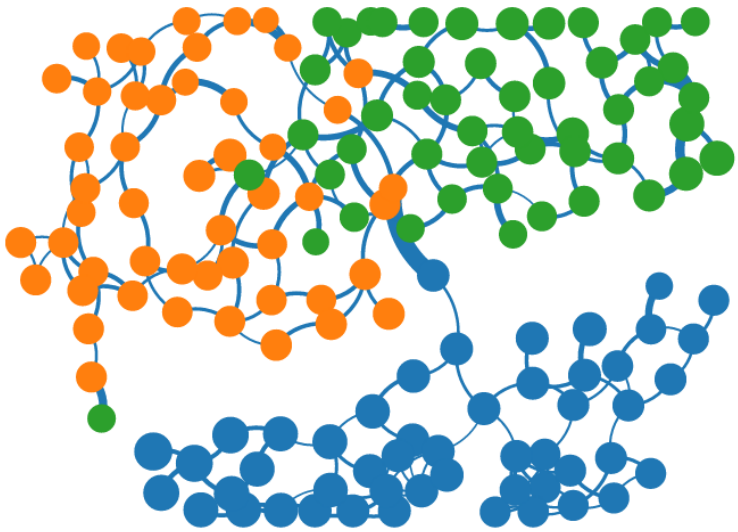


FIGURE 5.6 – Un graphe avec 150 noeuds devient difficilement interprétable

5.3.2 Visualisations géographiques

D3 propose aussi de générer des carte au format SVG et ainsi de pouvoir rapidement visualiser des données sur celles ci. Mon rôle sur ces visualisations a été d'étudier s'il était possible ou non d'intégrer de telles visualisations à VizAssist.

5.3.2.1 Introduction au format GeoJSON et TopoJSON

La spécification GeoJSON [2] est un format de donnée pour encoder une variété de structures de données géographiques, basée sur le format JSON.

```
{
  "type": "Feature",
  "geometry": {
    "type": "Point",
    "coordinates": [125.6, 10.1]
  },
  "properties": {
    "name": "Dinagat Islands"
  }
}
```

Listing 5.3 – Exemple de GeoJSON

Cette structure permet de décrire plusieurs types de formes géographiques, de la ligne au polygone. Chaque coordonnée $[x, y]$ représente la longitude et latitude d'un point.

De nombreuses applications Web existent pour obtenir des données géographiques de n'importe quel point de la planète depuis une carte [3] ou depuis une liste de nom de lieux publics [4]. Certains, comme Natural Earth [5], met directement les cartes à disposition, en haute résolution.

TopoJSON est l'adaptation du GeoJSON par D3, avec l'introduction de la topologie des terrains. GeoJSON souffre malheureusement de duplication de données : une frontière commune entre deux pays est inscrite deux fois dans l'objet, chacune appartenant à la géométrie du pays. TopoJSON permet ainsi d'effectuer des manipulations sur les géométries, en simplifiant par exemple la géométrie d'un pays, ainsi réduire la précision d'une carte mais clarifier les frontières.

5.3.2.2 Dessiner un planisphère

Une fois les coordonnées géographiques obtenues, D3 permet d'utiliser différentes projections géospatiales comme celle de Kavrayskiy [6] ou la projection de Mercator [7] pour dessiner notre carte.

Chaque pays est identifié selon son code ISO (FRA ou FR pour la France) qui permet de lui associer ensuite les valeurs que nous souhaitons.

La difficulté au final dans l'intégration d'une visualisation comme celle-ci repose sur les données à trouver pour utiliser ces visualisations. Encore une fois, des sites comme celui de la CIA [8] proposent diverses données sur les pays du monde pour promouvoir l'Open Data.

Enfin, les pays du monde peuvent être tellement variés, qu'il peut être difficile de reproduire correctement une visualisation de notre planète. En effet, si on reprend la figure 5.7, on a utilisé une échelle logarithmique pour représenter les données numériques, car la Chine premier pays du monde en terme de population, possède un milliard d'habitant de plus que les Etats-Unis, troisièmes du classement. Une échelle linéaire n'aurait pas pu montrer de telles différences entre les pays.

2. http://geojson.org/geojson-spec.html
3. http://teczno.com/squares
4. http://www.gpsvisualizer.com/geocoder/
5. http://www.naturalearthdata.com/
6. http://en.wikipedia.org/wiki/Kavrayskiy_VII_projection
7. http://en.wikipedia.org/wiki/Mercator_projection
8. cia.gov/library/publications/the-world-factbook/

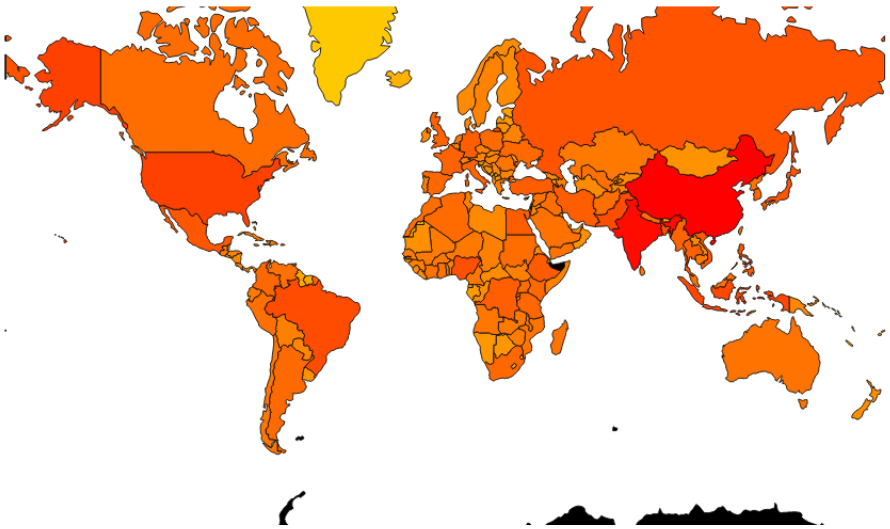


FIGURE 5.7 – Les populations par pays, selon une échelle logarithmique, effectué par VizAssist

Pour conclure cette étude, il est effectivement possible d'intégrer des visualisations géographiques à VizAssist. Cependant, cela demande du temps pour découvrir tous les aspects que cela implique pour comprendre le processus, puis pour trouver les cartes et enfin trouver les données compatibles.

Heureusement, de nombreux outils sont disponibles pour promouvoir et rendre accessibles ces visualisations. Dans les retours utilisateurs que nous avons a reçus, un professeur de l'université de Paris 1 a demandé s'il était possible d'utiliser des coordonnées latitudes et longitudes pour dessiner eux-mêmes leur carte. Comme nous l'avons vu pour le format GeoJSON, nous pouvons très bien convertir ces données au bon format pour laisser l'utilisateur dessiner sa propre visualisation, mais cela va rester très spécifique.

5.4 Ajout de fonctionnalité

Ma dernière tâche a été de trouver une solution pour que l'utilisateur puisse obtenir les visualisations qu'il souhaite. Toujours en gardant à l'esprit qu'on ne peut pas passer par une activité du serveur, il faut que l'utilisateur puisse télécharger les visualisations de son choix.

SVG Crowbar, développé par Mike Bostock, auteur de D3, est une première idée. Il s'agit d'un bookmarklet : un petit programme JavaScript pouvant être stocké dans un signet (ou marque page) d'un navigateur. En un clic, cet outil permet de télécharger les SVG affichés sur une page Web. Ce bookmark est cependant compatible qu'avec Chrome.

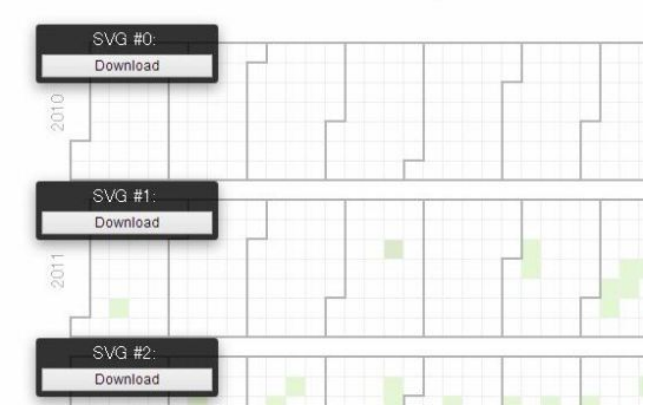


FIGURE 5.8 – SVG Crowbar permet de télécharger les SVG d'une page

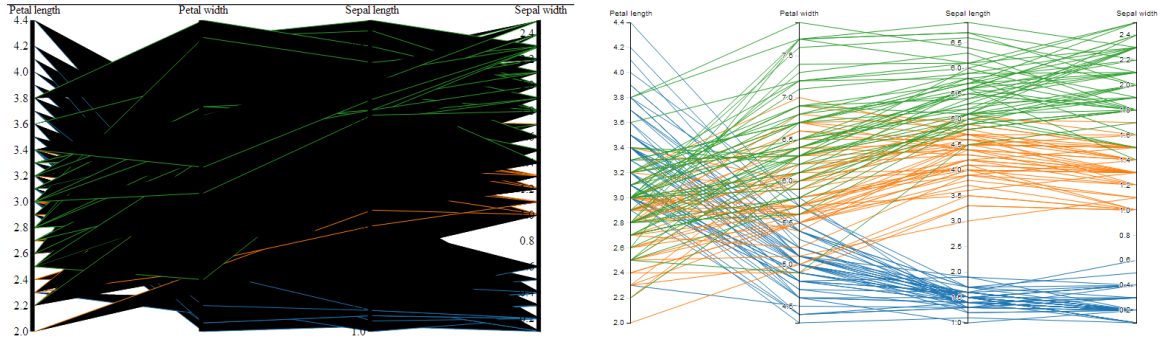


FIGURE 5.9 – Deux SVG identiques téléchargés par SVG Crowbar sur Firefox et Chrome

Le programme est simple, il cible tous les SVG du DOM dans un premier temps. Puis pour chacun, il recherche les différentes règles CSS associées et les associe à chaque SVG. Enfin, à l'aide de l'API File, il propose à l'utilisateur de télécharger le SVG de son choix.

Sur les autres navigateurs (autres que Chrome), un problème de compatibilité empêche à SVG Crowbar de lier correctement le CSS, si bien que la visualisation est téléchargé avec qu'une partie seulement du design du SVG.

Je me suis inspiré des mécanismes de SVG Crowbar pour l'intégrer à VizAssist en modifiant la façon dont sont récupérés les différentes règles CSS. La fonctionnalité n'est à l'heure actuelle pas encore en ligne. Cela devrait être effectué lors de l'intégration des résultats des projets de Web et Multimédia, qui arrivera dans les prochains jours. Cela signera notre dernier déploiement de l'application VizAssist pour cette année.

Conclusion

6.1 Conclusion de Rémy

6.1.1 Algorithme Génétique

Depuis mon Projet d'Ingénierie du Logiciel de l'an dernier, qui était basé sur les algorithmes génétiques, je me suis mis à apprécier cette métaheuristique. C'est pour cela, que j'ai voulu ce Projet de Fin d'Etude.

L'avantage de ce projet, est que j'ai pu intégrer ma propre AGI, en partant de rien. Cela fut très instructif en plus d'être intéressant.

En effet, la puissance de cette métaheuristique est réellement impressionnante et adaptable à tout type de sujet.

De plus, malgré sa puissance, il est possible de la mettre en place avec des langages de programmation tel que JavaScript, qui n'avait pas la réputation d'être très performant, à une époque aujourd'hui révolue.

6.1.2 Apprentissage

Mon cursus m'avait permis de découvrir le Web, dès mes premières années d'études supérieures. Cependant, bien que pensant connaître ce domaine, je me suis rendu compte, au fur et à mesure du déroulement du projet, que mes connaissances étaient très limitées, voir erronées en fonction du contexte.

J'ai pu découvrir de nouvelles technologies, librairies en JavaScript, telles que :

— D3
— NodeJS
— Modernizr
— Require

Ainsi que les bonnes pratiques du développement en JavaScript.

Je suis donc réellement enthousiaste quant à mes nouvelles connaissances, car le Web est un secteur que j'affectionne tout particulièrement, et ces connaissances me seront des plus utiles lors de ma vie professionnelle.

De plus, des problématiques sont ressorties de ce projet, notamment la compatibilité internavigateur qui reste un des principaux problème du développement Web. Ainsi, en voulant rendre notre application le plus compatible possible, nous avons découvert des outils permettant d'aider un développeur dans ces soucis de compatibilité.

6.1.3 Equipe

Mon projet de fin d'étude fut aussi positif de par l'obligation du travail en équipe. En effet, bien que durant notre cursus à Polytech nous ayons déjà effectué des projets en équipe, celui-ci fut le premier ayant la même taille qu'un projet industriel. Cela fut donc bénéfique quant à notre vie professionnelle futur. En effet, Jean et moi avons travaillé ensemble pendant 8 mois, et nous avons donc été obligé de faire des concessions l'un envers l'autre. Cependant, travailler conjointement avec quelqu'un sur un projet de cet ampleur fut une expérience très enrichissante.

Ensuite, Monsieur Venturini fut un tuteur très appréciable, car toujours disponible pour nous, quelque soit la raison. Il nous a conseillés et guidés durant l'intégralité du projet, ce qui fut un réel plaisir et nous a motivés à produire un rendu le plus fidèle possible à ce qu'il avait imaginé en écrivant ce sujet de projet de fin d'étude.

Enfin, nous avons accueilli les deux binômes de Web et Multimédia dans l'équipe de VizAssist. Ce fut intéressant car nous avons alors eut à ce moment un rôle de pédagogue, afin qu'ils puissent ajouter leurs visualisations le plus facilement possible.

Pour finir, j'espère que l'équipe qui reprendra VizAssist sera aussi motivée et intéressée que nous l'avons été durant notre projet de fin d'étude.

6.2 Conclusion de Jean

6.2.1 Sujet intéressant

Le développement d'application Web est un sujet que j'affectionne tout particulièrement. Avec ce projet, nous avons eu à faire à de nombreuses problématiques de cet environnement : les compatibilités inter-navigateurs, quelles technologies utiliser et quand.

La visualisation de données n'est pas un sujet que nous étudions à l'école, mais qui mérite de s'y intéresser avec l'explosion des banques Open Data et le mouvement Big Data, où on peut très rapidement trouvé des informations à exploiter.

Mon projet était au final très divers. J'ai dû jongler entre plusieurs courtes tâches, ce qui m'a permis d'obtenir des connaissances variées sur GeoJSON, les différentes projections géospatiales et les sites de banque de données.

6.2.2 Outils utilisés

Avoir pu approcher la visualisation au travers de D3 et de VizAssist, fut une très grande expérience. On a su se faire plaisir en développant les visualisations et l'application. Développer des visualisations avec D3 a un côté ludique. De plus, la liberté et la confiance que nous laissait M. Venturini nous a permis de pouvoir profiter de ce projet pour mettre en pratique des technologies récentes, parmi RequireJS et Modernizr, très utilisés dans le développement Web, et intéressantes pour notre connaissance personnelle.

Enfin m'être occupé du déploiement de l'application m'a permis de voir une autre facette du projet et de sortir la tête des visualisations pour m'occuper du workflow de mise en ligne de l'application. Git est un outil formidable, que nous aurions dû utiliser plus tôt, à mon avis, dans le développement de notre application. C'est dommage qu'on le voit si peu à l'école. C'est un outil qui peut être difficile à appréhender, mais qui offre des mécaniques plus intéressantes que SVN. Lorsque le réseau Wifi était surchargé, on aurait apprécié pouvoir commit localement nos modifications pour continuer de travailler. On ne fera pas la même erreur la prochaine fois. Cela n'empêche pas cela dit de garder SVN à côté, comme dépôt central.

6.2.3 Critiques

Je me suis beaucoup impliqué dans ce projet. J'ai été un peu déçu de ne pas utiliser NodeJS, qui aurait ouvert un autre pan de recherches et de connaissances. Avoir du JavaScript d'un bout à l'autre de notre projet, aurait permis de garder une meilleur cohérence dans notre développement. De plus, il existe de nombreux outils de gestion de projet et d'aide au développement, intéressants dans un environnement JavaScript, utilisés à partir de NodeJS.

Cependant, avec le recul et le temps que ce projet nous a pris, je me dis que ce fut une bonne décision pour cette année de rester sur Apache. PHP était un langage que nous connaissions mieux que le JavaScript avant de débuter ce projet. On a pu être rapide sur la mise en place du serveur pour se concentrer sur l'application cliente. On aurait peut être pas pu respecter les délais en utilisant NodeJS et nous l'aurions regretté.

Travailler en équipe avec Rémy fut une autre grande expérience. Nous n'avons pas les mêmes personnalités. J'ai tendance à être plus pointilleux sur les conventions et les bonnes pratiques que lui. Je pense que cela nous a demandés à tous les deux des efforts pour travailler ensemble, après tant d'heures, sur un même projet.

6.2.4 Perspectives d'évolution

J'espère que nos successeurs seront aussi passionnés que nous l'avons été. Il y a beaucoup de chose à ajouter à VizAssist. La galerie de D3 regorge d'exemples à mettre en place sur l'application. Nous n'avons qu'effleuré les différents layouts de la librairie. Autre que de nouvelles visualisations, par exemple, il peut être difficile de comprendre le format de donnée accepté par VizAssist. Le contrôle des données est inflexible à l'heure actuelle et pourrait aider l'utilisateur à corriger ses erreurs directement depuis l'application. Le format de fichier CSV commence aussi à atteindre ses limites du côté serveur. Sans compter les visualisations des projets Web et Multimédia, nous sommes à 25 visualisations pour 127 attributs visuels. La liste des visualisations va devenir difficile à maintenir. Le problème se retrouve aussi sur la liste des objectifs et de leur score d'appariement aux visualisations. Il serait intéressant d'envisager une interface administrateur sur le site pour aider au maintient de ces listes.

6.3 VizAssist

Après plus de 200 heures à l'école, et d'innombrables chez nous, passés sur ce projet, notre participation à VizAssist se termine. Le premier semestre nous a introduits à D3, au JavaScript et aux différentes contraintes que devaient prendre en compte notre solution. Durant le second, nous avons développé l'application, spécifiée au premier semestre.

Devoir garder les données du client en local a été le plus gros défi. Très peu de solutions existent à l'heure actuel mettant en place ces technologies. Heureusement que la File API était là, sinon nous aurions dû user d'astuces moins ergonomiques pour pouvoir respecter cette contrainte.

Avoir ensuite une souplesse à l'ajout facile de nouvelles visualisations s'est fait très aisément grâce à RequireJS et son architecture modulaire, ainsi qu'au modèle de données que M. Venturini avait imaginé via ses fichiers CSV.

Ce projet été une introduction pour nous au domaine de la visualisation de données et aux problématiques du choix de la visualisation la plus adaptée aux besoins, mais pas seulement. Le contenu de ce projet fut très varié. Soit par l'implémentation d'algorithme comme celui de Rémy, ou alors par la découverte de contenus géographiques pour Jean. De nombreux points du développement Web ont ainsi été découvert au travers de ce projet, qui fut très riche. M. Venturini a joué le rôle parfait du client indécis. De nombreuses réunions ont été nécessaires pour arriver à ce résultat, mais c'est dans ce contexte que s'utilisent le mieux les méthodes agiles : avoir des échanges réguliers entre l'équipe de projet et le client et savoir réagir rapidement et efficacement à toute modification. Nous l'avons bien compris après ces mois de collaboration.

Si ce projet était à refaire, nous ne referions pas tout ceci. Notre maîtrise du langage JavaScript a bien évolué durant ces 7 mois et nous avons conscience que certaines choses peuvent être mieux gérées. Cependant, nous n'avions pas les connaissances nécessaires au début du développement pour nous en rendre compte. Par exemple, il existe un objet History introduit par HTML5, que nous avons découvert que tout récemment, qui permet de sauvegarder l'état de l'application à l'instant voulu. Nous avons développé un système similaire sur l'application pour reproduire le même mécanisme. On pourrait ainsi gagner du temps et laisser le navigateur gérer l'historique de navigation.

Malgré cela, nous sommes fiers d'avoir produit la version en ligne de VizAssist. Les premiers retours utilisateurs sont tous très positifs. Un professeur voudrait par exemple proposer un sujet similaire, à plus petite échelle, lors de séances de TP. Nous serions ainsi cités en référence, ce qui est gratifiant après avoir passé tant de temps sur le projet.

Avec ce que vous savez maintenant de VizAssist, nous espérons vous avoir convaincu de son utilité, et que vous pourrez nous aider à l'améliorer, via vos suggestions.

Liste des visualisations

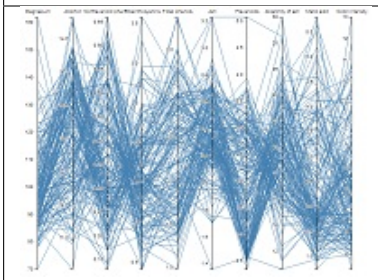
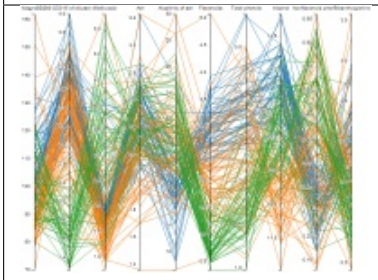
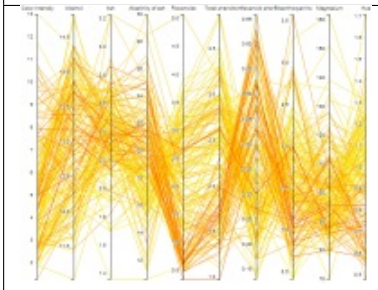
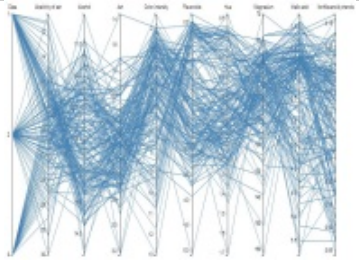
Visualisation	Description
	10 axes max parallel coordinates
	10 axes max parallel coordinates with class
	10 axes max parallel coordinates with measure
	10 axes max parallel coordinates with nominal attribute

TABLE A.1 – Liste des coordonnées parallèles

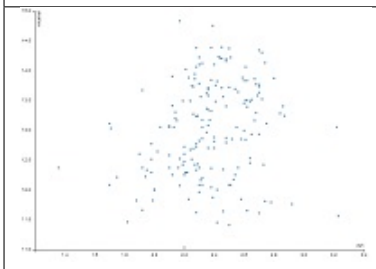
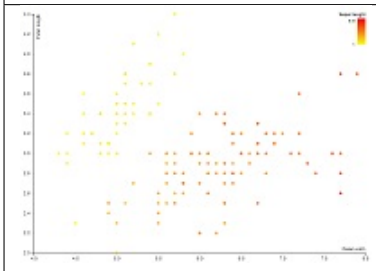
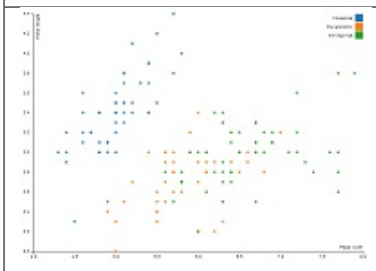
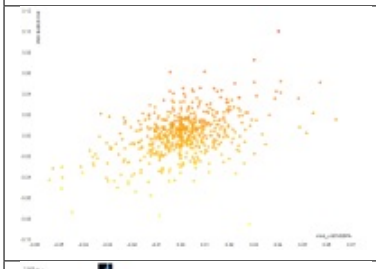
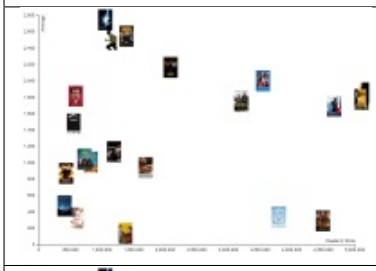
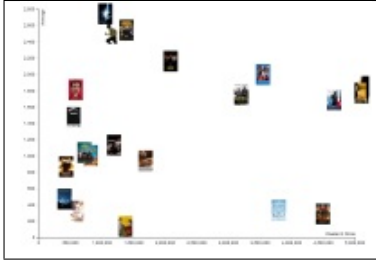
Visualisation	Description
	2D scatter plot
	2D scatter plot with colors for measure
	2D scatter plot with colors for class
	2D scatter plot with colors for ordinal
	2D scatter plot with images
	2D scatter plot with images and url

TABLE A.2 – Liste des nuages de points à deux dimensions

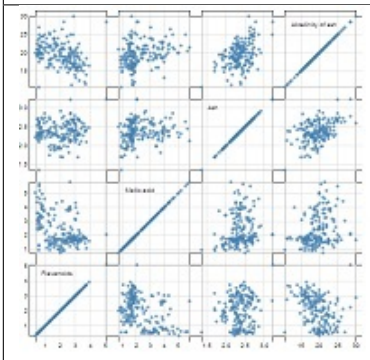
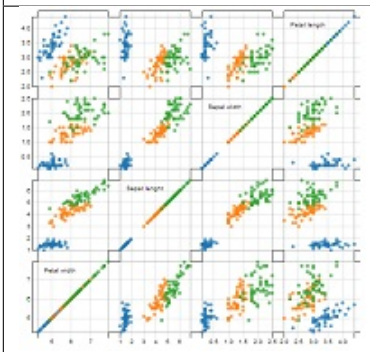
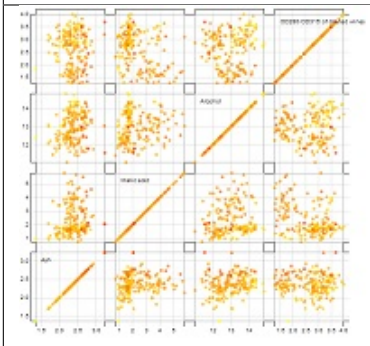
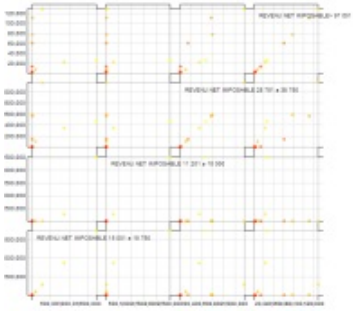
Visualisation	Description
	4 scatter plots matrix
	4 scatter plots matrix with colors for class
	4 scatter plots matrix with colors for measure
	4 scatter plots matrix with colors for ordinal class

TABLE A.3 – Liste des matrices de nuages de points

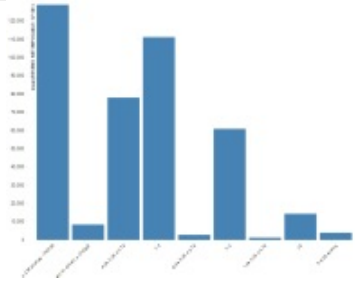
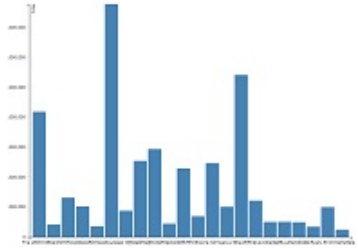
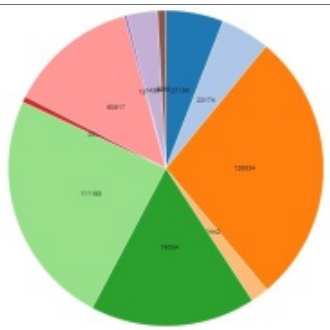
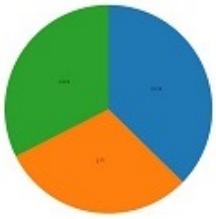
Visualisation	Description
	Histogram for ordinal attribute
	Histogram for class attribute
	Pie Chart for ordinal attribute
	Pie Chart for class attribute

TABLE A.4 – Liste des histogrammes et diagrammes camemberts

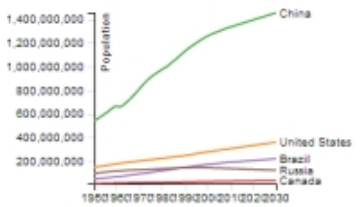
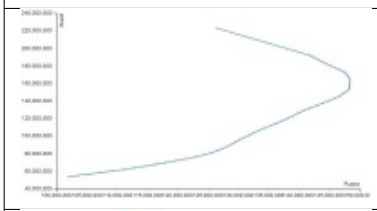
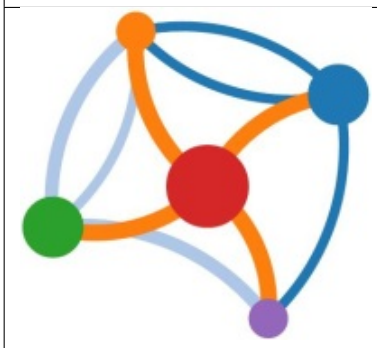
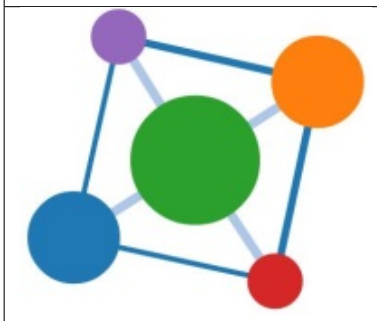
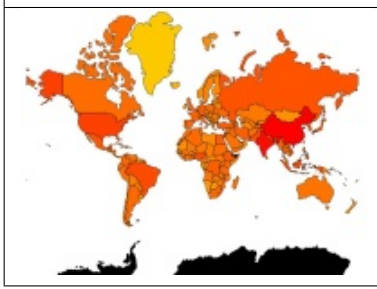
Visualisation	Description
	Time series graph
	Temporal 2D trajectory
	Directed Graph force-directed layout
	Undirected Graph force-directed layout
	World map for ordinal attribute

TABLE A.5 – Liste des times series, graphes et planisphères

Détail du déroulement des sprints

#	Tracker	Priorité	Sujet
562	Evolution	Normal	Habillage de l'application (CSS)
561	Evolution	Normal	Tester le navigateur de l'utilisateur
560	Evolution	Haut	Mise en place de l'environnement
583	Assistance	Normal	Poster partie 2
605	Evolution	Normal	Test - Server
604	Evolution	Normal	Dev - Server
603	Evolution	Normal	Test Obj et Visu
602	Evolution	Normal	ReadVisualizations
601	Evolution	Normal	ReadObjectives
594	Evolution	Normal	Ouverture et lecture des fichiers csv.
584	Assistance	Normal	CdS Sprint 1
607	Evolution	Normal	Test - PhpToJSON
606	Evolution	Normal	Dev - PhpToJSON

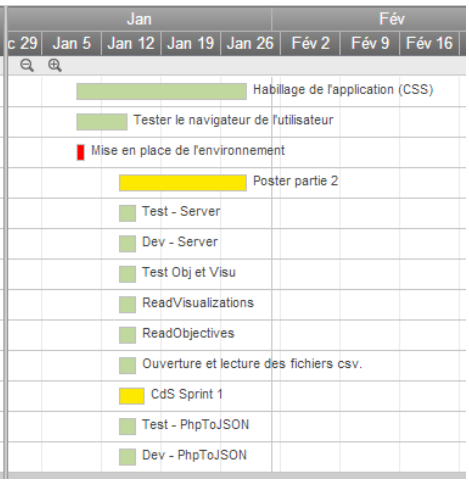


FIGURE B.1 – Détail des tâches du sprint 1

#	Tracker	Priorité	Sujet
640	Evolution	Normal	Test - LoadData
639	Evolution	Normal	Dev - LoadData
638	Evolution	Normal	Test - ShowData
637	Evolution	Normal	Dev - ShowData
636	Evolution	Normal	Test - ControlData
635	Evolution	Normal	Dev - Control Data
610	Evolution	Normal	ReadDemoFile Test
609	Evolution	Normal	ReadDemoFile
643	Evolution	Normal	Dev - Navigation

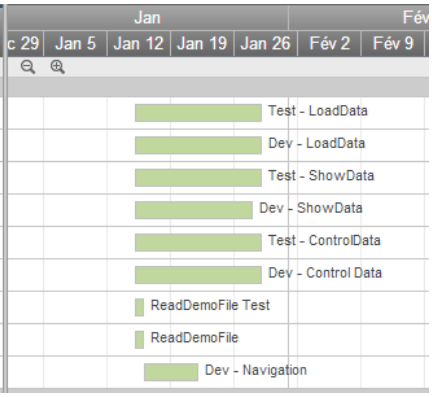


FIGURE B.2 – Détail des tâches du sprint 2

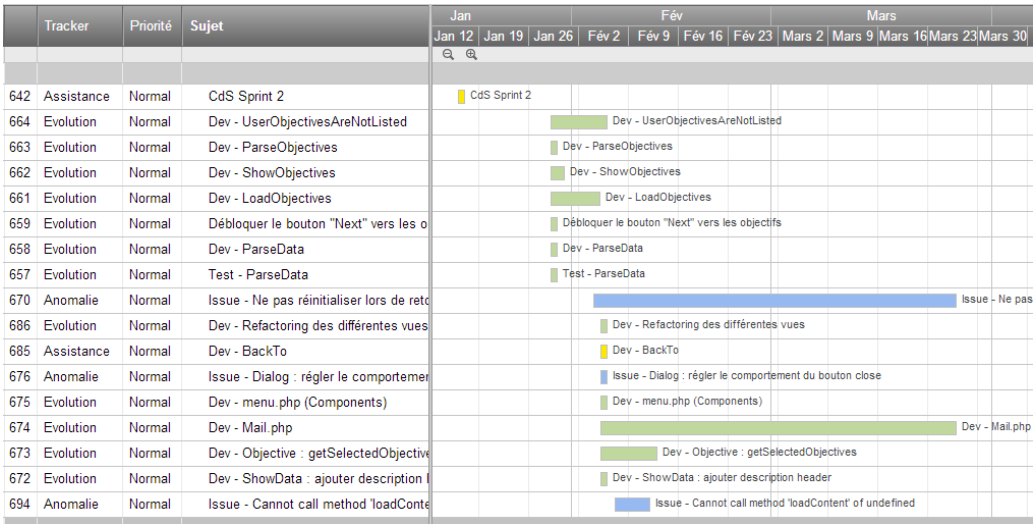


FIGURE B.3 – Détail des tâches du sprint 3

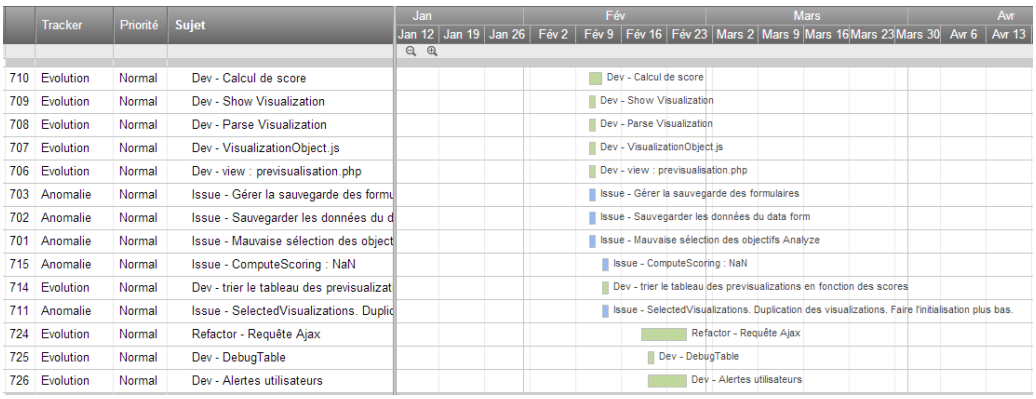


FIGURE B.4 – Détail des tâches du sprint 4 et 5, que nous avons fusionnés

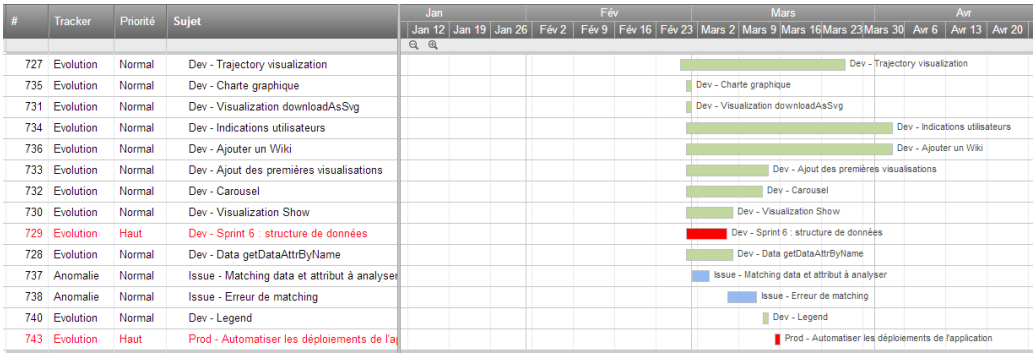
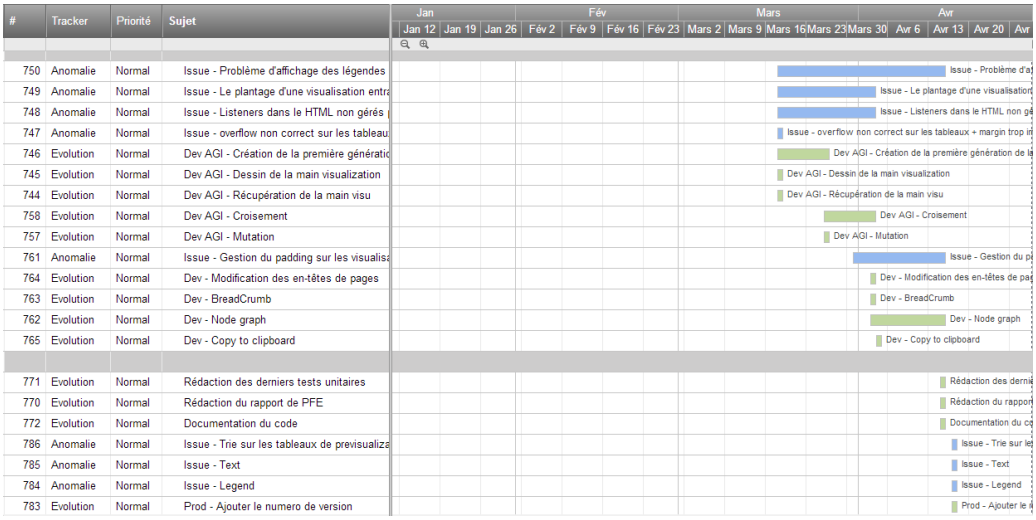


FIGURE B.5 – Détail des tâches du sprint 6



#	Tracker	Priorité	Sujet
750	Anomalie	Normal	Issue - Problème d'affichage des légendes
749	Anomalie	Normal	Issue - Le plantage d'une visualisation entra
748	Anomalie	Normal	Issue - Listeners dans le HTML non gérés
747	Anomalie	Normal	Issue - overflow non correct sur les tableau
746	Evolution	Normal	Dev AGI - Création de la première générati
745	Evolution	Normal	Dev AGI - Dessin de la main visualization
744	Evolution	Normal	Dev AGI - Récupération de la main visu
758	Evolution	Normal	Dev AGI - Croisement
757	Evolution	Normal	Dev AGI - Mutation
761	Anomalie	Normal	Issue - Gestion du padding sur les visualisa
764	Evolution	Normal	Dev - Modification des en-têtes de pages
763	Evolution	Normal	Dev - BreadCrumb
762	Evolution	Normal	Dev - Node graph
765	Evolution	Normal	Dev - Copy to clipboard
771	Evolution	Normal	Rédaction des derniers tests unitaires
770	Evolution	Normal	Rédaction du rapport de PFE
772	Evolution	Normal	Documentation du code
786	Anomalie	Normal	Issue - Trie sur les tableaux de previsualiza
785	Anomalie	Normal	Issue - Text
784	Anomalie	Normal	Issue - Legend
783	Evolution	Normal	Prod - Ajouter le numero de version

FIGURE B.6 – Détail des tâches du sprint 7

Glossaire

Apache : serveur HTTP permettant l'hébergement d'un site Web. La version utilisée ici permet d'interpréter du PHP afin de rendre le site dynamique.

AMP : Apache MySQL PHP est un package contenant les outils permettant le déploiement rapide d'un site Web embarquant un conteneur d'application Apache HTTP Serveur, PHP, une base de donnée MySQL et très souvent PHPMyAdmin délivrant une interface pour administrer la base MySQL.

AJAX : Asynchronous JavaScript And XML est un type d'architecture Web permettant l'échange rapide de donnée entre le client et le serveur. Cela permet d'effectuer des requêtes en JavaScript pour interroger le serveur et de modifier le contenu d'une page Web en conséquence.

Base de donnée : Lieu permettant de contenir des données informatiques, le plus souvent située sur un serveur. Il est ensuite possible d'interagir avec la base pour demander, modifier ou supprimer des données.

Base de donnée : défaut de conception informatique entrainant un mauvais fonctionnement du système. Une bonne conception et de longues séries de tests sont donc à établir pour réduire au maximum la présence de bug dans un programme.

CSV : Comma Separated Values est un type de fichier de données de la classe des Delimiter-Separated Values (fichiers DSV). Il contient un tableau de deux dimensions séparant chaque valeur par un délimiteur, dans le cas du CSV, une virgule (comma).

Fonction : code réalisant une tâche et pouvant renvoyant une valeur ou un objet.

D3js : Data Driven Document est une librairie JavaScript aidant à la visualisation de données. Spécialisée dans la création et la gestion d'image SVG, elle permet aussi de modifier le DOM.

DOM : Document Object Model est un standard W3C décrivant une interface Web indépendante de tout programme informatique. Elle peut être modifiée par différents langages, puis interprété par un logiciel tel qu'un navigateur Web.

Heat map : type de visualisation de donnée représentant une matrice 2D de données.

HTTP : Hypertext Transfer Protocol est un protocole de communication Client-Serveur développé pour le World Wide Web qui fait partie de la couche application selon le modèle OSI.

IDE : Integrated Developement Environment est un ensemble d'outils améliorant le cadre de travail d'un développeur. Il contient un éditeur de texte ainsi que divers outils pour assister le programmeur dans la rédaction de son code.

JavaScript : Langage à l'origine utilisée sur les pages Web pour manipuler le DOM et effectuer des requêtes en AJAX et permet de manipuler le DOM. On le retrouve maintenant dans l'interprétation de code côté serveur.

jQuery : Librairie JavaScript facilitant la rédaction de code JavaScript dans la gestion de requête AJAX ainsi que dans la manipulation du DOM.

Méthode : fonction appartenant à une classe.

MVC : Modèle Vue Contrôleur est un type d'architecture informatique dans les programmes. Cela permet de bien séparer les tâches entre les différentes classes. Certaines se chargent du rendu visuel (la

vue), tandis que d'autres du contrôle du contenu sur ces vues (le contrôleur) et enfin les modèles sont ces classes qui seront manipulées dans le contenu.

MySQL : système de gestion de base de donnée Open Source. C'est un serveur de base de donnée relationnelle SQL.

PHP : PHP : Hypertext Preprocessor est un langage compilé à la volée libre utilisé côté serveur pour rendre dynamique des pages Web. Il est capable ainsi de recevoir et traiter les requêtes AJAX et SQL pour répondre aux besoins de navigateur.

SVN : Apache Subversion est un logiciel distribué sous licence Apache et BSD et est un logiciel de gestion de version.

Test : enchaînement de tâches permettant de contrôler le bon fonctionnement d'un programme.

TSV : Tab-Separated Values est un type de fichier de données de la classe des Delimiter-Separated Values (fichiers DSV). Il contient un tableau de deux dimensions séparant chaque valeur par un délimiteur, dans le cas du TSV, une tabulation (tab).

Versionning : gestion de version. Permet dans un projet collaboratif de garder des traces des modifications tout au long du développement d'un projet.

Wizard : assistant logiciel qui permet d'automatiser certaines tâches comme l'installation ou le paramétrage d'un programme. On retrouve ainsi une suite d'étape pour aider l'utilisateur dans ses choix.

VizAssist - Assistant Web pour la visualisation de données

Département Informatique
5e année
2013 - 2014

Rapport de PFE

Résumé : Ce rapport présente le projet de fin d'étude de deux étudiants de l'école d'ingénieur Polytech Tours : Jean de Barochez et Rémy Pradignac. Ce projet a commencé le 26 septembre 2013 pour finir le 14 mai 2014 à leur soutenance. L'objectif du projet était de mettre en place l'application Web VizAssist, assistant les utilisateurs à choisir la visualisation la mieux adaptée à leurs données et à leurs objectifs liés à ces données. Ce document présente donc la gestion du projet et l'architecture de l'application. Le binôme s'est séparé début mars, Jean s'occupant du déploiement de l'application en ligne et Rémy de l'implémentation d'un algorithme génétique interactif.

Mots clefs : VizAssist, assistant Web utilisateur, visualisations, D3, SVG, RequireJS, Modernizr, File API, algorithme génétique interactif

Abstract: This report deals with the final project study of two students from the engineering school Polytech Tours : Jean de Barochez and Rémy Pradignac. This project began in the 29th of September 2013 to end the 14th of May 2014. The project a led to the deployement of VizAssist, a visualization Web assistant. This application will guide you to choose the visualization that best matches your data and your objectives. This document reports how the project was managed during this period, and the architecture of the application.

Keywords: VizAssist, user Web assistant, visualizations, D3, SVG, RequireJS, Modernizr, File API, interactive genetic algorithm

---

Encadrants
Gilles VENTURINI
gilles.venturini@univ-tours.fr

Université François-Rabelais, Tours

Étudiants
Jean de BAROCHEZ
jean.debarochez@etu.univ-tours.fr
Rémy PRADIGNAC
remy.pradignac@etu.univ-tours.fr

DI5 2013 - 2014