IUT Nice Côte d'Azur

Département informatique

Rapport de stage

effectué à l'INRIA

17 mai 2021 - 9 juillet 2021

Fedi GHALLOUSSI

2e année de DUT Informatique

Centre Inria Sophia Antipolis - Méditerranée
2004 Route des Lucioles,
06902 Valbonne

Encadrant à l'INRIA
Luc Hogie
Responsable académique
Leo Donati

1 Remerciements

Tout d'abord, je tiens à remercier Monsieur Luc Hogie de m'avoir offert cette opportunité de stage et de m'avoir conseillé et accompagné tout au long de celui-ci.

Mes remerciements s'adressent aussi à toutes les personnes que j'ai rencontrées à l'INRIA et qui m'ont très bien accueilli.

Je souhaite de plus remercier Monsieur Leo Donati pour son implication et ses conseils prodigués lors de mon stage.

Enfin, je remercie Monsieur Erol Acundeger et Madame Béatrice Brun d'avoir été là pour répondre à toutes mes questions préalables au stage.

2 Résumé

Ce document a pour but de rapporter les missions que j'ai réalisées pendant mon stage de fin d'études de mon DUT informatique. Mon stage se déroulait au sein de l'INRIA Sophia Antipolis - Méditerranée pendant une durée de huit semaines. Le stage s'est déroulé en grande partie en télétravail tout en étant en présentiel plus ou moins deux jours par semaine selon les besoins.
L'INRIA est un centre de recherche et j'ai intégré l'équipe COATI de celle-ci. Ce stage a donc été pour moi l'occasion de découvrir le domaine de la recherche qui ne m'était pas du tout familier.

L'objectif de ce stage est de développer une plateforme web qui expose des services web pour la création et la modification de graphes mais aussi et surtout pour la visualisation de ceux-ci. Ce service de visualisation devra permettre d'observer des graphes tout en permettant de personnaliser en profondeur la forme de ceux-ci. On devra aussi pouvoir modifier dynamiquement le graphe tout en visualisant les changements. Des librairies web existent pour faire une partie de cela, mais aucune ne convient à tous les besoins cités. Ainsi, ma tâche principale était de développer le front end de ces services. J'ai pu faire cela notamment à l'aide du langage Javascript et de la librairie Vis.js, spécialisé dans l'affichage de graphe.

3 Abstract

The purpose of this document is to report the assignments that I carried out during my internship at the end of my studies of my two-year degree in IT. My internship took place at INRIA Sophia Antipolis - Méditerranée for a period of eight weeks. The internship took place largely by teleworking while being face-to-face more or less two days a week as needed.
INRIA is a research center and I joined its COATI team. This internship was therefore an opportunity for me to discover the field of research which was not at all familiar to me.

The objective of this internship is to develop a web platform that exposes web services for the creation and modification of graphs but also and above all for the visualization of these. This visualization service should make it possible to observe graphs while allowing in-depth customization of their shape. We must also be able to dynamically modify the graph while visualizing the changes. Web libraries exist to do a part of this, but none are suitable for all the stated needs. So, my main task was to develop the front end of these services. I was able to do this in particular using the javascript language and the Vis.js library, which specializes in graph display.

Table des matières

4 Présentation de l'entreprise

FIGURE 1 – Logo de l'INRIA

Le centre de recherche Inria Sophia Antipolis-Méditerranée a été créé en 1983. Sa dynamique s'inscrit dans le développement du site de Sophia Antipolis à Nice, avec l'Université Côte d'Azur. Il est également implanté à Montpellier, où il accompagne le développement de l'Université de Montpellier.

Inria est présent sur Sophia Antipolis et Nice avec 34 équipes-projets dont la moitié sont communes, et depuis 2003 sur le site de Montpellier avec, en 2019, 4 équipes-projets communes. Le centre a également une équipe à Bologne en Italie et une autre avec l'Université d'Athènes. Son action mobilise 500 personnes, scientifiques et personnels d'appui à la recherche et à l'innovation, issues de 55 nationalités.

Ses axes scientifiques prioritaires sont de développer la thématique de l'intelligence artificielle en lien avec le projet structurant du 3IA Côte d'Azur, notamment sur les sujets biologie-santé incluant médecine, neurosciences et biologie computationnelles, les liens entre IA, géométrie, données hétérogènes et modélisation et robotique collaborative pour les environnements ouverts et dynamiques et de développer les activités de recherche dans le domaine de l'informatique ubiquitaire, en particulier dans les thématiques du génie logiciel, de la sécurité, fiabilité et certification des logiciels.

Le centre a de nombreux partenariats dont le CNRS. L'équipe COATI (Combinatorics, Optimization, and Algorithms for Telecommunications) que j'ai rejoint est un des fruits de ce partenariat. Cette équipe a un grand passif avec les graphes et la théorie des graphes et s'inscrit donc parfaitement dans le contexte de ce stage.

5 Cahier des charges

5.1 Terminologie

- Graphe : Ensemble de vertices (ou noeuds) et d'arcs ou d'edge liant certains couples de vertices.
- Vertex(noeud) : En théorie des graphes, un sommet, aussi appelé nœud et plus rarement point, est l'unité fondamentale d'un graphe. Deux sommets sont voisins s'ils sont reliés par un lien. Deux sommets sont indépendants s'ils ne sont pas voisins.
- Edge : Lien non orienté entre deux sommets
- Arc : Lien orienté entre deux sommets

5.2 Contexte et motivation

Le but général est de faire une plateforme web qui expose des services web pour la création et la modification de graphes du type removeVertex, addArc ou removeArc par exemple. Sur chacun des éléments du graphe, on doit être capable de définir la taille, la couleur, le label et beaucoup d'autres types de personnalisation. Les graphes seront hébergés sur le serveur et pourront donc être manipulés par les programmes clients dans n'importe quel langage à travers les services web.

Le but de la partie dont j'ai la charge est de visualiser le graphe avec toutes les propriétés visuelles qui lui sont associées ainsi que de pouvoir répercuter les changements dynamiques du graphe en direct sur la visualisation.
Certaines solutions plus ou moins similaires existent mais elles ne regroupent pas toutes les conditions et certaines sont trop lourdes et peu flexibles.

5.3 Description fonctionnelle

5.3.1 Énoncés et caractérisation des fonctions de service

1. Afficher le graphe stocké
   - But : Pouvoir voir le graphe qu'on a stocké
   - Priorité : Maximale
2. Afficher les personnalisations visuelles des éléments du graphe
   - But : Pouvoir personnaliser le graphe en profondeur
   - Priorité : Maximale

3. Affecter n'importe quelle propriété à une personnalisation du graphe
   - But : Pouvoir affecter une propriété personnalisé de l'utilisateur à la personnalisation qu'il veut
   - Priorité : Moyenne
4. Affecter n'importe quelle propriété au label des éléments du graphe
   - But : Pouvoir accéder à la valeur d'une propriété de tous les éléments du graphe très rapidement
   - Priorité : Moyenne
5. Afficher les changements dynamiques sur le graphe en direct
   - But : Pouvoir visualiser directement l'impact des changements fait sur le graphe
   - Priorité : Maximale

5.4 Imposition de conception

- Personnalisation poussée
- Dynamisme

5.5 Planning prévisionnel

5.5.1 Charge totale du projet

Tâche	Durée prévisionnelle (heures)
Compréhension et familiarisation avec le sujet	10
Recherche, état de l'art et analyse du sujet	30
Rédaction du cahier des charges	10
Conception du logiciel	21
Développement du logiciel	202
Total	273

5.5.2 Durée prévisionnel du stage

Charge de travail du projet : 273 heures
Date du projet : du 17/05/2021 au 09/07/2021

5.5.3 Calendrier prévisionnel

Tâche	Semaine du 17 mai	Semaine du 24 mai	Semaine du 31 mai
Compréhension et familiarisation avec le sujet			
Recherche, état de l'art et analyse du sujet			
Rédaction du cahier des charges			
Conception du logiciel			
Développement du logiciel			

Tâche	Semaine du 7 juin	Semaine du 14 juin	Semaine du 21 juin
Compréhension et familiarisation avec le sujet			
Recherche, état de l'art et analyse du sujet			
Rédaction du cahier des charges			
Conception du logiciel			
Développement du logiciel			

Tâche	Semaine du 28 juin	Semaine du 5 juillet
Compréhension et familiarisation avec le sujet		
Recherche, état de l'art et analyse du sujet		
Rédaction du cahier des charges		
Conception du logiciel		
Développement du logiciel		

6 Etat de l'art et logiciels existants

6.1 Neo4j

6.1.1 Présentation

FIGURE 2 – Logo de Neo4j

Neo4j est un système de gestion de base de données au code source libre basée sur les graphes. Neo4j permet de représenter les données en tant que nœuds reliés par un ensemble d'arcs, ces objets possédant leurs propres propriétés. Les propriétés sont constituées d'un couple de clé-valeurs de type simple tel que chaînes de caractères ou numérique.

6.1.2 Avantage

- Permet d'observer énormément de noeuds (jusqu'à 34 milliards)
- Permet de personnaliser le graphe en profondeur comme voulu

6.1.3 Défauts

- Extrêmement lourd
- Peu flexible

6.2 Corese

6.2.1 Présentation

FIGURE 3 – Logo de Corese et de l'INRIA

Corese est un moteur de recherche web sémantique. Il est basé sur le formalisme des graphes conceptuels et permet de rechercher des informations en exploitant une base d'annotations RDF reposant sur une ontologie RDFS (un vocabulaire conceptuel standardisé). Il permet d'afficher des graphes et de les modifier à l'aide du langage SPARQL.
Il est développé par une équipe de l'INRIA

6.2.2 Avantage

- Proximité et contact avec les créateurs car ils sont de l'INRIA
- Graphes sémantiquement riche permettant d'assigner beaucoup d'informations
- Permet d'observer de nombreux noeuds

6.2.3 Défauts

- Très complexe d'utilisation
- Visualisation et personnalisation peu avancé

7 Réalisation

7.1 Environnement

7.1.1 Javascript

FIGURE 4 – Logo de Javascript

Le JavaScript est un langage de programmation créé en 1995. Le JavaScript est aujourd'hui l'un des langages de programmation les plus populaires et il fait partie des langages web dits « standards » avec le HTML et le CSS. Son évolution est gérée par le groupe ECMA International qui se charge de publier les standards de ce langage. On dit que c'est un standard du web car les principaux navigateurs web savent tous « lire » ces langages et les interprètent généralement de la même façon ce qui signifie qu'un même code va généralement produire le même résultat dans chaque navigateur. Pour définir ce qu'est le JavaScript et le situer par rapport aux autres langages, il faut savoir que :

- Le JavaScript est un langage dynamique ; c'est-à-dire un langage qui va nous permettre de générer du contenu dynamique pour nos pages web.
- Le JavaScript est un langage (principalement) côté client ; Un langage «côté client » ou « client side » est un langage qui va être exécuté dans le navigateur des utilisateurs qui demandent la page.
- Le JavaScript est un langage interprété. Cela signifie qu'il va pouvoir être exécuté directement sous réserve qu'on possède le logiciel interpréteur.
- Le JavaScript est un langage orienté objet.

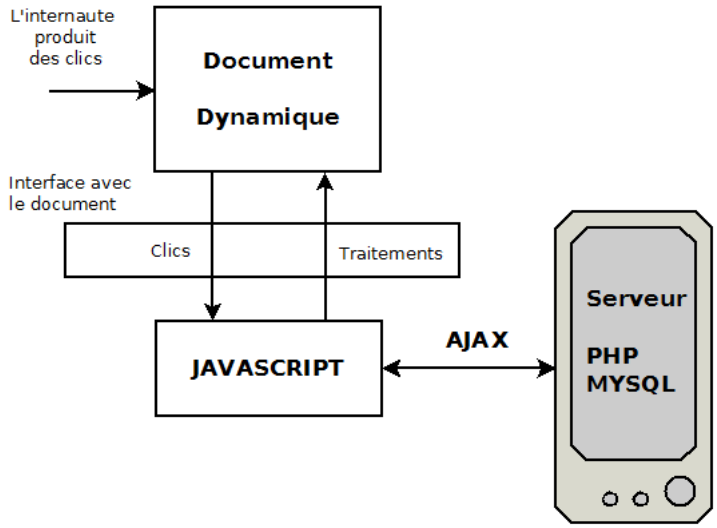

FIGURE 5 – Fonctionnement de Javascript

L'ensemble de la partie de l'application que j'ai réalisé a été faites en Javascript. Ce choix a été évident puisque c'est le langage qui domine très largement le développement web. De plus, l'application existante était déjà faite en Javascript.

7.1.2 Vis.js

FIGURE 6 – Logo de Vis.js

Vis.js est une bibliothèque de visualisation dynamique sur un navigateur. La bibliothèque est conçue pour être facile à utiliser, pour gérer de grandes quantités de données dynamiques et pour permettre la manipulation et l'interaction avec les données. La bibliothèque se compose des composants DataSet, Timeline, Network, Graph2d et Graph3d. Dans notre cas, nous utiliserons seulement les composants Network et Dataset.

Il y a plusieurs librairies pour l'affichage de graphe en Javascript mais mon choix s'est porté sur Vis.js. Tout d'abord, le code existant utilisait déjà cette librairie donc afin de faciliter l'utilisation de celui-ci, il était plus simple de reprendre cette librairie. De plus, Vis.js permet une personnalisa-

tion très pointue du graphe qui correspond parfaitement à ce que l'on veut faire. En outre, la librairie permet de manipuler de nombreux noeuds tout en ayant la possibilité de modifier dynamiquement les éléments du graphe. Enfin, le système physique proposé par la librairie et la disposition intelligente des noeuds ont conforté ce choix.

7.1.3 jQuery et AJAX

FIGURE 7 – Logo de jQuery et d'AJAX

Ajax pour Asynchronous JavaScript and XML, correspond à un groupe de méthodes et de moyens visant à permettre d'établir une communication asynchrone entre le navigateur et le serveur.
Ajax permet d'effectuer des modifications parcellaires sur une page web, sans recharger l'ensemble de la page internet.
jQuery permet de réaliser des appels Ajax de manière très simple. Les appels Ajax sont préalablement instanciés par du code en JavaScript. Dans les grandes lignes, on met en place un gestionnaire d'événements pour anticiper les actions à réaliser avant l'appel et après le retour du serveur (fonction de rappel ou callback) puis on effectue des appels Ajax en jQuery au serveur. Ajax faisant appel à des technologies différentes, nativement une requête XMLHttpRequest peut se comporter de différentes façons d'un navigateur à l'autre. jQuery est là pour pallier ce problème et fournit un cadre pour réaliser des appels Ajax plus standardisés et surtout garantir qu'une requête Ajax s'exécute de la même façon sur chaque navigateur.

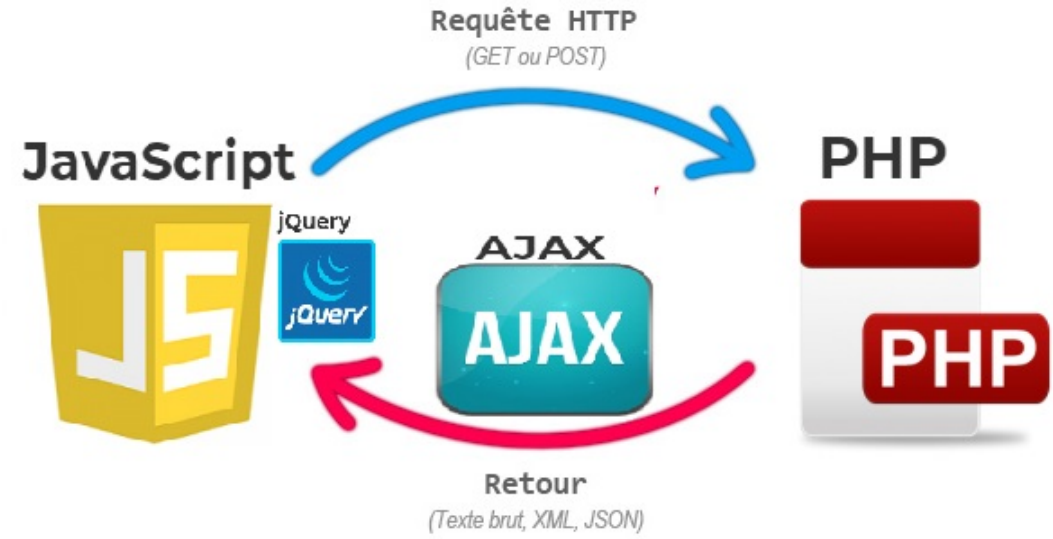

FIGURE 8 – Fonctionnement de jQuery et d'AJAX

Ces technologies ont été choisis afin de récupérer les données du back end, notamment les graphes et les changements dynamiques apportés à ceux-ci.

7.2 Code existant

Le projet était à la base seulement une interface de monitoring d'un système de composants Idawi qui comportait un module affichant un graphe. Un autre stagiaire a pu faire cela avant que j'arrive.
Cependant, ce qui devait être juste une interface de monitoring d'Idawi a évolué pour devenir une librairie de graphes, ce à quoi je prends part.
Ainsi, il y avait donc un code existant permettant d'afficher des graphes que j'ai pu reprendre. Cependant, celui-ci était profondément lié à Idawi et donc pas utilisable complètement en l'état. Ainsi, j'ai décidé de garder la structure globale du code (notamment les classes utilisées) ainsi que l'interface existante permettant d'affecter des personnalisations du graphe aux propriétés.
Il n'a par contre pas été possible de reprendre une majorité du code en lui-même car il faisait complètement référence aux données d'Idawi qu'il affichait.

7.3 Analyse et conception

Les fichiers de l'application se présente ainsi :

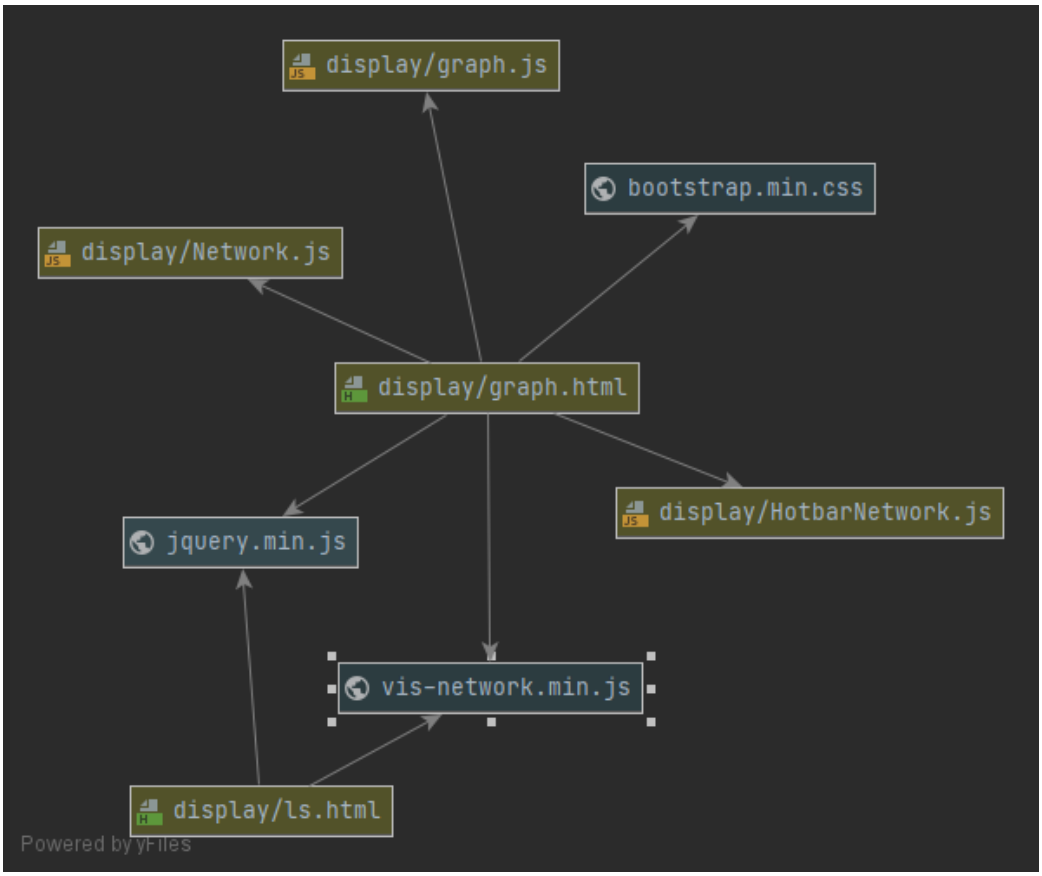

FIGURE 9 – Fichiers du projet

L'application se résume à une page html (graph.html) qui affiche notre interface à l'aide de nos trois fichiers javascript :

- Network.js qui contient les classes des éléments de notre graphe
- HotbarNetwork.js qui contient la classe permettant l'affichage du menu de personnalisation des propriétés
- graph.js qui fait les appels Ajax et traite les JSON à l'aide des deux autres fichiers javascript

Enfin, comme dit auparavant, on utilise Vis.js et jQuery.

Le diagramme de nos classes Javascript se présente ainsi :

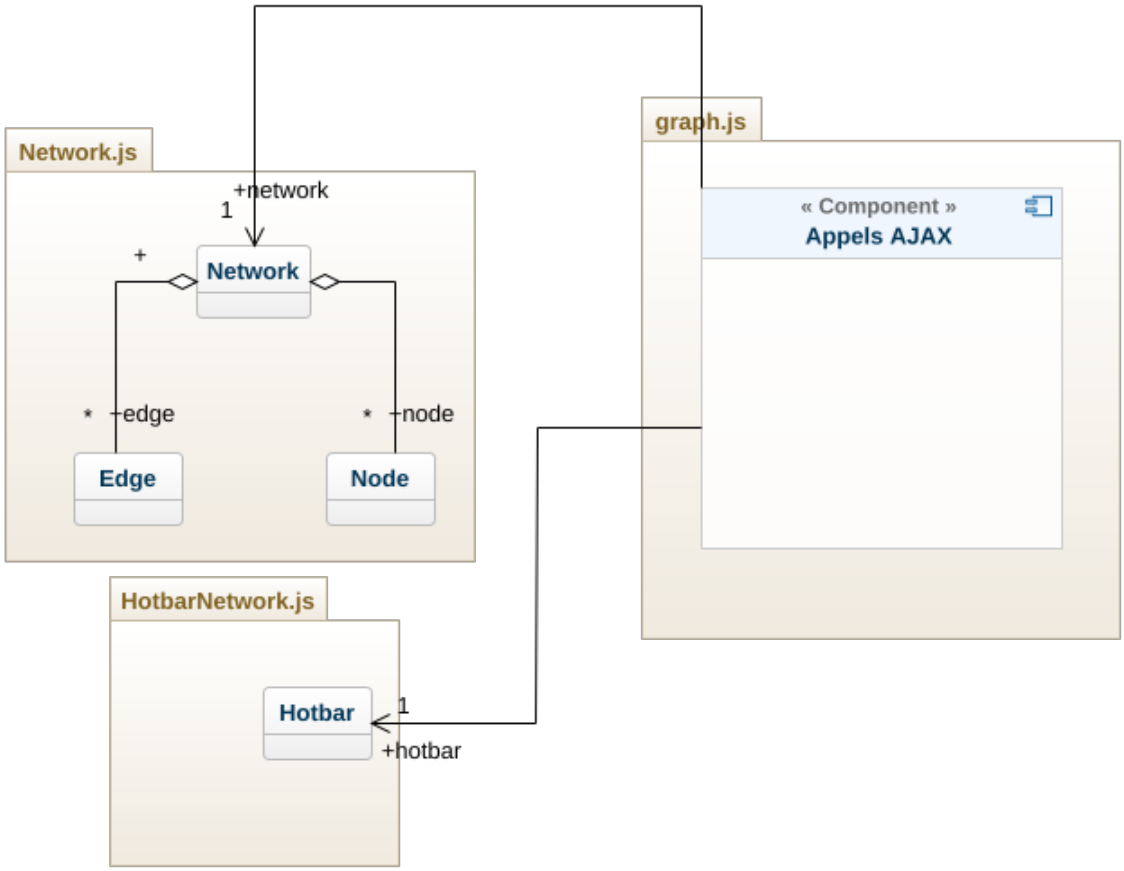

FIGURE 10 – Digramme de classe

Nous avons choisi de faire une classe Network représentant les graphes, une classe Node représentant les noeuds d'un graphe ainsi qu'une classe Link représentant les liens entre les noeuds. Chaque élément Network peut contenir des éléments de la classe Node et Link. Le choix d'utiliser ces classes et de ne pas utiliser celles de Vis.js est lié à la facilité d'utilisation mais surtout à la personnalisation des propriétés gérées non nativement qui nous oblige à faire cela.

Les échanges avec le serveur se présentent ainsi :

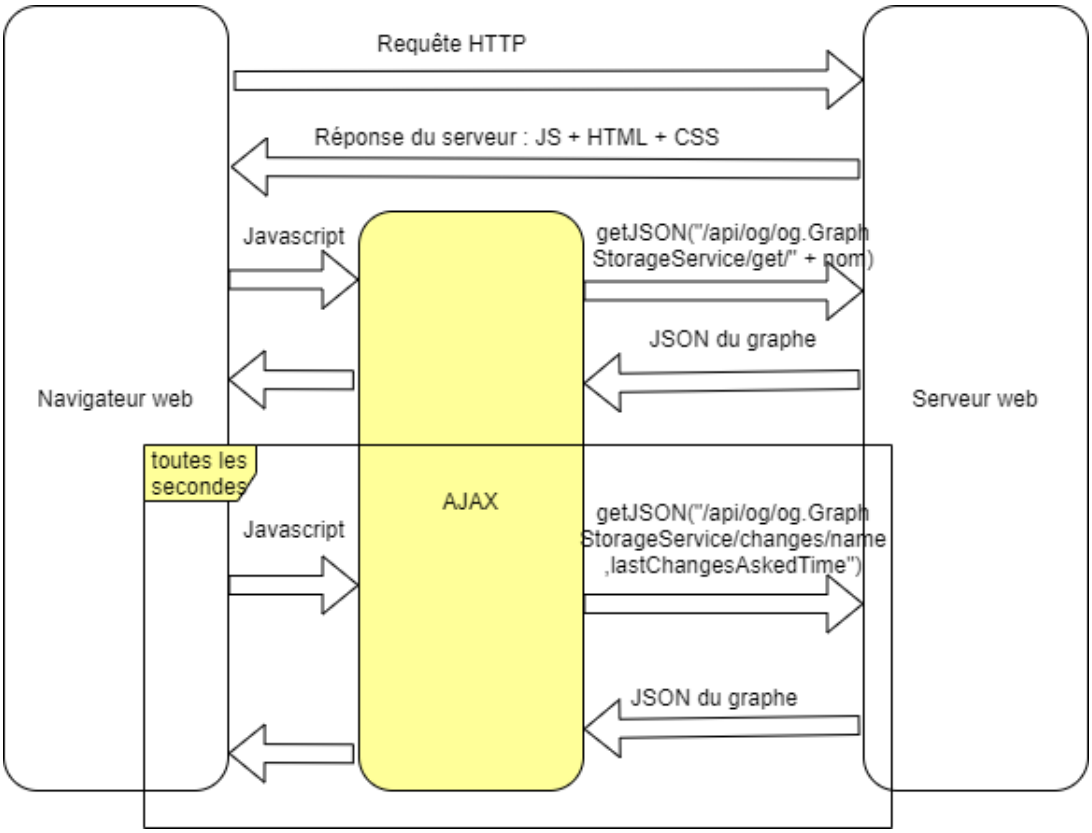

FIGURE 11 – Echanges avec le serveur

On va donc récuperer les JSON fournis par le serveur à l'aide d'Ajax et ce sont ceux-ci qui vont nous permettre de visualiser le graphe en temps réel.

7.4 Développement

7.4.1 Afficher le graphe à partir du JSON récupéré

La première étape consiste à récupérer le JSON du graphe donné en paramètre de l'URL. Comme dit précédemment, la solution choisie pour faire cela est Ajax. Ainsi, on récupère un objet JSON de la forme suivante :


```
"value": {
  "props": {
    "default vertex mass": "1",
    "default edge label": "relation",
    "default vertex size": "20",
    "default edge directed": "yes",
    "default vertex foo": "1",
    "default edge width": "1"
  },
  "vertices": [
    {
      "id": -9148163119997931148,
      "props": {
        "bar": "11.012019487646125",
        "borderWidth": "0",
        "label": "mlxbdx"
      }
    }
  ],
  "edges": [
    {
      "from": -7996837736701854242,
      "to": 3191478325383994979,
      "id": 836895940776313420,
      "props": {}
    }
  ]
},
"type": "og.GraphStorageService$GraphInfo"
```


FIGURE 12 – Allure JSON du graphe

Ce JSON est composé d'une partie "props" qui représente les propriétés par défaut des éléments du graphe. Par exemple, "default edge label : relation" donnera le label "relation" à tous les liens du graphe.
Il est aussi composé d'une partie "vertices" qui représente les noeuds du graphe. Chaque noeud est lui-même représenté par un id et des "props" (propriétés) qui lui sont associés.
Enfin, la partie "edges" représente les liens du graphe. Chaque lien est représenté par trois long : from (l'id du noeud d'où part le lien), to (l'id du noeud où arrive le lien) et id. Ils ont aussi des propriétés.

Afin d'afficher ce graphe visuellement, Vis.js, la librairie choisi nécessite deux choses : un dataset composé des noeuds et des liens du network ainsi que des options qui détermineront les valeurs par défaut des propriétés des éléments ou la gestion de la physique par exemple.
Ainsi, on initialise le dataset du graphe Vis avec la parties "vertices" et "edges"

du JSON et la partie "options" du graphe Vis avec la partie "props".

7.4.2 Traiter les propriétés reconnues des éléments

Pour traiter les propriétés de chaque noeud et lien, on crée les objets Network et les objets Edge et Node associés. Ensuite, on boucle sur tous les noeuds et liens du graphe pour appliquer une fonction permettant de traiter les paramètres de chaque élément.
Cette fonction boucle sur la liste de propriétés de l'élément et appelle la fonction permettant de modifier la propriété visuellement sur le graphe Vis et son dataset.

7.4.3 Traiter les propriétés par défaut des éléments

Pour traiter les propriétés par défaut, on utilise la même méthode que pour les propriétés spécifiques, sauf que l'on place l'appel de la fonction qui va traiter les paramètres par défaut avant celle qui traite les propriétés spécifiques des éléments du graphe. Ainsi, on va d'abord appliquer les paramètres par défaut, puis, si des données spécifiques sont données, on écrase les données par défaut.

7.4.4 Personnalisation des propriétés

On a déjà parlé du cas des propriétés gérés nativement, mais le graphe d'un utilisateur peut souvent contenir d'autres informations qu'on ne peut gérer nativement. Ainsi, nous avons choisi de faire un menu permettant de personnaliser la visualisation de ces données que l'on ne connaît pas mais aussi de personnaliser l'effet des propriétés même si elles sont déjà gérées nativement si l'utilisateur le souhaite.
Pour faire cela, lorsqu'on traite les propriétés d'un élément, on les stocke dans un tableau de propriétés. Ensuite, la classe HotbarNetwork intervient. C'est dans cette classe qu'on crée le menu permettant la personnalisation.
Le menu se présente ainsi :

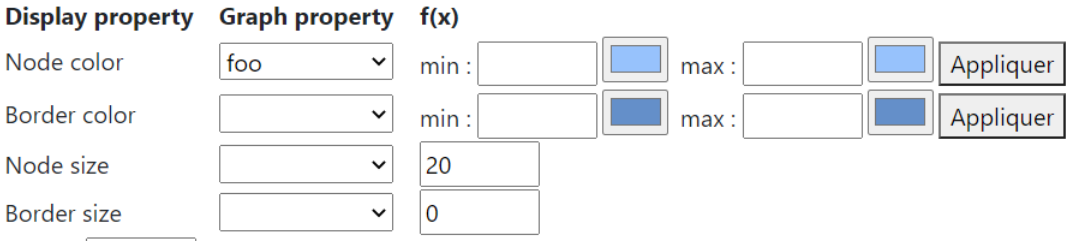

FIGURE 13 – Interface du menu

On choisit de personnaliser les éléments suivants :

- **Node size :** L'utilisateur entre une fonction f. Tous les noeuds ayant la propriété choisie auront désormais une taille de f(x) avec x la valeur de la propriété choisi
- **Border size :** De même, l'utilisateur entre une fonction f. Tous les noeuds ayant la propriété choisie auront désormais une border size de f(x) avec x la valeur de la propriété choisi
- **Node color :** L'utilisateur entre le minimum et le maximum de la valeur de la propriété choisie, ainsi qu'une couleur de départ du gradient et une couleur finale. Selon la valeur de la propriété, la couleur du noeud deviendra de la couleur associée à la valeur du gradient.
- **Border color :** De même, l'utilisateur entre le minimum et le maximum de la valeur de la propriété choisie, ainsi qu'une couleur de départ du gradient et une couleur finale. Selon la valeur de la propriété, la couleur du border du noeud deviendra de la couleur associée à la valeur du gradient.

7.4.5 Changement du label associé aux éléments

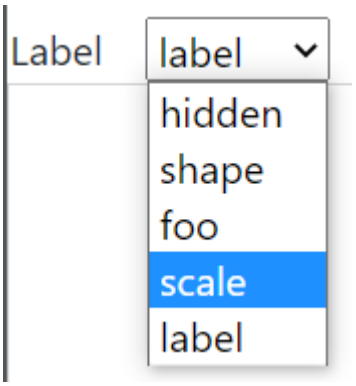

FIGURE 14 – Interface changement label

On peut changer le label de tous les éléments du graphe en fonction de la valeur de n'importe quelle propriété. Ainsi, si une information précise sur les éléments doit être vue rapidement sur le graphe, on a la possibilité de la voir directement.
Pour réaliser cela, on stocke le nom de toutes les propriétés qu'on a pu traiter et on les propose dans le select qu'on voit sur la figure 14. Lorsque l'utilisateur choisi une propriété, on boucle sur tous les éléments du graphe et on change le label associé sur le graphe Vis avec la valeur de la propriété si elle existe.

7.4.6 Amélioration de l'affichage

Plusieurs problèmes d'affichage se posent : tout d'abord, les noeuds vont très souvent en dehors de la fenêtre. Pour changer cela, on applique des changements au système physique de la visualisation de Vis.js. Tout d'abord, on applique une gravité centrale afin de faire converger le graphe au centre de la fenêtre. On limite aussi la force de répulsion des éléments.
Enfin, Vis.js propose une option censée permettre l'amélioration de la disposition des noeuds qui a bien marché dans notre cas, et qu'on laisse donc activé.

7.4.7 Changement dynamique du graphe

Le graphe peut être modifié dynamiquement et la visualisation doit permettre de voir ces changements s'effectuer en direct. Pour cela, on récupère les changement dans un autre JSON. La récupération de ce JSON se fera aussi en Ajax mais cette fois, on utilise un attribut qui permet de savoir le temps auquel le dernier changement a été pris en compte. Ainsi, on sauvegarde à chaque appel du JSON ce temps-là et on l'utilise pour le prochain appel afin de se limiter aux changements qui ne sont pas encore pris en compte.
Le JSON des changements se présente dans la forme suivante :


```
{
  "value": [
    {
      "vertexInfo": {
        "id": -6054516734464777405
      },
      "date": 1.6248642946210647E9,
      "type": "AddVertex"
    },
    {
      "elementID": -6054516734464777405,
      "date": 1.6248642946211312E9,
      "type": "RemoveVertex"
    },
    {
      "edgeInfo": {
        "from": -2374462833833014068,
        "to": -2374462833833014068,
        "id": -7225761638703636295
      },
      "date": 1.6248642976224418E9,
      "type": "AddEdge"
    }
  ],
  "type": "java.util.ArrayList"
}
```


FIGURE 15 – Allure JSON des changements

Ce JSON est composé d'une liste de changements. Ces changements peuvent être de plusieurs types : "AddVertex" pour l'ajout d'un noeud, "AddEdge" pour l'ajout d'un lien, "RemoveVertex" pour la suppression d'un sommet et "RemoveEdge" pour la suppression d'un lien.
Dans tous les cas, on dispose des informations nécessaires à l'identification de l'élément (id pour un noeud et id, from, to pour un lien).
Enfin, pour chaque changement, on dispose de la date de celui-ci.
Ainsi, on fait un appel Ajax au JSON toutes les secondes, ensuite on boucle sur tous les changements et selon leurs types, on modifie le graphe en conséquence.

8 Présentation des résultats

Les résultats suivants sont partiels car ils ont été retranscrit deux semaines avant la fin du stage soit au trois quarts de la durée du stage.

L'interface se présente ainsi :

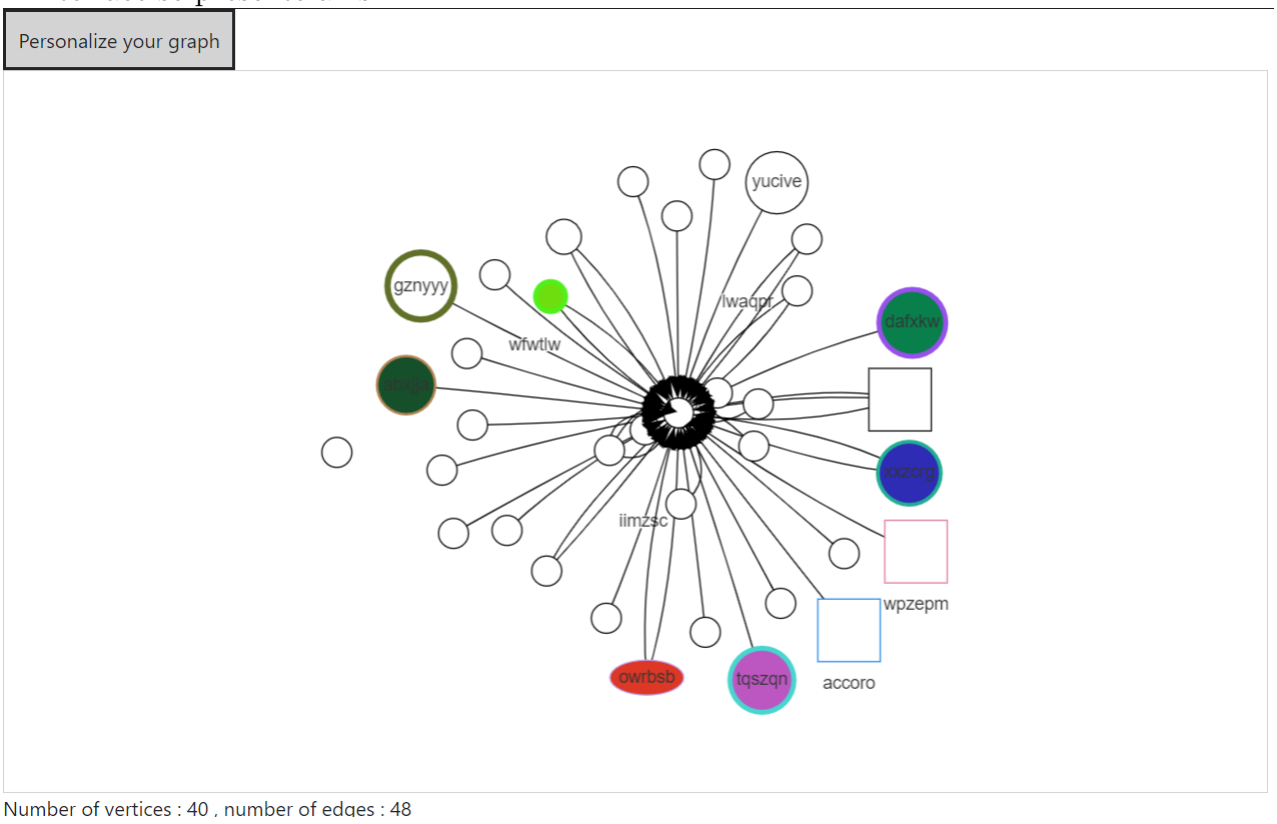

FIGURE 16 – Interface de l'application

En affichant le menu de personnalisation des propriétés :

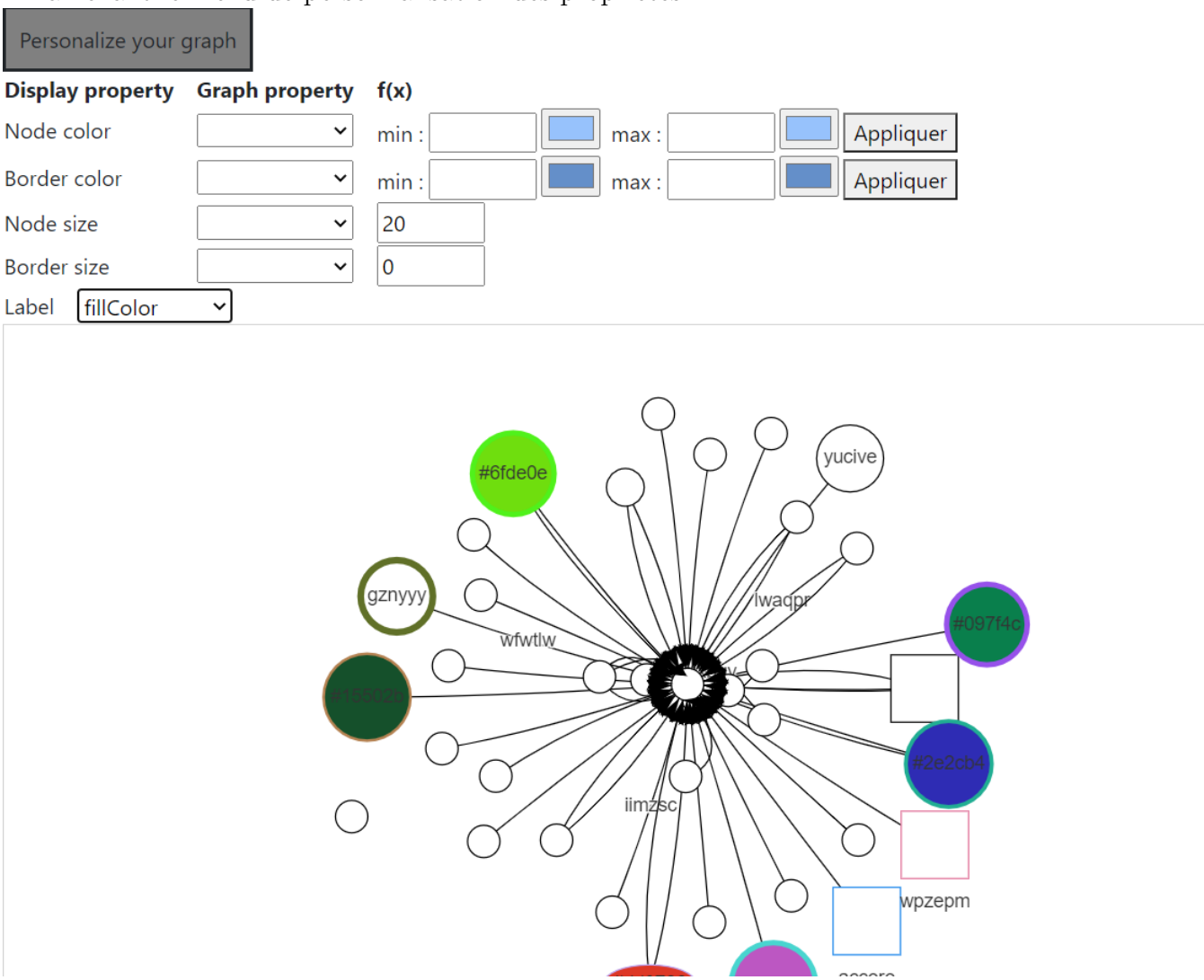

FIGURE 17 – Interface de l'application avec le menu

Le cahier des charges a été respecté car :

- on a la possibilité de visualiser un graphe à partir du JSON fourni
- on peut visualiser la personnalisation par les propriétés données
- on peut personnaliser l'effet de toutes propriétés des éléments du graphe
- on peut affecter au label de tous les éléments n'importe quel valeur de propriété

- on affiche dynamiquement tous les changements du graphe

9 Critique des résultats

Les résultats sont ceux attendus dans le cahier des charges.
On pourrait cependant améliorer l'application en :

- améliorant l'interface,
- en résolvant quelques petits bugs
- en donnant plus d'options de personnalisation à l'utilisateur. On pourra notamment donner toutes les possibilités d'affectation à l'utilisateur car pour l'instant, il n'y en a que quatre.
- en testant en profondeur l'appli
- en améliorant

Le maximum de tout cela sera fait durant les deux dernières semaines du stage.

10 Difficultés rencontrés

10.1 Reprise d'un code déjà existant

La première difficulté rencontré lors de ce stage a été la reprise du code déjà existant.
En effet, c'était une première pour moi de developper avec un existant et ça a été plus compliqué que prévu. La compréhension à tout d'abord été assez difficile car le code était très peu commenté et il n'est pas évident de se plonger dans un code Javascript aussi dense sans explications.
De plus, il m'a fallu du temps pour bien séparer le code à garder qui était utilisable dans le projet et le code non utilisable.
Malgré tout, je pense finalement avoir bien tiré parti de ce code existant pour gagner du temps dans mon developpement.

10.2 Gestion des propriétés des éléments du graphe

La seconde difficulté rencontré est la gestion des propriétés des éléments du graphe fourni.
Lorsque la propriété et sa valeur sont gérée nativement par Vis.js, il n'y a pas de soucis, il suffit de faire le changement sur le graphe. Seulement, plusieurs

cas problématiques se posent. Tout d'abord, lorsque la propriété n'est pas reconnu. Pour résoudre cela, j'ai mis en place le système de personnalisation de l'effet de la propriété sur le graphe.
Un autre cas beaucoup moins facile à gérer est lorsque la valeur associée à la propriété n'est pas reconnu. Par exemple, Vis.js ne gère qu'un nombre limité de forme de noeud et dans le cas où la valeur de la forme du noeud est "trapèze" par exemple, on ne peut pas utiliser cela. J'ai tout d'abord pensé à utiliser d'autres formes ressemblantes par exemple pour ce cas mais il n'est au final pas possible de couvrir toutes les possibilités de formes ou de nuances de couleurs par exemple.
Ainsi, j'ai décidé d'utiliser des valeurs par défauts pour toutes les valeurs non reconnues. Il n'est cependant pas idiot de couvrir le cas des valeurs non reconnues par Vis.js mais qui peuvent être souvent utilisé comme par exemple la forme "rectangle".

11 Conclusion

Pour conclure, ce stage a été pour moi très enrichissant dans plusieurs domaines.

Tout d'abord, j'ai découvert le milieu de la recherche qui ne m'était pas du tout familier. J'ai appris beaucoup de choses sur le travail dans un centre de recherche que je ne soupçonnait pas du tout. J'ai par exemple pu découvrir la liberté et la créativité dont les chercheurs ont l'occasion de faire preuve, ce qui, je pense n'est pas aussi commun dans une entreprise privé. Je tiens ainsi à remercier une nouvelle fois M. Luc Hogie, mon tuteur à l'INRIA ainsi que toutes les personnes rencontrées à l'INRIA qui m'auront fait découvrir tout cela malgré le fait que le stage s'est déroulé en majeure partie en télétravail.

J'ai de plus eu l'occasion de faire face à des situations que je rencontrerai tout au long de ma vie professionnelle et dont l'expérience que j'ai eu durant ce stage m'aidera à bien appréhender celles-ci.

Enfin, j'ai eu l'occasion de produire une application qui, je l'espère, servira à de nombreuses personnes au sein de l'INRIA et en dehors.

Table des figures