BOONE Kévin
Informatique 2

RAPPORT DE STAGE

Réalisation d'un outil Web de visualisation des différences entre deux versions de la base de données Simbad

1 avril 2019 / 7 juin 2019

OBSERVATOIRE ASTRONOMIQUE DE STRASBOURG - 2018/2019

Observatoire astronomique de Strasbourg | ObAS

Tuteurs en entreprise :
Mme. OBERTO Anaïs
&
M. MANTELET Grégory

Tuteur pédagogique :
M. DIEUDONNÉ Stéphane

Établissement d'accueil :
Observatoire astronomique de Strasbourg
CDS (Centre de Données astronomiques de Strasbourg)

Établissement :
IUT Saint-Dié-des-Vosges

UNIVERSITÉ DE LORRAINE | IUT Saint-Dié-des-Vosges CDS

Remerciements

Tout d'abord, je tiens à remercier mon maître de stage, M. SCHAAFF André, ingénieur de recherche à l'Observatoire Astronomique de Strasbourg, pour m'avoir accompagné et guidé avec bienveillance tout au long de ce stage sein de l'établissement. Par la même occasion, je remercie Mme. OBERTO Anaïs et M. MANTELET Grégory, ingénieurs de recherche à l'Observatoire Astronomique de Strasbourg, mes deux co-encadrants, dont leurs indications et précieux conseils m'ont permis de progresser au mieux durant ce stage. De plus, je tiens à remercier Mme. BROUTY Marianne, documentaliste, représentante de l'ensemble des documentalistes, pour m'avoir conseiller au fil de mon projet sur les diverses fonctionnalités et l'apparence de cet outil Web.

Je suis également reconnaissant envers M. DUC Pierre-Alain, directeur de l'Observatoire Astronomique de Strasbourg, ainsi qu'envers M. ALLEN Mark, directeur du Centre de Données astronomiques de Strasbourg, pour m'avoir permis d'effectuer mon stage dans ce lieu si merveilleux et si proche des étoiles, et pour m'avoir permis d'assister à tous les événements organisés pour les équipes de l'établissement (visites, séance de planétarium, réunions, cafés, séminaires, …).

De manière générale, j'adresse toute ma gratitude à l'ensemble des employés de l'Observatoire pour leur accueil chaleureux, ainsi que pour leur gentillesse, leur accompagnement et leur bienveillance tout au long de mon stage, faisant ainsi de ma première expérience professionnelle un souvenir à la fois marquant et agréable.

Enfin, je tiens à remercier toute l'équipe enseignante de l'IUT Informatique de Saint-Dié-des-Vosges pour leur accompagnement et leur dévotion au cours de ces deux années de DUT et plus particulièrement M. DIEUDONNÉ, enseignant, responsable des stages et tuteurs référent au sein de l'IUT, m'ayant permis de réaliser ce stage dans l'Observatoire, et ainsi avoir indirectement contribué à la bonne réalisation de ce stage.

Table des matières

Résumé

Afin de valider mon diplôme universitaire, j'ai décidé d'effectuer un stage de 10 semaines à l'Observatoire Astronomique de Strasbourg plus particulièrement au sein de l'équipe du Centre de Données astronomiques de Strasbourg. J'ai pu découvrir le monde réel de l'astronomie autre que par Internet, ainsi que le travail dans le monde de l'entreprise. Je suis en effet grandement attiré depuis des années par l'astronomie, c'est pourquoi ce stage fut pour moi une aubaine, principalement par la réalisation de ce projet en Java.

J'ai réalisé un site Web (interface cliente et serveur) de comparaison de données astronomique. La partie cliente est composée d'un affichage web en HTML et CSS, ainsi que du code JavaScript permettant de lire les données retournées par notre serveur sur la page Web. La partie serveur consiste quant à elle à effectuer les traitements de comparaison en Java, et de retourner le résultat au format JSON à l'aide d'une Servlet. J'ai ainsi donc dû me familiariser avec de nouvelles classes Java, une librairie Java permettant d'effectuer des comparaisons entre des chaînes de caractères, et la librairie JQuery en JavaScript. Le serveur Apache Tomcat a aussi permis la communication entre la partie serveur réalisée en Java, et la partie cliente en HTML.

Cette expérience m'a beaucoup appris, tant d'un point de vue personnel que professionnel. J'ai ainsi pu consolider et améliorer mes compétences dans divers langages de programmation enseignés à l'IUT. Durant ce stage, j'ai pu acquérir un certain nombre de compétences et de connaissances, aussi bien en informatique qu'en astronomie. Ainsi, travailler dans cet environnement professionnel a été pour moi une confirmation de mes envies : ce lieu et ce domaine me passionnent toujours autant.

Summary

In order to validate my university degree, I decided to do a 10-weeks internship at the Astronomical Observatory of Strasbourg, most particularly within the team of the CDS. I had the opportunity to discover the real world of astronomy other than by the Internet, as well as working in the corporate world. Indeed, I have been attracted for years by astronomy, that is why this internship was for me a boon, by the realization of this project mainly in Java in this environment.

I realized a website (client interface and server) astronomical data comparison. The client part consists of a web display in HTML and CSS, as well as JavaScript code to read the data returned by our server on the web page. The server part consists of performing comparison processing in Java, and returning the result in JSON format using a Servlet. So I had to familiarize myself with new Java classes, a Java library to perform comparisons between strings, and the JQuery and FLOT (consist of displaying a diagram) libraries in JavaScript. The Apache Tomcat server also allowed communication between the server part made in Java, and the client part in HTML.

This experience taught me a lot, both from a personal and a professional point of view. I was able to consolidate and improve my skills in various programming languages taught at the IUT. During this internship, I was able to acquire a certain number of skills and knowledge, both in computer science and astronomy. Thus, working in this professional environment was for me a confirmation of my desires: this place and this field always fascinate me as much.

Introduction

Étudiant en 2ème année de DUT informatique à l'IUT de Saint-Dié-des-Vosges, j'ai réalisé un stage de fin d'études afin de valider mon diplôme. Il s'agit d'un stage à triple enjeu : valider les compétences acquises durant les deux années de formation à l'IUT, approfondir et acquérir de nouvelles connaissances dans différents domaines, ainsi que découvrir le métier d'informaticien dans un milieu professionnel.

Ce stage a été réalisé à l'Observatoire Astronomique de Strasbourg et tout particulièrement au Centre de Données Astronomiques de Strasbourg. Il avait pour sujet de réaliser un outil web permettant de visualiser les différences entre deux versions de la base de données Simbad, principalement pour les documentalistes. Celles ci ajoutent de nouvelles données, obtenues grâce à des revues scientifiques par exemple, pour les intégrer dans les différentes bases de données. J'ai choisi ce stage car durant ma formation j'ai pu développer de nombreuses compétences, et un attachement pour le langage orienté objet Java. De plus, l'astronomie est un domaine qui me passionne tout particulièrement et j'aimerais en faire mon futur métier. C'est donc pourquoi je me suis dirigé vers cette offre.

D'une durée de 10 semaines, mon stage a consisté à développer une interface web permettant d'afficher les informations d'un objet céleste, et ainsi pouvoir comparer différentes versions de sauvegarde de cet objet dans la base de données pour mettre en évidences les changements apportés chaque mois. Cette interface web a donc été réalisée en Java pour le serveur, et JavaScript pour récupérer et afficher les données côté client. Celle-ci est utilisée exclusivement par les documentalistes pour leur faciliter la comparaison d'objets célestes lors de la mise à jour quotidienne des différentes bases de données du CDS. Elle n'est donc accessible que sur le réseau de l'Observatoire.

Au cours de ce stage, j'ai écrit un rapport retraçant le contenu de celui ci. C'est pourquoi dans une première partie, j'introduirais l'établissement d'accueil. À la suite de cela, j'aborderais le cahier des charges m'ayant permis de mieux cerner les enjeux du projet. Concernant ma troisième partie, j'expliquerais la méthodologie de travail adoptée durant ce stage. Enfin, je développerais l'ensemble de mon travail au cours de ces dix semaines. Et pour finir, je terminerais ce rapport sur une conclusion personnelle.

Fiche de synthèse

- Raison sociale : Centre de Données astronomiques de Strasbourg
- Objet social : collecte, analyse et distribution de données
- Siège social : 11 rue de l'Université 67000 Strasbourg France
- Forme juridique et capital social : Unité mixte de recherche
- Effectif de l'entreprise : Environ 80 salariés (dont 34 pour le CDS)
- Chiffre d'affaires : Ne génère pas de bénéfices
- Nom du dirigeant : Docteur ALLEN Mark G.
- Numéro de téléphone : +33 3 68 85 24 75 / +33 3 68 85 24 17
- Messagerie : cds-question@unistra.fr
- SIRET : 13000545700010
- NAF : Enseignement supérieur (8542Z)

PARTIE 1 : L'Observatoire

Présentation

L'Observatoire a été fondé en 1881 alors que l'Alsace et la Moselle étaient encore sous occupation allemande après la guerre franco-prussienne de 1870. À la fin de cette guerre, il fut décidé que Strasbourg représenterait la puissance. C'est ainsi que l'Observatoire fut construit, de même que le jardin botanique et une université. L'Observatoire se compose de 3 bâtiments. Le premier bâtiment comporte deux lunettes méridiennes sous deux coupoles différentes et abrite une bibliothèque contenant de nombreuses archives astronomiques. Le deuxième bâtiment contient de nombreux bureaux et les appartements du directeur général des services de l'Université. Et enfin, le troisième bâtiment est l'édifice le plus important de l'Observatoire puisqu'il contient la lunette appelée « Grand Réfracteur ».

Il s'agit d'une lunette astronomique de 48,7 cm d'ouverture et de 7 m de focale, elle est située sous une Grande Coupole. Faite en fer, la coupole pèse 34 tonnes pour 9,2 m de diamètre, qui assure la protection de la lunette. Celle-ci fut développée en 1877 et était la plus grande d'Europe. À l'heure actuelle, elle reste tout de même la 3ème plus grande de France. Elle a été utilisée pour de nombreuses observations notamment des comètes, d'étoiles variables (étoiles dont la luminosité varie), des nébuleuses, du système solaire, … Cependant, avec la pollution lumineuse de la ville de Strasbourg, les observations deviennent de plus en plus compliquées, c'est pourquoi elle n'est maintenant utilisée que pour des démonstrations, ou certaines observations du ciel proche.

L'Observatoire Astronomique de Strasbourg est une unité mixte de recherche dont les tutelles sont le CNRS (Centre National de Recherche Scientifique) et l'Université de Strasbourg. Il ne s'agit donc pas d'une entreprise à proprement parlé mais d'un établissement public, et ne génère donc pas de bénéfices.

Ressources humaines

Multiculturel, l'Observatoire Astronomique de Strasbourg compte aujourd'hui environ 80 personnes (ingénieurs, techniciens, et astronomes) travaillant dans ce lieu dans le but de contribuer aux progrès de la connaissance astronomique.
Au vu du nombre de salariés au sein de cet Observatoire, on pourrait croire que les différentes équipes et profils de métier ne se rencontrent que très peu, voir pas du tout, mais il n'en est rien.
Bien au contraire, l'une des opportunités de ce stage est de pouvoir travailler et côtoyer au quotidien des astronomes, des informaticiens ou encore des documentalistes provenant des quatre coins du monde.

À l'aide d'une communication interne adaptée à la situation, tout cela a pu être rendu possible. Principalement avec des méthodes d'interactions traditionnelles tels que les réunions de travail, les appels téléphoniques, les casiers personnels de rangements ou encore les messageries électroniques. Le centre de recherche dispose aussi de plateformes en ligne sur un réseau privée comme le "Twiki" ou encore GitLab permettant aux différents membres de communiquer et de partager un certain nombre de fichiers.

Il est important de noter que des séminaires sont organisés chaque vendredi afin de présenter le travail d'une ou plusieurs personnes, internes ou non à l'Observatoire. Cela permet de constater l'avancé des différents travaux présentés, mais aussi de pouvoir intervenir, partager et rencontrer de nouvelles personnes.

Équipes de recherches

Vers les années 1970, les télescopes commençaient à surpasser progressivement celui de l'Observatoire. C'est pourquoi il a fallu réorienter ses objectifs. La structure est divisée en 2 équipes de travail, le CDS (Centre de Données astronomiques de Strasbourg) et GALHECOS (Galaxies, High Energy, Cosmology, Compact Objects & Stars).

CDS

Figure 1: Logo du CDS

Un nouveau groupe de travail a ainsi vu le jour en 1972, le CDS, ayant différents objectifs :

- rassembler de nombreuses données astronomiques informatiquement.
- améliorer ces données par le biais de valeur ajoutée sous forme d'analyse de documents scientifiques.
- redistribuer l'ensemble des données à la communauté scientifique en effectuant des recherches en parallèle.

C'est au sein de cette structure que j'ai effectué mon stage de 10 semaines.

Le CDS est actuellement composé d'une trentaine de personnes (astronomes, documentalistes et informaticiens), qui étudient et distribuent à la communauté scientifique mondiale comme au grand public des données astronomiques, sous forme d'articles ou de bases de données. L'évolution de l'informatique, des techniques de stockage de données et du réseau a permis au CDS de stocker davantage de données au fil des années. Il devient vite une référence en matière de données astronomiques, en travaillant en collaboration avec des instituts renommés tels que l'ESA (European Space Agency) ou encore la NASA (National Aeronautics and Space Administration).

Le CDS a reçu en 2014 et 2016 le label DSA (Data Seal of Approval) garantissant que les moyens ont été mis en œuvre pour la préservation et la dissémination en accord avec les exigences du label.

GALHECOS

Un second groupe de travail a vu le jour en 2018, GALHECOS (Galaxies, High Energy, Cosmology, Compact Objects & Stars). Les membres de ce groupe étudient principalement les Galaxies (Trou noir central, population stellaire et leur dynamique, etc..) ainsi que les sources émettrices en rayon X et objets compacts (étoiles à neutrons, naines blanches, etc..).

Services proposés

Simbad

Figure 2: Logo de Simbad

SIMBAD (Set of Identifications, Measurements, and Bibliography for Astronomical Data) est une base de données d'objets astronomiques. Cette base de données recense diverses informations telles que la position dans le ciel, la magnitude,la vitesse de rotation, etc.. . L'ensemble des données est regroupé sur plus de 10 millions d'objets différents, avec plus de 25 millions d'identifiants différents et 300 000 références bibliographiques. Toutes ces données représentent un total de 60 Go de données stockées. Il s'agit d'une base de données ouverte au public via le site Web du CDS, visité en moyenne 500 000 fois par jour par des astronomes du monde entier.

VizieR

VIZIER est un service de catalogues astronomiques géré par le CDS. Il s'agit d'un ensemble de données scientifiques sous forme de tables regroupant diverses mesures sur de nombreux objets du ciel. Disponible depuis 1996 sur le site du CDS, il reste un outils libre, comme tous les outils proposés par le CDS. Il offre la possibilité aux utilisateurs d'effectuer des requêtes pour en extraire les données suivant les critères désirés. À sa création, la base de données proposait uniquement 680 catalogues représentant environ 3 Go de stockage. Aujourd'hui, elle possède plus de 40 000 tables, ce qui fait d'elle la bibliothèque de catalogues astronomiques la plus complète disponible sur Internet.

Figure 3: Logo de VizieR

Aladin

Figure 4: Logo de Aladin

ALADIN est une application réalisée en Java permettant de visualiser le ciel tel un atlas céleste. Cet outil permet de localiser, visualiser et comparer des données astronomiques (images, catalogues, cubes de données). Il a été rendu public en 1999, et est rapidement devenu l'outil de référence en matières d'observations visuelles du ciel. Les images sont puisées dans une base de données, il est cependant possible d'afficher des données supplémentaires par dessus depuis les bases de données de SIMBAD et de VIZIER, pour permettre d'afficher dans une interface plus attractive et visuelle les données de ces différentes bases de données.

Depuis quelques années, une version Web en JavaScript est également disponible : AladinLite.

Profil du dirigeant de l'Observatoire

Figure 5: Photo de Monsieur DUC Pierre-Alain

L'Observatoire Astronomique de Strasbourg est dirigé depuis 2017 par Monsieur DUC Pierre-Alain. Diplômé d'un doctorat en Astronomie et Astrophysique de l'Université de Paris VI, il effectue ses recherches au CNRS comme directeur de recherche

Profil du dirigeant du CDS

Figure 6: Photo de Monsieur ALLEN Mark G.

Le Centre de Données astronomiques de Strasbourg (CDS) est dirigé depuis Septembre 2015 par Monsieur ALLEN Mark. Diplômé d'un doctorat en Astronomie et Astrophysique de l'Université Nationale Australienne. Devenu directeur de recherche au CNRS, son travail porte essentiellement sur l'utilisation des e-infrastructures pour faire progresser l'astronomie

PARTIE 2 : Le cahier des charges

Contexte

De nombreuses observations et nouvelles mesures sont effectuées chaque jour, et leur publication ou mise à disposition de tous prend un certain temps. L'Observatoire et le Centre de Données astronomiques de Strasbourg ont tous deux pour mission de rassembler le plus de données possible via les revus scientifiques, les articles, ou tout autre document officiel. Pour mener à bien cela, le Centre de Données astronomiques de Strasbourg dispose de documentalistes, analysant chaque jour des documents astronomiques pour en extraire les informations pertinentes.

Cependant, analyser des centaines de pages de textes par jour n'est pas de tout repos, il s'agit d'un travail de longue recherche. Ainsi les documentalistes travaillent longuement sur l'extraction de données pour y remplir les différentes bases de données, mais aussi sur la comparaison des données trouvées avec celles déjà présentent dans les bases de données. Cette étape de comparaisons consiste aussi à rechercher d'anciennes informations qui auraient pu être supprimées par inadvertance, ou même d'essayer de comprendre la variation et la cohérence de certaines données. Tout cela constitue la principale valeur ajoutée du Centre de Données astronomiques de Strasbourg ; il s'agit d'un plus dans l'analyse des données pour leurs bases de données. Les documentalistes ne disposent pas d'outils adaptés à leurs besoins. Il leur faudrait un outil capable d'effectuer une comparaison de l'état d'un objet donné à deux dates différentes, et d'obtenir ensuite le résultat, avec un affichage permettant d'avoir les deux versions. Cela leur permettrait de visualiser l'ensemble des informations à comparer sur une seule page, et donc de mieux cerner le résultat final.

Existant

Les documentalistes utilisent donc un programme en ligne de commandes nommé « oldsim » et permettant de récupérer les informations d'objets célestes sous le format de fichier « parfile » (voir glossaire). Ainsi, les informations sont affichées dans un terminal Linux, laissant le soin aux documentalistes de rechercher les lignes désirées dans une suite de plusieurs fichiers affichés. Pour les objets célestes comprenant le plus d'informations, il devient vite difficile de comparer une même ligne à deux dates différentes. Plus le fichiers est volumineux, plus cela devient compliqué d'effectuer ce type de comparaison sur plusieurs fichier. Il nous arrive alors de vouloir afficher plusieurs fichiers à la suite. Cependant, le tampon devient alors vite plein, et nous perdons donc les informations précédentes.

Généralement, pour comparer deux versions différentes d'un fichier dans la base de données, les documentalistes utilisent plusieurs terminaux Linux. Cela leur permettant d'obtenir une meilleure lisibilité des informations pour faciliter la comparaison. Il faut cependant la place suffisante pour pouvoir afficher plusieurs terminaux sur un même écran, au détriment d'une meilleure lecture des données.

Besoins

Principalement dû au travail fastidieux de comparaison, il a donc été proposé de développer un outil permettant d'afficher deux versions différentes de la base de données Simbad, dans le but de faciliter la lisibilité des informations. Celui ci se présentera sous la forme d'une application Web, accessible dans le réseau interne de l'Observatoire par les documentalistes. Le tout devra être simple à prendre en main, avec une interface permettant de modifier rapidement et simplement les informations des dates et du nom de l'objet céleste.

Principales missions à réaliser

Au seins de ce stage, j'ai été amené à réaliser une application Web, guidé par mes tuteurs et leurs précieux conseils : ils m'ont conseillé au moyen d'une liste de missions évolutives au fil des semaines.

Dans un premier temps, j'ai commencé par me familiariser avec une librairie Java appelée diffutils, permettant d'indiquer des modifications dans une liste de chaînes de caractères. Il s'agit d'une des étapes principales à la réalisation de cette application Web, puisqu'elle consiste à gérer le marquage des zones différentes entre deux listes.
Dans un second temps, j'ai dû commencer à me familiariser avec la notion de servlets. Il s'agit de classes Java permettant de répondre à des requêtes par HTTP, à travers un serveur d'application Web (Tomcat dans mon cas). Tout ceci m'a permis de réaliser les interactions entre mes pages Web et mon programme Java principal me servant à gérer les comparaisons.
Finalement, j'ai dû approfondir mon enseignement du JavaScript, ainsi que la librairie JQuery et la méthode de gestion de requêtes HTTP permise par AJAX. Cela m'a permis d'effectuer le transfert d'informations entre le fichier Java et la page Web par le biais de requêtes AJAX et d'objets JSON.

Contraintes

Plusieurs aspects sont cruciaux dans la réalisation de cette mission et doivent être respectés impérativement.

Le premier concerne la véracité des informations affichées, c'est à dire qu'il ne faut pas que les informations retournées par mon application soit fausses. Il faut que le contenu du fichier soit exact, partout dans l'application. Si le fichier est affiché en entier, il est facile de vérifier sa création à partir de la date indiquée en début de document. Cependant, si nous n'affichons que certaines données, il devient vite difficile de vérifier s'il s'agit bien du bon résultat. Mais si les indications de différences lors d'une comparaison s'avèrent parfois inexactes, cela reste parfaitement lisible, et il devient facile de discerner les erreurs dues à la recherche.

Le deuxième point crucial se situe dans l'aspect ergonomique. Il ne faut pas que la page Web soit trop complexe pour ne pas perturber les habitudes des documentalistes. Ainsi, l'interface doit être facile à prendre en main par tout type de personne. Cette interface se chargera d'afficher l'ensemble des données, et ne pas se restreindre à une partie. Car il arrive que les documentalistes veulent la totalité des données, et pas seulement certaines lignes, cela leur permet de contextualiser leurs origines. De plus, l'utilité des différents champs à remplir doit être suffisamment intuitive pour qu'une personne découvrant ce projet soit capable d'utiliser cette application, sans explications préalables. Et pour finir, le temps d'exécution d'une requête doit se faire en un temps raisonnable. Étant donnée la taille de la base de données, et le nombre d'informations que possèdent certains objets célestes, il est inévitable que certaines requêtes prendront du temps à l'exécution. Cependant, il convient d'optimiser au mieux le programme, et ne pas faire de calculs ou de requêtes à la base de données de façon inutile.

En troisième point, nous pouvons aborder l'intégration de notre projet avec les logiciels déjà présents sur les serveurs. Il s'agit principalement de créer une application portable, pour qu'elle puisse être installée à n'importe quel emplacement, et sans dépendre d'autres logiciels pour fonctionner. Cela inclut, par exemple, le fait de pouvoir utiliser l'application sans devoir modifier le code source pour y indiquer le chemin de l'exécutable du programme oldsim déjà existant.

Et pour finir, cette application devra respecter les commentaires de Marianne Brouty, notre documentaliste référente, qui apportera un regard critique sur l'avancé de ce projet. Elle nous indiquera donc les fonctionnalités à intégrer, et divers changements ergonomiques et fonctionnels. L'avis de Marianne nous sera donc très précieux pour mieux appréhender les besoins réels des documentalistes.

Solutions possibles

Dans un premier temps, comme dit précédemment, la comparaison de deux fichiers peut se faire de plusieurs manières : en comparant l'ensemble du fichier sous forme de chaîne de caractères, ou en le découpant sous forme d'objets Java, permettant ainsi d'y effectuer la recherche de différences sur de plus petites parties de texte. La première est celle privilégiée dans un premier temps, principalement pour apprendre à utiliser correctement la librairie diffutils, mais aussi puisqu'elle nécessite moins de temps d'implémentation. En effet, il nous suffit juste de récupérer l'ensemble des lignes d'un fichier que nous pouvons ensuite stocker dans une liste de chaînes de caractères, alors que la seconde méthode nous oblige à découper le fichier sous forme d'objets avant de les traiter par la suite.

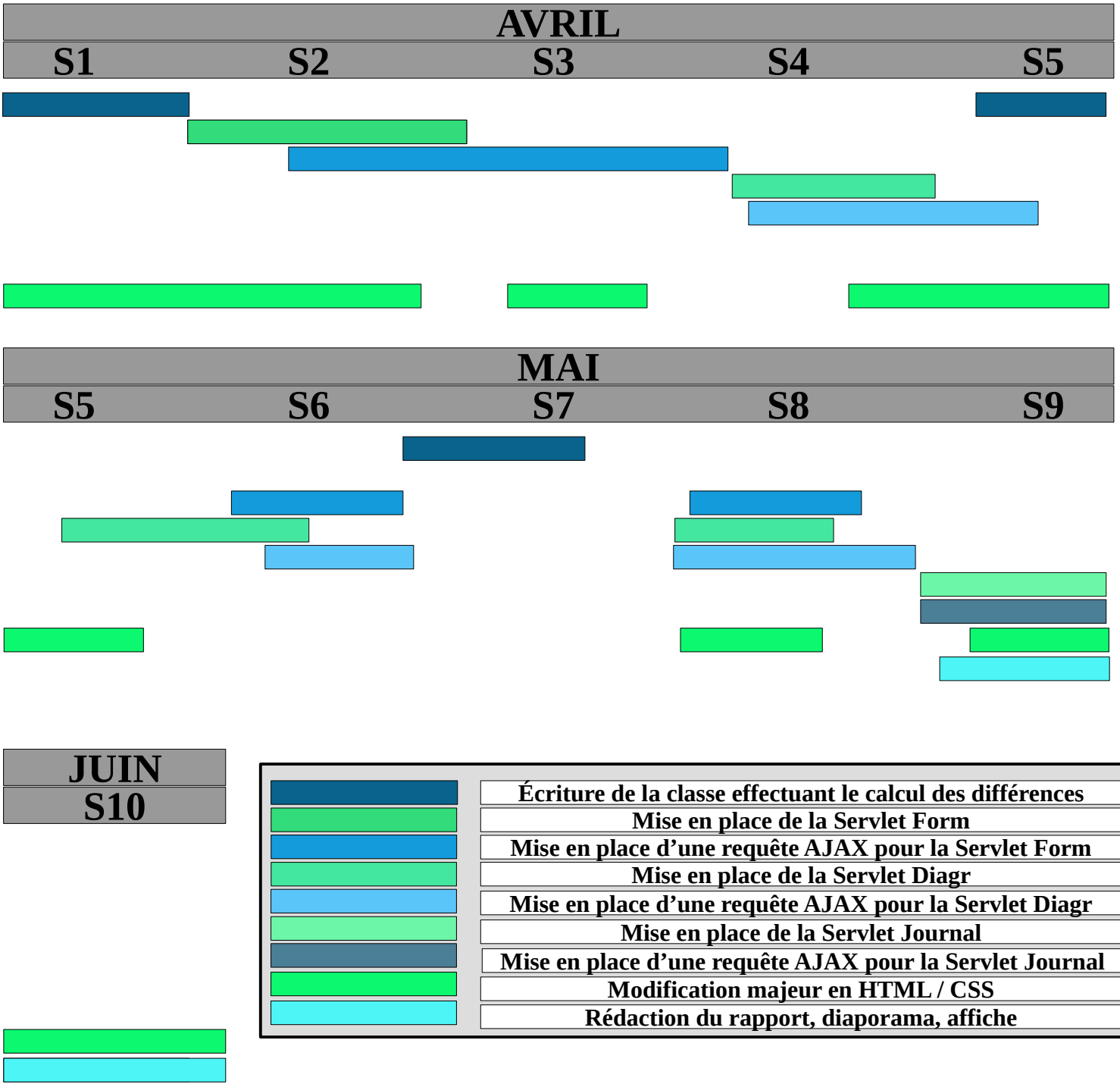
Ensuite, concernant la véracité des informations, la mise en place de tests unitaires Java (Junit) permet de vérifier le bon fonctionnement du programme, ainsi que les informations qui en ressortent pour ne pas fournir de mauvais résultats. Grâce à cela, nous pouvons vérifier qu'une fois la comparaison terminée, les balises entourant les modifications soient placées aux bons endroits, et sur les bonnes lignes. Et dans un second temps, nous allons tester le bon déclenchement des erreurs pouvant être causées lors de l'utilisation de notre application.

En plus de cela, les conseils de Marianne et de mes tuteurs m'ont permis d'évaluer au mieux les critères d'ergonomie à prendre en compte, c'est à dire, éviter une même couleur pour mettre en évidence plusieurs informations n'ayant pas de lien direct. En plus de cela, l'agencement de la page doit se faire avec attention afin d'avoir à la fois un environnement aéré et compact, pour ne pas trop s'éparpiller mais ne pas se sentir oppressé non plus. Ainsi, le haut de la page doit être dédié aux différents paramètres de recherche, tels que le nom de l'objet céleste et les dates de versions à comparer. En complément à cela, les deux formulaires permettant de sélectionner deux dates différentes sont séparés par un diagramme récapitulatif du nombre de différences par mois pour la période sélectionnée. Outre le complément d'informations, cela permet de marquer une séparation visuelle entre les deux fichiers.

Pour finir, afin d'effectuer nos différents calculs, nous utilisons plusieurs requêtes AJAX permettant de mieux segmenter les utilisations, et donc de créer des requêtes spécifiques à chaque problème. De plus, cela nous permet d'effectuer plusieurs calculs en parallèle. Par exemple, sur la page principale de comparaison, nous pouvons rechercher les différences entre deux fichiers, et en même temps générer un diagramme du nombre de différences entre ces deux périodes de temps. Segmenter son code possède de nombreux avantages. L'un d'entre eux est un gain de temps de calcul, et donc un gain de temps d'affichage de la page ; cela offre une meilleure expérience utilisateur grâce à une plus grande réactivité. Mais il s'agit avant tout d'une bonne pratique de programmation, permettant d'améliorer la lisibilité et la maintenabilité du code.

PARTIE 3 : La méthodologie

Calendrier



Environnement de travail

Ce stage a été effectué dans l'Observatoire astronomique de Strasbourg, plus particulièrement dans sa bibliothèque. Dans celle ci, nous étions six stagiaires, quatre personnes au rez-de-chaussé, et deux, dont je faisais partie, à l'étage. Nous étions dans un environnement très calme, peu de personnes hormis nos tuteurs venaient dans cette partie de l'Observatoire. Par chance, le bureau d'un de mes tuteurs était à la sortie de la bibliothèque, cela fut donc très facile d'aller le voir en cas de problème.

Cependant, ils venaient aussi d'eux même. Généralement plusieurs fois par semaine, nous pouvions faire le point sur l'avancé du projet, répondre à d'éventuelles questions, mais aussi m'assigner les tâches futures à réaliser ou modifier. Chaque jour, je pouvais travailler à mon rythme, et faire le point en fin de journée de mon avancement. Les visites se faisaient principalement au alentours de 15h, peu de temps avant que je ne quitte mon lieu de travail. Il me restait suffisamment de temps pour effectuer un compte rendu de leur visite, pour me remémorer le lendemain les tâches à réaliser et leurs conseils.

Concernant les horaires de travail, nous devions être présents dans l'établissement de 10h à 12h et de 14h à 16h. Ainsi, je travaillais généralement de 8h à 16h20, pour être en accord avec les horaires de départ du train, et les contraintes de l'Observatoire. De plus, cela me permettait de répartir équitablement mes heures de travail entre le matin et l'après-midi.

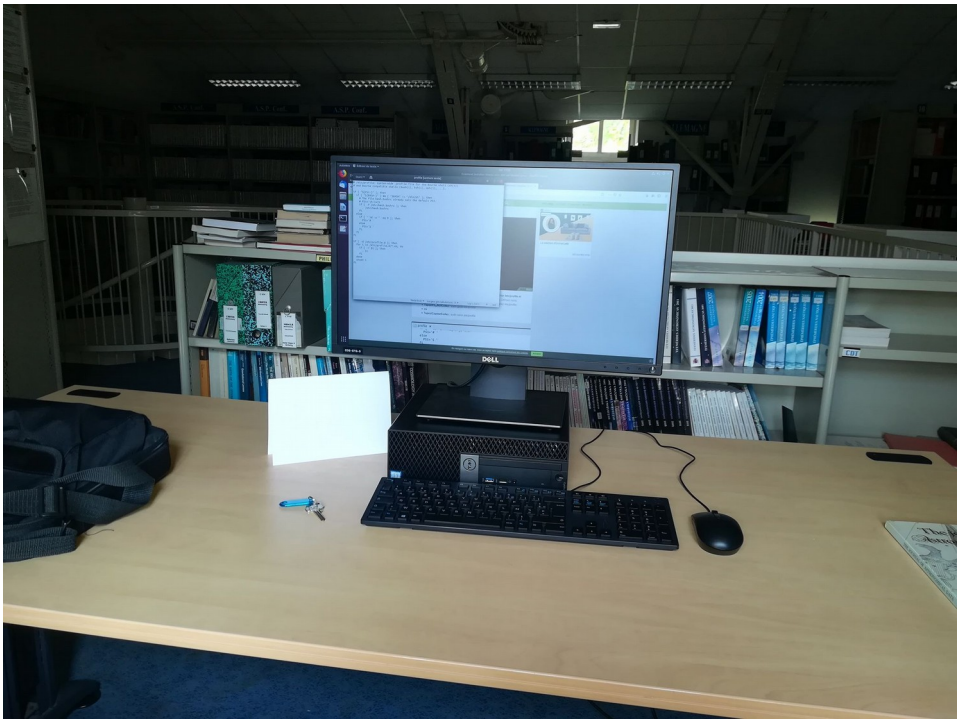


Figure 7: Mon bureau de travail

Logiciels et outils en ligne utilisés

UBUNTU

J'ai effectué l'intégralité de mon stage sur un poste de travail sous Linux Ubuntu, sur lequel je possédais les droits administrateurs. Ainsi, j'ai eu une grande liberté concernant ma façon de travailler, en pouvant installer les logiciels que je souhaitais pour ne pas me sentir dépaysé. Étant un grand utilisateur de Windows 10, cela m'a également permis de parfaire ma connaisse du système d'exploitation Linux.

ECLIPSE

Le logiciel Eclipse est un environnement de développement permettant de réaliser des projets informatiques dans de nombreux langages. Il s'agit d'un logiciel gratuit, open source, disponible sur Linux. Étant l'environnement de développement principalement utilisé à l'IUT, je n'ai pas été perdu lors de mon travail à l'Observatoire. Il m'a permis de développer mon projet web, et de l'exécuter en local sur mon ordinateur grâce à un plugin intégrant directement Tomcat dans Eclipse.

TWIKI

Il s'agit d'une plate-forme de travail collaboratif utilisée par les différents membres du Centre de Données astronomiques de Strasbourg. Ainsi, chaque stagiaire possède sa propre page dédiée au suivis du stage. Chaque page a pour objectif de garder une trace écrite de notre avancement tout au long du stage , mais aussi des différentes difficultés rencontrées, de nos tâches à effectuer, ou tout autre information pertinente sur l'avancé de ce stage.

GITLAB

GitLab est un outil de partage en ligne, ayant pour principale fonction le stockage de code informatique. Ainsi, celui-ci peut-être récupéré par n'importe qui ayant accès à notre dépôt. Le principal intérêt est le travail de groupe pour les projets et c'est dans ce but que nous l'utilisons. Nous pouvons donc y déposer le code de notre application pour que chacun puisse travailler avec une version constamment à jour entre chaque membre. Pour finir, GitLab offre la possibilité de faire de l'intégration continue, c'est à dire d'exécuter du code après y avoir déposé le moindre fichier. C'est très pratique dans le cas où nous souhaitons maintenir notre application à jour, sans devoir la re-déployer manuellement.

Langages de programmation utilisés

HTML

Le HTML n'est pas à proprement parler un langage de programmation, mais plutôt un langage dit de balisage conçu pour représenter les pages web. C'est en utilisant tout une architecture de balises que nous pouvons créer des pages web. Il s'agit donc du langage utilisé pour créer l'agencement de la page web de mon application.

CSS

Tout comme le HTML, le CSS n'est pas un langage de programmation, mais un langage de description et de formatage de style. Il décrit l'apparence des éléments de divers documents, principalement du HTML ou du XML par exemple. Ainsi, nous l'utiliserons pour l'aspect visuel de l'application, tel que le positionnement ou la coloration.

JavaScript

Il s'agit du langage de programmation web le plus utilisé. Il permet de rendre les pages dynamiques, mais aussi de gérer les interactions avec l'utilisateur. De plus, c'est un langage permettant d'effectuer des calculs aussi bien synchrones que asynchrones, c'est à dire qu'il peut ne pas attendre la fin totale d'une instruction pour continuer son exécution dans le programme. Il permet de gérer facilement les évènements déclenchés par l'utilisateur tels que des clics. En effet, cela permet de ne pas bloquer l'application tant que les résultats ne sont pas encore disponibles, et ainsi de pouvoir afficher des animations d'indication de progression par exemple. Dans notre cas, cela nous servira principalement au déclenchement de fonctions lors d'un clic sur des boutons, ou même d'appuis des touches du clavier.

JQuery

JQuery n'est pas un langage de programmation, mais une librairie du langage JavaScript. Elle facilite le parcours et la modification des éléments du DOM, et offre donc une écriture plus lisible à lire et à comprendre. Ainsi, les événements clavier ou souris sont plus faciles à gérer de part une écriture plus succincte, ainsi que des méthodes facilitant la gestion. De plus, manipuler des valeurs contenues dans des champs de la page HTML devient relativement simple.

Java

Il s'agit d'un langage de programmation orienté objet, portable sur de nombreux systèmes d'exploitations. Dans notre cas, il nous a permis d'effectuer les principales opérations de comparaisons, puis de retourner les résultats sous forme d'un fichier JSON. Ainsi, ils sont facilement exploitables par du code JavaScript.

XML

Très ressemblant au HTML, il s'agit là aussi d'un langage de balisage. Il est cependant plus polyvalent, et permet donc de définir sa propre grammaire au niveau des balises. Nous l'avons utilisé concernant la configuration de nos servlets.

PARTIE 4 : La mise en œuvre

Organisation de la page Web

Différents modes

Notre application se basant sur une interface pour les interactions utilisateur, il est donc nécessaire de présenter le corps de notre page. Cette page se compose tout d'abord de deux boutons radio permettant de changer de mode de présentation des résultats, et ainsi, de changer les informations sur la page. Ensuite, il y a un champs permettant d'accueillir le nom de l'objet céleste à rechercher, ainsi que deux boutons. Le premier permet d'effectuer la recherche, et le second vide entièrement la zone contenant le nom. Pour finir, une liste déroulante est visible à droite de la page, pour choisir les champs à rechercher dans un fichier (Elle n'est cependant pas fonctionnelle par

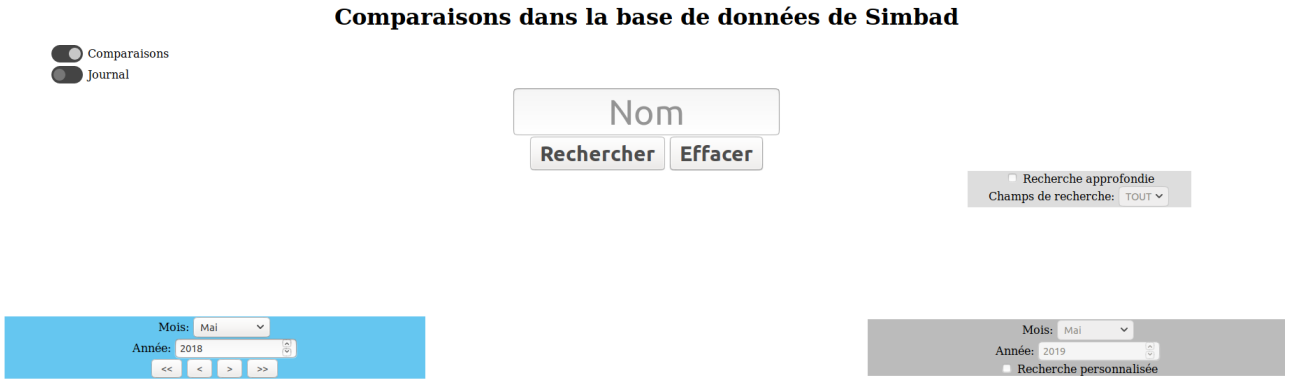


Figure 8: Page d'accueil de mon interface Web

manque de temps).

Mode de comparaison standard

Dans ce mode, la page comporte deux formulaires, permettant respectivement de choisir les deux dates avec lesquelles nous effectueront la comparaison. Après avoir choisies ces dates, nous pouvons effectuer la recherche. Un diagramme apparaît au centre de l'écran, entre les deux formulaires. Le diagramme représente le nombre de différences par mois ; par défaut seuls les 12 mois précédant la date choisie la plus récente sont affichées. Et concernant le résultat principal de la requête, nous obtenons le contenu des deux versions de l'objet céleste choisi. Dans chacune d'elles, les lignes surlignées correspondent aux lignes possédant des modifications. Les caractères différents sont quant à eux mis en évidence en les formatant en gras.

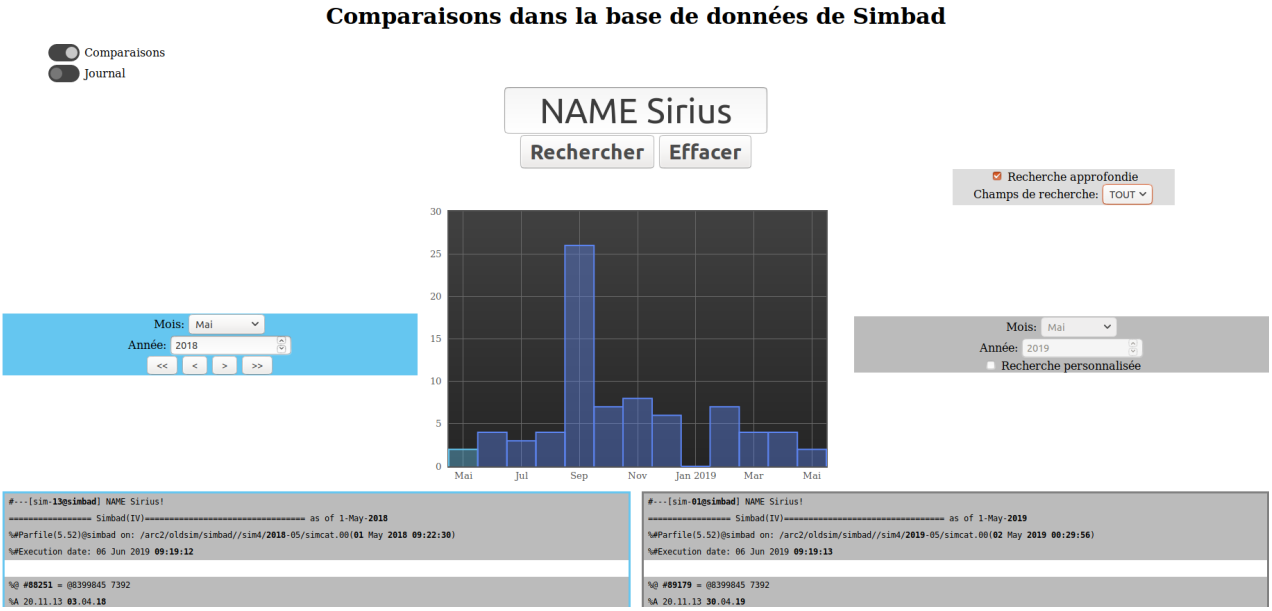


Figure 9: Aperçu de l'interface après une requête

Mode de comparaison affiché sous forme de journal

Dans ce mode, seules les lignes possédant des modifications sont affichées. Un tableau à deux colonnes est ainsi présenté. La colonne de gauche contient les mois, alors que la colonne de droite contient les lignes modifiées. Le code couleur adopté permet d'indiquer à l'utilisateur si la ligne est supprimée, ajoutée ou modifiée.

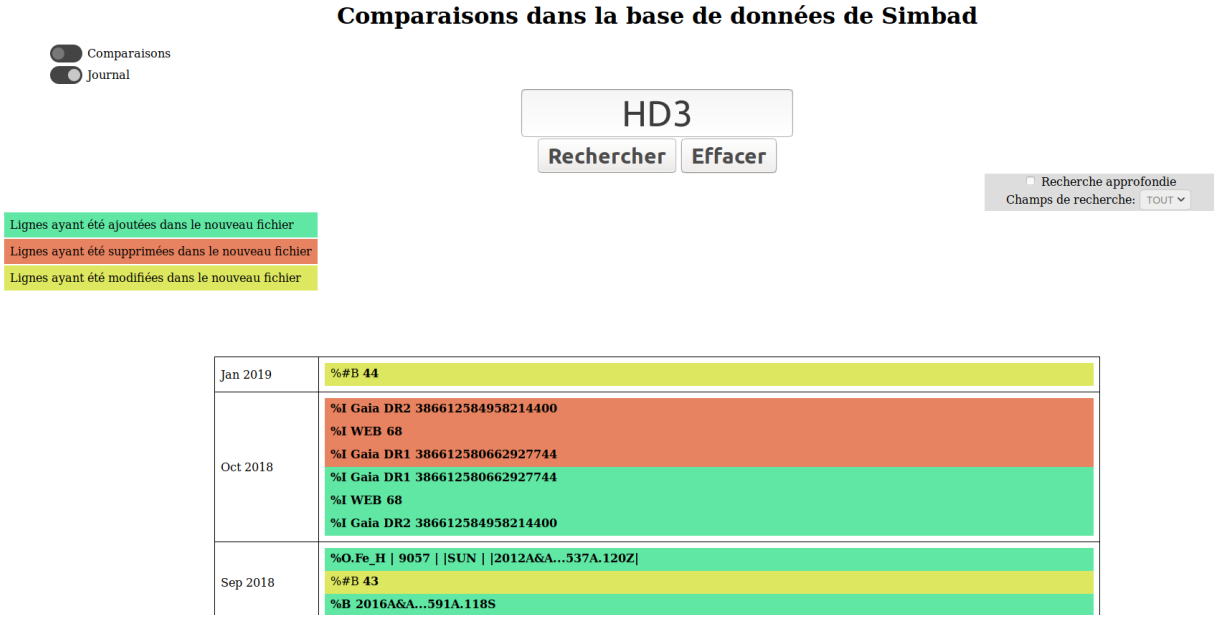


Figure 10: Aperçu du mode journal après une requête

Compréhension et utilisation de diffutils

Création d'un projet Eclipse Gradle

Durant ce stage, mes premières tâches étaient de réussir à afficher sous la forme de deux tableaux, les différences entre deux fichiers chargés en local. Pour ce faire, il m'a été proposé de travailler avec la librairie Java « diffutils », disponible sur GitHub.
Pour utiliser cette librairie, nous avons tout d'abord créé un projet Gradle sous Eclipse pour nous permettre d'inclure automatiquement la librairie (et toutes ses dépendances) sous la forme d'un JAR le plus facilement possible. Ce projet gradle aura une grande importance durant le projet. C'est grâce à lui que nous pourrons générer une archive WAR, et donc la déployer sur le serveur pour obtenir une application plus portable. En plus de cela, nous avons pu faire de l'intégration continue depuis GitLab, avec un fichier de configuration nommé .gitlab-ci.yml. Après chaque commit sur le projet du GitLab, une archive est générée, puis déployée sur un serveur.

Une fois cela effectué, il nous a fallu configurer ce projet, en modifiant le fichier build.gradle, généré à la création de notre projet.

```
apply plugin: 'war'
apply plugin: 'eclipse-wtp'

/* ************************************************************************
 * * DEFAULT TASK TO RUN (when running `gradle` with no parameter)        *
 * ************************************************************************ */

defaultTasks = ["war"]


/* ************************************************************************
 * * PROJECT                                                              *
 * ************************************************************************ */

description = "Navigate through Simbad history."

archivesBaseName = 'oldsim'
javadoc.title = archivesBaseName

version = '1.0.beta'
project.ext.implVersion = version+'.0'


/* ************************************************************************
 * * ECLIPSE-WTP                                                          *
 * ************************************************************************ */

eclipse {
  wtp {
  facet {
    facet name: 'jst.web', version: '3.0'
    facet name: 'java', version: '1.8'
  }
 }
}
```

Figure 11: Exemple de contenu du fichier build.gradle

Nous avons tout d'abord indiqué l'emplacement des fichiers sources du projet, et des différentes ressources nécessaires, tels que les fichiers à comparer en local. Une fois cela effectué, il nous a fallu préciser la version de Java utilisée, ici Java 12. Et pour finir, nous avons indiqué nos différentes dépendances, nécessaires au bon fonctionnement de l'application. C'est ainsi que nous avons pu spécifier notre librairie diffutils, ainsi qu'une librairie Servlet, indispensable dans notre cas à la création d'une application communiquant entre Java et une page Web. Cette dernière est une librairie distribuée par Oracle, et inclut toutes les classes nécessaires à l'écriture de Servlets.

Compréhension de la librairie diffutils

Après un certain temps passé à essayer de comprendre le fonctionnement de cette librairie, j'ai pu découvrir ce dont j'avais besoin, la méthode principale effectuant la recherche des différences.

```
//create a configured DiffRowGenerator
DiffRowGenerator generator = DiffRowGenerator.create()
        .showInlineDiffs(true)
        .inlineDiffByWord(true)
        .oldTag(f -> f ? "<span class=\"modified\">" : "</span>")
        .newTag(f -> f ? "<span class=\"modified\">" : "</span>")
        .build();

ArrayList<String> finalOld = new ArrayList<String>();
ArrayList<String> finalNew = new ArrayList<String>();

try {
    List<DiffRow> rows = generator.generateDiffRows(lStr1, lStr2);
```

Figure 12: Méthode permettant la recherche des différences

Grâce à cela, j'ai pu obtenir deux listes de chaînes de caractères, contenant chaque ligne de chaque fichier, avec leurs différences entourées de balises HTML. Il s'agit donc du cœur de l'application : l'identification des parties de chaque fichier à mettre en évidence.
Nous pouvons observer le fonctionnement de la création d'un nouvel Objet Java, un DiffRowGenerator obtenu par le biais de la librairie diffutils. Deux méthodes ayant permis sa création nous intéressent particulièrement :

- La méthode oldTag() permet d'ajouter les balises pour le premier fichier en paramètre.
- La méthode newTag() permet d'ajouter les balises pour le second fichier.

Nous pouvons donc effectuer différents traitements suivant le fichier. Nous pouvons même choisir la balise de début et de fin. Il faut cependant utiliser des fonctions anonymes, ou expressions lambda, de la classe Lambda en Java.

La classe Lambda Java

Les expressions lambda permettent l'implémentation d'une closure en Java, c'est à dire avoir la possibilité d'effectuer des références vers des méthodes. Il est donc possible de simplifier grandement le code en en réduisant le nombre de lignes. Dans notre exemple, nous avons utilisé une expression lambda testant la valeur de retour des fonctions, et ainsi savoir si celle-ci s'applique au début d'une modification, ou au contraire à la fin. De cette manière, nous pouvons donc personnaliser l'encadrement de chaque modification.

Voici le diagramme UML (de classe) de mes trois classes Servlet, ainsi que ma classe principale effectuant les opérations de différences.

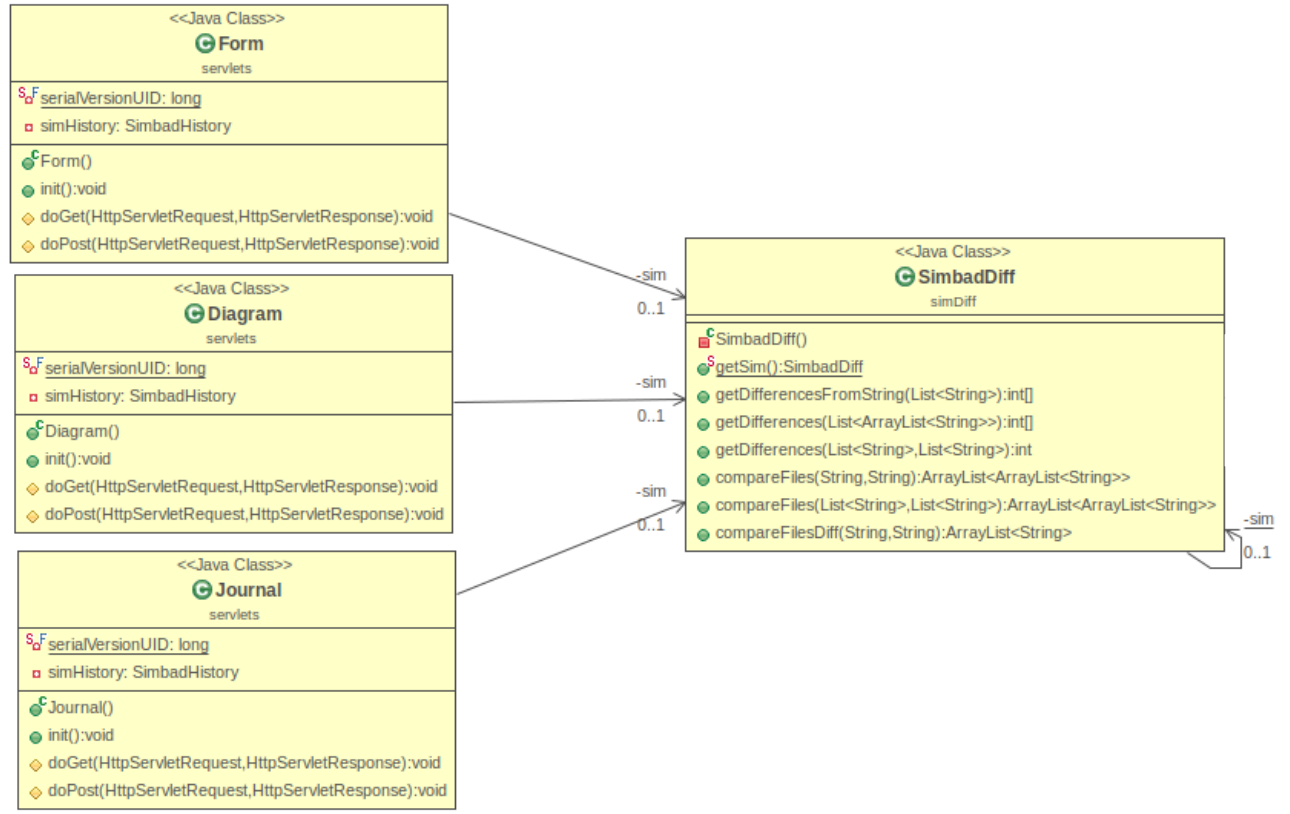


Figure 13: Diagramme de classe du Projet (sans les classes de test)

Contenu des fichiers retourné par l'API nous retournant le contenu des fichiers parfile

Dans un premier temps, les fichiers à comparer étaient enregistrés en local, et je devais donc les lire à partir d'un flux d'octets. Mais mes deux tuteurs, Anaïs et Grégory ont rapidement pris en charge le développement d'une API, me permettant d'effectuer les connexions à la base de données. Il a ainsi été possible d'obtenir le contenu de n'importe quel fichier à l'aide d'un nom d'objet, d'une année et d'un mois. Grâce à cela, j'ai pu élargir mes tests et vérifier le bon fonctionnement du programme sur divers objets, plus ou moins volumineux. Cependant, mon programme ne prenait pas encore en charge le résultat de retour de leur recherche, qui était une chaîne de caractères.

Surcharge des méthodes de comparaison pour utiliser l'API

Une fois l'API en main, j'ai adapté mon code pour qu'il prenne en charge le résultat retourné par l'API. Ainsi, je disposais de deux méthodes. Toujours celle prenant en paramètre les deux listes contenant chaque ligne des fichiers. J'ai cependant créé une nouvelle méthode prenant en paramètres deux chaînes de caractères représentant le contenu des deux fichiers sur lesquels effectuer la recherche de différences. Ainsi, seule ma méthode permettant de lire le flux d'octets a due être supprimée, car maintenant obsolète. Elle fût donc remplacée par celle ci, dans laquelle j'ai recréé deux listes de chaînes de caractères à l'aide d'une découpe des chaînes par le caractère d'échappement « \n » et de la méthode split() de la classe String.

Ajout de méthode de calcul du nombre de différences entre deux fichiers

Dans le but d'afficher un graphique du nombre de différences entre deux fichiers deux à deux sur une période donnée, nous avons ajouté plusieurs méthodes permettant de calculer ce nombre de différences.
Dans un premier temps, nous avons écrit une méthode permettant de calculer le nombre de différences entre deux fichiers. Ainsi, il nous suffisait de compter le nombre de fois où la balise de différence était écrite dans le fichier (sauf les 7 premières lignes étant donné qu'elles varient constamment d'un fichier à l'autre, car il s'agit de plusieurs lignes indiquant les dates du fichier, ou de son emplacement lors de la sauvegarde).
Une seconde méthode prenant en paramètre le contenu de plusieurs fichiers sous la forme de liste de lignes de chacun. Une fois cela effectué, celle-ci appelle la dernière méthode, qui effectue une boucle entre deux fichiers deux à deux et ainsi y calculer le nombre de différences. Une fois cela effectué, nous pouvons stocker le résultat final dans un tableau d'entiers, util pour notre diagramme.

Gestion des exceptions

En plus du programme, une classe de test JUnit a été réalisée pour permettre à l'application de ne retourner que des valeurs contrôlées. Des tests sur le calcul du nombre de différences et sur le retour d'un fichier dont les modifications sont balisées en HTML. La position des balises, ainsi que leur nom ont été vérifiés, pour ne pas obtenir de code HTML erroné.

Utilisation d'un Design Pattern Singleton

Pour finir, afin de garantir un unique accès à la base données et ainsi, n'avoir aucun conflit d'accès, j'ai mis en place un Design Pattern Singleton, prenant en charge le cas d'accès multiples à l'aide des threads, autorisant plusieurs requêtes simultanées à la base de données. Chaque thread attendra la fin de création de chaque instance de la classe, afin d'obtenir la même instance de l'objet.

```
// Attributs
/** Unique objet SimbadDiff pour les comparaisons dans la base de données*/
private static SimbadDiff sim;

// Méthodes
/** Construit un objet par defaut avec connection avec la base de données
 * @throws SimbadDbException */
@SuppressWarnings("unused")
private SimbadDiff() throws SimbadDbException {
    // Initialisation de Simbad
    Simbad simbad = new Simbad("/simbad.ini");
}

/** Construit un objet SimbadDiff en garantissant un singleton
 * @return Un objet singleton SimbadDiff
 * @throws SimbadDbException
 */
public static synchronized SimbadDiff getSim() throws SimbadDbException {
    if (sim == null)
        sim = new SimbadDiff();
    return sim;
}
```

Figure 14: Mise en place du Design Pattern Singleton

Transfert de données via la classe Servlet

Création d'une nouvelle Servlet

Avec la création d'une Servlet Java, nous avons décidé de faire de notre projet actuel, un projet Eclipse de type web dynamique. J'ai donc dû migrer le projet existant pour le transformer en projet web dynamique me permettant d'utiliser la classe Servlet. Une fois cela effectué, une nouvelle architecture de fichiers a alors été générée. De plus, un fichier de configuration nommé web.xml permet de déclarer nos nouvelles Servlets, ainsi que les éventuelles paramètres à leur donner.

Nous avons donc un nouveau fichier nommé web.xml nous permettant de déclarer nos nouvelles Servlets à utiliser. Tout au long de ce projet, j'ai créé trois Servlets, ayant chacune une utilité différente.

Valeur de retour des Servlets

Une fois en possession de notre page Web, nous pouvons nous demander comment envoyer les résultats des différents calculs par le biais des Servlets. Plusieurs possibilités s'offrent à nous.

Retourner une page HTML

Dans un premier temps, la solution la plus évidente et rapide à mettre en place consiste à retourner une page HTML. Ainsi il ne nous reste plus qu'à reconstruire une nouvelle page en y écrivant directement le résultat de notre requête. Cependant, il nous faut recréer la page à chaque requête, ce qui est très long à effectuer dans le cas de gros résultats à retourner. De plus, ce n'est pas très pratique d'obtenir une nouvelle page Web à chaque fois. C'est pourquoi nous avons opté pour la seconde option.

Retourner un fichier JSON

Cette option correspond d'avantage en terme de résultat produit. C'est à dire que nos servlets retournent un fichier JSON contenant des résultats associés à des clés.

Format d'affichage d'un fichier JSON :

```
{
   "oldVersion" : [
                     "id" : "ligne"
                          …
                  ],
 "newVersion" : [ … ]
}
```

Ainsi, pour retourner le contenu d'un fichier, nous associons celui ci au nom d'une clé de notre choix. Et pour pouvoir effectuer sa lecture, il suffit de lire les données contenues au niveau de cette clé. En plus d'être plus polyvalent en terme de contenu, il l'est aussi au niveau de sa hiérarchie, grâce à son système de clé et de valeur permettant de retourner différents types de valeurs. De plus, lire ce type de fichier n'impose pas le rechargement de la page, mais simplement une insertion dans les champs demandés directement dans le navigateur à l'aide de JavaScript.

Servlet pour la comparaison de deux fichiers

Comparaison effectuée une seule fois

La première Servlet que j'ai créé, nommée Form, m'a permis de récupérer deux versions mensuelles du contenu d'un objet astronomique, et d'identifier les différences. Pour effectuer cela, elle récupère les valeurs des champs de date contenues dans le code HTML. Cette action est réalisable par le biais des « id » que nous avons spécifié sur les éléments HTML correspondants. JQuery nous permet ensuite de récupérer ces valeurs pour les transmettre à notre Servlet, qui elle même effectue la requête à l'API. Une fois cette opération effectuée, nous avons créé des objets JSON à partir des classes Java JSONArray et JSONObject. Pour ce faire, chaque contenu de fichier sera attribué à une clé. Il s'agit d'un identificateur permettant de récupérer la valeur qu'elle indique, ici il y aura deux clés pour deux fichiers. Ainsi, il nous suffit juste de rechercher une clé pour avoir directement accès au premier ou au second fichier, suivant la demande. Cette opération s'effectue lors d'une demande de comparaison simple, c'est à dire en effectuant une recherche via le bouton de recherche ou un appui de la touche ENTRÉE sur la page Web. Cependant, cliquer sur les boutons « < » ou « > » effectue aussi cette recherche simple puisqu'elle nécessite uniquement le fichier précédant celui sélectionné dans le formulaire de gauche, en plus du fichier déjà sélectionné dans le formulaire de droite.

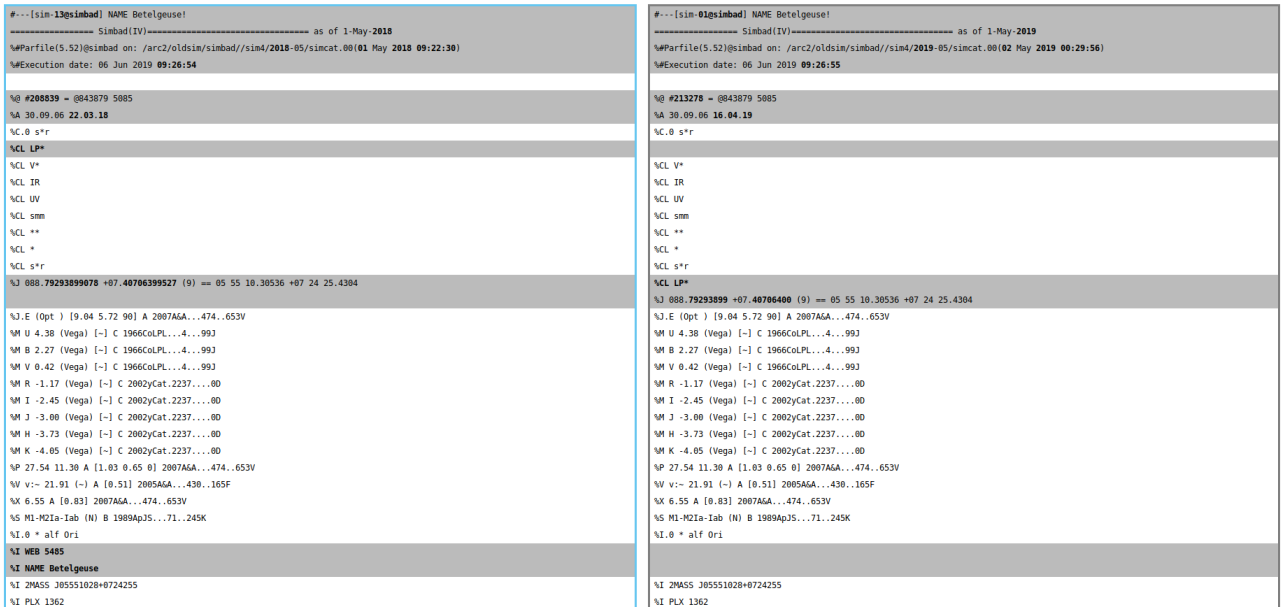
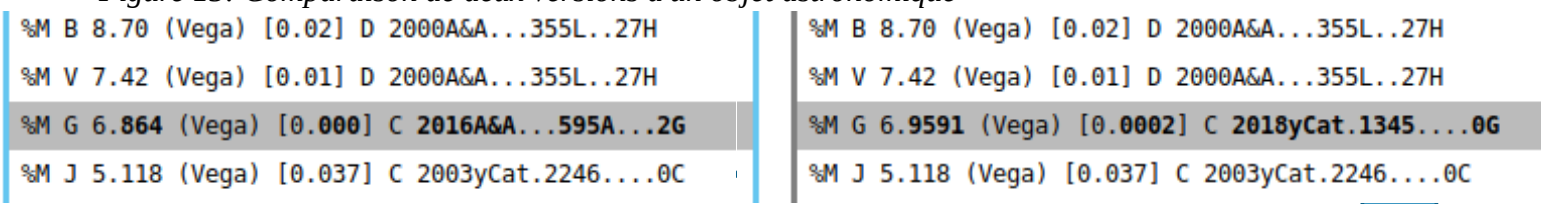


Figure 15: Comparaison de deux versions d'un objet astronomique



Comparaison effectuée sur douze fichiers consécutifs

Cette nouvelle fonctionnalité a été ajoutée en même temps que les boutons « << » et « >> » du formulaire. Lors d'un clic sur ceux ci, nous effectuons une comparaison deux à deux sur douze fichiers consécutifs. Étant donné que nous ne connaissons pas la date de création de l'objet astronomique recherché, il nous faut un nombre de recherches limités, pour ne pas effectuer des calculs indéfiniment. Nous effectuons donc la recherche de différences entre les douze fichiers. Une fois cette opération terminée, nous calculons le nombre de différences deux à deux entre tous les fichiers de notre liste. Si entre deux fichiers il n'existe aucune modification, nous vérifions la valeur suivante. Si nous arrivons à la fin du tableau et qu'il n'existe toujours pas de différence, nous allons considérer qu'il n'existe plus de différences pour cet objet, et donc nous ne continuons pas la recherche. Ainsi, nous ne retournons rien, exceptée une exception pour indiquer qu'aucune différence n'a été trouvée. Cependant, si lors du parcours nous rencontrons deux fichiers possédant des différences, alors nous retournons ce fichier sur la page Web.

Servlet pour la génération du diagramme

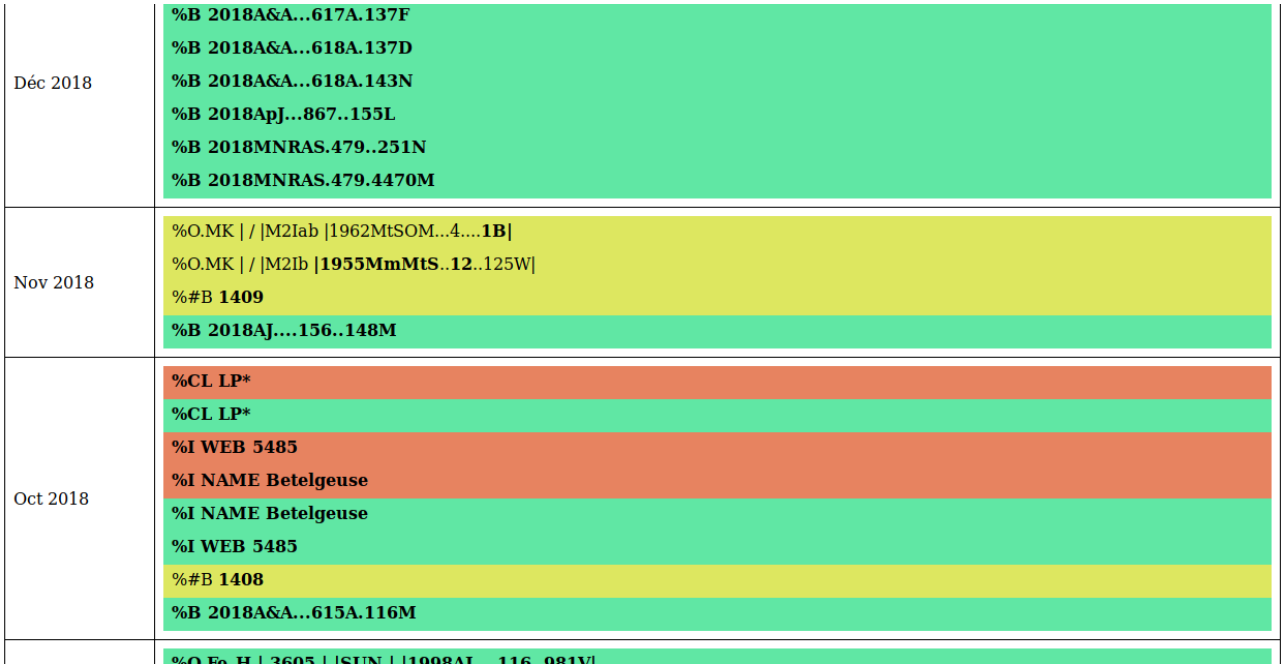


Figure 16: Exemple d'affichage du mode journal pour un objet astronomique

Calcul du nombre d'axes

Cette deuxième servlet est entièrement consacrée à l'envoi des données nécessaires à la bonne génération du diagramme. Dans le corps principal, nous commençons par remplir une liste des fichiers dont nous aurons besoin. Pour connaître ce nombre, il nous faut savoir les deux dates du formulaire. A partir de cela, nous voulons afficher en permanence un multiple de 12 mois sur le diagramme, en partant du mois du formulaire de droite et en remontant le temps. Ainsi, la date du formulaire de droite est toujours affichée, et la différence de temps entre le formulaire de droite et de gauche permet de savoir quel multiple de 12 mois afficher.

Remplissage du fichier JSON

Une fois l'ensemble des fichiers récupérés, il nous faut calculer le nombre de différences entre chaque fichier deux à deux. Pour finir, nous allons écrire manuellement notre fichier JSON

sans passer par des Objets. Il nous faudra donc parfaitement maîtriser la syntaxe des fichiers JSON, pour ne pas oublier de caractères, et bien hiérarchiser nos données de retour. Ainsi nous allons créer un champs contenant les valeurs du diagramme, composé d'un mois, et d'une valeur. Seul le mois suffit, puisque l'année est trouvable via le formulaire de droite et la position dans le diagramme.

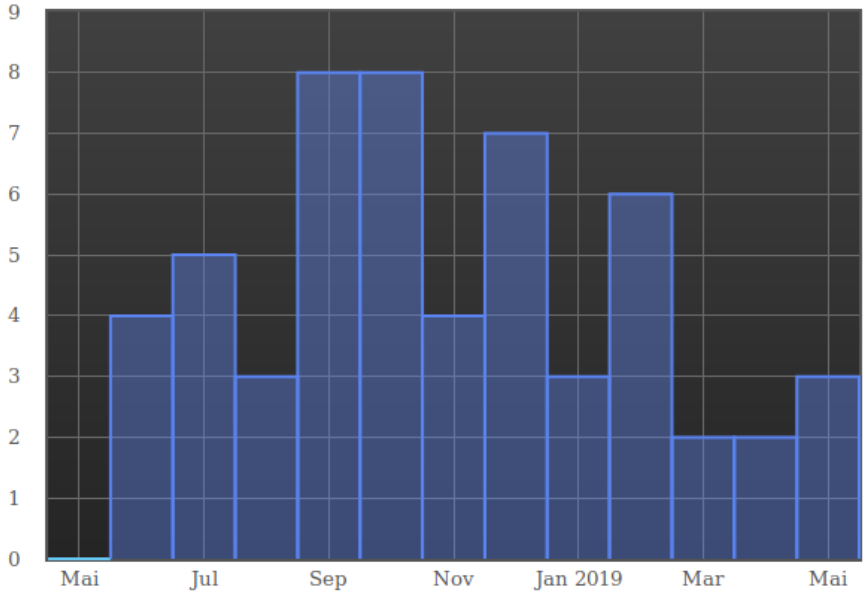


Figure 17: Diagramme représentant le nombre de différences entre deux versions dans le temps.

Servlet pour le transfert de données du mode Journal

Pour finir, il s'agit de la dernière servlet que j'ai mis en place. Elle permet d'afficher seulement les ajouts, suppressions et modifications des informations d'un objet chaque mois. Ainsi, les mois contenant ces critères sont affichés, avec le contenu ajouté, modifié et supprimé entre celui ci et le fichier précédant. Il y a donc un code couleur pour chaque ligne affichée. Les lignes en vert sont celles ayant été ajoutées, celles en rouge sont un aperçu de la ligne telle qu'elle était avant sa suppression. Et enfin, en jaune sont colorées les lignes ayant subies des modifications. Nous pouvons passer la souris sur la ligne pour voir la ligne avant sa modification.

Configuration du chemin de l'exécutable de oldsim

Comme dit précédemment, le fichier web.xml fait office de fichier de configuration de l'ensemble de nos servlets. Ainsi, il est possible de déclarer une nouvelle variable contenant le chemin de l'exécutable de oldsim, disponible pour chaque servlet. Cela permet ainsi de ne disposer qu'à un seul endroit ce chemin d'exécution dans le code. Ce qui devient très vite pratique si nous voulons que notre application soit portable, principalement lors de son installation sur les serveurs de l'Observatoire.

Gestion des erreurs et exceptions

Tout au long de ce projet, j'ai veillé à ce que mon programme fonctionne le plus parfaitement possible. Ainsi, en cas de problème d'intervalle de valeur, de type, ou même de

problème interne, j'ai mis en place une gestion des exceptions permettant de traiter ces cas. Cela m'a permis d'effectuer de nombreux tests, garantissant le bon fonctionnement du programme. Des messages d'erreurs sont ainsi affichés sur la page HTML pour faire comprendre à l'utilisateur la nature et l'origine du problème. Cela sert aussi aux différents développeurs en tant qu'indications permettant de mieux cerner un problème, pour mieux le résoudre.
En plus de cela, j'ai effectué différents tests unitaires pour veiller au bon déclenchement de mes exceptions. Ces différents tests me permettent de vérifier que les exceptions ne se lancent qu'en cas de nécessité. Il faut donc veiller à bien tester les valeurs aux bornes que nous fixons, pour ne pas avoir des cas mals gérés. Par exemple, un mois ne peut être compris qu'entre Janvier et Décembre, de même que la date ne peut pas dépasser celle courante, nous ne connaissons pas encore les données futur.

Communication des données à l'aide du JavaScript

Utiliser un fichier JavaScript nous permet de dynamiser notre page. Ainsi, nous pouvons ajouter et modifier les informations dans la page en temps réel.

Utilisation de JQuery

Pour faciliter le transfert des informations de la page HTML au code JavaScript, j'ai choisi d'utiliser JQuery. Avec l'aide de cette librairie, j'ai la possibilité de récupérer la valeur contenue dans les champs HTML en connaissant simplement un "id" ou une classe de l'élément. L'écriture est ainsi raccourcie, d'autant plus que ce genre d'opération est très fréquente dans mon programme, de même que l'écriture de données dans des champs. Pour une application interne, l'utilisation de la librairie JQuery n'est pas très dérangeant. Principalement car le temps d'exécution du programme ne sera pas beaucoup plus long, car le programme n'est pas très conséquent. JQuery facilite donc grandement la lisibilité du code, et donc sa maintenance. Enfin, pour pouvoir récupérer ou modifier des données dans une page, il faut mettre le code dans une fonction nommée "ready" ne s'exécutant qu'une seule fois, à la fin de la création de la page.

Utilisation d'AJAX

Enfin, pour pouvoir effectuer les transferts entre le client (pageWeb/HTML) et le serveur (les Servlets), AJAX réalisera la requête sous forme d'un appel de fonction asynchrone. Nous avons donc un appel à notre servlet Java, en prenant soin de passer les paramètres nécessaires. Cette requête s'effectue de manière asynchrone, c'est à dire qu'elle n'attend pas sa finalisation pour exécuter le reste du code. Nous avons donc un calcul en tâche de fond, qui une fois terminé, retourne ses résultats dans le contenu de la page Web.

Création du diagramme

La création du diagramme a été une grosse partie, comprenant de nombreuses fonctionnalités. J'ai choisis d'utiliser la librairie Javascript "flot" permettant de réaliser différents types de diagramme, dont la forme qui nous intéressait: l'histogramme. Pour pouvoir créer cela, de nombreux réglages sont nécessaires. Pour faire un diagramme, il nous faut un tableau contenant pour chaque case un index et une valeur, mais aussi diverses options. Comme dit précédemment, notre diagramme aura une taille variable suivant les dates sélectionnées: toujours un multiple de douze auquel on ajoutera une dernière barre. Ensuite nous allons remplir notre tableau de données, avec les données obtenues par la servlet Diagram. Ainsi, notre tableau complété, nous pouvons passer aux valeurs sur les axes. Ici, en abscisses nous aurons les mois, et en ordonnées le nombre de différences. Concernant les ordonnées, il n'y aura aucun problème Cependant, les abscisses vont nécessiter plusieurs calculs. D'une part la détermination du mois de Janvier est primordiale pour y intégrer l'année. Il reste la plus grosse partie, afficher seulement certains mois sur l'axe suivant le nombre d'années affichées à la fois. C'est à dire, si une seule année est affichée, alors l'ensemble des mois peut être affiché. Or, si trois ans sont présents en même temps, nous allons afficher un mois sur trois, en veillant à avoir le mois de Janvier suivi de l'année constamment affichés pour ne pas masquer les changements d'année. Une fois tout cela effectué, nous avons changé la couleur de

l'ensemble des barres, et mis d'une couleur différente la barre représentant le mois de visualisation actuelle. Cela permet de mieux se repérer sur le diagramme. Ce diagramme va ensuite être intégré dans une balise HTML sous la forme de canvas.

Les différentes requêtes AJAX

Chaque Servlet Java ne possède qu'une requête AJAX associé, c'est pourquoi il n'y en a que trois.

Requête AJAX du formulaire de comparaisons

Celle ci se concentre sur la récupération du contenu de différents fichiers, extraits de la base de données. Cette requête traite aussi bien les comparaisons entre deux fichiers qu'avec une liste de fichiers. Il a donc été mis en place deux manières de les différencier. Deux variables sont dédiées à leur différenciation. La première permet de tester s'il s'agit d'une requête deux à deux entre les mois renseignés dans les formulaires d'une requête avec le fichier suivant ou précédent, ou bien d'une requête recherchant le prochain fichier contenant des modifications. La seconde variable teste s'il s'agit d'une recherche pour les fichiers précédents, ou pour les fichiers suivants. Quelque soit les tests, la requête AJAX écrit les données reçues sur la page Web, dans un emplacement dédié.

Requête AJAX du diagramme

Cette fois ci, la requête permettant la création du diagramme est beaucoup plus simple. Elle fait simplement appel à la fonction permettant de créer le diagramme, en passant en paramètre le résultat de la requête envoyée à la Servlet. Une fois le résultat du calcul du nombre de différences retourné, nous créons le diagramme à partir de ces données, et des dates associées.

Requête AJAX du mode journal de modifications

Pour finir, la dernière requête AJAX permet de seulement afficher les lignes modifiées chaque mois d'un fichier. Ainsi, l'appel au programme Java nous retourne les mois et les données associées. Il ne reste plus qu'à reconstruire chaque ligne du tableau dans lequel nous allons insérer nos lignes. Celles ci contiendront ainsi les données formatées pour chaque mois possédant des modifications.

Chargements de la page

Lors du clic sur le bouton Rechercher, d'un changement de mode, ou même lors du scroll de la page dans le mode journal, une requête est effectuée. Tout d'abord, dans le mode de comparaison standard, nous effectuons une requête au simple clic du bouton Rechercher, de plus, cliquer sur un bouton « < » ou « >> » par exemple contribue aussi à effectuer une requête, de même qu'un clic sur le diagramme. Tout cela se gère dans le JavaScript, en écoutant les événements de clic. Concernant la page Web dans le mode d'affichage du journal, lors d'un scroll, nous affichons progressivement les mois suivants. Cela est dû à l'écoute d'un événement scroll et en veillant bien évidemment qu'une seule requête ne s'effectue à la fois. En effet, grâce à une variable booléenne, nous pouvons contrôler le nombre de requêtes AJAX, et si l'une d'entre elles démarre, nous bloquons la possibilité d'en effectuer une nouvelle. C'est seulement une fois sa complétion atteinte que de nouvelles requêtes peuvent être exécutées.

Perspectives d'évolutions

De nombreuses modifications et ajouts sont envisageables, principalement lors de la détection des différences. Il arrive parfois qu'un décalage de lignes se produise lors de la recherche de différences. Il s'agit d'un problème de mise en forme causé par la librairie diffutils.

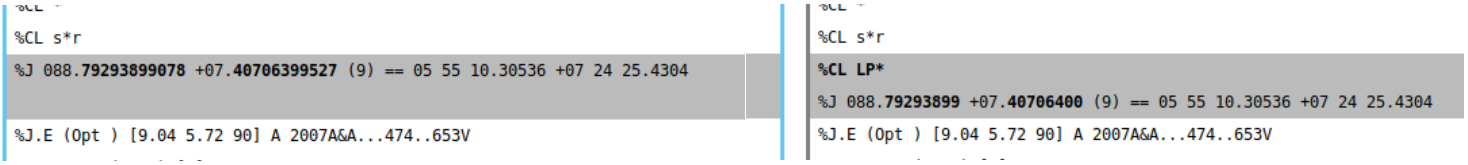


Figure 18: Problème de mise en forme

Étant donné le problème de décalage lors de l'affichage, ainsi qu'une détection faussée par endroit, convertir les chaines de caractères en objets Java structurés aurait été possible. Ainsi, chaque objet Java représenterait un objet astronomique, permettant une plus grand souplesse dans l'affichage des données. En le déclarant ainsi, la librairie diffutils ne traitera qu'une petite quantité de données à la fois, et donc limitera les erreurs. De plus, ce passage du mode chaîne de caractères au mode objet aurait été l'occasion de comparer leur temps d'exécution, et ainsi de pouvoir en déduire la version la plus rapide d'exécution.

Concernant la partie fonctionnalité, de nombreux ajouts sont possibles. Par exemple : l'ajout d'un filtre, ne permettant d'afficher que certaines lignes des fichiers suivant la clef parfile de chaque ligne (c'est à dire, une information particulière d'un objet céleste). Ainsi, les documentalistes auraient pu se focaliser sur certaines parties d'un fichier ; par exemple, les références bibliographiques. Dans le mode d'affichage du journal, une possibilité envisagée serait de pouvoir cliquer sur une date souhaitée pour nous afficher la comparaison du fichier entre cette date et la précédente dans le mode de comparaison classique. Cela aurait permis de faciliter la mise en contexte des différences dans la description complète de l'objet astronomique. Il s'agit en plus de cela d'une fonctionnalité améliorant l'ergonomie de l'application, évitant les changements intempestifs entre les deux modes d'affichage des informations.

Au niveau de la présentation des données, l'affichage de la date de création du premier fichier d'un objet céleste aurait pu être recherchée, et ainsi indiquée sur notre page Web. Pour cela, il avait été proposé de parcourir l'ensemble des fichiers disponibles d'un objet céleste, et s'il manquait au moins 12 fichiers consécutifs, nous considérions que cet objet a été ajouté à la base de donnée à cette date. Pour cela, j'aurais eu recours à la notion de Thread en Java afin de calculer simultanément des différences entre plusieurs couples de mois. Il aurait par conséquent fallu trouver le moyen de retourner l'information d'apparition, puisque durant la requête parallèle, la réponse aurait mis plus de temps à parvenir du côté client. Il aurait donc fallu lancer la recherche de la date de création dans une requête AJAX séparée de la recherche des différences.
Enfin, l'écriture d'un manuel d'utilisation expliquant en détail les fonctionnalités de cette application Web serait un véritable plus pour la compréhension et l'utilisation du programme. Ainsi n'importe quel utilisateur, ancien ou nouveau, pourrait le consulter en cas de doute sur le fonctionnement du programme. De plus, le questionnement sur la manière dont ont été obtenues ces

informations pourrait être levé via ce manuel. Par exemple, il pourrait contenir des explications sur les différents boutons, le code couleur utilisé, l'importance et l'utilité des différents modes, de la manière dont doivent être analysées les données, ainsi qu'une brève présentation de la liste des problèmes existants.

Conclusion

Après ces dix semaines au coeur de l'Observatoire, au sein de l'équipe du CDS, j'ai beaucoup appris.
Tout d'abord, sur le plan technique, j'ai pu renforcer mes connaissances avec le langage de programmation orienté objet Java. L'utilisation de la librairie « diffutils » ainsi que la classe Servlet d'Oracle m'ont appris à mieux me documenter, et découvrir des solutions que je n'avais encore jamais vu à l'IUT. Il faut parfois utiliser des notions qui nous étaient encore inconnues, mais aussi se servir du travail d'autres informaticiens. Concernant le JavaScript, j'ai eu l'occasion d'apprendre le JQuery et AJAX, étudiés durant le quatrième semestre à l'IUT, pour les personnes ayant choisis le parcours Web.
Ensuite, sur le plan social, les échanges entre membres de l'Observatoire m'ont permis de découvrir le milieu professionnel. J'ai donc pu observer les conditions de travail d'un vrai projet, ainsi que les contraintes qui y sont attachées. J'ai cependant eu la chance d'avoir des horaires de travail plutôt souples, ce qui m'a permis de venir travailler aux horaires qui me convenaient le mieux.
Pour finir, l'environnement de travail m'a particulièrement séduit. Le milieu astronomique reste pour moi la passion que j'aime le plus, c'est pourquoi j'ai particulièrement apprécié venir travailler ici. Chaque matin, ce fût par envie que je m'apprêtais à travailler, plutôt que par obligation.
Ce stage a donc contribué à m'offrir une vision plus claire du milieu professionnel. Et je ne renoncerais pas à travailler dans ce domaine.

Glossaire

Les objets célestes

Au seins de l'Observatoire, de nombreux domaines sont abordés. Principalement l'étude des différents corps célestes, régissant dans l'espace. Les bases de données du CDS contiennent tout type d'objet céleste. C'est à dire des étoiles, des galaxies, des nébuleuses, etc.

Un fichier parfile

La base de données Simbad fournit de nombreuses informations sur chaque objet céleste. Afin de sauvegarder toutes ces informations sur un objet à une date donnée, un format de texte particulier fût définit : le format « parfiles ». Dans un tel fichier, chaque ligne se présente sous la forme d'un couple clef-valeur. Il existe autant de clefs que de types information permettant de décrire un objet céleste. Chaque clef commence par un % suivi d'une ou plusieurs lettres.

```
#---[sim-14@simbad] NAME Vega!
================= Simbad(IV)================================ as of 1-Apr-2018
%#Parfile(5.52)@simbad on: /arc2/oldsim/simbad//sim4/2018-04/simcat.00(01 Apr 2018 09:55:30)
%#Execution date:                                          05 Jun 2019 14:47:57

%@ #124965 = @2900336 19821
%A 30.09.06 28.03.18
%C.0 dS*
%CL *
%CL PM*
%CL V*
%CL **
%CL IR
%CL X
%CL UV
%CL dS*
%CL smm
%J 279.23473478702 +38.78368895624 (9) == 18 36 56.33635 +38 47 01.2802
%J.E (Opt ) [3.51 2.81 90] A 2007A&A...474..653V
%M U 0.03 (Vega) [~] C 2002yCat.2237....0D
%M B 0.03 (Vega) [~] C 2002yCat.2237....0D
%M V 0.03 (Vega) [~] C 2002yCat.2237....0D
%M R 0.07 (Vega) [~] C 2002yCat.2237....0D
%M I 0.10 (Vega) [~] C 2002yCat.2237....0D
%M J -0.18 (Vega) [~] C 2003yCat.2246....0C
%M H -0.03 (Vega) [~] C 2003yCat.2246....0C
```

Figure 19: Exemple de fichier parfile

JSON

JSON (JavaScript Object Notation) est une notation textuelle permettant de représenter un ensemble structuré de données en JavaScript (par exemple un objet) . Il est principalement utilisé pour l'échange de données avec un serveur. Il nous est très pratique lorsque nous devons retourner le contenu des fichiers entre notre code Java et JavaScript, par exemple.

HTML

HyperText Markup Language

CSS
Cascading Style Sheet

JUnit
Framework de test unitaire pour le langage de programmation Java.

Servlet
Classe Java permettant de créer dynamiquement des données au sein d'un serveur HTTP.

DOM
Interface permettant à des scripts de modifier le contenu du navigateur Web.

SIMBAD
Set of Identifications, Measurements, and Bibliography for Astronomical Data

CDS
Centre de Données astronomique de Strasbourg

GALHECOS
Galaxies, High Energy, Cosmology, Compact Objects & Stars

NASA
National Aeronautics and Space Administration

ESA
European Space Agency

Webographie

Site de GitHub
https://github.com/java-diff-utils/java-diff-utils/blob/master/src/test/resources/mocks/original.txt
Lien original de la librairie diffutils

Site officiel de la librairie FLOT
https://www.flotcharts.org/

Site officiel de Oracle
https://docs.oracle.com/javase/7/docs/api/java/lang/String.html Documentation String
https://docs.oracle.com/javase/7/docs/api/java/util/regex/Matcher.html Documentation Matcher
https://docs.oracle.com/javase/8/docs/api/java/io/BufferedReader.html Documentation Reader

Site d'apprentissage OpenClassrooms
https://openclassrooms.com/fr/courses/26832-apprenez-a-programmer-en-java/22404-apprehendez-la-genericite-en-java Documentation sur la généricité en Java
https://openclassrooms.com/fr/courses/1304236-redigez-en-markdown Documentation Markdown
https://openclassrooms.com/forum/sujet/les-suffixes-amp-lt-et-amp-gt-73061
https://openclassrooms.com/fr/courses/626954-creez-votre-application-web-avec-java-ee/621914-formulaires-le-b-a-ba Documentation interaction Java et JavaScript
https://openclassrooms.com/fr/courses/1631636-simplifiez-vos-developpements-javascript-avec-jquery/1631981-premiers-pas Documentation JQuery
https://openclassrooms.com/fr/courses/146276-tout-sur-le-javascript/145081-lobjet-date
Documentation Date JavaScirpt

Forum communautaire Developpez
https://www.developpez.net/forums/d861917/environnements-developpement/eclipse/eclipse-java/changer-nature-d-projet-java-projet-type-web-dynamique/ Documentation projet Eclipse
https://www.developpez.net/forums/d605454/java/developpement-web-java/servlets-jsp/recuperer-envoyer-valeurs-formulaire-servlet/ Documentation Servlet Java
https://www.developpez.net/forums/d1645058/java/developpement-web-java/servlets-jsp/requete-ajax-java-ee/ Documentation AJAX
https://www.developpez.net/forums/d1688894/java/general-java/debuter/methode-generique-declaree-t-exemple/ Documentation sur la généricité en Java

Site StackOverflow
https://stackoverflow.com/questions/19568142/how-to-read-json-sent-by-ajax-in-servlet
Documentation interaction Java et JavaScript via AJAX

Blog de cours en ligne
http://blog.paumard.org/cours/java-api/chap03-expression-regulieres-syntaxe.html
Documentation syntaxe Expressions régulières

Annexes

Code source de ma Servlet Form :

```
package servlets;
import java.io.IOException;
import java.io.PrintWriter;

import java.util.ArrayList;
import java.util.List;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.xml.parsers.ParserConfigurationException;

import org.json.JSONArray;
import org.json.JSONException;
import org.json.JSONObject;

import cds.simbad.database.SimbadDbException;
import cds.simbad.oldsim.ShellCommandException;
import cds.simbad.oldsim.SimbadHistory;
import simDiff.SimbadDiff;
import utilitaires.ExceptionUtils;

/**
 * Servlet implementation class From
 */
public class Form extends HttpServlet {
    // Attributs
    private static final long serialVersionUID = 1L;
    /** Unique objet SimbadDiff pour les comparaisons dans la base de données*/
    private SimbadDiff sim;
    private SimbadHistory simHistory;

   // Méthodes
   /** Initialise notre objet SimbadDiff permettant d'effectuer les comparaisons */
   public void init() throws ServletException {
       super.init();
       try {
           sim = SimbadDiff.getSim();
           // Initialisation de SimbadHistory:
           simHistory = new SimbadHistory(getServletContext().getInitParameter("oldsimURL"));
           // Pour débugage seulement:
           simHistory.debug = true;
           simHistory.checkDate = true;
       } catch (SimbadDbException e) {
           e.printStackTrace();
       } catch (ParserConfigurationException e) {
           e.printStackTrace();
       }
   }

   /**
    * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)
    */
   protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
       String nomForm = request.getParameter("object_name");
       String mois1Form = request.getParameter("date_month1");
       String annee1Form = request.getParameter("date_year1");
       String mois2Form = request.getParameter("date_month2");
       String annee2Form = request.getParameter("date_year2");

       String stringComparaison = request.getParameter("simpleComparaison");

       PrintWriter writer = response.getWriter();
```

```
try {
    ExceptionUtils.testBooleanClicButton(stringComparaison);

    boolean booleanComparaison = Boolean.parseBoolean(stringComparaison);
    // Boutons SUBMIT, < et >
    if (booleanComparaison) {
        try {
            // Test les différents cas de mauvais paramètres
            ExceptionUtils.testParametersServ(nomForm, mois1Form, annee1Form);
            ExceptionUtils.testParametersServ(nomForm, mois2Form, annee2Form);

            ArrayList<ArrayList<String>> chaine;

            chaine = sim.compareFiles(simHistory.getParfile(nomForm, Integer.parseInt(annee1Form), Integer.parseInt(mois1Form)),
                    simHistory.getParfile(nomForm, Integer.parseInt(annee2Form), Integer.parseInt(mois2Form)));

            JSONArray oldFile = new JSONArray(chaine.get(0));
            JSONArray newFile = new JSONArray(chaine.get(1));

            JSONObject result = new JSONObject();
            result.put("oldFile", oldFile);
            result.put("newFile", newFile);

            response.setContentType("application/json; charset=UTF_8");
            writer.append(result.toString());
        } catch (NullPointerException | IllegalArgumentException | ShellCommandException | JSONException e) {
            e.printStackTrace();
            List<String> lstError = new ArrayList<String>();
            lstError.add(e.getMessage());
            response.setContentType("application/json; charset=UTF_8");
            ExceptionUtils.writeErrorDocument(lstError, writer);
        }
    // Boutons << et >>
    } else {
        // Test les différents cas de mauvais paramètres
        ExceptionUtils.testParametersServ(nomForm, mois1Form, annee1Form);

        String directionDroite = request.getParameter("directionDroite");
        ExceptionUtils.testBooleanClicButton(directionDroite);

        boolean booleanDirectionDroite = Boolean.parseBoolean(directionDroite);

        int mois = Integer.parseInt(mois1Form);
```

```
        int annee = Integer.parseInt(annee1Form);
        int mois1Int = Integer.parseInt(mois1Form);
        int annee1Int = Integer.parseInt(annee1Form);

        ArrayList<String> files = new ArrayList<String>();
        int nbValDiag = 12;
        // Bouton <<
        if (!booleanDirectionDroite) {
            for (int i = 0; i < nbValDiag+1; i++) {
                mois--;
                if (mois < 10)
                    mois1Form = "0"+mois;
                else
                    mois1Form = ""+mois;
                if (mois < 1) {
                    mois = 12;
                    annee--;
                    mois1Form = ""+mois;
                    annee1Form = ""+annee;
                }
                try {
                    // Ajouter les élément de la fin vers le début, sinon l'histogramme sera à l'envers
                    files.add(0, simHistory.getParfile(nomForm, Integer.parseInt(annee1Form), Integer.parseInt(mois1Form)));
                } catch (NullPointerException | IllegalArgumentException | ShellCommandException e) {
                    e.printStackTrace();
                }
            }
        // Bouton >>
        } else {
            for (int i = 0; i < nbValDiag+1; i++) {
                try {
                    // Ajouter les élément de la fin vers le début, sinon l'histogramme sera à l'envers
                    files.add(simHistory.getParfile(nomForm, Integer.parseInt(annee1Form), Integer.parseInt(mois1Form)));
                } catch (NullPointerException | IllegalArgumentException | ShellCommandException e) {
                    e.printStackTrace();
                }
                mois++;
                if (mois < 10)
                    mois1Form = "0"+mois;
```

```
                else
                    mois1Form = ""+mois;
                if (mois > 12) {
                    mois = 1;
                    annee++;
                    mois1Form = ""+mois;
                    annee1Form = ""+annee;
                }
            }
        }

        int[] diffs = sim.getDifferencesFromString(files);

        int ind = (booleanDirectionDroite ? 0 : diffs.length-1);

        if (!booleanDirectionDroite) {
            while (ind > 0 && diffs[ind] <= 0)
                ind--;

            if (ind < 0 && diffs[ind] <= 0)
                throw new NullPointerException("Il n'exite pas de fichier différent de celui demandé dans les 12 derniers mois !");

            mois1Int = mois1Int-12+ind;
            if (mois1Int <= 0) {
                mois1Int += 12;
                annee1Int--;
            }
        } else {
            while (ind < diffs.length && ind < nbValDiag && diffs[ind] <= 0) {
                ind++;
            }

            if (ind >= nbValDiag && diffs[ind] <= 0)
                throw new NullPointerException("Il n'exite pas de fichier différent de celui demandé dans les 12 prochains mois !");

            mois1Int = mois1Int+ind+1;
            if (mois1Int > 12) {
                mois1Int -= 12;
                annee1Int++;
            }
        }

        try {
            ArrayList<ArrayList<String>> chaine;
            chaine = sim.compareFiles(simHistory.getParfile(nomForm, annee1Int, mois1Int),
                    simHistory.getParfile(nomForm, Integer.parseInt(annee2Form), Integer.parseInt(mois2Form)));

            JSONArray oldFile = new JSONArray(chaine.get(0));
            JSONArray newFile = new JSONArray(chaine.get(1));

            List<String> lstDateActu = new ArrayList<String>();
            lstDateActu.add(mois1Int < 10 ? "0"+mois1Int : ""+mois1Int);
            lstDateActu.add(""+annee1Int);
            JSONArray dateActu = new JSONArray(lstDateActu);

            JSONObject result = new JSONObject();
            result.put("oldFile", oldFile);
            result.put("newFile", newFile);
            result.put("dateActu", dateActu);

            response.setContentType("application/json; charset=UTF_8");
            writer.append(result.toString());
        } catch (NullPointerException | IllegalArgumentException | ShellCommandException | JSONException e) {
            e.printStackTrace();
            List<String> lstError = new ArrayList<String>();
            lstError.add(e.getMessage());
            response.setContentType("application/json; charset=UTF_8");
            ExceptionUtils.writeErrorDocument(lstError, writer);
        }
    }
} catch (NullPointerException npe) {
    npe.printStackTrace();
    List<String> lstError = new ArrayList<String>();
    lstError.add(npe.getMessage());
    response.setContentType("application/json; charset=UTF_8");
    ExceptionUtils.writeErrorDocument(lstError, writer);
}
}
```

Code source de ma Servlet Diagr :

```
package servlets;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.ArrayList;
import java.util.List;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.xml.parsers.ParserConfigurationException;

import cds.simbad.database.SimbadDbException;
import cds.simbad.oldsim.ShellCommandException;
import cds.simbad.oldsim.SimbadHistory;
import simDiff.SimbadDiff;
import utilitaires.ExceptionUtils;

/**
 * Servlet implementation class Diagram
 */
public class Diagram extends HttpServlet {
    // Attributs
    private static final long serialVersionUID = 1L;
    /** Unique objet SimbadDiff pour les comparaisons dans la base de données*/
    private SimbadDiff sim;
    private SimbadHistory simHistory;

    // Méthodes
    /** Initialise notre objet SimbadDiff permettant d'effectuer les comparaisons */
    public void init() throws ServletException {
        super.init();
        try {
            sim = SimbadDiff.getSim();
            // Initialisation de SimbadHistory:
            simHistory = new SimbadHistory(getServletContext().getInitParameter("oldsimURL"));
            // Pour débugage seulement:
            simHistory.debug = true;
            simHistory.checkDate = true;
        } catch (SimbadDbException e) {
            e.printStackTrace();
        } catch (ParserConfigurationException e) {
            e.printStackTrace();
        }
```

```java
/**
 * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)
 */
protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
    String nomForm = request.getParameter("object_name");
    String mois1Form = request.getParameter("date_month1");
    String annee1Form = request.getParameter("date_year1");
    String mois2Form = request.getParameter("date_month2");
    String annee2Form = request.getParameter("date_year2");

    PrintWriter writer = response.getWriter();
    try {
        // Test les différents cas de mauvais paramètres
        ExceptionUtils.testParametersServ(nomForm, mois1Form, annee1Form);
        ExceptionUtils.testParametersServ(nomForm, mois2Form, annee2Form);

        int mois2FormInt = Integer.parseInt(mois2Form);
        int annee2FormInt = Integer.parseInt(annee2Form);
        int mois1FormInt = Integer.parseInt(mois1Form);
        int annee1FormInt = Integer.parseInt(annee1Form);

        ExceptionUtils.testSuperiority(annee1FormInt, annee2FormInt);

        // Nombre de valeurs dans le diagramme
        int nbValDiagr = 0;
        // Ajout de 12 valeurs pour chaque année supplémentaire à afficher
        while (annee1FormInt+nbValDiagr/12 < annee2FormInt || (annee1FormInt+nbValDiagr/12 == annee2FormInt && mois1FormInt < mois2FormInt)) {
            nbValDiagr += 12;
        }
        if (nbValDiagr == 0)
            nbValDiagr +=12;

        ArrayList<String> files = new ArrayList<String>();
        for (int i = 0; i < nbValDiagr+2; i++) {
            try {
                // Ajouter les éléments de la fin vers le début, sinon l'histogramme sera à l'envers
                files.add(0, simHistory.getParfile(nomForm, Integer.parseInt(annee2Form), Integer.parseInt(mois2Form)));
            } catch (NullPointerException | IllegalArgumentException | ShellCommandException e) {
                e.printStackTrace();
            }
            // Décrémentation d'un mois, on retire un an et on remets le mois à 12 si on arrive au début d'année
            mois2FormInt--;
            if (mois2FormInt < 10)
                mois2Form = "0"+mois2FormInt;
            else
                mois2Form = ""+mois2FormInt;
            if (mois2FormInt < 1) {
                mois2FormInt = 12;
                annee2FormInt--;
                mois2Form = ""+mois2FormInt;
                annee2Form = ""+annee2FormInt;
            }
        }

        // Nombre de différences représenté par les barres du diagramme
        int[] diffs = sim.getDifferencesFromString(files);
        mois2FormInt += 2;

        response.setContentType("application/json; charset=UTF_8");
        writer.append("{ \"diagr\": [ ");
        if (diffs.length != 0) {
            for (int i = 0; i < diffs.length-1; i++) {
                writer.append("[");
                if (mois2FormInt > 12)
                    mois2FormInt -= 12;
                writer.append(mois2FormInt+",");
                mois2FormInt++;
                writer.append(diffs[i]+"], ");
            }
            writer.append("[");
            if (mois2FormInt > 12)
                mois2FormInt -= 12;
            writer.append(mois2FormInt+",");
            writer.append(diffs[diffs.length-1]+"] ] }");
        } else {
            writer.append("] }");
        }
    } catch (IllegalArgumentException iae) {
        iae.printStackTrace();
        List<String> lstErreurs = new ArrayList<String>();
        lstErreurs.add(iae.getMessage());
        response.setContentType("application/json; charset=UTF_8");
        ExceptionUtils.writeErrorDocument(lstErreurs, writer);
    }
}
```

Code source de ma Servlet Journal :

```
package servlets;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.ArrayList;
import java.util.List;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.xml.parsers.ParserConfigurationException;

import org.json.JSONArray;
import org.json.JSONException;
import org.json.JSONObject;

import cds.simbad.database.SimbadDbException;
import cds.simbad.oldsim.ShellCommandException;
import cds.simbad.oldsim.SimbadHistory;
import simDiff.SimbadDiff;
import utilitaires.ExceptionUtils;

/**
 * Servlet implementation class Journal
 */
public class Journal extends HttpServlet {
    // Attributs
    private static final long serialVersionUID = 1L;
    /** Unique objet SimbadDiff pour les comparaisons dans la base de données*/
    private SimbadDiff sim;
    private SimbadHistory simHistory;

    // Méthodes
    /** Initialise notre objet SimbadDiff permettant d'effectuer les comparaisons */
    public void init() throws ServletException {
        super.init();
        try {
            sim = SimbadDiff.getSim();
            // Initialisation de SimbadHistory:
            simHistory = new SimbadHistory(getServletContext().getInitParameter("oldsimURL"));
            // Pour débugage seulement:
            simHistory.debug = true;
            simHistory.checkDate = true;
        } catch (SimbadDbException e) {
```

```
            e.printStackTrace();
        } catch (ParserConfigurationException e) {
            e.printStackTrace();
        }
    }

    /**
     * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)
     */
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        String nomForm = request.getParameter("object_name");
        String moisForm = request.getParameter("date_month");
        String anneeForm = request.getParameter("date_year");

        PrintWriter writer = response.getWriter();
        try {
            // Test les différents cas de mauvais paramètres
            ExceptionUtils.testParametersServ(nomForm, moisForm, anneeForm);

            ArrayList<ArrayList<String>> lstContenuFichier = new ArrayList<ArrayList<String>>();
            ArrayList<ArrayList<String>> lstDates = new ArrayList<ArrayList<String>>();

            // Date la plus récente
            int anneeRecente = Integer.parseInt(anneeForm);
            int moisRecent = Integer.parseInt(moisForm);
            // Date avec un mois de décalage
            int anneeAvant = anneeRecente;
            int moisAvant = moisRecent;

            JSONObject journal = new JSONObject();

            for (int i = 0; i < 12; i++) {
                if (moisAvant <= 1) {
                    moisAvant = 12;
                    anneeAvant--;
                } else {
                    moisAvant--;
                }
                try {
                    lstContenuFichier.add(sim.compareFilesDiff(simHistory.getParfile(nomForm, anneeRecente, moisRecent),
                        simHistory.getParfile(nomForm, anneeAvant, moisAvant)));
                    int m = moisAvant%12;
                     int y = anneeAvant+moisAvant/12;
                     ArrayList<String> tmpDays = new ArrayList<String>(2);
                     tmpDays.add(""+m);
                     tmpDays.add(""+y);
                     lstDates.add(tmpDays);
                } catch (NullPointerException | IllegalArgumentException | ShellCommandException e) {
                     System.out.println(e.getMessage());
                }
                anneeRecente = anneeAvant;
                moisRecent = moisAvant;
            }

            JSONArray diary = new JSONArray(lstContenuFichier);
            JSONArray days = new JSONArray(lstDates);
            journal.put("diary", diary);
            journal.put("days", days);

            response.setContentType("application/json; charset=UTF_8");
            writer.append(journal.toString());
        } catch (IllegalArgumentException | JSONException iae) {
            iae.printStackTrace();
            List<String> lstErreurs = new ArrayList<String>();
            lstErreurs.add(iae.getMessage());
            response.setContentType("application/json; charset=UTF_8");
            ExceptionUtils.writeErrorDocument(lstErreurs, writer);
        }
    }
```

Code source de la Classe principale SimDiff :

```java
package simDiff;
import com.github.difflib.algorithm.DiffException;
import com.github.difflib.text.DiffRow;
import com.github.difflib.text.DiffRowGenerator;

import cds.simbad.database.Simbad;
import cds.simbad.database.SimbadDbException;
import utilitaires.ExceptionUtils;

import java.io.IOException;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;

public class SimbadDiff {
    // Attributs
    /** Unique objet SimbadDiff pour les comparaisons dans la base de données*/
    private static SimbadDiff sim;

    // Méthodes
    /** Construit un objet par defaut avec connection avec la base de données
     * @throws SimbadDbException */
    @SuppressWarnings("unused")
    private SimbadDiff() throws SimbadDbException {
        // Initialisation de Simbad
        Simbad simbad = new Simbad("/simbad.ini");
    }

    /** Construit un objet SimbadDiff en garantissant un singleton
     * @return Un objet singleton SimbadDiff
     * @throws SimbadDbException
     */
    public static synchronized SimbadDiff getSim() throws SimbadDbException {
        if (sim == null)
            sim = new SimbadDiff();
        return sim;
    }

    /** Calcule le nombre de ligne contenant au moins une balise HTML de différences entre plusieurs fichiers
     * @param files List de String (contenant le contenu du fichier)
     * @return Tableau d'entier contenant le nombre de différences successives
     * @throws IOException
     */
    public int[] getDifferencesFromString(List<String> files) throws IOException {
```

```
        ExceptionUtils.testListGetDifferencesString(files);

        ArrayList<ArrayList<String>> list = new ArrayList<ArrayList<String>>();
        for (int i = 0; i < files.size(); i++) {
            try {
                list.add(new ArrayList<String>(Arrays.asList(((String)files.get(i)).split("\\n"))));
            } catch (NullPointerException npe) {
                list.add(null);
            }
        }

        return getDifferences(list);
    }

    /** Calcule le nombre de ligne contenant au moins une balise HTML de différences entre plusieurs fichiers
     * @param files List d'une List de String (contenant pour chaque fichier dans la première list chacune des lignes de la second list par fichier)
     * @return Tableau d'entier contenant le nombre de différences successives
     * @throws IOException
     */
    public int[] getDifferences(List<ArrayList<String>> files) throws IOException {
        ExceptionUtils.testListGetDifferencesListString(files);
        int[] diff = new int[files.size()-1];

        ArrayList<ArrayList<ArrayList<String>>> list = new ArrayList<ArrayList<ArrayList<String>>>();
        for (int i = 0; i < files.size()-1; i++) {
            List<String> s = (List<String>)(files.get(i));
            List<String> s1 = (List<String>)(files.get(i+1));
            try {
                list.add(compareFiles(s, s1));
            } catch (NullPointerException npe) {
                list.add(null);
            }
        }

        for (int i = 0; i < list.size(); i++) {
            try {
                diff[i] = getDifferences(list.get(i).get(0), list.get(i).get(1));
            } catch (NullPointerException npe) {
                diff[i] = -1;
            }
        }

        return diff;
    }

    /** Calcule le nombre de ligne contenant au moins une balise HTML de différences entre deux fichiers
     * @param file1 List de String contenant chacune des ligne du premier fichier
     * @param file2 List de String contenant chacune des ligne du second fichier
     * @return Entier indiquant le nombre de différences
     */
    public int getDifferences(List<String> file1, List<String> file2) {
        ExceptionUtils.testListGetDifferencesInt(file1, file2);

        int nbdiff = 0;

        for (int j = 7; j < file1.size(); j++) {
            if (file1.get(j).contains("<span class=") || file2.get(j).contains("<span class="))
                nbdiff++;
        }

        return nbdiff;
    }

    /** Retourne la comparaison de deux fichiers passé en paramètre sous forme de deux List de String
     * @param file1 String contenant le contenue du premier fichier séparé par des retour chariot (\n) pour le convertir en list suivant chacune des lignes
     * @param file2 String contenant le contenue du second fichier séparé par des retour chariot (\n) pour le convertir en list suivant chacune des lignes
     * @return List d'une List de String, contenant pour chaque emplacement, une List de String pour chacun des fichiers, correspondant à chacune de leurs lignes
     */
    public ArrayList<ArrayList<String>> compareFiles(String file1, String file2) {
        ExceptionUtils.testStringCompareFile(file1, file2);

        List<String> lfile1 = Arrays.asList(file1.split("\\n"));
        List<String> lfile2 = Arrays.asList(file2.split("\\n"));

        return compareFiles(lfile1, lfile2);
    }

    /** Retourne la comparaison de deux fichiers passé en paramètre sous forme de deux List de String
     * @param lStr1 List de String contenant chacune des ligne du premier fichier
     * @param lStr2 List de String contenant chacune des ligne du second fichier
     * @return List d'une List de String, contenant pour chaque emplacement, une List de String pour chacun des fichiers, correspondant à chacune de leurs lignes
     */
    public ArrayList<ArrayList<String>> compareFiles(List<String> lStr1, List<String> lStr2) {
```

```
        //create a configured DiffRowGenerator
        DiffRowGenerator generator = DiffRowGenerator.create()
                .showInlineDiffs(true)
                .inlineDiffByWord(true)
                .oldTag(f -> f ? "<span class=\"modified\">" : "</span>")
                .newTag(f -> f ? "<span class=\"modified\">" : "</span>")
                .build();

        ArrayList<String> finalOld = new ArrayList<String>();
        ArrayList<String> finalNew = new ArrayList<String>();

        try {
            List<DiffRow> rows = generator.generateDiffRows(lStr1, lStr2);

            String str1 = "<div class=\"modif\">";
            String str2 = "<div class=\"line\">";
            String str3 = "</div>";

            // Pour chacune des ligne
            for (DiffRow row : rows) {
                if (row.getOldLine().contains("<span class=") || row.getNewLine().contains("<span class=")) {
                    finalOld.add(str1+row.getOldLine()+str3);
                    finalNew.add(str1+row.getNewLine()+str3);
                } else {
                    finalOld.add(str2+row.getOldLine()+str3);
                    finalNew.add(str2+row.getNewLine()+str3);
                }
            }
        } catch (DiffException de) {
            de.printStackTrace();
        }

        ArrayList<ArrayList<String>> list = new ArrayList<ArrayList<String>>(2);
        list.add(finalOld);
        list.add(finalNew);
        return list;
    }

    /** Retourne les lignes du second fichier ayant été modifié entre deux fichiers passé en paramètre sous forme de deux List de String
     * @param file1 String contenant le contenue du premier fichier séparé par des retour chariot (\n) pour le convertir en list suivant chacune des lignes
     * @param file2 String contenant le contenue du second fichier séparé par des retour chariot (\n) pour le convertir en list suivant chacune des lignes
     * @return List de String contenant chaque ligne du second fichier ayant des modifications avec le premier fichier
     */
    public ArrayList<String> compareFilesDiff(String file1, String file2) {
        ArrayList<String> listTest = compareFiles(file2, file1).get(0);

        ArrayList<String> listDonne = compareFiles(file2, file1).get(1);

        ArrayList<String> list2 = new ArrayList<String>();
        for (int i = 7; i < listDonne.size(); i++) {
            if (listDonne.get(i).contains("<div class=\"modif\">")) {
                String testChaine1 = listTest.get(i).replace("<div class=\"modif\">", "").replace("</div>", "");
                String testChaine2 = listDonne.get(i).replace("<div class=\"modif\">", "").replace("</div>", "");
                if (testChaine1.isEmpty() && !testChaine2.isEmpty()) {
                    listDonne.set(i, listDonne.get(i).replace("<div class=\"modif\">", "<div class=\"modif donneesAjout\">"));
                } else if (!testChaine1.isEmpty() && testChaine2.isEmpty()){
                    listDonne.set(i, listTest.get(i).replace("<div class=\"modif\">", "<div class=\"modif donneesSuppr\">"));
                } else {
                    testChaine1 = testChaine1.replace("<span class=\"modified\">", "");
                    testChaine1 = testChaine1.replace("</span>", "");
                    listDonne.set(i, listDonne.get(i).replace("<div class=\"modif\">", "<div class=\"modif donneesModif\" title=\""+testChaine1+"\">"));
                }
                list2.add(listDonne.get(i));
            }
        }
        if (list2.isEmpty())
            throw new IllegalArgumentException("Le fichier ne posséde pas de nouvelles différences !");
        return list2;
    }
}
```

Code source de ma Classe gérant les exceptions ExceptionUtils :

```java
package utilitaires;
import java.io.PrintWriter;
import java.util.ArrayList;
import java.util.Calendar;
import java.util.List;
import java.util.stream.Collectors;

import org.json.JSONArray;
import org.json.JSONException;
import org.json.JSONObject;

public class ExceptionUtils {
    // Méthodes
    /** Effectue les différents test sur les valeurs fournis en paramètre pour nos Servlets
     * @param name Nom d'objet céleste, ne peut pas être null
     * @param month Mois de recherche, compris entre [1, 12], ou un mois de moins que celui courant pour l'année courante, ne peut pas être null
     * @param year Année de recherche, compris entre [1950, année en cours], ne peut pas être null
     */
    public static void testParametersServ(String name, String month, String year) {
        if (name == null || name.trim().isEmpty())
            throw new IllegalArgumentException("Le nom de l'objet n'a pas été renseigné !");

        if (month == null || year == null || month.trim().isEmpty() || year.trim().isEmpty())
            throw new IllegalArgumentException("La date n'a pas été renseigné !");

        if (month.matches("^[^0-9]+$"))
            throw new IllegalArgumentException("Le mois doit être exprimé au format numérique !");
        if (year.matches("^[^0-9]+$"))
            throw new IllegalArgumentException("L'année doit être exprimé au format numérique !");

        int monthInt = Integer.parseInt(month);
        int yearInt = Integer.parseInt(year);

        Calendar calendar = Calendar.getInstance();

        // Test si les valeurs fournis sont bien dans les bon intervalles
        if (monthInt <= 0 || monthInt > 12)
            throw new IllegalArgumentException("La valeur du mois n'est pas valide ! Elle doit être comprise entre 1 et 12 inclus !");
        if (yearInt <  1950)
            throw new IllegalArgumentException("La valeur de l'année n'est pas valide ! Elle doit être comprise supérieure à 1950 !");
        if (yearInt > calendar.get(Calendar.YEAR))
            throw new IllegalArgumentException("La valeur de l'année n'est pas valide ! Elle doit être inférieure ou égale à "+calendar.get(Calendar.YEAR));

        if (yearInt == calendar.get(Calendar.YEAR) && monthInt > calendar.get(Calendar.MONTH))
            throw new IllegalArgumentException("Les valeurs du mois et de l'année sont incompatibles ! La date maximal est d'un mois avant ce jour, soit le : "+
                (calendar.get(Calendar.MONTH) < 10 ? "0"+calendar.get(Calendar.MONTH) : calendar.get(Calendar.MONTH))+"/"+calendar.get(Calendar.YEAR)+" !");
    }

    /** Effectue des tests sur les List pour la méthode getDifferences prennant en paramètre une List de List de String
     * @param files List de List de String contenant plusieurs ligne de plusieurs fichiers à tester
     */
    public static void testListGetDifferencesListString(List<ArrayList<String>> files) {
        if (files.size() == 0)
            throw new NullPointerException("La List passé en paramètre est vide !");
    }

    /** Effectue des tests sur les List pour la méthode getDifferences prennant en paramètre une List de String
     * @param files List de String contenant plusieurs ligne d'un fichier à tester
     */
    public static void testListGetDifferencesString(List<String> files) {
        if (files.size() == 0)
            throw new NullPointerException("La List passé en paramètre est vide !");
    }

    /** Effectue des tests sur les List pour la méthode getDifferences retournant un entier
     * @param file1 List de String du premier fichier (contenant chacune des lignes du fichier)
     * @param file2 List de String du second fichier (contenant chacune des lignes du fichier)
     */
    public static void testListGetDifferencesInt(List<String> file1, List<String> file2) {
        if (file1.size() == 0 || file2.size() == 0)
            throw new NullPointerException("Le contenu d'au moins une des List est vide !");
        if (file1.size() < 7 || file2.size() < 7)
            throw new NullPointerException("Le contenu d'au moins une des List est incorrect, il manque des lignes au fichier !");
    }

    /** Effectue des tests sur la tailles des fichiers pour la méthode compareFile
     * @param file1 Chaîne de caractère contenant le contenu du premier fichier
     * @param file2 Chaîne de caractère contenant le contenu du second fichier
     */
    public static void testStringCompareFile(String file1, String file2) {
        if (file1 == null || file2 == null || file1.trim().isEmpty() || file2.trim().isEmpty())
            throw new NullPointerException("Les fichiers passés en paramètre sont vides, ou peut-être inexistants !");
    }
```

```java
    /** Effectue des tests sur la validité et l'existance du booleen de test
     * @param bool Booleen de test
     */
    public static void testBooleanClicButton(String bool) {
        if (bool == null || bool.trim().isEmpty())
            throw new NullPointerException("Le boolean de test n'a pas été renseigné !");
    }

    public static void testSuperiority(int year1, int year2) {
        if (year1 > year2)
            throw new IllegalArgumentException("Les années ne sont pas dans le bon ordre !");
    }

    /**
     * Soumet une réponse HTTP en JSON listant les erreurs données en paramètre.
     * @param lstErrors Liste des erreurs à retourner.
     * @param writer    Flux de sortie HTTP.
     */
    public static void writeErrorDocument(final List<String> lstErrors, final PrintWriter writer) {
        // Formate la liste des erreurs en JSON:
        String errorDoc;

        try {
            // Essaie d'abord proprement avec les classes d'org.JSON:
            JSONArray error = new JSONArray(lstErrors);
            JSONObject result = new JSONObject();
            result.put("error", error);
            errorDoc = result.toString();
        }catch(JSONException je) {
            // En cas d'erreur, écrit manuellement l'objet JSON:
            errorDoc = "{\"error\": ["+lstErrors.stream().map(n -> "\""+n+"\"").collect(Collectors.joining(","))+"]}";
        }

        // Envoie ce document d'erreur JSON:
        writer.append(errorDoc);
    }
}
```

Code source de mon fichier JavaScript :

```
var date = new Date();

//Labels du diagramme (recalculé à chaque rafraichissement du diagramme)
var labelDiagr = [];
// Annee du diagramme (recalculé à chaque rafraichissement du diagramme)
var anneeDiagr = [];
// Mois en String suivant une valeur numérique (ne change jamais)
var tabMonth = [
    "Jan", "Fev", "Mar", "Avr", "Mai", "Jui",
    "Jul", "Aoû", "Sep", "Oct", "Nov", "Déc"
];
// Boolean de test, permettant d'assurer un seul traitement dans le mode Journal à la fois
var isRunning = false;
//Booleen de test pour le scroll
var ticking = false;
//Position de scroll de la souris dans la page
var positionScroll = 0;
//Annee actuelle (diminue d'un an par un an après chaque affichage dans le journal)
var anneeActuJourn = date.getFullYear();
// Mois actuelle (ne change pas de valeur)
var moisActuJourn = (date.getMonth() < 10 ? '0'+date.getMonth() : date.getMonth());
//Mois actuelle sous forme de String (avec le 0 au début ou non) (évolue suivant le fomrulaire de recherche)
var month = moisActuJourn;

//Retourne un tableau de deux boolean, le premier indique si on calcule pour le mode journal, et le second pour le mode comparaison
function dataProcessing() {
    if ($("#modeDiary").is(":checked"))
        return [true, false];
    if ($("#modeCompare").is(":checked"))
        return [false, true];
}

// Crée et affiche le diagramme
function createDiagr(data) {
    console.debug("Diagramme en cours de création");
    if (data.error == null) {
        // Nombre de barres du diagramme
        var nbAxe = data.diagr.length;
        // Nombre d'année à afficher sur le diagramme
        var nbAnnees = Math.trunc((nbAxe-1)/12)+1;
        if (nbAxe != 0) {
            // Indicateur permettant de savoir quel mois afficher dans la légende
            var cptAxe = (nbAnnees-((data.diagr[0][0]-1) % nbAnnees)) % nbAnnees;
        }
```

```
        // Valeur de l'année au fil du parcours de la légende
        var diagYear = $("#date_annee2").val()-Math.trunc((nbAxe-1)/12);

        var ticks = [];
        // Remplissage de la paire [emplacement sur la légende / mois]
        for (var i = 0; i < nbAxe; i++) {
            labelDiagr[i] = data.diagr[i][0];
            if (data.diagr[i][0]-1 == 0)
                diagYear++;
            anneeDiagr[i] = diagYear;
            // Ajout à la légende
            if (cptAxe <= 0) {
                ticks[i] = [i, tabMonth[data.diagr[i][0]-1]];
                // S'il s'agit du mois de Janvier ajout de l'année
                if (data.diagr[i][0]-1 == 0) {
                    ticks[i][1] = ticks[i][1] + "\n"+diagYear;
                }
                cptAxe = nbAnnees-1;
            } else {
                cptAxe--;
            }
        }

        // Contient les valeur à représenter sur le diagramme, ainsi que la couleur de la barre
        var diffs = [nbAxe];
        // Boolean de test permettant de savoir si une barre a déjà été coloré
        var isColored = false;
        // Parcours du tableau inverse pour le cas où le mois selectionné serait le dernier du diagramme, et donc le même que le premier
        for (var i = nbAxe-1; i >= 0; i--) {
            // Si le mois selectionné corresponds bien au mois actuelle, et s'il s'agit en plus du dernier du diagramme
            if (!isColored && data.diagr[i][0] == $("#date_mois1").val() && nbAxe <= 13 && $("#date_annee1").val() == $("#date_annee2").val()) {
                diffs[i] = {data: [[i, data.diagr[i][1]]], color: "#65c6f0"}
                isColored = true;
            // Si le mois selectionné corresponds bien au mois actuelle, et s'il d'un mois compris entre les indices 0 et 12 du diagramme
            } else if (!isColored && data.diagr[i][0] == $("#date_mois1").val() &&
                        $("#date_annee1").val() != $("#date_annee2").val() &&
                        ((nbAxe >= 14 && i <= 11) ||
                        (nbAxe <= 13 && i < 11))) {
                diffs[i] = {data: [[i, data.diagr[i][1]]], color: "#65c6f0"};
                isColored = true;
```

```
            // Pas de coloration particulière
            } else
                diffs[i] = {data: [[i, data.diagr[i][1]]], color: "#5c84f0"};

        }

        var options = {
            series: {
                bars: {
                    show: true
                }
            },
            bars: {
                align: "center",
                barWidth: 1
            },
            xaxis: {
                ticks: ticks,
                // couleur des barres horizontales dans le diagrammes
                tickColor: "#6a6a6a"

            },
            yaxis: {
                // couleur des barres verticales dans le diagrammes
                tickColor: "#6a6a6a"
            },
            grid: {
                hoverable: true,
                clickable: true,
                borderWidth: 2,
                // Couleurs avec un dégradé en fond du diagramme
                backgroundColor: { colors: ["#414141", "#252525"] }
            }
        };
        $.plot($("#diagram"), diffs , options);
    } else {
        console.debug(data.error);
    }
}
```

```
// Effectue une recherche dans le mode Journal
function diarySearch() {
    $.ajax({
        type: 'GET',
        url: 'journal',
        data: {
            object_name: $("#nom").val(),
            date_month: moisActuJourn,
            date_year: anneeActuJourn
        },

        success: function(data) {
            console.debug("Journal en cours d'écriture");
            anneeActuJourn--;
            var contenu = "";
            if (data.error == null) {
                // ajout des éléments dans le tableau HTML
                for (var i = 0; i < data.diary.length; i++) {
                    contenu += "<tr>";
                    contenu += "<td class=\"tableDate\">"+tabMonth[data.days[i][0]]+" "+data.days[i][1]+"</td>";
                    contenu += "<td class=\"tableContenue\">"+data.diary[i].join('\n')+"</td>";
                    contenu += "</tr>";
                }
                $("#resultJournal").append(contenu);
            } else {
                $("#erreur").html(data.error.join('\n'));
                $("#resulJour").empty();
            }
        },

        error: function(result, statut, error) {
            console.debug("Problème formulaire !", result);
            console.debug("statut", statut);
            console.debug("error", error);
        },

        complete: function() {
            console.debug("Journal terminé !");
            $("#loading").hide();
            // indique que la requête est terminé
            isRunning = false;
        }
    });
```

```
// Effectue une recherche dans le mode de comparaison, pour le formulaire
function compareSearchForm(simpleComparaisonBool, directionDroiteBool) {
    $.ajax({
        type: 'GET',
        url: 'form',
        data: {
            object_name: $("#nom").val(),
            date_month1: $("#date_mois1").val(),
            date_year1: $("#date_annee1").val(),
            date_month2: $("#date_mois2").val(),
            date_year2: $("#date_annee2").val(),
            simpleComparaison: simpleComparaisonBool,
            directionDroite: directionDroiteBool
        },

        success: function(data) {
            if (data.error == null) {
                console.debug("Formulaire en cours d'écriture");
                if (directionDroiteBool == null) {
                    console.debug("Clique sur le bouton RECHERCHE, la touche ENTREE, ou un changement de mode");
                } else if (simpleComparaisonBool && directionDroiteBool) {
                    console.debug("Clique sur le bouton >");
                } else if (simpleComparaisonBool && !directionDroiteBool) {
                    console.debug("Clique sur le bouton <");
                } else if (!simpleComparaisonBool) {
                    console.debug("Clique sur le bouton << ou >>");
                    $("#date_mois1").val(data.dateActu[0]);
                    $("#date_annee1").val(data.dateActu[1]);
                    compareSearchDiagr();
                }
                $("#oldfile").html(data.oldFile.join('\n'));
                $("#newfile").html(data.newFile.join('\n'));
            } else {
                if (directionDroiteBool != null) {
                    if (simpleComparaisonBool && directionDroiteBool) {
                        if (month == 1) {
                            month = 12;
                            year--;
                        } else {
                            if (month < 11)
                                month = '0'+(month-1);
                            else
                                month--;
                        }
```

```
                    $("#date_mois1").val(month);
                    $("#date_annee1").val(year);
                } else if (simpleComparaisonBool && !directionDroiteBool) {
                    if (month == 12) {
                        month = "01";
                        year++;
                    } else {
                        if (month < 9) {
                            month++;
                            month = '0'+month;
                        } else
                            month++;
                    }
                    $("#date_mois1").val(month);
                    $("#date_annee1").val(year);
                }
            }
            $("#erreur").html(data.error.join('\n'));
            $("#oldfile").empty();
            $("#newfile").empty();
        }
    },

    error: function(result, statut, error) {
        console.debug("Problème formulaire !", result);
        console.debug("statut", statut);
        console.debug("error", error);
    },

    complete: function() {
        console.debug("Fin d'écriture du formulaire");
        month = $("#date_mois1").val();
        year = $("#date_annee1").val();
    }
});

month = $("#date_mois1").val();
year = $("#date_annee1").val();
}
```

```
// Effectue une recherche dans le mode de comparaison, pour le diagramme
function compareSearchDiagr() {
    $.ajax({
        type: 'GET',
        url: 'diagram',
        data: {
            object_name: $("#nom").val(),
            date_month1: $("#date_mois1").val(),
            date_year1: $("#date_annee1").val(),
            date_month2: $("#date_mois2").val(),
            date_year2: $("#date_annee2").val()
        },

        success: function(data) {
            if (data.error == null) {
                console.debug("Diagramme en cours de réalisation");
                createDiagr(data);
                $("#result").show();
            } else {
                $("#diagram").empty();
                $("#erreur").html(data.error.join('\n'));
            }
        },

        error: function(result, statut, error) {
            console.debug("Problème diagramme !", result);
            console.debug("statut", statut);
            console.debug("error", error);
        },

        complete: function() {
            console.debug("Diagramme terminé !");
            $("#loading").hide();
        }
    });
}

// Effectue le traitement du bouton de recherche (suivant le mode selectionné)
function submitButton(diary, compare) {
    $("#erreur").empty();
    $("#loading").show();
    if (diary)
        diarySearch();
    if (compare) {
        compareSearchForm(true, null);
        compareSearchDiagr();
    }
}
```

```
$(document).ready(function(){
    // Hauteur de la page
    var hauteurPage = $(document).height()-$(window).height();

    // Valeur maximal d'année suivant l'année en cours
    $("#date_annee1").attr("max", date.getFullYear());
    $("#date_annee2").attr("max", date.getFullYear());

    $("#date_mois1").val(month);
    $("#date_annee1").val(date.getFullYear()-1);
    $("#date_mois2").val(month);
    $("#date_annee2").val(date.getFullYear());

    // Valeur de la date en cours d'affichage (évolue suivant le fomrulaire de recherche)
    var year = $("#date_annee1").val();

    // Changement d'état des radios boutons
    $(".mode").on("change", function(e) {
        if ($("#modeCompare").is(":checked")) {
            $("#compar").show();
            $("#diagram").empty();
            $("#journal").hide();
            submitButton(false, true);
        } else {
            $("#compar").hide();
            $("#journal").show();
            submitButton(true, false);
        }
    });
    // Au clique du bouton Effacer, supprime le contenu du champs contenant le nom
    $("#clear").click(function() {
        $("#nom").val("");
    });


    // A chaque clique sur la combobox pour le champs des dates à droite, change l'état de modification
    $("#scales").click(function() {
        if ($("#scales").is(":checked")) {
            $("#date_mois2").prop("disabled", false);
            $("#date_annee2").prop("disabled", false);
        } else {
            $("#date_mois2").prop("disabled", true);
            $("#date_annee2").prop("disabled", true);
```

```
    }
});

// A chaque clique sur la combobox pour le champs de recherche personnalisée, change l'état de modification
$("#advanced").click(function() {
    if ($("#advanced").is(":checked")) {
        $("#categories").prop("disabled", false);
    } else {
        $("#categories").prop("disabled", true);
    }
});

// Envoie des données pour recevoir le contenu des fichiers (après un clique sur le bouton de recherche)
$("#submit").click(function() {
    anneeActuJourn = date.getFullYear();
    moisActuJourn = (date.getMonth() < 10 ? '0'+date.getMonth() : date.getMonth());
    $("#resultJournal").empty();
    var bool = dataProcessing();
    submitButton(bool[0], bool[1]);
});

// Effectue une recherche lors d'un appuis sur la touche ENTREE n'importe où dans la page
$("#nom").keypress(function(e) {
    if (e.which != 13)
        return;
    anneeActuJourn = date.getFullYear();
    moisActuJourn = (date.getMonth() < 10 ? '0'+date.getMonth() : date.getMonth());
    $("#resultJournal").empty();
    var bool = dataProcessing();
    submitButton(bool[0], bool[1]);
});

// Affiche les données à la date un mois avant (après un clique sur le bouton < )
$("#prec").click(function() {
    if ($("#result").is(":visible")) {
        $("#loading").show();
        $("#erreur").empty();

        if (month == 1) {
            month = 12;
            year--;
        } else {
            if (month < 11)
                month = '0'+(month-1);
            else
                month--;
        }
        $("#date_mois1").val(month);
        $("#date_annee1").val(year);

        compareSearchForm(true, false);
        compareSearchDiagr();
    }
});

// Affiche les données à la date un mois aprés (après un clique sur le bouton > )
$("#suiv").click(function() {
    if ($("#result").is(":visible")) {
        $("#loading").show();
        $("#erreur").empty();

        if (month == 12) {
            month = "01";
            year++;
        } else {
            if (month < 9) {
                month++;
                month = '0'+month;
            } else
                month++;
        }
        $("#date_mois1").val(month);
        $("#date_annee1").val(year);

        compareSearchForm(true, true);
        compareSearchDiagr();
    }
});

// Affiche les données à une date antérieur dès qu'il y a une modification (après un clique sur le bouton << )
$("#precExt").click(function() {
    if ($("#result").is(":visible")) {
        $("#loading").show();
        $("#erreur").empty();
```

```
        compareSearchForm(false, false);
    }
});

// Affiche les données à une date supérieur dès qu'il y a une modification (après un clique sur le bouton >> )
$("#suivExt").click(function() {
    if ($("#result").is(":visible")) {
        $("#loading").show();
        $("#erreur").empty();
        compareSearchForm(false, true);
    }
});

// Affiche les informations d'un objet céleste dans le mode journal suivant le scroll sur la page
$(window).scroll(function(){
    if ($("#modeDiary").is(":checked")) {
        positionScroll = window.scrollY;
        if (!ticking) {
            window.requestAnimationFrame(function() {
                // Valeur minimal (5% de l'écran en partant du bas) pour activer le chargement des données
                if (!isRunning && anneeActuJourn > 1950 && positionScroll > $(document).height()-0.05*$(document).height()-$(window).height()) {
                    isRunning = true;
                    submitButton(true, false);
                }
                ticking = false;
            });
        }

        ticking = true;
    }
});

// Effectue une recherche lors d'un clique sur une barre du diagramme (modifie le resultat de recherche, les champs associé, et le diagramme)
$("#diagram").bind("plotclick", function(event, pos, obj){
    if (!obj)
        return;
    $("#date_mois1").val(labelDiagr[obj.seriesIndex] < 10 ? '0'+labelDiagr[obj.seriesIndex] : labelDiagr[obj.seriesIndex]);
    $("#date_annee1").val(anneeDiagr[obj.seriesIndex]);
    submitButton(false, true);
});
```

```
    // Changement de cursor suivant la position du diagrame
    $("#diagram").bind("plothover", function(event, pos, obj){
        if(obj)
            $("#diagram").css("cursor","pointer","important");
        else
            $("#diagram").css("cursor","default", "important");
    });
});
```

Architecture du projet