

ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
UNIVERSITÉ DU QUÉBEC

RAPPORT DE PROJET PRÉSENTÉ À
L'ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

COMME EXIGENCE PARTIELLE
À L'OBTENTION DE MAÎTRISE AVEC PROJET EN GENIE LOGICIEL

PAR
Victor DUPUY

EVALUATION ET REINGENIERIE DU PROJET GOAT

MONTREAL, LE 15 AVRIL 2016



Victor Dupuy, 2016



Cette licence [Creative Commons](https://creativecommons.org/licenses/by-nc-nd/4.0/) signifie qu'il est permis de diffuser, d'imprimer ou de sauvegarder sur un autre support une partie ou la totalité de cette œuvre à condition de mentionner l'auteur, que ces utilisations soient faites à des fins non commerciales et que le contenu de l'œuvre n'ait pas été modifié.

PRÉSENTATION DU JURY

CE RAPPORT DE PROJET A ÉTÉ ÉVALUÉ

PAR UN JURY COMPOSÉ DE :

Professeur Alain April, directeur de projet
Département de génie logiciel et TI à l'École de technologie supérieure

Dr. Pavel Hamet, co-directeur de projet
Département de médecine de l'Université de Montréal

Dr. Michael Philipps, jury externe
Precision Medecine Advertiser Inc

Witold Suryn, jury
Département de génie logiciel et TI à l'École de technologie supérieure

REMERCIEMENTS

Pour leur aide, patience, et leurs conseils ainsi que toutes les connaissances et expériences que j'ai pu acquérir grâce à eux, je tiens à remercier :

Professeur Alain APRIL (directeur de recherche, département de génie logiciel et des TI à l'École de Technologie Supérieure) pour avoir accepté de m'engager sur ce projet, ainsi pour la confiance et l'autonomie qu'il m'a octroyé ces derniers mois.

Docteur Pavel HAMET (codirecteur de recherche, chercheur et professeur de médecine à l'Université de Montréal) pour m'avoir accepté dans son service de recherche.

Docteur Michael PHILLIPS (spécialiste en médecine de précision) pour m'avoir accompagné dans ce projet au sein du CRCHUM et permis de bien saisir les enjeux du projet GOAT.

Béatriz KANZKI (bio-informaticienne stagiaire) pour sa collaboration tout au long du projet.

Le reste de l'équipe du laboratoire de recherche du Dr. Pavel Hamet au CRCHUM, qui a su m'accueillir avec bienveillance et a toujours été à l'écoute de mes propositions.

EVALUATION ET REINGENIERIE DU PROJET GOAT

Victor DUPUY

RÉSUMÉ

Ce rapport a pour but de présenter ma démarche d'évaluation et de réingénierie du projet « Genetic Output Analysis Tool » (GOAT), au sein de l'équipe de recherche du docteur Pavel Hamet au Centre de Recherche du Centre Hospitalier Universitaire de Montréal (CRCHUM), en collaboration avec l'École de Technologie Supérieure (ETS).

GOAT est une application web destinée à assister les chercheurs en génomique dans leurs recherches, en leur fournissant des graphes interactifs et des tableaux dynamiques, afin qu'ils puissent explorer rapidement et facilement les données en leur possession, très volumineuses. L'objectif du projet est de faire d'ouvrir GOAT en open-source, afin de maximiser son utilisation dans la communauté des chercheurs et de permettre l'ajout de fonctionnalités par d'autres développeurs externes.

A mon arrivée dans l'équipe du projet, GOAT consistait en une version presque fonctionnelle, mais pas maintenable et sans potentiel d'évolution. L'objectif de ce projet en maîtrise appliquée, de 15 crédits, visait à évaluer l'état du projet GOAT et à proposer puis mettre en place des solutions de réingénierie, afin d'atteindre les standards de qualité requis pour un tel outil.

Ce rapport permet au lecteur de suivre ma démarche pas-à-pas, à savoir une découverte du contexte, puis une évaluation du projet, suivie d'une phase de conception et enfin d'une phase de développement.

Mots-clés : réingénierie du logiciel, logiciel de visualisation génétique.

EVALUATION AND REFACTORING OF THE GOAT PROJECT

Victor DUPUY

ABSTRACT

This report aims to present my approach for the evaluation and reengineering of the « Genetic Output Analysis Tool » (GOAT) project, in Doctor Pavel Hamet's research team inside the CRCHUM, in collaboration with the ETS.

GOAT is a web application destined to assist the genomic researchers in their research, by supplying interactive plots and dynamic data grids, so that they can quickly and easily explore their data, which is massive.

The objective of the project is to open-source GOAT, in order to maximize its use in the research community, and to allow other external developers to add new features.

When I arrived in the project team, GOAT consisted of an almost functional version, but not maintainable nor upgradable or scalable. The objective of this Master degree project of 15 credits, was to evaluate the state of the GOAT project, and to propose then implement reengineering solutions, in order to reach the quality standards of such a tool.

This report allows the reader to follow my approach step by step, namely a context discovery phase, then a project evaluation phase, followed by a conception phase, and finally a development phase.

Keywords : software refactoring, genetic visualisation software.

TABLE DES MATIÈRES

	Page
INTRODUCTION	1
CHAPITRE 1 CONTEXTE	3
1.1 Le CRCHUM.....	3
1.2 Le service	3
1.3 Le projet GOAT.....	3
1.3.1 Objectifs	3
1.3.2 Logiciel libre	5
1.4 Technologies utilisées	5
1.4.1 Back-end	5
1.4.2 Front-end.....	6
1.4.3 Stockage des données	6
1.5 Ma mission	6
1.6 CONCLUSION.....	7
CHAPITRE 2 ETAT DE L'ART – REINGENIERIE.....	9
2.1 Définition.....	9
2.2 Application	9
2.2.1 À petite échelle	9
2.2.2 À grande échelle	10
2.3 Quand refactorer?	10
2.3.1 Code dupliqué.....	11
2.3.2 Modules avec trop de responsabilités	11
2.3.3 Nommage peu représentatif	11
2.4 Conclusion.....	12
CHAPITRE 3 EVALUATION DU PROJET GOAT	13
3.1 Fonctionnalités.....	13
3.2 Méthodologie de travail en place.....	14
3.3 Architecture Logicielle.....	14
3.4 Défauts de qualité interne majeurs.....	15
3.4.1 Couplage très élevé.....	15
3.4.2 Cohésion faible	15
3.4.3 Routage incohérent	16
3.4.4 Peu de validation des champs d'entrée utilisateur.....	16
3.4.5 Données sensibles en clair sur Github	16
3.4.6 Front-end non responsive.....	17
3.4.7 Code difficile à déchiffrer	17
3.5 Conclusion.....	17

CHAPITRE 4 REINGENIERIE – CONCEPTION.....	19
4.1 Modifier ou recommencer ?.....	19
4.2 Appréhension du domaine	19
4.3 Exigences.....	20
4.3.1 Exigences fonctionnelles.....	21
4.3.2 Exigences non-fonctionnelles.....	21
4.4 Architecture Logicielle	22
4.4.1 Architecture logicielle haut niveau	22
4.4.2 Architecture logicielle front-end.....	24
4.4.2.1 Description	24
4.4.2.2 Fonctionnalités	26
4.4.2.3 Visuels	27
4.4.3 Back-end.....	27
4.4.3.1 Description	27
4.4.3.1 Modules	28
4.4.3.1.1 Web.....	28
4.4.3.1.2 AreaSelection	29
4.4.3.1.3 Manhattan	29
4.4.3.1.4 SNP for one Gene.....	30
4.4.3.1.1 Autocomplete	30
4.4.3.1.1 Phenotypes	30
4.4.3.1.1 Tools	31
4.5 Conclusion	31
CHAPITRE 5 REINGENIERIE – DEVELOPPEMENT	33
5.1 Outils de développement	33
5.1.1 JetBrains PyCharm.....	33
5.1.2 Atom.....	33
5.1.3 Cmdr	33
5.2 Technologies de développement.....	34
5.2.1 Contrôle de version	34
5.2.2 Build automatique.....	35
5.2.2.1 Babel.....	35
5.2.2.2 Browserify.....	35
5.2.2.3 Gulp	36
5.3 Technologies de l'application.....	36
5.3.1 Front-end	36
5.3.1.1 React.js.....	36
5.3.1.2 Flux	40
5.3.1.3 Griddle	42
5.3.1.4 Bootstrap	42
5.3.2 Back-end.....	42
5.3.2.1 Django.....	42
5.3.2.2 Bokeh	43

CHAPITRE 6 DISCUSSION.....	45
6.1 Bilan de l'expérience	45
6.1.1 Positif	45
6.1.2 Difficultés rencontrées	46
6.2 Recommandations.....	46
CONCLUSION	49
BIBLIOGRAPHIE	51
ANNEXE I EXIGENCES FONCTIONNELLES	55
ANNEXE II EXIGENCES NON-FONCTIONNELLES	57
ANNEXE III FONCTIONNALITES ET ETATS « FRONT-END »	59
ANNEXE IV CAS D'UTILISATION DE L'INTERFACE DE GOAT	61
ANNEXE V MODULES « BACK-END »	71
ANNEXE VI VISUELS	75

LISTE DES FIGURES

	Page
Figure 1-1 : Capture d'écran de l'application actuellement utilisée au CRCUM	4
Figure 4-1 : Vue en couches de l'architecture de GOAT	23
Figure 4-2 : Vue Composants et Connecteurs de GOAT	24
Figure 4-3 – Fonctionnement de la SPA de GOAT	25
Figure 4-4 : Vue modulaire de l'architecture « back-end »	28
Figure 5-1 : Input dont la valeur entrée est invalide.....	39
Figure 5-2 : Input dont la valeur entrée est valide.....	39
Figure 5-3 : Fonctionnement de Flux	41
Figure 5-4 : Exemple de graphe Bokeh [32] (directement tiré de la documentation).....	43

LISTE DES ABRÉVIATIONS, SIGLES ET ACRONYMES

ETS : École de technologie Supérieure.

CRCHUM : Centre de Recherche du Centre Hospitalier Universitaire de Montréal.

SPA : Single Page Application.

SNP : Single-Nucleotide Polymorphism, ou Polymorphisme d'un seul nucléotide.

GOAT : Genetic Output Analysis Tool.

GWAS : Genome-Wide Association Study.

HTML : HyperText Markup Language

CSS : Cascading Style Sheet

SQL : Structured Query Language

DOM : Document Object Model

JSON : JavaScript Object Notation

AJAX : Asynchronous JavaScript and XML

UdeM : Université de Montréal

SASS : Syntactically Awesome Style Sheets

IDE : Integrated Development Environment, ou Environnement de Développement

LISTE DES ALGORITHMES

Algorithme 2-1 : Deux classes JavaScript identiques, la seconde avec un nommage représentatif du contenu	12
Algorithme 5-1 : Tableau contenant la liste des noms de modules.....	37
Algorithme 5-2 : Classe React renvoyant un composant « MenuItem ».....	37
Algorithme 5-3 : Composant React créant dynamiquement un menu	38

INTRODUCTION

Ces dernières années, les médecins chercheurs en génomiques ont eu accès à des données dont le volume et la complexité croissent très rapidement. De fait, il est maintenant vital pour leurs recherches qu'ils aient à leur disposition des logiciels capables de récupérer, traiter et afficher ces données dans une forme utile et compréhensible par un humain. Avec les technologies modernes, notamment celles tirées du web, il est désormais possible de réaliser cela dans un délai acceptable. Cela permet d'automatiser un grand nombre des tâches des bio-informaticiens afin qu'ils puissent mobiliser leur temps à des opérations moins répétitives.

Le logiciel Genetic Output Analysis Tool (GOAT) est un projet conjointement mené par une équipe d'étudiants de l'École de Technologie Supérieure (ÉTS) et des employés du Centre de Recherche du Centre Hospitalier Universitaire de Montréal (CRCHUM). Ils travaillent à ce projet de logiciel libre sous la supervision du professeur Alain April. L'objectif du projet est de réaliser une interface web permettant aux chercheurs, statisticiens et bio-informaticiens de l'équipe du docteur Pavel Hamet, d'accéder aux données génomiques disponibles dans son laboratoire de recherche en diabète et d'en tirer rapidement des conclusions à l'aide de tableaux et de graphes interactifs.

Il s'agit d'un projet de recherche appliquée de 15 crédits concluant une maîtrise en génie logiciel effectuée à l'ÉTS. Mon rôle dans le cadre de ce projet fût d'évaluer l'état de la qualité interne du prototype GOAT dans sa version en date du début janvier 2016, puis de faire des propositions de réingénierie, et d'en effectuer un maximum dans le temps qui m'était imparti.

CHAPITRE 1

CONTEXTE

1.1 Le CRCHUM

Le projet de recherche a lieu au CRCHUM, le centre de recherche du Centre Hospitalier Universitaire de Montréal, lui-même rattaché à l'Université de Montréal (UdeM).

1.2 Le service

Le service dans lequel j'ai travaillé, ces derniers mois, tente d'établir un traitement ciblé pour chaque patient du diabète et de certaines maladies cardiovasculaires. Pour cela, il dispose des données génomiques de plusieurs cohortes de volontaires, et utilise les données d'ADN pour la recherche.

1.3 Le projet GOAT

1.3.1 Objectifs

L'équipe de recherche du Dr. Hamet utilise actuellement une application web créée à l'interne, qui permet de faire des requêtes dans leur base de données. Les membres de l'équipe obtiennent en retour un grand nombre d'informations, affichées dans des tableaux statiques.

**KNOWLEDGE
DATABASE :
HOME**

Search by marker
RS_id

Search by gene
Symbol

List of markers
RS_list

List of gene
Symbol list

Search by gene
Name

Search Phenotype
Advance 2313 patients ☐
Advance 3449 patients ☐
Patents ☐
Monica ☐
Meta-analysis ☐
Incident ☐

Figure 1-1 : Capture d'écran de l'application actuellement utilisée au CRCUM

L'objectif du logiciel libre GOAT est d'offrir une version plus « moderne » de cette application, en fournissant des fonctionnalités supplémentaires comme des graphes interactifs affichant les biomarqueurs « Single-Nucleotide Polymorphism » (SNPs) significatifs, des liens dynamiques vers des bases de données de référence extérieures au laboratoire, des comparaisons entre phénotypes, et bien plus. Ces tâches nécessitent de récupérer les données au niveau « back-end » du logiciel GOAT et de les traiter avant de les envoyer au « front-end » de GOAT qui se chargera de les afficher de manière appropriée. Les utilisateurs finaux de GOAT seront des médecins chercheurs en génomiques et des statisticiens partout dans le monde.

La version actuelle a déjà connu un travail de réingénierie du « front-end », effectuée par Cédric Urvoy dans le cadre de son projet de Maîtrise de 15 crédits. Les détails de son intervention sont disponibles ici : <http://publicationslist.org/data/a.april/ref-514/RapportMaitriseTi%20-CedricUrvoy.pdf>.

1.3.2 Logiciel libre

Il a été décidé que GOAT serait distribué sous licence Apache v2 [1], c'est-à-dire qu'il s'agira d'un logiciel libre. L'intérêt de cette démarche est qu'elle permet d'impliquer d'autres personnes volontaires dans le développement et l'évolution future de l'application. De plus, cela implique de penser l'architecture logicielle de GOAT de façon particulière afin de faciliter sa compréhension et l'ajout futur de fonctionnalités en minimisant le risque de dégrader la qualité interne de la version de référence. Plus de précisions seront disponibles concernant les implications de ce choix sur l'architecture logicielle, au chapitre 4 : Conception de ce rapport.

1.4 Technologies utilisées

Sans empiéter sur les informations qui seront présentées au chapitre 3 : Évaluation, il est quand même intéressant de rappeler toutes les technologies qui étaient utilisées pour développer le prototype de GOAT à mon arrivée début janvier 2016.

1.4.1 Back-end

Le « back-end » de l'application GOAT, c'est-à-dire le côté serveur, utilisait le cadriciel Django, basé sur le langage de programmation Python. Ce choix avait été motivé car la plupart des bio-informaticiens connaissent bien le Python et pourraient donc maintenir et modifier GOAT dans le futur, sans avoir à apprendre un nouveau langage. Django [2] est un cadriciel créé pour faciliter la création d'applications web au niveau « back-end » d'un logiciel. Il permet la création d'applications parallèles accessibles via un routeur relié à un serveur web. Il est basé sur une architecture « Model-View-Template » qui d'envoyer au « front-end » du logiciel GOAT des vues générées dynamiquement à partir des données récupérées en base de données et des gabarits « HyperText Markup Language » (HTML) [3].

1.4.2 Front-end

Le « front-end », c'est-à-dire la partie client de l'application, était basée sur plusieurs technologies. D'abord, les gabarits HTML transmis par Django incluant les données demandées par l'utilisateur. Il s'agit simplement d'un fichier HTML dans lequel certains champs sont remplis et/ou répétés en fonction des données. L'apparence de l'application était écrite en SASS [4] compilé vers du « Cascading Style Sheet » (CSS) [5] à l'aide de la technologie Gulp [6]. L'interaction utilisateur était basée sur du JavaScript avec la librairie JQuery [7]. Enfin, les tableaux générés à partir des données reçues par le serveur étaient basés sur la technologie de Facebook React.js, combinée à un plugin spécialisé nommé Griddle [8].

1.4.3 Stockage des données

Les données étaient stockées dans une base de données MySQL [9]. Il s'agit d'une technologie libre de gestion de bases de données relationnelles. MySQL vient avec un langage de requêtes nommé SQL [10], qui permet de lire, modifier, insérer ou détruire des données dans la base.

1.5 Ma mission

Ma mission était, en premier lieu, d'évaluer l'état de la qualité interne du prototype GOAT, tant sur le plan technique que méthodologique et architectural. Une fois cette tâche effectuée, il me fallait proposer des pistes d'améliorations et de réingénierie logicielle, puis en effectuer un maximum. L'objectif implicite était d'introduire le plus grand nombre de pratiques exemplaire possibles afin que le projet GOAT soit et demeure modifiable et maintenable à l'avenir pour la communauté libre.

1.6 CONCLUSION

Ce chapitre a introduit les objectifs du projet de recherche appliquée, le domaine d'affaire impliqué, le contexte de réalisation ainsi que les technologies du domaine du libre impliquées. Le suivant présente l'état de l'art en réingénierie logicielle.

(1.1)

CHAPITRE 2

ETAT DE L'ART – REINGENIERIE

Ce chapitre a pour vocation de rappeler l'état de l'art dans le domaine de la réingénierie logicielle. Il se base en grande partie sur le contenu du livre *Code Complete (2nd Edition)* de Steve McConnell [31], utilisé dans le cadre de la maîtrise en génie logiciel de l'ETS.

2.1 Définition

Dans son livre *Code Complete (2nd Edition)*, Steve McConnell reprend une définition de Martin Fowler (1999), qui qualifie la réingénierie de « changement effectué sur la structure interne d'un logiciel pour le rendre plus simple à comprendre et moins cher à modifier, sans changer son comportement apparent » (traduit de l'anglais). La réingénierie permet donc d'obtenir un logiciel maintenable, modifiable et compréhensible sans perturber son bon fonctionnement.

2.2 Application

2.2.1 À petite échelle

Un logiciel peut tout à fait posséder une architecture logicielle de qualité, mais avoir été mal programmé. Si aucune pratique exemplaire du génie logiciel n'a été utilisée dans la phase de développement, il est alors probable que le produit livré soit rempli de défauts techniques, entraînant de fait une qualité interne faible. Dans ce genre de cas, le code est souvent mal structuré et/ou mal écrit et commenté.

La réingénierie à petite échelle du code d'un logiciel est utile quand utilisée au fur et à mesure du développement. Cela permet de garder le code clair et concis, ce qui augmente sensiblement la maintenabilité du logiciel. D'autre part, la réingénierie ne devant pas modifier le comportement apparent d'un logiciel, il est nécessaire d'écrire des tests qui seront

relancés à chaque modification du code. Ainsi, le développeur peut facilement voir quel est l'impact des modifications qu'il effectue, et corriger ces dernières en cas de problème.

2.2.2 À grande échelle

L'architecture d'un logiciel dépend en grande partie des attributs de qualité internes décidés lors de la phase d'identification des exigences. Quand ces attributs ne sont pas pris en compte dans la conception d'un logiciel, ce dernier voit sa qualité se dégrader avec le temps et les modifications apportées. Afin de pallier à cela, il est parfois nécessaire de modifier l'architecture du logiciel pour correspondre réellement aux attributs de qualité.

De plus, l'ajout de fonctionnalités non prévues au début d'un projet peut amener les développeurs à coder ces dernières sans suivre l'architecture prévue au départ. C'est particulièrement le cas quand un logiciel est en phase de production, et que le moindre changement peut entraîner des erreurs empêchant les utilisateurs d'utiliser le programme. Il arrive alors que l'équipe de développement cherche à rapidement fournir les nouvelles fonctionnalités, sans se soucier de l'impact de leurs ajouts sur la qualité du logiciel.

Appliquer une réingénierie à cette échelle sur un logiciel peut cependant s'avérer risqué, car cela entraîne souvent l'apparition d'un grand nombre d'erreurs dans le fonctionnement de ce dernier. Si les conséquences sont assez minimes si le logiciel est dans sa phase de développement, cela peut poser problème en phase de production. En effet, les utilisateurs ne peuvent alors plus utiliser le logiciel dans des conditions optimales, voire plus du tout. En cas de désaccord majeur entre l'état d'un logiciel et les attributs de qualité prévus au départ du projet, il est préférable de reconcevoir entièrement l'architecture, en sélectionnant soigneusement les éléments de qualité présents dans la version défailante.

2.3 Quand refactorer?

Il existe un grand nombre de pratiques de réingénierie, chacune adaptée à un contexte particulier. Cette partie présente trois contextes dans lesquelles il est conseillé de les appliquer.

2.3.1 Code dupliqué

Si le code contient plusieurs fois les mêmes instructions, alors il est probable que la moindre modification d'une de ces répétitions nécessite de modifier toutes les autres. Cela fait perdre du temps au développeur. De plus, si ce dernier oublie de modifier une partie des répétitions, le logiciel ne fonctionnera pas comme prévu.

Il est alors conseillé de regrouper ces répétitions sous un-e même module/classe/fonction, qui sera appelé-e quand nécessaire. Une modification ne devra alors être effectuée qu'une seule fois.

2.3.2 Modules avec trop de responsabilités

Il arrive que des modules aient un grand nombre de fonctionnalités différentes, voire même qu'ils aient des effets collatéraux sur le fonctionnement d'autres modules. On dit alors que la cohésion de ce module est faible (le concept de cohésion est expliqué plus en détail dans le chapitre 3 : Évaluation du rapport). Un module ne devrait pas avoir plusieurs responsabilités, mais se concentrer sur une fonctionnalité du logiciel. Ainsi, un changement sur la fonctionnalité n'impliquera pas de rechercher longtemps les modules à modifier. Il est donc parfois nécessaire de séparer un module en plusieurs modules indépendants, chacun chargé d'une tâche précise.

2.3.3 Nommage peu représentatif

Un code doit être facilement compréhensible quelque-soit le développeur qui le lit, même si ce dernier n'a pas participé au développement. Il arrive parfois que les noms de classes, fonctions ou variables soient peu représentatifs de leur contenu. Le développeur perd alors beaucoup de temps à analyser le code et comprendre son utilité, avant même de pouvoir le modifier. La figure 2-1 montre, de façon exagérée, à quel point des noms de variables, classes et fonctions représentatifs de leur contenu peut aider à comprendre rapidement du code.

Algorithme 2-1 : Deux classes JavaScript identiques, la seconde avec un nommage représentatif du contenu

```
// Version avec noms peu représentatifs
class P{
  constructor(n, a){
    this.n = n;
    this.a = a;
  }
  changeN(n){
    this.n = n;
  }
};

// Version après réingénierie
class Player{
  constructor(name, age){
    this.name = name;
    this.age = age;
  }
  changeName(newName){
    this.name = newName;
  }
};
```

2.4 Conclusion

Ce chapitre a permis de mettre en évidence ce qu'est la réingénierie d'un logiciel et les bénéfices qu'elle apporte quand elle est appliquée correctement. Le chapitre suivant présente l'évaluation de la qualité interne de GOAT et liste les observations basées sur la lecture du code de l'application et la synthèse des discussions avec les membres de l'équipe de projet.

CHAPITRE 3

EVALUATION DU PROJET GOAT

Ce chapitre présente les observations effectuées lors de la phase d'évaluation du projet, à savoir une analyse des fonctionnalités fournies par la version prototype, de la méthodologie de travail mise en place, de l'architecture logicielle et une liste des défauts techniques identifiés.

3.1 Fonctionnalités

Lors de l'évaluation de la qualité interne du prototype GOAT, les fonctionnalités offertes étaient les suivantes :

- **Login** : Un système de session était en place, permettant aux utilisateurs inscrits d'accéder à l'application seulement s'ils étaient connectés avec leurs identifiants et mot de passe. Dans le cas contraire, ils étaient redirigés vers une page de login.
- **Graphe Manhattan statique et tableau de SNPs significatifs** : En rentrant un phénotype, un ou plusieurs gènes, et un ou plusieurs SNPs (par leur rsID), l'utilisateur pouvait accéder après environ cinq minutes à une image statique représentant le graphe Manhattan du GWAS souhaité, ainsi qu'un tableau listant tous les SNPs significatifs du phénotype identifié. Le temps d'attente était d'environ **cinq minutes** une fois la requête effectuée avant que ne s'affiche ces informations et image.
- **Area Selection** : En cliquant sur un SNP particulier dans le tableau précédent, il était possible pour l'utilisateur d'accéder à un graphe interactif représentant une portion plus restreinte du graphe Manhattan, centrée autour de ce SNP, accompagné d'un tableau listant tous les SNPs locaux. L'application mettait environ 4 à 5 secondes avant d'afficher ce graphe et ce tableau. De plus, la fonctionnalité ne fonctionnait pas tout le temps, et faisait parfois planter le serveur Django, résultant en une erreur affichée directement dans le navigateur et forçant l'utilisateur à rafraîchir la page pour utiliser GOAT de nouveau.

3.2 Méthodologie de travail en place

Il n'y avait pas de méthodologie de travail établie, à savoir que tout membre de l'équipe avec une idée la proposait à un des étudiants travaillant sur le projet, sans suivre un processus particulier ni documenter quoi que ce soit. Il y avait pourtant un document de vision du projet, décrivant ses objectifs et fonctionnalités prévues. Il semblerait que l'absence de processus établi pour ajouter ou modifier une fonctionnalité ait entraîné le fait que l'équipe n'a pas mis-à-jour le document avec les nouveaux besoins exprimés. Cela a mené à une accumulation de fonctionnalités listées « à venir », souvent mal définies et s'écartant parfois des besoins exprimés en premier lieu. Par exemple, une fonctionnalité de comparaison de plusieurs phénotypes a été demandée, et ce sans préciser si les chercheurs souhaitaient un graphe, un tableau listant les SNPs significatifs communs, deux tableaux côte à côte, ou les trois en même temps.

Voici un autre exemple. La fonctionnalité d'accès à un GWAS se fait actuellement en entrant le nom d'un gène dans un formulaire. Lors d'une discussion avec un statisticien de l'équipe, il a été soulevé qu'il voulait plutôt une liste des SNPs significatifs présents sur ce gène.

Enfin, certains membres de l'équipe avaient commencé à modifier l'application alors même qu'elle n'en était qu'à sa version prototype, résultant en une série de défauts qui rendait GOAT inutilisable.

3.3 Architecture Logicielle

Pour préciser la situation, aucune architecture logicielle n'a été conçue, par la bio-informaticienne responsable, avant de coder GOAT. Même si l'étudiant David Lauzon a préparé une architecture logicielle à l'aide du cadriciel Django, il a été ignoré. Le « back-end » de GOAT a été codé en découvrant le cadriciel Django, ce qui a eu pour conséquence de ne pas vraiment suivre les recommandations de ce dernier.

De plus, le code « back-end » fourni était extrêmement procédural et était contenu dans un seul fichier, impliquant une absence totale de modularité résultant en un couplage extrêmement important. En effet, chaque fonctionnalité était liée aux autres. Concrètement, cela signifie qu'il était impossible d'accéder à la fonctionnalité « Area Selection » sans

passer par la génération du graphe Manhattan (qui prenait cinq minutes à afficher les données) !

3.4 Défauts de qualité interne majeurs

3.4.1 Couplage très élevé

Un couplage fort est présent quand les différents modules d'une application sont très liés les uns aux autres, et que modifier l'un a pour conséquence de modifier le comportement des autres.

Dans le prototype GOAT, chaque fonctionnalité était complètement liée aux autres. En effet, les données étaient toutes chargées lors de l'accès au graphe Manhattan puis conservées dans une variable globale, côté serveur. Ces données étaient ensuite traitées de différentes manières en fonction de la fonctionnalité choisie par l'utilisateur.

Cette conception résultait en une attente de cinq minutes pour avoir l'information d'un simple tableau et d'un graphe statique (c.-à-d. une image au format PNG) même si l'utilisateur souhaitait accéder à « Area Selection ».

De plus, modifier une fonctionnalité très couplée porte un grand risque d'impacter les autres, ce qui ralentissait l'évolution de GOAT. En effet, si l'accès au graphe statique et au tableau faisait défaut, il était alors impossible d'utiliser le reste de l'application.

3.4.2 Cohésion faible

La cohésion représente le degré de spécialisation des éléments d'un programme. Il est de conseillé de viser une forte cohésion dans son application, car cela signifie que chaque module est dédié à une tâche spécifique, limitant ainsi les risques de contamination à d'autres modules en cas de modification. La cohésion forte est souvent associée à un couplage faible, et vice-versa.

Ici, en plus d'être liées, les différentes fonctionnalités étaient toutes contenues dans une même fonction qui contenait quasiment toute la logique de l'application. C'est l'inverse même de ce qu'il faut faire, à savoir viser une cohésion forte.

3.4.3 Routage incohérent

En temps normal, une application web contient un module nommé « routeur », chargé de rediriger les requêtes HTML du « front-end » vers les parties correspondantes du « back-end ». Ces requêtes sont majoritairement de deux types : « GET », servant à obtenir des données ou fichiers, et « POST », servant à envoyer des données au serveur.

GOAT est une application qui ne fait que recevoir des données pour les afficher de différentes façons. Ainsi, un ingénieur logiciel s'attend à trouver plusieurs routes « GET » correspondant à chaque fonctionnalité. Hors, l'ensemble des fonctionnalités de l'application était situées derrière une route « POST : /SGene ». La fonction contenant toute la logique serveur redirigeait la requête utilisateur vers la fonctionnalité correspondante en fonction des données envoyées dans cette dernière.

Cette conception possède un défaut majeur lors de la redirection d'une requête HTML, car cela engendre un code de contrôle de la requête (c.-à-d. pour rediriger vers la bonne fonctionnalité) extrêmement long et inutile : c'est normalement le rôle du routeur. La gestion des routes est d'ailleurs normalement confiée à un module natif du cadre Django, ne nécessitant qu'une configuration minime.

3.4.4 Peu de validation des champs d'entrée utilisateur

L'utilisateur pouvait entrer n'importe quelle information dans les champs d'entrée de GOAT. En cas de requête non valide, l'utilisateur recevait le même message que si sa recherche ne donnait pas de résultat pour une entrée valide. Ainsi, l'entrée « *jdsjgff[{}@--_ç)* » retournait « No phenotype ! ». De la même manière, il était possible d'entrer du SQL dans l'entrée et ainsi interagir directement sur la base de données.

3.4.5 Données sensibles en clair sur Github

La version du prototype de GOAT, postée initialement sur Github [11], contenait des informations de connexion à la base de données du CRCHUM, qui auraient pu être

recupérées et utilisées à mauvais escient. Un simple fichier `.gitignore`, qui liste tous les dossiers et fichiers qui ne devraient pas être présents sur le repository Git [24], permet pourtant de se prémunir de ce genre de risques.

3.4.6 Front-end non responsive

Une application moderne, en 2016, se doit de fonctionner sur toutes les tailles d'écran. Même si, dans le cas de GOAT, il n'était pas requis dans les exigences d'opérer sur des tablettes ou des téléphones portables, il aurait été quand même sage de s'assurer que les graphes interactifs soient adaptés à la taille de l'écran de l'utilisateur en tout temps. Hors, ces derniers étaient de taille fixe et relativement petite : 900 pixels de large pour 500 pixels de haut. Les utilisateurs étant doté d'écrans de tailles variables d'écrans, allant du petit ordinateur portable au grand écran fixe de 24 pouces, l'application actuelle n'était pas « responsive ».

3.4.7 Code difficile à déchiffrer

Les noms de variables dans le code source « back-end » ne sont absolument pas représentatifs de ce qu'elles contiennent, ce qui m'a posé problème pour comprendre certaines fonctions. De plus, le code ne contenait aucun commentaire, ce qui a été un frein à son analyse.

3.5 Conclusion

La version du prototype GOAT de janvier 2016 souffrait de plusieurs défauts. L'absence d'architecture logicielle et d'utilisation de bonnes pratiques du génie logiciel durant son développement ont conduit à une application fonctionnelle, mais peu performante et difficilement maintenable.

Le prochain chapitre détaille les actions entreprises afin de pallier à cette situation dans la version 1.0 de GOAT.

CHAPITRE 4

REINGENIERIE – CONCEPTION

Ce chapitre présente les différentes réflexions et décisions de conception menées durant la réingénierie de la version prototype de GOAT. Il s'attarde en premier lieu sur la décision de recommencer le projet de zéro, puis sur la nécessité de bien appréhender le domaine de la recherche en génomique. Il décrit ensuite la redéfinition des exigences fonctionnelles et non-fonctionnelles. Enfin, il donne un aperçu détaillé de la nouvelle architecture logicielle, tant « front-end » que « back-end ».

4.1 Modifier ou recommencer ?

GOAT devra évoluer dans le temps, mais aussi devra être ouvert à la participation d'autres développeurs situés partout dans le monde. Hors, avec les défauts actuels de qualité interne, il aurait été plus long de tout séparer que de recommencer sur de bonnes bases. Il a donc été décidé de recommencer de zéro, en réutilisant éventuellement quelques parties existantes au besoin, afin que la prochaine version soit maintenable et facile de modification.

4.2 Appréhension du domaine

Avant d'entreprendre toute action, il a semblé nécessaire de mieux comprendre le domaine de la génétique, et les concepts liés à GOAT. Cela ne s'est pas fait en un jour, car la recherche en génomique est loin d'être triviale. Ainsi, du temps a été consacré avec les bio-informaticiens pour bien saisir tous les concepts nécessaires au projet.

Une liste des concepts important a été dressée pour bien comprendre les données manipulées par GOAT et le traitement effectué par les chercheurs lors de son utilisation. Il n'était pas nécessaire de tout comprendre, seulement la partie du travail reliée à l'utilisation du logiciel. Pour cela, l'approche *Domain Driven Design* d'Eric J. Evans [12] a été utilisée, dans laquelle il est conseillé d'établir le plus vite possible un langage commun « ubiquitous language »

permettant aux développeurs et aux experts d'un domaine de communiquer sans risquer d'incompréhension ou de mauvaise interprétation. Ces concepts sont les suivants :

- **Phénotype** : Au sens utilisé par l'équipe, un phénotype est la manifestation d'un trait particulier pour un individu. Par exemple : une insuffisance rénale, une maladie cardio-vasculaire, le diabète, ou même la mort.
- **Chromosome** : Les chromosomes sont des filaments d'ADN présents, chez l'humain, dans les noyaux des cellules. Ces filaments contiennent des suites de gènes. Il y en a 23 paires chez l'être humain. L'équipe du CRCHUM n'étudie que les 22 paires non caractéristiques du sexe (c'est-à-dire tous sauf les chromosomes X et Y), apparemment plus difficiles à traiter.
- **Gène** : Un gène est un fragment d'ADN, identifiable par sa position sur un chromosome. Il contient une partie du code ADN, constitué de quatre bases A, C, G et T. Il est caractérisé par un nom unique.
- **SNP** : Un SNP (ou Single-Nucleotide Polymorphism) est une variation d'une paire de bases du génome. Ils sont donc tous liés à un gène (celui contenant la paire qui varie). Il est identifiable par un ID (le rsID, unique) et sa position sur le chromosome. Ces deux informations ont cependant tendance à changer avec le temps au fur et à mesure de l'avancée des connaissances dans le domaine.
- **GWAS** : Un GWAS (ou Genome-wide Association Study) est une étude statistique des différentes variantes génomiques (les SNPs) afin de déterminer s'ils sont liés à un trait (ici, un phénotype).

4.3 Exigences

Il a été nécessaire de mettre à jour le document d'exigences de GOAT, afin de repartir sur une base de fonctionnalités représentant réellement les besoins des chercheurs, et identifiant des attributs de qualité interne incontournables pour l'évolution future du projet.

4.3.1 Exigences fonctionnelles

Pour identifier les fonctionnalités souhaitables dans la prochaine version de GOAT, les membres de l'équipe qui l'utiliseront dans le futur ont été rencontrés.

La méthodologie a été la suivante : les fonctionnalités déjà présentes dans GOAT ont été décrites en demandant de confirmer leurs besoins ou si des modifications étaient requises. Dans un deuxième temps, est-ce que des fonctionnalités supplémentaires sont requises. De ces discussions, les conclusions suivantes ont été documentées:

- Les fonctionnalités déjà présentes étaient intéressantes mais incomplètes, et leur ordre d'utilisation (c.-à-d. Tableau et graphe statique avant d'accéder à l'Area Selection) ne représentaient pas la façon dont les chercheurs travaillent;
- Les chercheurs voulaient pouvoir accéder à chaque fonctionnalité indépendamment les uns des autres;
- Même s'ils acceptaient d'attendre un certain délai avant d'obtenir un résultat, cinq minutes pour accéder à un tableau leur semblaient beaucoup trop longues.

Une liste de onze exigences fonctionnelles a donc été approuvée. Ces exigences ont été triées par valeur d'entrée (input). Elles sont décrites dans les tableaux présents en annexe I.

4.3.2 Exigences non-fonctionnelles

Afin de concevoir une architecture adaptée, il était nécessaire d'identifier les exigences non-fonctionnelles de l'application. C'était plus difficile, car les membres de l'équipe du docteur Hamet n'étaient pas habitués à définir cela.

Pour faire cela, je me suis inspiré de la norme ISO 25010 [33], qui liste un certain nombre d'exigences séparées en sous-attributs de qualité. De cette liste ont été sélectionnées les exigences les plus adaptées aux besoins des utilisateurs finaux de GOAT. Ces exigences sont :

- **L'efficacité de rendement** : Elle permet de quantifier le temps nécessaire à l'affichage des données, acceptable par l'utilisateur.

- **La facilité d'utilisation** : Elle traite de la pertinence des données affichées, des prérequis nécessaires pour utiliser le logiciel, de la manière dont le logiciel doit prévenir l'entrée de données invalides par l'utilisateur, et des dimensions quantifiables de l'interface utilisateur.
- **La fiabilité** : Cette exigence est davantage reliée à la façon dont le logiciel est capable de se relancer en cas d'erreurs et de garder une trace de ces dernières.
- **La sécurité** : Dans le contexte de GOAT, elle revient essentiellement à être capable de tracer tout accès aux données via l'application.
- **La maintenabilité** : Elle traite de la modularité du code, de la réutilisabilité de ses composants, et de la possibilité d'éventuelles modifications à l'avenir.

Ces exigences sont décrites en détail dans les tableaux présents en annexe II.

A partir des exigences fonctionnelles et non-fonctionnelles décrites précédemment, une architecture a été conçue. Cette dernière est détaillée dans la partie suivante.

4.4 Architecture Logicielle

Au moment de revoir l'architecture logicielle de GOAT, il a fallu prendre en compte les exigences fonctionnelles et non-fonctionnelles, ainsi que la perspective d'ouvrir GOAT à la communauté libre. Il a aussi fallu tenir compte de la limite de temps établie pour ce projet de recherche appliquée (c.-à-d. jusqu'à la fin de la session pour effectuer cette réingénierie). Cela a influencé les décisions et les choix technologiques, détaillés dans les prochaines sections.

4.4.1 Architecture logicielle haut niveau

La nouvelle version de l'architecture logicielle de GOAT sera composée de trois couches logiques, à savoir : 1) une base de données MySQL; 2) un serveur web utilisant le cadriciel Django; et une « Single-Page Application » (SPA) [13] utilisant le cadriciel React [14] et Flux [15].

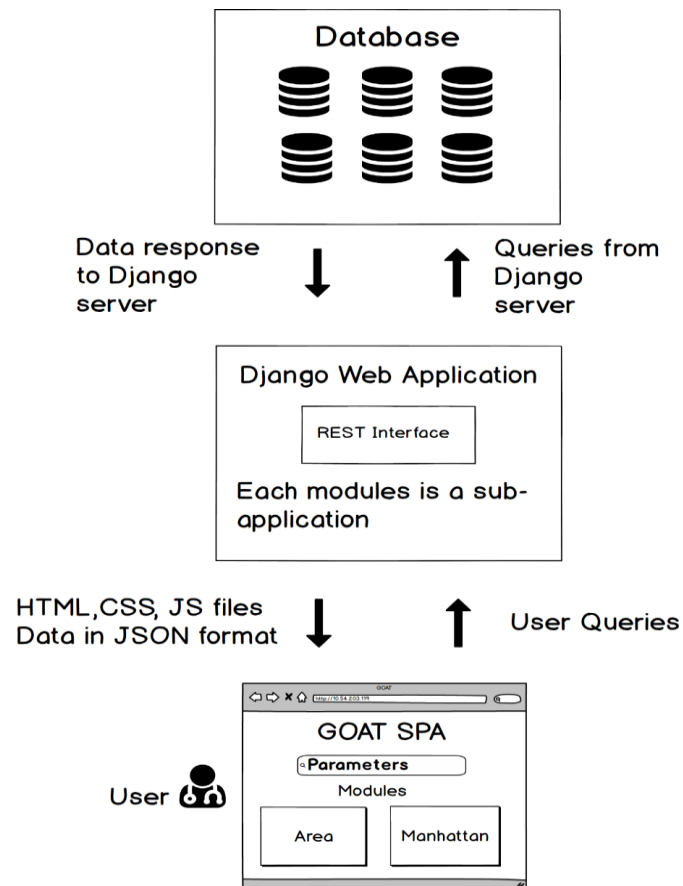


Figure 4-1 : Vue en couches de l'architecture de GOAT

Dans cette nouvelle conception, l'utilisateur interagit avec une interface utilisateur et émet des requêtes au serveur pour accéder à une fonctionnalité. Cette requête est transmise de façon asynchrone à l'aide de la technologie AJAX [16] au serveur web Django qui se charge d'interroger la base de données MySQL afin de récupérer les données nécessaires, les traiter, puis les envoyer à l'application « front-end » qui se chargera de les afficher correctement à l'utilisateur.

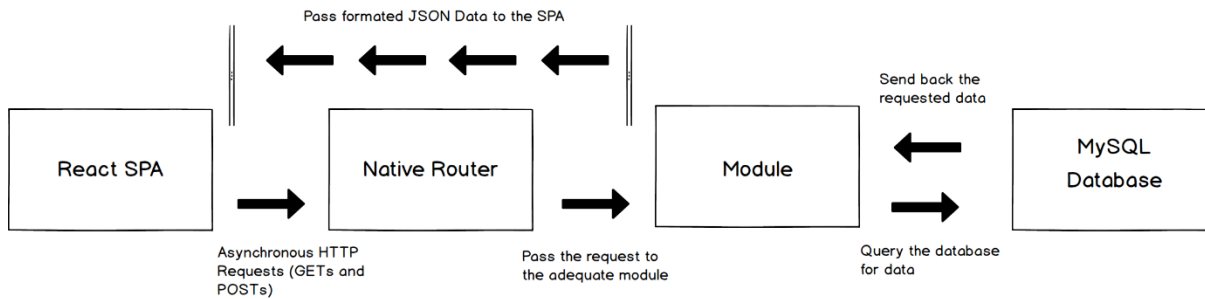


Figure 4-2 : Vue Composants et Connecteurs de GOAT

L'idée principale à la base de cette nouvelle architecture logicielle est de séparer toutes les fonctionnalités dans leurs modules propres, dans le « front-end » tout comme dans le « back-end », afin de minimiser le temps d'accès aux données pour l'utilisateur (ENF1) et maximiser la modularité, la modifiabilité et la réutilisabilité du code (ENF4).

4.4.2 Architecture logicielle front-end

4.4.2.1 Description

La nouvelle version de GOAT sera donc conçue avec la nouvelle approche de conception nommée SPA. Les applications web modernes, en 2016, devraient prendre avantage de l'approche SPA. Contrairement aux conceptions d'il y a quelques années, la conception SPA repose sur l'utilisation d'un seul fichier HTML, et beaucoup plus sur le langage JavaScript [17].

Le contenu d'une SPA se modifie dynamiquement en fonction des actions de l'utilisateur et des données reçues par le serveur, sans rechargement. Par exemple, quand l'utilisateur se connecte à son compte, l'état de l'application passe de « Login » à « Home ». Le formulaire de connexion disparaît alors et est remplacé par une liste des modules disponibles. Le fichier HTML est toujours le même, mais son contenu est changé dynamiquement en fonction de l'état de l'application.

L'application « front-end » est composée d'un état « Home », qui affiche la liste des fonctionnalités à l'utilisateur. Chaque fonctionnalité est contenue dans un module

indépendant, accessible directement depuis l'écran d'accueil, ou via un autre module si nécessaire.

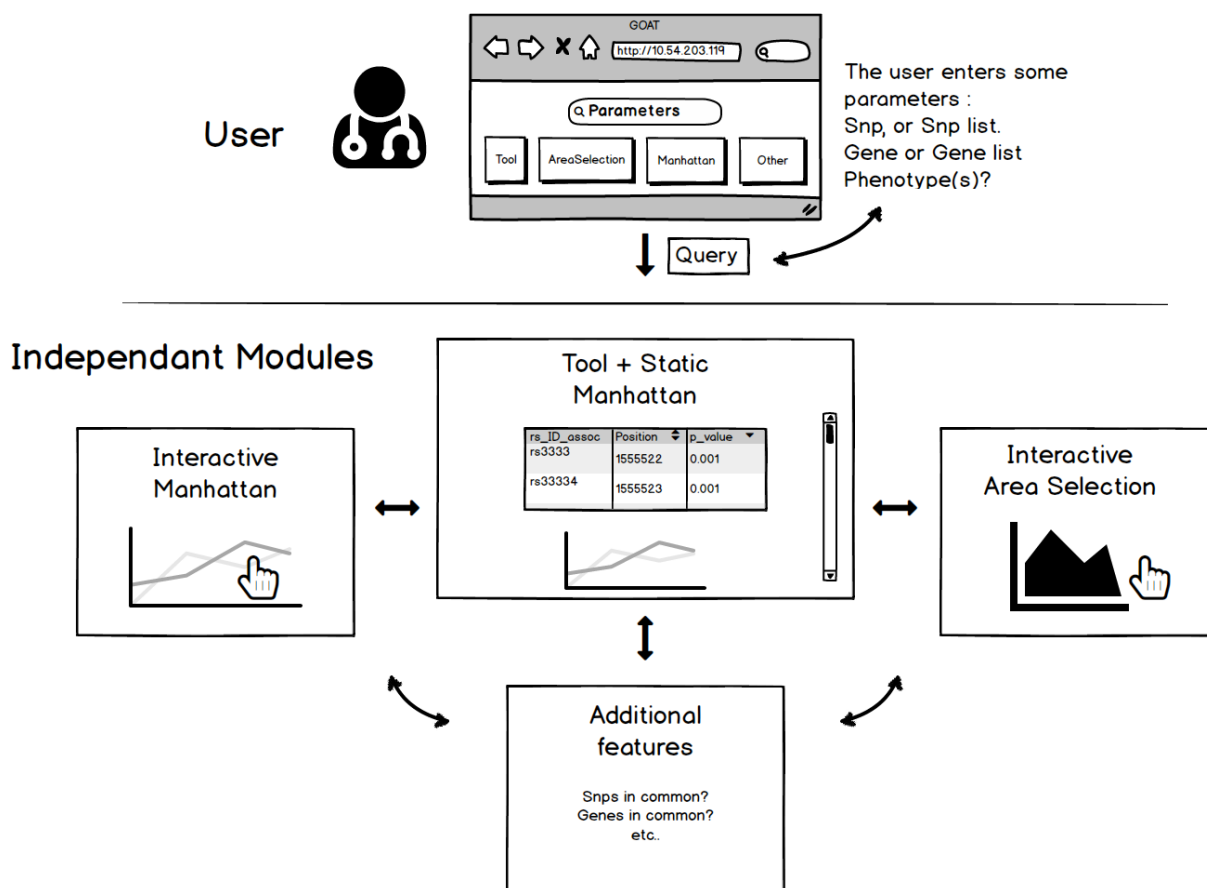


Figure 4-3 – Fonctionnement de la SPA de GOAT

Quand l'utilisateur souhaite accéder à une fonctionnalité, il clique sur le bouton correspondant, ce qui modifie alors l'état de l'application. Les données nécessaires sont récupérées et affichées en accord avec la fonctionnalité.

L'intérêt de séparer chaque fonctionnalité des autres, est que les données seront bien plus rapides à récupérer qu'auparavant, ce qui permettra d'accélérer grandement l'accès à ces dernières. Il y avait un délai de cinq minutes avant d'obtenir une réponse dans la version précédente et ce pour accéder à la première fonctionnalité. Il était aussi nécessaire d'afficher un graphe Manhattan statique avant d'accéder au graphe interactif AreaSelection. Dans la nouvelle version, il sera non-seulement possible d'accéder en dix à vingt secondes aux

graphes interactifs (**ENF1**), mais aussi directement à chaque fonctionnalité, sans être obligé de passer par une autre.

4.4.2.2 Fonctionnalités

En tenant compte des exigences fonctionnelles, il a été possible d'établir une liste de fonctionnalités. Chaque fonctionnalité correspond à un état spécifique de l'application. Ainsi, la version 1.0 de GOAT contiendra cinq états fonctionnels :

- **Login** : L'utilisateur peut se connecter avec son identifiant et son mot de passe.
- **AreaSelection** : L'utilisateur, en entrant le nom d'un phénotype, le numéro d'un chromosome et une position, peut accéder à un graphe interactif centré autour de cette position.
- **GeneQuery** : L'utilisateur, en entrant un rsID, le nom d'un gène ou celui d'un phénotype, peut accéder à un tableau listant les SNPs significatifs et leurs propriétés, en rapport avec les données d'entrée.
- **Manhattan** : L'utilisateur, en entrant un rsID, le nom d'un gène ou d'un phénotype, peut accéder à un graphe Manhattan interactif représentant les SNPs significatifs, en rapport avec les données d'entrée.
- **SnpForGene** : L'utilisateur, en entrant le nom d'un gène, peut accéder à la liste des SNPs significatifs présents sur ce gène, sur le gène précédent, et le gène suivant.

Le tableau A III-1 en annexe III liste ces états ainsi que les exigences fonctionnelles qu'ils satisfont.

De plus, certains états se voient allouer un état intermédiaire, correspondant au formulaire d'entrée nécessaire à son bon fonctionnement. Le tableau A III-2 liste ces états intermédiaires.

A partir de la liste précédente, il a été possible d'établir les cas d'utilisation de l'interface utilisateur de GOAT. Ces derniers traitent des différentes réactions du système aux actions utilisateurs. Ainsi, chaque fonctionnalité représentant des données comporte trois cas

d'utilisation, pour des données d'entrée au format valide, invalide, et valide mais ne renvoyant aucune donnée.

Ces cas d'utilisation sont décrits en détails dans les tableaux de l'annexe IV.

La partie suivante contient des captures d'écrans du « front-end » GOAT permettant de mieux comprendre les fonctionnalités précédemment décrites.

4.4.2.3 Visuels

Les captures d'écran présentes en annexe V montrent à quoi ressemble désormais le « front-end » de GOAT.

4.4.3 Back-end

4.4.3.1 Description

Étant donné que le « front-end » ne dépend plus des gabarits mais est maintenant basé sur React, le « back-end » est désormais uniquement chargé seulement de récupérer des données en base de données, les traiter, puis les faire passer à la SPA.

Chaque fonctionnalité doit maintenant correspondre à une route spécifique, afin qu'un ajout ou une modification d'une fonctionnalité n'ait pas d'impact sur les autres. De plus, chaque fonctionnalité est contenue dans son propre module, nommé « app » dans Django.

La figure 4-4 montre une vue modulaire de l'architecture back-end.

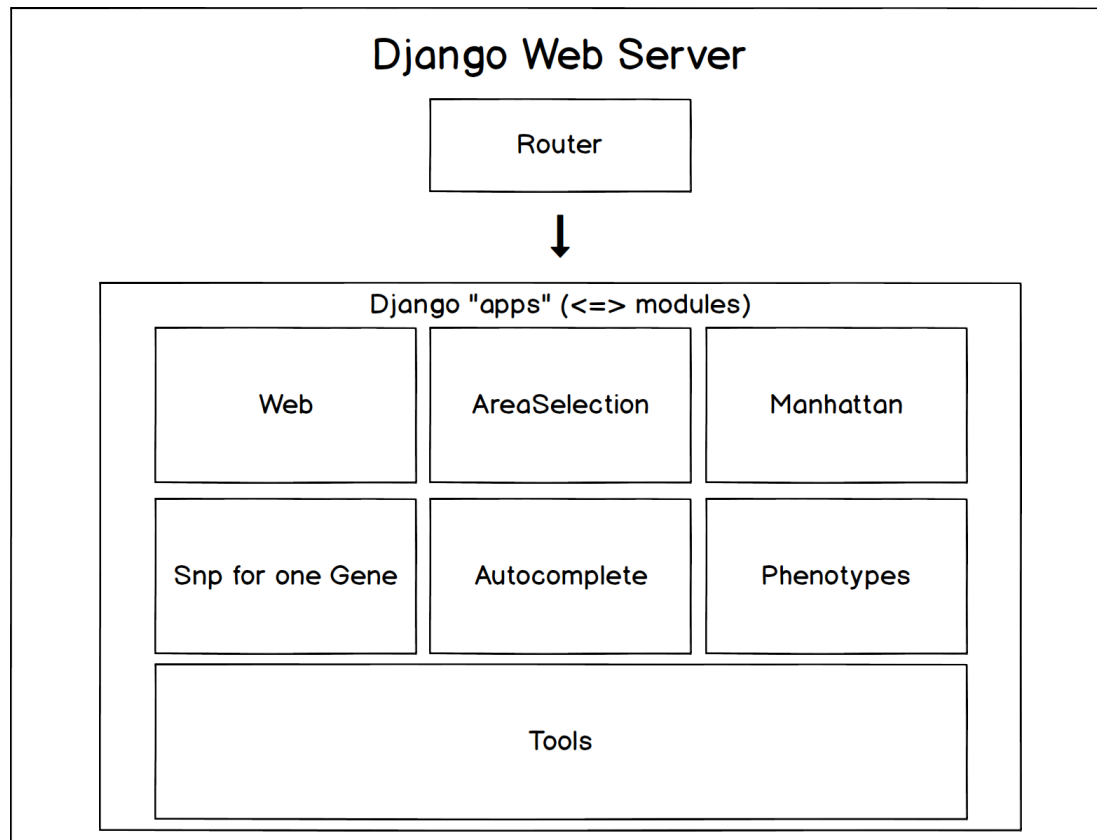


Figure 4-4 : Vue modulaire de l'architecture « back-end »

La section suivante détaillera ces modules.

4.4.3.1 Modules

Cette partie détaille les routes d'accès des différents modules de la version 1.0 de GOAT. Elle contient également pour chacun d'entre eux un ou plusieurs exemples permettant de comprendre comment la Single Page Application communique avec les modules, par l'intermédiaire d'un routeur et de requêtes HTTP.

4.4.3.1.1 Web

Ce module est chargé d'envoyer le fichier HTML nécessaire au bon fonctionnement de l'interface utilisateur (la SPA).

Route : `http://serveradress/`

Exemple : Si l'utilisateur souhaite accéder à l'application, il se rend à l'adresse allouée ce qui a pour effet d'automatiquement effectuer une requête GET à l'adresse :

« `http://serveradress/` ».

NB : « serveradress » correspond à l'adresse IP allouée à l'application. Elle varie selon l'environnement d'installation de GOAT.

4.4.3.1.2 AreaSelection

Ce module est chargé de récupérer, traiter et faire passer à la SPA les données nécessaires à l'affichage d'un graphe interactif et d'un tableau de SNPs significatifs correspondants aux données entrées par l'utilisateur.

Route :

`http://serveradress/areaSelection/chromosome/position/phenotype/userWidth/userHeight`

Exemple : Si l'utilisateur, dont l'écran est large de 1600px (pixels) et haut de 900px, souhaite accéder au graphe interactif areaSelection, sur le chromosome 5, centré en la position 6000000 et sur le phénotype « All cause death », l'application effectuera une requête GET à l'adresse suivante :

« `http://serveradress/areaSelection/5/6000000/All cause death/1600/900` »

4.4.3.1.3 Manhattan

Ce module est chargé de récupérer, traiter et faire passer à la SPA les données nécessaires à l'affichage d'un graphe Manhattan interactif correspondant aux données entrées par l'utilisateur.

Route : `http://serveradress/manhattan/type/value/userWidth/userHeight`

Exemple : Si l'utilisateur, dont l'écran est large de 1500px et haut de 600px, souhaite accéder au graphe Manhattan du phénotype « All cause death », l'application effectuera une requête GET à l'adresse :

« [http://serveradress/manhattan/phenotype/All cause death/1500/600](http://serveradress/manhattan/phenotype/All%20cause%20death/1500/600) »

4.4.3.1.4 SNP for one Gene

Ce module récupère et envoie à la SPA une liste de SNPs significatifs présents sur le gène, le gène précédent et le gène suivant celui choisi par l'utilisateur.

Route : <http://serveradress/snpsForGene/gene>

Exemple : Si l'utilisateur souhaite accéder à la liste des SNPs significatifs présents sur le gène « SPOCK3 », le précédent ou le suivant, l'application effectuera une requête GET à l'adresse :

« <http://serveradress/snpsForGene/spock3> »

4.4.3.1.1 Autocomplete

Ce module récupère et envoie à la SPA les données nécessaires à l'auto-complétion des champs d'entrée des formulaires de gènes et de phénotypes.

Route : <http://serveradress/autocomplete/type/value>

Exemple 1: Si l'utilisateur souhaite accéder à la liste des gènes dont le nom commence, contient ou termine par « SPO », l'application effectuera une requête à l'adresse :

« <http://serveradress/autocomplete/gene/spo> »

Exemple 2: Si l'utilisateur souhaite accéder à la liste des phénotypes dont le nom commence, contient ou termine par « All », l'application effectuera une requête GET à l'adresse :

« <http://serveradress/autocomplete/phenotype/all> »

4.4.3.1.1 Phenotypes

Ce module récupère et envoie à la SPA la liste des phénotypes présents en base de données.

Route : `http://serveradress/phenotypes/`

Exemple : Si l'utilisateur souhaite accéder à la liste des phénotypes l'application effectuera une requête à l'adresse :

« `http://serveradress/phenotypes/` »

4.4.3.1.1 Tools

Il contient plusieurs modules, utilisés par plusieurs ou tous les autres modules de l'application, comme la connexion à la base de données, la vérification de la significativité d'un SNP ou d'un gène pour un phénotype, etc... Il ne communique pas directement avec la SPA.

Les tableaux présents en annexe VI détaillent plus précisément chacun des modules, et s'attardent sur leurs données d'entrée et de sortie.

4.5 Conclusion

L'ouverture du projet en open-source nécessitait que l'ajout de fonctionnalités n'ait pas d'influence sur le code préexistant. De plus, la redéfinition des exigences fonctionnelles et non-fonctionnelles a impliqué de repenser la façon dont l'utilisateur interagirait avec l'interface utilisateur. La nouvelle version de GOAT a été conçue autour d'une architecture modulaire, destinée à être maintenable à l'avenir, et ouverte à la modification. La séparation du code en modules indépendants, tant sur le « front-end » que sur le « back-end », permet d'ajouter de nouvelles fonctionnalités sans risquer d'introduire des bugs dans l'application.

La partie suivante présente les décisions de conception précédemment décrites ont été mises en œuvre.

CHAPITRE 5

REINGENIERIE – DEVELOPPEMENT

Ce chapitre décrit en premier lieu les outils de développement utilisés, puis les technologies employées pour faciliter le développement. Il liste ensuite les technologies sélectionnées pour mettre en œuvre les décisions de conception. Enfin, il présente un aperçu visuel de l'application dans sa version 1.0.

5.1 Outils de développement

5.1.1 JetBrains PyCharm

Pycharm [18] est un IDE [19] spécialisé dans la création de projets en langage Python. Il offre de nombreux services tels que l'auto-complétion, la détection dynamique d'erreurs, la configuration d'un environnement virtuel Python, et bien d'autres fonctionnalités.

Pycharm a été sélectionné pour développer la partie « back-end » du projet, le cadriciel Django étant basé sur Python.

5.1.2 Atom

Atom [20] est un éditeur de texte gratuit et libre de droits, créé par les fondateurs de Github. Pensé pour être facilement paramétrable, il permet l'installation de nombreux plugins créés par la communauté de développeurs. Il intègre nativement certains des outils les plus utilisés par les développeurs « front-end » pour accélérer l'écriture du code, comme Emmet [21], JSHint [22] ou CSSLint [23].

5.1.3 Cmdr

Cmdr [29] est un émulateur de console. C'est un outil intéressant car il offre une interface utilisateur plus agréable à utiliser que la console Windows native, de par sa configuration de

couleurs, de police d'écriture, ou encore la possibilité d'afficher plusieurs consoles en une fenêtre. Cela a été utile pour afficher simultanément une console dédiée au serveur, une autre dédiée au « build » automatique de l'application « front-end », et une dernière pour le contrôle de version avec Git [24].

5.2 Technologies de développement

Cette partie du rapport va lister les technologies utilisées pour développer la nouvelle version de GOAT, contrôler ses différentes versions dans le temps, et partager le code en open-source.

5.2.1 Contrôle de version

Git est un système open-source de contrôle de versions décentralisé. Il a été créé par Linus Trolvads, le créateur de Linux. Il n'y a pas qu'un repository unique hébergé sur un serveur, mais plusieurs repository locaux installés sur les machines des développeurs du projet, ainsi qu'un repository "public" qui permet de partager son travail et récupérer celui des autres. Afin d'éviter de divulguer des éléments sensibles, comme les informations de connexion à la base de données, ces informations ont été placées dans des fichiers spécifiques importées quand nécessaires, et un fichier « .gitignore » liste ces derniers, qui ne seront pas détectés par Git.

L'ouverture en open-source de GOAT se faisant sur la plateforme Github [11], cela laissait peu de place quant au choix de technologie de contrôle de version, car Github impose d'utiliser Git. Nous avons donc créé un repository privé sur la plateforme Github. Une fois la première version de GOAT validée, un compte spécifique à GOAT sera créé et un repository ouvert contiendra tout le code source du projet.

Github est une application web accessible via un navigateur à l'adresse suivante : <https://github.com/>.

Sur Github, on a accès à une interface graphique facilitant l'utilisation de Git. On peut ainsi créer un repository d'un clic, le partager avec un lien généré dynamiquement, ou encore

configurer ses accès externes (public ou privé). Chaque utilisateur de Github possède un profil similaire à celui d'un réseau social, indiquant les projets auxquels il a participé, à la manière d'un portfolio. Il peut également discuter de ses projets publiquement avec des développeurs expérimentés, ou demander l'autorisation de collaborer à un projet qui l'intéresse, faisant de Github un environnement de développement collaboratif.

Il est possible de "Fork" un repository, ce qui revient à le cloner sur son compte pour l'améliorer ou créer un nouveau projet en se basant dessus. Il est possible de notifier les développeurs originaux d'un projet à l'aide d'une "Pull Request", afin que ces derniers puissent regarder les modifications et éventuellement les inclure dans le projet de référence.

5.2.2 Build automatique

Les technologies utilisées dans le « front-end » (présentées dans la partie suivante) ne peuvent être lues directement par un navigateur, car leur syntaxe comporte beaucoup d'éléments d'ECMAScript 2015, la dernière version de JavaScript. Pour pallier à cela, j'ai utilisé les technologies suivantes.

5.2.2.1 Babel

Babel [25] est un compilateur JavaScript qui transforme un code de version 6 (EC2015) en version 5, lisible par tous les navigateurs internet. Cela permet d'utiliser un grand nombre de fonctionnalités, comme les classes, les « arrow-functions » (équivalent JavaScript des lambda-expressions), ou encore les variables locales.

5.2.2.2 Browserify

Browserify [26] permet d'importer des « modules » JavaScript depuis d'autres fichiers. C'est la pierre angulaire de tout projet JavaScript car il permet d'organiser son code en plusieurs fichiers indépendants. Il est donc nécessaire pour rendre son code « front-end » modulaire.

5.2.2.3 Gulp

Gulp [6] est un outil d'automatisation qui permet de lancer des tâches complexes sur des fichiers. Dans mon cas, j'ai utilisé Gulp pour observer les changements dans les fichiers JavaScript de l'application « front-end », et automatiquement compiler le code en version 5, puis créer un seul fichier JavaScript contenant toute l'application. Cela permet d'avoir, en un seul fichier, tout le code nécessaire au bon fonctionnement de GOAT, ce qui évite d'éventuelles erreurs en cas de problème lors du chargement d'un des fichiers de l'application.

De plus, le fait de pouvoir exécuter le code de montage de l'application au moindre changement dans un fichier a accéléré le développement car il suffisait alors de recharger la page pour utiliser la dernière version.

5.3 Technologies de l'application

Cette partie va lister les technologies utilisées dans la nouvelle version de GOAT pour implémenter l'architecture décrite dans le chapitre précédent. Une attention particulière sera apportée au « front-end », car les technologies utilisées sont radicalement différentes de l'ancienne version de l'application.

5.3.1 Front-end

5.3.1.1 React.js

React.js [14] est une librairie créée par Facebook afin de pallier aux limites de maintenabilité du patron majoritaire en développement web : le MVC. React offre plusieurs avantages : Il permet de créer des composants indépendants et réutilisables, contenant toute la logique nécessaire à leur fonctionnement. Par exemple, un Input vérifiant lui-même le format des données d'entrée.

De plus, il fonctionne avec un langage nommé JSX, qui permet d'écrire dans une syntaxe proche du HTML tout en pouvant y intégrer de la logique en JavaScript.

Enfin, il est possible d'éviter les répétitions et les modifications de code, en créant des composants dynamiquement en fonction de données. L'exemple suivant montre la façon (simplifiée) dont le menu de GOAT a été créé, à partir d'un seul composant réutilisé plusieurs fois pour chaque objet contenu dans la liste des modules. Pour ajouter une fonctionnalité au menu, il suffit d'ajouter un objet dans la liste, sans jamais avoir à toucher au composant Menu.

Algorithme 5-1 : Tableau contenant la liste des noms de modules

```
module.exports = [  
  {  
    content : "HOME"  
  },  
  {  
    content : "GENE QUERY"  
  },  
  {  
    content : "AREA SELECTION"  
  },  
  {  
    content : "MACHINE LEARNING"  
  }  
];
```

Algorithme 5-2 : Classe React renvoyant un composant « MenuItem »

```
module.exports = React.createClass({  
  render : function(){  
    return (  
      <li>  
        <h2>{this.props.content}</h2>  
      </li>  
    )  
  }  
});
```

React permet de passer des données à un composant à travers les « props », de façon à y accéder dans le code du composant. Ici, la « prop » passée est nommée « content ».

Algorithme 5-3 : Composant React créant dynamiquement un menu

```
var MenuList = require('MenuList'); //On importe la liste d'éléments du menu
var MenuItem = require('MenuItem'); //On importe le composant MenuItem

module.exports = React.createClass({
  render : function(){
    //On crée une fonction qui génère un MenuItem
    var createMenuItem = function(item){
      return <MenuItem content={item.content}/>
    }
    return(
      <div>
        { /* Pour chaque élément de la liste, on crée un composant */ }
        {MenuList.map(createMenuItem)}
      </div>
    );
  }
});
```

Chaque composant possède un état qui lui est propre, dans lequel il est possible de stocker des données. Par exemple, quand un utilisateur entre un mot dans un input, on peut stocker la valeur d'entrée dans l'état de l'application et ainsi travailler dessus en temps réel. Les Inputs de GOAT vérifient par exemple la validité de la valeur d'entrée à l'aide de *Regex*, et changent dynamiquement leur couleur d'arrière-plan en rouge (si invalide) ou vert (si valide). Cette validité étant stockée dans l'état du composant, il est possible d'empêcher l'utilisateur de faire une requête à partir de la valeur qu'il a rentrée.

Dans l'exemple suivant, on voit que la valeur d'entrée ne suit pas le format requis (rsXXXXXX avec X un chiffre entre 0 et 9). Dans l'état (State) du composant, on assigne *false* à la validité, et un arrière-plan rose/rouge.

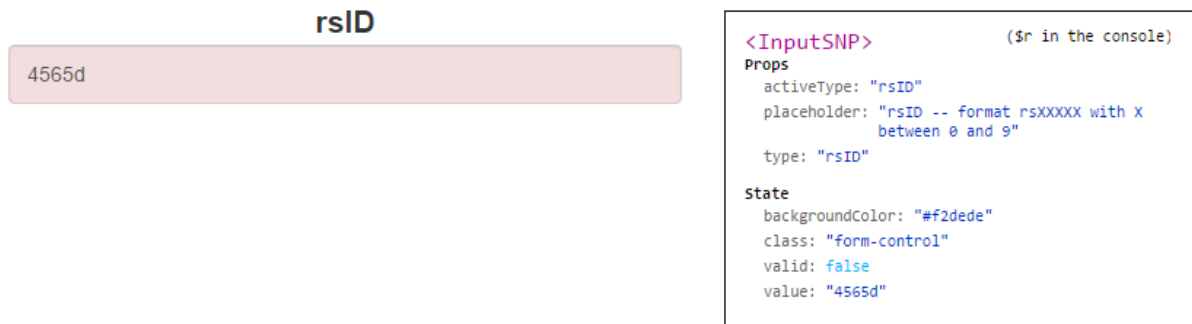


Figure 5-1 : Input dont la valeur entrée est invalide

Dans l'image suivante, la valeur entrée est au contraire valide. On assigne *true* à la propriété *valid*, qui servira alors au composant supérieur (le formulaire entier) à autoriser la requête GET pour récupérer les données nécessaire à la fonctionnalité que l'utilisateur souhaite atteindre.

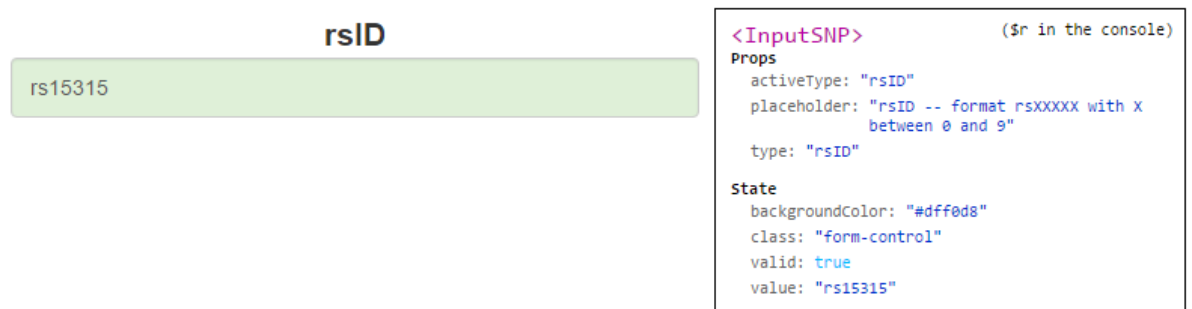


Figure 5-2 : Input dont la valeur entrée est valide

Les composants s'affichent en fonction de leur état et de ceux des composants supérieurs. En somme, le « front-end » devient une machine à états, chacun définissant un type de contenu à afficher.

Chaque composant peut appeler d'autres sous-composants et leur fournir des données, via les « props ». Cela permet de créer une arborescence de composants dans laquelle les données circulent du haut vers le bas : on parle de « **Unidirectional data-flow** ».

En cas d'interaction utilisateur ou de modification des données d'un composant, React compare automatiquement le DOM (Document Object Model, décrivant tous les éléments de la page) avec son propre DOM Virtuel, et ne modifie que ce qui est nécessaire pour tenir compte de ces changements. Cela a pour conséquence de fournir de très bonnes performances pour l'application.

Enfin, le fait que les composants soient complètement indépendants les uns des autres permet d'utiliser de nombreux composants créés par d'autres développeurs et mis à disposition gratuitement sur des sites de partage de code, tels que <http://react-components.com/>. D'autres développeurs ont même créés de véritables librairies basées sur React. GOAT utilise d'ailleurs une de ces librairies nommée Griddle pour créer des tableaux de données interactifs.

5.3.1.2 Flux

React est très puissant mais ne s'occupe que de la partie « Vue » de l'application « front-end ». Afin de coordonner facilement les différents états de l'application, Facebook a mis à disposition des développeurs un patron nommé Flux [15], spécialement créé pour faciliter la gestion d'événements utilisateurs et les transmissions de données.

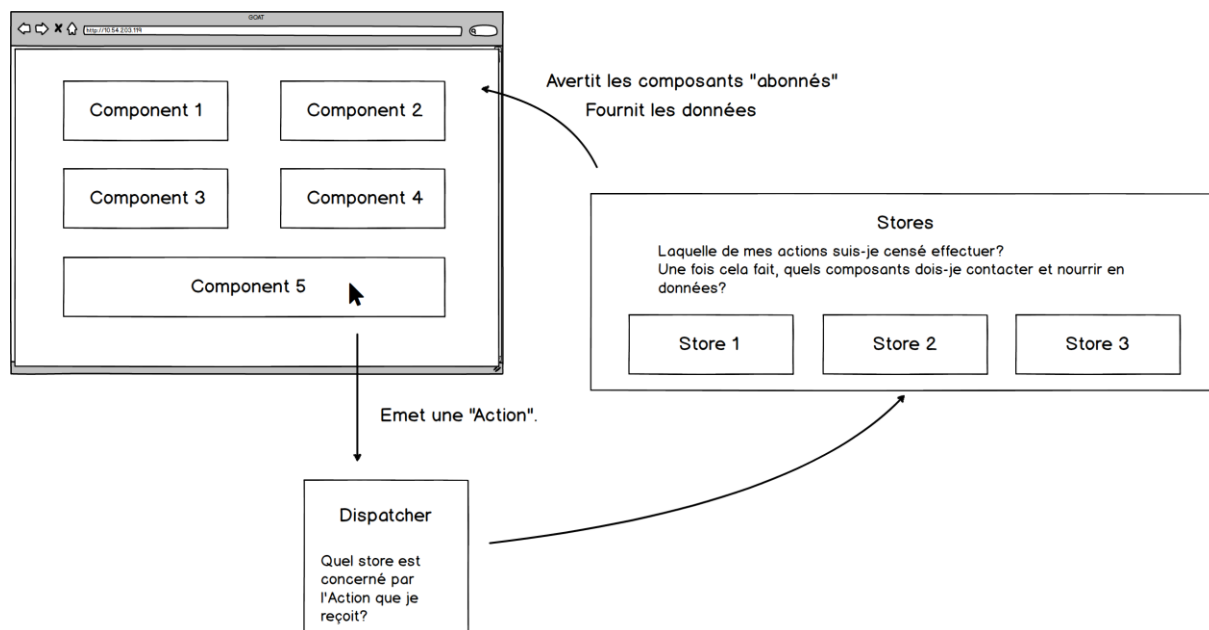


Figure 5-3 : Fonctionnement de Flux

Quand un utilisateur effectue une action sur un composant, ce dernier émet un évènement nommé « Action », qui contient ou non des données et possède un nom particulier.

Cette « **Action** » est reçue par un module nommé « Dispatcher », chargé de la rediriger vers les « Store » abonnés à cette dernière.

Les « **Stores** » contiennent la logique de l'application, ainsi que la logique nécessaire pour aller récupérer ou envoyer les données sur le serveur. Une fois l'Action effectuée, ils broadcastent un évènement contenant le nouvel état de l'application à tous les composants actifs, afin qu'ils puissent être modifiés ou supprimés en conséquence.

Le **Dispatcher** est fourni par Facebook, qui a mis le code à disposition sur son repository GitHub. L'implémentation du reste de Flux est à la charge des développeurs, ce pourquoi ne nombreuses implémentations ont été créées. Les deux plus populaires sont Reflux et Redux. Ayant de l'expérience avec Reflux, c'est la solution qui a été sélectionnée.

5.3.1.3 Griddle

Griddle [8] est une composant React qui permet de créer un tableau dynamique à partir d'une liste d'objets JSON. Il avait été sélectionné par le Cédric Urvoy, qui avait travaillé sur la version précédente du « front-end », grâce à plusieurs fonctionnalités essentielles pour explorer les listes de SNPs de la fonctionnalité Gene Query.

En effet, il est possible de n'afficher qu'un petit nombre de SNPs correspondant à la capacité d'affichage de l'écran de l'utilisateur, et d'en créer dynamiquement quand l'utilisateur fait défiler le tableau vers le bas. Le nombre de SNPs se comptant en milliers, cela apporte un gain de fluidité de la page.

Il est également possible pour l'utilisateur de chercher un élément particulier dans le tableau grâce à un champ de recherche intégré, de cacher certaines colonnes ou encore de filtrer l'affichage par ordre alphabétique ou croissant/décroissant par colonne.

5.3.1.4 Bootstrap

Créé par Twitter, Bootstrap [30] est un cadriciel CSS qui fournit un certain nombre de classes permettant de rapidement créer des applications au design clair et professionnel. Il n'est pas nécessaire d'avoir des connaissances approfondies en CSS l'utiliser. La dimension open-source du projet m'a poussé à choisir Bootstrap pour gérer l'apparence de l'application, car cela évitera aux futurs développeurs de travailler sur GOAT sans avoir à apprendre le CSS en profondeur.

5.3.2 Back-end

5.3.2.1 Django

Django [2] est un framework basé sur Python et dédié à la création d'applications web. Il a déjà été introduit dans le chapitre 1, et son utilisation dans GOAT a été définie dans le chapitre 3. Aussi, son fonctionnement ne sera pas redéfini ici.

5.3.2.2 Bokeh

Bokeh [27] est une technologie jeune et en pleine évolution, dédiée à la représentation interactive de données dynamiques dans un navigateur internet. Elle est constituée de deux parties, l'une destinée au serveur et l'autre destinée au navigateur.

La partie serveur permet de dessiner un graphe en utilisant un certain nombre d'outils connus des développeurs Python, en y ajoutant un lot de fonctionnalités tels que la possibilité de zoomer avec la molette de la souris, en sélectionnant une zone du graphe, mais encore d'enregistrer l'image ou bien redimensionner le graphe. Elle génère alors deux balises HTML, l'une contenant du code JavaScript, l'autre une balise « <div> » à laquelle est assigné un ID dynamique, dans laquelle sera dessiné le graphe.

La partie client est une librairie contenant un lot d'outils de représentation graphique. Elle permet d'utiliser les canvas. Il est également possible d'utiliser le moteur WebGL, qui demande d'avoir une carte graphique mais possède des performances bien supérieures aux canvas.

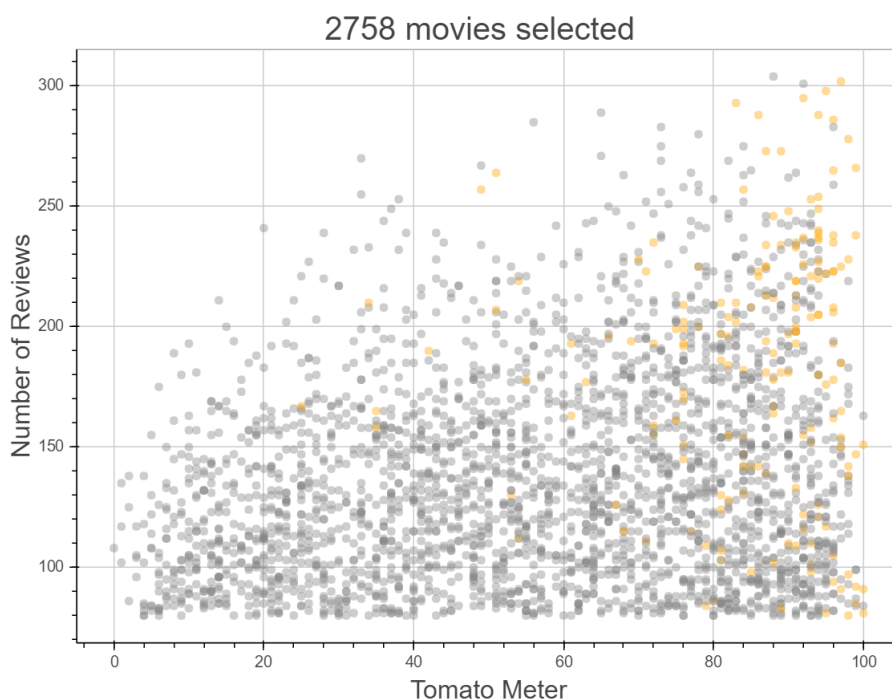


Figure 5-4 : Exemple de graphe Bokeh [32] (directement tiré de la documentation)

CHAPITRE 6

DISCUSSION

Cette partie présente mon ressenti de cette expérience, ainsi que des recommandations pour la suite du projet.

6.1 Bilan de l'expérience

6.1.1 Positif

Ce projet m'a offert la possibilité de découvrir un domaine qui jusqu'alors m'étais totalement étranger : la recherche en génomique. Pouvoir travailler au contact d'une équipe pluridisciplinaire a été une expérience très enrichissante. En effet, j'ai pris conscience du fait qu'un développeur ne doit pas seulement s'appuyer sur des compétences techniques pour mener à bien sa mission, mais doit également comprendre le domaine dans lequel il évolue afin de travailler le plus efficacement possible. De la même manière, j'ai fait l'effort, tout au long de ce rapport, d'expliquer les décisions architecturales et de conception ainsi que les choix de technologies aux membres de l'équipe concernés, ce qui a eu pour effet de faciliter nos échanges au jour le jour.

J'ai également pu travailler en conditions réelles avec des technologies modernes, telles que React et Bokeh, ce qui est toujours agréable pour un ingénieur logiciel.

Enfin, je pense avoir pu faire prendre conscience aux bio-informaticiens du laboratoire du CRCHUM de la nécessité d'appliquer les pratiques exemplaires de génie logiciel dans leur travail, comme le fait de mettre en place une architecture logicielle qui permet d'écrire un code modulaire, et de profiter des nombreux bénéfices qui en découlent.

6.1.2 Difficultés rencontrées

La phase initiale du projet a été la plus compliquée, car il m’a fallu en même temps de comprendre le code du prototype initial GOAT et de me familiariser au domaine de la recherche en génomique. Cela n’a pas facilité l’analyse de l’état de départ du projet.

De plus, je n’ai pas eu tout de suite accès au réseau du CRCHUM, ce qui a ralenti mon travail car je devais emprunter les codes d’accès d’autres membres de l’équipe quand ces derniers ne les utilisaient pas, ce qui était assez peu pratique.

Il m’a également fallu faire face à l’absence de bonnes pratiques du génie logiciel chez les bio-informaticiens. Je suis arrivé à la conclusion que la formation actuelle des bio-informaticiens d’une seule année est inadéquate. Ils ont aussi l’habitude de créer des scripts qu’ils ne modifient jamais (ou peu), ils n’ont donc pas la connaissance, ni la formation qui leur permette de saisir l’importance de la maintenabilité d’un logiciel. Ceci a pu être constaté dans la façon de coder et l’absence d’architecture de l’ancienne version de GOAT.

Enfin, il est arrivé que des membres viennent directement me voir pour ajouter des fonctionnalités, ce qui n’a pas forcément été simple à gérer en plein développement. Nous nous sommes donc concentrés sur une version 1.0 de GOAT contenant toutes les fonctionnalités présentées dans ce document, et avons décidé d’ajouter les fonctionnalités demandées qu’en fin de projet, s’il restait du temps.

6.2 Recommandations

Pour l’instant, la base de données ne contient que quelques phénotypes. A terme, il est possible qu’elle contienne plus d’une centaine de phénotypes. Il est probable que le temps d’attente avant d’afficher certaines fonctionnalités augmente sensiblement avec l’augmentation du nombre de phénotypes, même si la séparation des fonctionnalités en modules indépendants, évitant ainsi de charger toutes les données nécessaires à GOAT d’un coup, va grandement limiter ce ralentissement.

Toutefois, il existe maintenant de nombreuses technologies capables de traiter rapidement de grandes quantités de données, qui pourraient non seulement empêcher un tel ralentissement, mais même accélérer la vitesse de traitement des données de GOAT. La technologie qui me

semble la plus adaptée est Spark [28]. En effet, il est possible d'utiliser Spark en programmant directement en Python, ce qui permettrait aux bio-informaticiens de s'approprier l'outil rapidement. De plus, j'ai effectué quelques tests sur les fichiers qui ont été utilisés pour créer la base de données SQL sur laquelle est branché GOAT. Ces derniers sont lisibles par Spark, ce qui laisse entrevoir une possibilité de travailler directement sur ces derniers sans avoir à créer une nouvelle base de données intermédiaire entre les données du CRCHUM et GOAT, comme c'est le cas actuellement.

S'agissant de la dimension open-source du projet, il va falloir que quelqu'un se charge de faire évoluer le projet et des relations avec les développeurs extérieurs qui voudraient s'impliquer dans ce dernier. L'idéal serait de créer un compte Github dédié et de publier le code sur un référentiel public de ce dernier.

CONCLUSION

Ce projet consistait en l'évaluation, puis en la réingénierie du projet GOAT. Mon intervention a été réalisée en trois phases distinctes, à savoir une phase d'évaluation de la version précédente de GOAT, une phase de conception et une phase de développement.

Devant l'état du prototype, il a été plus simple de recommencer le projet de zéro, en empruntant à l'ancienne version certains éléments fonctionnels.

Il m'a ainsi été possible de concevoir une nouvelle version, à l'architecture fondée sur des bases saines et respectueuses d'exigences fonctionnelles et non-fonctionnelles mises-à-jour pour correspondre au mieux aux attentes des futurs utilisateurs de l'application. La nouvelle version de GOAT a été pensée dès le départ pour être maintenable, et surtout ouverte à la modification. Son ouverture en open-source va lui permettre d'évoluer en même temps que les besoins des équipes de recherche.

Ce projet aura été pour moi une bonne expérience. Sur le plan technique, d'abord, car il m'aura permis d'appliquer les connaissances acquises dans le cadre de ma maîtrise à l'ETS et de programmer dans des technologies modernes et innovantes. Sur le plan humain, ensuite, car les membres de l'équipe de recherche étaient d'origines et de formations diverses.

BIBLIOGRAPHIE

- [1] The Apache Software Foundation. (2004). *Apache License, Version 2.0*. En ligne.
<<http://www.apache.org/licenses/LICENSE-2.0>> Consulté le 15 mars 2016.
- [2] Django Software Foundation. [s.d.]. *The web framework for perfectionists with deadlines*
| *Django*. En ligne. <<http://www.apache.org/licenses/LICENSE-2.0>> Consulté le 12
janvier 2016.
- [3] World Wide Web Consortium. [s.d.]. *HTML 5.1*. En ligne : <<https://w3c.github.io/html/>>
Consulté le 05 avril 2016.
- [4] Hampton Catlin, Natalie Weizenbaum, Chris Eppstein et de nombreux contributeurs.
2006. *Sass*, version 3.4.18. Logiciel, MIT-License.
- [5] W3Schools [s.d.]. *CSS*, version 3. En ligne :
<<http://www.w3schools.com/cssref/default.asp>> Consulté le 25 mars 2016.
- [6] GulpJS. 2014. *Gulp*, version 3.9.0. Logiciel, MIT-License. Récupéré sur
<<https://github.com/gulpjs/gulp>> le 12 janvier 2016.
- [7] The jQuery Foundation. *jQuery*, version 1.11.3. Cadriceil JavaScript. Récupéré sur
<<https://code.jquery.com/jquery-1.11.3.js>> le 12 janvier 2016.
- [8] Ryan Lanciaux, Joel Lanciaux et de nombreux contributeurs. *Griddle*, version 0.9.0.
Module JavaScript basé sur le cadriceil React.js. Récupéré sur
<<https://github.com/GriddleGriddle/Griddle>> le 02 février 2016.

- [9] Oracle. *MySQL*, version 5.6.28. Logiciel. Récupéré sur <http://dev.mysql.com/downloads/installer/> le 22 janvier 2016.
- [10] James R. Groff, Paul N. Weinberg et Andy Oppel. 2010. *SQL, the complete reference*, 3rd ed. ISBN : 978-0-07-159255-0. USA : McGraw-Hill, 882 p.
- [11] Github, Inc. [2016]. *Github*. En ligne : <https://github.com/> Consulté le 10 janvier 2016.
- [12] Evans, Eric J. (2004). *Domain Driven Design : Tackling Complexity in the Heart of Software*, 2nd ed. ISBN : 0-321-12521-5. Boston : Pearson Education, 529p.
- [13] Wikipedia Foundation, Inc, (2016). *Single-Page Application*. En ligne : https://en.wikipedia.org/wiki/Single-page_application Consulté le 22 mars 2016.
- [14] Facebook, Inc, (2013). *A JavaScript Library for user interfaces | React*. En ligne : <https://facebook.github.io/react/> Consulté le 20 janvier 2016.
- [15] Facebook, Inc, (2014). *Flux | Application Architecture for Building User Interfaces*. En ligne : <https://facebook.github.io/flux/> Consulté le 25 janvier 2016.
- [16] W3Schools, [s.d]. *AJAX Introduction*. En ligne : http://www.w3schools.com/ajax/ajax_intro.asp Consulté le 25 janvier 2016.
- [17] Mozilla Developer Network, (2016). *JavaScript | MDN*. En ligne : <https://developer.mozilla.org/fr/docs/Web/JavaScript> Consulté le 22 mars 2016.
- [18] JetBrains, s.r.o. 2011. *JetBrains Pycharm*, version 5.0.3. Logiciel. Boston (USA) : JetBrains, s.r.o. Récupéré sur : <https://www.jetbrains.com/pycharm/download/#section=windows>

- [19] Wikipedia Foundation, Inc, (2003). *Environnement de développement*. En ligne : https://fr.wikipedia.org/wiki/Environnement_de_d%C3%A9veloppement > Consulté le 3 avril 2016.
- [20] Github, Inc. 2014. *Atom*, version 1.6.2. Logiciel, MIT-License. Récupéré sur <https://atom.io/>.
- [21] Sergey Chikuyonok. 2012. *Emmet*, version Atom 2.4.1. Logiciel.
- [22] Rick Waldron, Caitlin Potter, Mike Sherov, Mike Pennisi et Luke Page. 2011. *JSHint*. Logiciel.
- [23] Nicolas C. Zakas et Nicole Sullivan. 2011. *CSSLint*. Logiciel.
- [24] Linus Trovald. 2005. *Git*, version 1.9.5. Logiciel. Software Freedom Conservancy. Open-source (GNU General Public License version 2.0).
- [25] Sebastian McKenzie. 2014. *Babel*, version 6.5.0. Logiciel. Open-source (MIT License).
- [26] James Halliday. 2014. *Browserify*, version 12.0.0. Logiciel. Open-source (MIT License).
- [27] Continuum Analytics. 2012. *Bokeh*, version 0.11.1. Logiciel. Continuum Analytics, All rights reserved.
- [28] The Apache Software Foundation. 2014. *Apache Spark*, version 1.0.0. Logiciel. Open-source (License Apache v2.0).
- [29] Samuel Vasko et la « cmdrdev team ». 2015. *Cmdr*, version 1.2. Logiciel. Open-source (License MIT).

- [30] Twitter. 2011. *Bootstrap*, version 3.3.6. Logiciel. Open-source (License MIT).
- [31] McConnell, Steve. 2004. *Code Complete*, 2nd éd, chapitre 24. ISBN : 978-0-7356-1967-8. Redmond : Microsoft Press, 931 p.
- [32] Continuum Analytics, (2012). *Bokeh Application*. En ligne :
<<http://demo.bokehplots.com/apps/movies>> Consulté le 10 avril 2016.
- [33] ISO/IEC 25010. 2011. *Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) – System and software quality models*. Norme ISO/IEC JTC 1/SC 7. 34p.

ANNEXE I

EXIGENCES FONCTIONNELLES

Tableau-A I-1 : Exigences fonctionnelles basées sur l'input identifiant et mot de passe

Inputs	Description de l'exigence
Identifiant, Mot de passe	EF1 : L'utilisateur doit pouvoir se connecter à l'application avec son identifiant personnel et son mode de passe.

Tableau-A I-2 : Exigences fonctionnelles basées sur l'input rsID

Inputs	Description de l'exigence
rsID	EF2 : L'utilisateur, en fournissant un rsID, (identifiant d'un SNP) doit pouvoir obtenir la liste des phénotypes dans lesquels ce SNP est significatif.
	EF3 : L'utilisateur, en fournissant un rsID, doit pouvoir obtenir, pour chaque phénotype dans lequel il est significatif, un tableau représentant l'ensemble des SNPs significatifs à ce phénotype.
	EF4 : L'utilisateur, en fournissant un rsID, doit pouvoir obtenir, pour chaque phénotype dans lequel il est significatif, un graphe Manhattan interactif du GWAS correspondant.

Tableau-A I-3: Exigences fonctionnelles basées sur l'input gène

Inputs	Description de l'exigence
Nom d'un gène	EF5 : L'utilisateur, en fournissant le nom d'un gène, doit pouvoir obtenir la liste des phénotypes dans lesquels ce gène contient des SNPs significatifs.
	EF6 : L'utilisateur, en fournissant le nom d'un gène, doit pouvoir obtenir, pour chaque phénotype dans lequel il contient des SNPs significatifs, un tableau représentant l'ensemble des SNPs significatifs à ce phénotype.
	EF7 : L'utilisateur, en fournissant le nom d'un gène, doit pouvoir obtenir, pour

	chaque phénotype dans lequel il contient des SNPs significatifs, un graphe Manhattan interactif du GWAS correspondant.
	EF8 : L'utilisateur, en fournissant le nom d'un gène, doit pouvoir obtenir la liste des SNPs significatifs présent sur ce gène, sur le gène précédent, et le gène suivant.

Tableau-A I-4 : Exigences fonctionnelles basées sur l'input phénotype

Inputs	Description de l'exigence
Nom d'un phénotype	EF9 : L'utilisateur, en fournissant le nom d'un phénotype, doit pouvoir obtenir un tableau représentant l'ensemble des SNPs significatifs à ce phénotype.
	EF10 : L'utilisateur, en fournissant le nom d'un phénotype, doit pouvoir obtenir un graphe Manhattan interactif du GWAS correspondant.

Tableau-A I-5: Exigences fonctionnelles basées sur les inputs phénotype, chromosome et position

Inputs	Description de l'exigence
Numéro de chromosome, Position, Nom d'un phénotype	EF11 : L'utilisateur, en fournissant conjointement le numéro de chromosome (de 1 à 22), une position et le nom d'un phénotype, doit pouvoir accéder à un graphe Manhattan interactif centrée et zoomé sur cette position, accompagné d'un tableau recensant tous les SNPs de la zone et leurs caractéristiques.

ANNEXE II

EXIGENCES NON-FONCTIONNELLES

Tableau-A II-1: Exigence non-fonctionnelle : Efficacité de rendement

ENF1 : Efficacité de rendement	
Comportement vis-à-vis du temps	L'utilisateur doit pouvoir accéder à n'importe quelle fonctionnalité en moins de 30 secondes.

Tableau-A II-2 : Exigence non-fonctionnelle : Facilité d'utilisation

ENF2 : Facilité d'utilisation	
Pertinence	Les données affichées doivent correspondre à la requête de l'utilisateur.
Opérabilité / Facilité d'apprentissage	L'application doit être utilisable par n'importe quel utilisateur sachant remplir un formulaire web.
Protection de l'utilisateur à faire une erreur	L'application doit prévenir la soumission de données d'input incorrectes et avertir l'utilisateur le cas échéant, sans planter.
Interface utilisateur esthétique	L'interface utilisateur doit s'adapter à tous les écrans de largeur supérieure à 1300 pixels. La langue de l'UI est l'anglais.

Tableau-A II-3 : Exigence non-fonctionnelle : Fiabilité

ENF3 : Fiabilité	
Disponibilité	L'application doit être utilisable par toute personne munie d'identifiants valides et connecté au réseau du CRCHUM.
Tolérance aux pannes / Rétablissement	L'application doit pouvoir se relancer toute seule en cas de problème, et écrire dans un fichier de logs l'erreur éventuelle.

Tableau-A II-4: Exigence non-fonctionnelle : Sécurité

ENF4 : Sécurité	
Confidentialité	Tout accès aux données sera inscrit dans un fichier de logs spécifique. Les informations inscrites dans le fichier sont les suivantes : • Timestamp (au format UTC) • Nom d'utilisateur • Adresse IP de l'utilisateur • Description de l'interaction utilisateur • Source des données • Nom des tables de données concernées • Clé primaire.

Tableau-A II-5: Exigence non-fonctionnelle : Maintenabilité

ENF5 : Maintenabilité	
Modularité	Le code de l'application doit être le plus modulaire possible, afin de séparer au maximum les responsabilités de chaque module et ainsi limiter l'impact d'une quelconque modification sur d'autres modules.
Réutilisabilité	La séparation des responsabilités doit permettre, en particulier sur l'interface utilisateur, la réutilisation de certains modules afin de faciliter l'ajout de nouvelles fonctionnalités.
Modificabilité	L'architecture de l'application doit être pensée pour permettre de modifier n'importe quel module en ayant un minimum d'impact sur les autres.

ANNEXE III

FONCTIONNALITES ET ETATS « FRONT-END »

Tableau-A III-1: Description des exigences « font-end »

Fonctionnalité	Description	Exigences Fonctionnelles	État
Login	L'utilisateur peut se connecter avec son identifiant et mot de passe pour accéder à l'application.	EF1	Login
AreaSelection	L'utilisateur, en entrant un chromosome, un phénotype et une position, peut accéder à un graphe interactif centré autour de cette position.	EF11	AreaSelection
Gene Query	L'utilisateur, en entrant un rsID, un gène ou un phénotype, peut accéder à un tableau listant les SNPs significatifs du phénotype choisi (ou correspondant au rsID ou gène en entrée) et leurs propriétés.	EF2 EF3 EF5 EF6 EF9	GeneQuery
Interactive Manhattan	L'utilisateur, en entrant un RSID, un gène ou un phénotype, peut accéder à un graphe Manhattan interactif représentant le GWAS du au phénotype choisi (ou correspondant au rsID ou gène en entrée).	EF4 EF7 EF10	Manhattan
Snps for one Gene	L'utilisateur, en entrant le nom d'un gène, doit pouvoir obtenir la liste des SNPs significatifs présents sur ce gène, sur le gène précédent, et le gène suivant.	EF8	SnpsForGene

Tableau-A III-2: États et États intermédiaires correspondants

État	État intermédiaire correspondant
GeneQuery	GeneQueryParams
AreaSelection	AreaSelectionParams
SnPForGene	SnPForGeneParams
Manhattan	GeneQueryParams NB : Les données d'entrée pour obtenir le graphe Manhattan interactif sont les mêmes que pour obtenir le tableau de SNPs significatifs. De fait, l'état intermédiaire est le même que pour GeneQuery, et le formulaire comportera deux boutons donnant accès à chacune des deux fonctionnalités.

ANNEXE IV

CAS D'UTILISATION DE L'INTERFACE DE GOAT

Tableau-A IV-1: Login

Cas d'utilisation 1-1 : Login	
Given	
Précondition	L'utilisateur est connecté au réseau auquel le serveur où est installé GOAT est relié.
When	
Action	L'utilisateur entre un identifiant et un mot de passe valides
Then	
Post conditions	L'utilisateur est connecté. L'état de l'application passe de « Login » à « Home ». L'application affiche la liste des modules disponibles.

Tableau-A IV-2: Login 2

Cas d'utilisation 1-2 : Login	
Given	
Précondition	L'utilisateur est connecté au réseau sur lequel le serveur où est installé GOAT est installé.
When	
Action	L'utilisateur entre un identifiant et un mot de passe invalides
Then	
Post conditions	L'utilisateur n'est connecté. L'état de l'application reste « Login ». L'application affiche un message d'erreur.

Tableau-A IV-3: Choix de module

Cas d'utilisation 2-1 : Choix de module	
Given	
Précondition	L'utilisateur est connecté à l'application. L'application est en état « Home ».
When	
Action	L'utilisateur clique sur un module.
Then	
Post conditions	L'état de l'application passe de « Login » à celui intermédiaire du module concerné. L'application affiche le formulaire d'entrée des données du module concerné.

Tableau-A IV-5: Gene Query

Cas d'utilisation 3-1 : Gene Query	
Given	
Précondition	L'utilisateur est connecté à l'application. L'application est en état « GeneQueryParams », et affiche le formulaire d'entrée de données.
When	
Action	L'utilisateur entre un rsID, le nom d'un gène ou celui d'un phénotype dans l'input correspondant. Il clique ensuite sur le bouton de validation du formulaire.
Données d'entrée	Un rsID valide, ou le nom d'un gène valide, ou le nom d'un phénotype valide.
Then	
Post conditions	L'état de l'application passe de « GeneQueryParams » à « GeneQuery ».

	L'application récupère les données, et affiche le tableau des SNPs significatifs correspondants aux données d'entrée.
--	---

Tableau-A IV-6 : Gene Query 2

Cas d'utilisation 3-2 : Gene Query	
Given	
Précondition	L'utilisateur est connecté à l'application. L'application est en état « GeneQueryParams », et affiche le formulaire d'entrée de données.
When	
Action	L'utilisateur entre un rsID, le nom d'un gène dans l'input correspondant. Il clique ensuite sur le bouton de validation du formulaire.
Données d'entrée	Un rsID valide, ou le nom d'un gène valide, mais n'étant significatif pour aucun phénotype.
Then	
Post conditions	L'état de l'application reste « GeneQueryParams ». L'application indique un message indiquant l'absence de phénotype correspondant aux données d'entrées.

Tableau-A IV-7: Gene Query 3

Cas d'utilisation 3-3 : Gene Query	
Given	
Précondition	L'utilisateur est connecté à l'application. L'application est en état « GeneQueryParams », et affiche le formulaire d'entrée de données.
When	
Action	L'utilisateur entre un rsID, le nom d'un gène ou celui d'un phénotype dans l'input correspondant.

	Il clique ensuite sur le bouton de validation du formulaire.
Données d'entrée	Un rsID invalide, ou le nom d'un gène invalide, ou le nom d'un phénotype invalide.
Then	
Post conditions	L'état de l'application reste « GeneQueryParams ». L'application indique l'invalidité de la donnée d'entrée.

Tableau-A IV-8: Manhattan

Cas d'utilisation 4-1 : Manhattan	
Given	
Précondition	L'utilisateur est connecté à l'application. L'application est en état « GeneQueryParams », et affiche le formulaire d'entrée de données.
When	
Action	L'utilisateur entre un rsID, le nom d'un gène ou celui d'un phénotype dans l'input correspondant. Il clique ensuite sur le bouton de validation du formulaire.
Données d'entrée	Un rsID valide, ou le nom d'un gène valide, ou le nom d'un phénotype valide.
Then	
Post conditions	L'état de l'application passe de « GeneQueryParams » à « Manhattan ». L'application récupère les données via le serveur, et affiche le graphe Manhattan interactif correspondant aux données d'entrée.

Tableau-A IV-9: Manhattan 2

Cas d'utilisation 4-2 : Manhattan	
Given	
Précondition	L'utilisateur est connecté à l'application. L'application est en état « GeneQueryParams », et affiche le formulaire d'entrée de données.
When	
Action	L'utilisateur entre un rsID, le nom d'un gène dans l'input correspondant. Il clique ensuite sur le bouton de validation du formulaire.
Données d'entrée	Un rsID valide, ou le nom d'un gène valide, mais n'étant significatif pour aucun phénotype.
Then	
Post conditions	L'état de l'application reste « GeneQueryParams ». L'application indique un message indiquant l'absence de phénotype correspondant aux données d'entrées.

Tableau-A IV-10: Manhattan 3

Cas d'utilisation 4-3 : Manhattan	
Given	
Précondition	L'utilisateur est connecté à l'application. L'application est en état « GeneQueryParams », et affiche le formulaire d'entrée de données.
When	
Action	L'utilisateur entre un rsID, le nom d'un gène ou celui d'un phénotype dans l'input correspondant. Il clique ensuite sur le bouton de validation du formulaire.
Données d'entrée	Un rsID invalide, ou le nom d'un gène invalide, ou le nom d'un phénotype invalide.

Then	
Post conditions	L'état de l'application reste « GeneQueryParams ». L'application indique l'invalidité de la donnée d'entrée.

Tableau-A IV-11: Area Selection

Cas d'utilisation 5-1 : AreaSelection	
Given	
Précondition	L'utilisateur est connecté à l'application. L'application est en état « AreaSelectionParams », et affiche le formulaire d'entrée de données.
When	
Action	L'utilisateur entre un chromosome, une position et un phénotype dans les inputs correspondant. Il clique ensuite sur le bouton de validation du formulaire.
Données d'entrée	Un numéro de chromosome valide, une position valide, et le nom d'un phénotype valide.
Then	
Post conditions	L'état de l'application passe de « AreaSelectionParams » à « AreaSelection ». L'application récupère les données, et affiche le tableau des SNPs significatifs correspondants aux données d'entrée et le graphe interactif, centré dans la position indiquée en entrée. Le graphe interactif est adapté à la taille d'écran de l'utilisateur.

Tableau-A IV-12: Area Selection 2

Cas d'utilisation 5-2 : AreaSelection	
Given	
Précondition	L'utilisateur est connecté à l'application. L'application est en état « AreaSelectionParams », et affiche le formulaire d'entrée de données.
When	
Action	L'utilisateur entre un chromosome, une position et un phénotype dans les inputs correspondant. Il clique ensuite sur le bouton de validation du formulaire.
Données d'entrée	Un numéro de chromosome invalide, une position invalide, ou le nom d'un phénotype invalide.
Then	
Post conditions	L'état de l'application reste « AreaSelectionParams ». L'application interroge le serveur, et un message d'erreur indiquant que la donnée d'entrée invalide.

Tableau-A IV-13: Area Selection 3

Cas d'utilisation 5-3 : AreaSelection	
Given	
Précondition	L'utilisateur est connecté à l'application. L'application est en état « GeneQuery », et affiche un tableau de SNPs.
When	
Action	L'utilisateur clique sur le rsID d'un SNP sur une des lignes du tableau.
Données d'entrée	Un numéro de chromosome valide, une position valide, ou le nom d'un phénotype valide (présents dans le tableau).
Then	

Post conditions	L'état de l'application devient « AreaSelection ». L'application récupère les données, et affiche le tableau des SNPs significatifs correspondants aux données d'entrée et le graphe interactif, centré dans la position indiquée en entrée. Le graphe interactif est adapté à la taille d'écran de l'utilisateur.
-----------------	---

Tableau-A IV-14: SNP for Gene

Cas d'utilisation 6-1 : SnpForGene	
Given	
Précondition	L'utilisateur est connecté à l'application. L'application est en état « SnpForGeneParams », et affiche le formulaire d'entrée de données.
When	
Action	L'utilisateur entre le nom d'un gène. Il clique ensuite sur le bouton de validation du formulaire.
Données d'entrée	Un nom de gène valide.
Then	
Post conditions	L'état de l'application passe de « SnpForGeneParams » à « SnpForGene ». L'application affiche la liste des SNPs significatifs présents sur le gène en entrée, le gène suivant et le gène précédent.

Tableau-A IV-15: SNP for Gene 2

Cas d'utilisation 6-2 : SnpForGene	
Given	
Précondition	L'utilisateur est connecté à l'application. L'application est en état « SnpForGeneParams », et affiche le formulaire d'entrée de données.
When	
Action	L'utilisateur entre le nom d'un gène. Il clique ensuite sur le bouton de validation du formulaire.
Données d'entrée	Un nom de gène invalide.
Then	
Post conditions	L'état de l'application reste « SnpForGeneParams ». L'application affiche un message d'erreur.

ANNEXE V

MODULES « BACK-END »

Tableau-A V-1: Description Module Web

Module	Inputs	Outputs
Web	Aucun	Renvoie le fichier index.html.
Route : / Exemple : Si l'utilisateur souhaite accéder à l'application, il se rend à l'adresse allouée ce qui a pour effet d'automatiquement effectuer une requête GET à l'adresse « http://serveradress/ ».		

Tableau-A V-2: Description Module AreaSelection

Module	Inputs	Outputs
AreaSelection	- Chromosome : entre 1 et 22 - Position : nombre entier positif - Phénotype : nom d'un phénotype existant en base de données - Largeur de l'écran de l'utilisateur - Hauteur de l'écran de l'utilisateur	Objet JSON contenant : - Script Bokeh pour générer le graphe interactif, adapté à la taille de l'écran de l'utilisateur - Table d'objets contenant les informations sur les SNPs nécessaires pour générer le tableau interactifs - ID de la div à utiliser pour le script Bokeh

Route :

<http://serveradress/areaSelection/chromosome/position/phenotype/userWidth/userHeight>

Exemple : Si l'utilisateur, dont l'écran est large de 1600px (pixels) et haut de 900px, souhaite accéder au graphe interactif areaSelection, sur le chromosome 5, centré en la position 6000000 et sur le phénotype « All cause death », l'application effectuera une requête GET à l'adresse suivante :

« [http://serveradress/areaSelection/5/6000000/All cause death/1600/900](http://serveradress/areaSelection/5/6000000/All%20cause%20death/1600/900) »

Tableau-A V-3: Description Module Manhattan

Module	Inputs	Outputs
Manhattan	- Type d'input, pouvant prendre trois valeurs : « phenotype » « gene » « rsID » - Valeur du rsID, du gène ou du phenotype entré - Largeur de l'écran de l'utilisateur - Hauteur de l'écran de l'utilisateur	Objet JSON contenant : - Script Bokeh pour générer le graphe interactif, adapté à la taille de l'écran de l'utilisateur - ID de la div à utiliser pour le script Bokeh

Route : <http://serveradress/manhattan/type/value/userWidth/userHeight>

Exemple : Si l'utilisateur, dont l'écran est large de 1500px et haut de 600px, souhaite accéder au graphe Manhattan du phenotype « All cause death », l'application effectuera une requête GET à l'adresse :

« [http://serveradress/manhattan/phenotype/All cause death/1500/600](http://serveradress/manhattan/phenotype/All%20cause%20death/1500/600) »

Tableau-A V-4: Description Module Snp For Gene

Module	Inputs	Outputs
Snp for One Gene	Le nom d'un gène	Objet JSON contenant : Tableau contenant une liste d'objets, chacun correspondant à un SNP.
Route : http://serveradress/snpsForGene/gene Exemple : Si l'utilisateur souhaite accéder à la liste des SNPs significatifs présents sur le gène « SPOCK3 », le précédent ou le suivant, l'application effectuera une requête GET à l'adresse : « http://serveradress/snpsForGene/spock3 »		

Tableau-A V-5: Description Module AutoComplete

Module	Inputs	Outputs
AutoComplete	- Type d'input, pouvant prendre deux valeurs : « phenotype » « gene » - La valeur du gène ou du phénotype entré en input	Objet JSON contenant : Tableau contenant une liste d'objets, chacun correspondant à un SNP.
Route : http://serveradress/autocomplete/type/value Exemple 1: Si l'utilisateur souhaite accéder à la liste des gènes dont le nom commence, contient ou termine par « SPO », l'application effectuera une requête à l'adresse :		

« http://serveradress/autocomplete/gene/spo»

Exemple 2: Si l'utilisateur souhaite accéder à la liste des phénotypes dont le nom commence, contient ou termine par « All », l'application effectuera une requête GET à l'adresse :

« http://serveradress/autocomplete/phenotype/all»

Tableau-A V-6: Description Modules Phenotypes

Module	Inputs	Outputs
Phenotypes	Aucun	Objet JSON contenant : Tableau contenant la liste des phénotypes présents en base de données.
Route : http://serveradress/phenotypes/ Exemple : Si l'utilisateur souhaite accéder à la liste des phénotypes l'application effectuera une requête à l'adresse : « http://serveradress/phenotypes/»		

ANNEXE VI

VISUELS

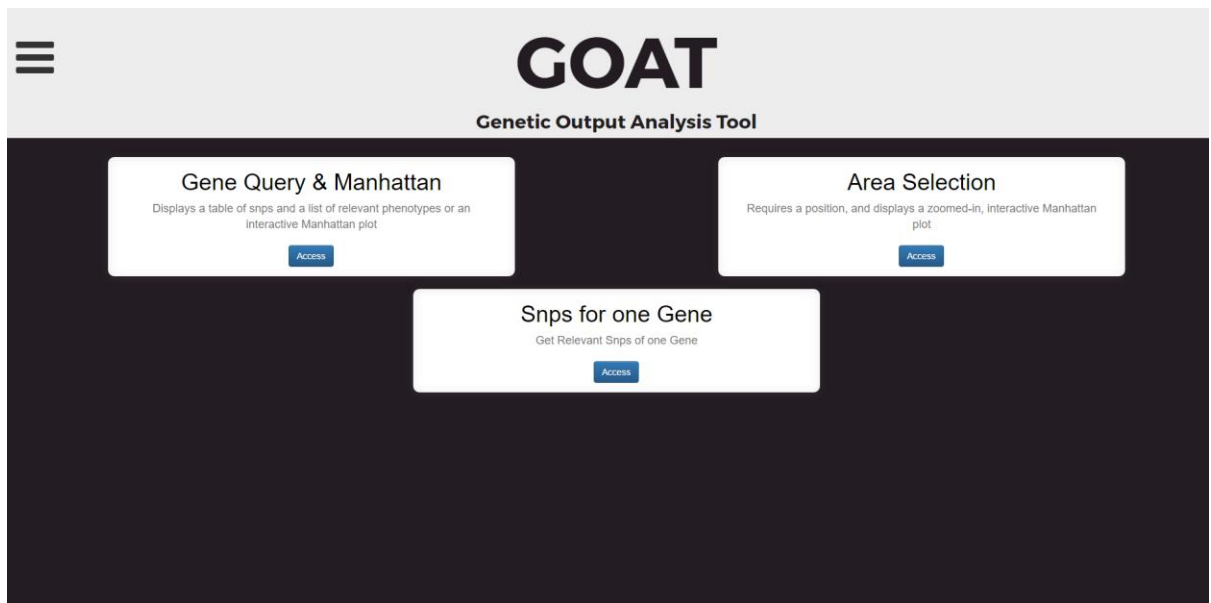


Figure-A VI-1 : Écran d'accueil

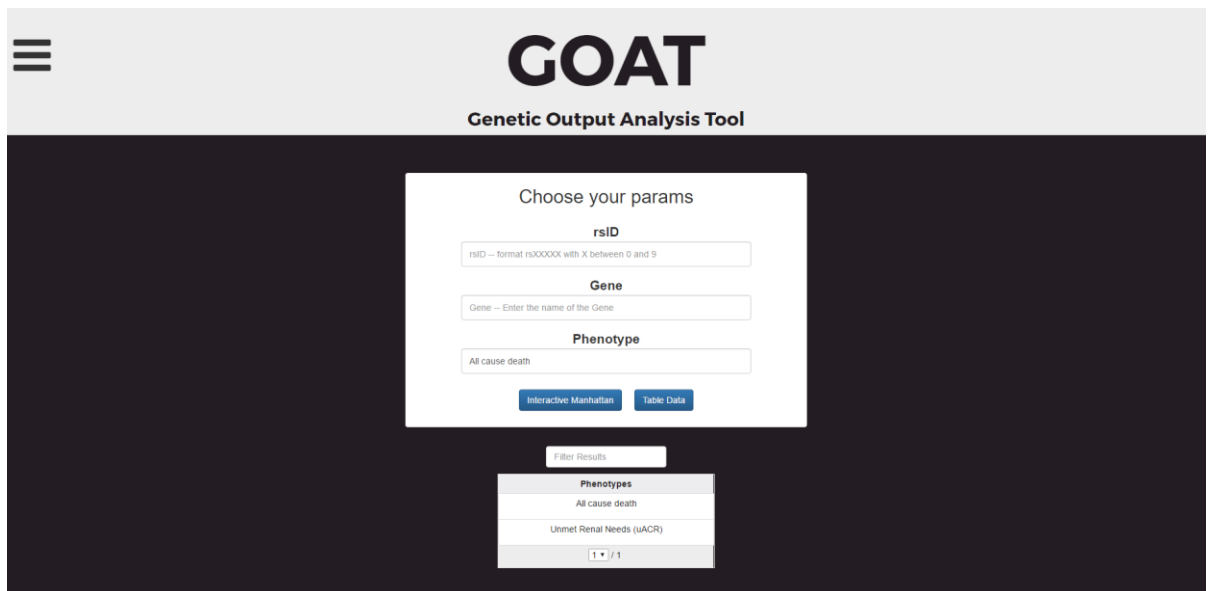


Figure-A VI-2 : Écran Gene Query Params

Gene Query

Filter Results

Phenotypes

All cause death

1 / 1

Filter Results

SETTINGS

rs_ID	Chr	P-value	GeneBefore	Gene	GeneAfter	Pos	experiment	info_assoc	Allele A	Allele B	cohort AA	cohort BB	Beta Assoc	maf	all_OR	Covariates
rs77108552	18	0.00091456		DOK5	CD226	67312065	4	0.97745	G	A	3059.3	14.307	0.50905	0.058574	1.5848	age sex pc1 pc2
rs7216372	17	0.00050988	CASC17		LOC102723505	69625434	4	1	G	C	2785	31	0.42419	0.10075	1.4217	age sex pc1 pc2
rs8077947	17	0.00094118	CASC17		LOC102723505	69626105	4	0.99791	G	C	2795.5	32	0.40551	0.09938	1.4019	age sex pc1 pc2
rs4350607	17	0.00087987	CASC17		LOC102723505	69626669	4	0.99684	C	T	2794.8	31.998	0.40793	0.099474	1.4037	age sex pc1 pc2
rs4456541	17	0.00084705	CASC17		LOC102723505	69627020	4	0.99678	A	T	2794.4	31.998	0.40917	0.099533	1.4047	age sex pc1 pc2
rs4479291	17	0.0005865	CASC17		LOC102723505	69630585	4	1	G	A	2793	32	0.41558	0.099739	1.4082	age sex pc1 pc2
rs7845649	8	0.00075694	CNGB3		CNBD1	87873220	4	0.99608	T	C	1095.4	640.55	0.24827	0.43405	1.277	age sex pc1 pc2
rs55688002	17	0.00068518	CASC17		LOC102723505	69641991	4	0.98804	A	G	2804.8	33.616	0.41705	0.098266	1.4263	age sex pc1 pc2
rs12603056	17	0.00063004	CASC17		LOC102723505	69642189	4	0.98733	A	G	2804.7	32.682	0.42076	0.098144	1.4292	age sex pc1 pc2
rs55685923	17	0.00058505	CASC17		LOC102723505	69644442	4	0.98046	T	G	2805.3	34.158	0.42347	0.098267	1.4328	age sex pc1 pc2
rs3798269	6	0.0008742	CLIC5	CLIC5	ENPF4	45981962	4	1	C	T	662	1190	0.24087	0.42346	1.2739	age sex pc1 pc2

Figure-A VI-3 : Écran Gene Query

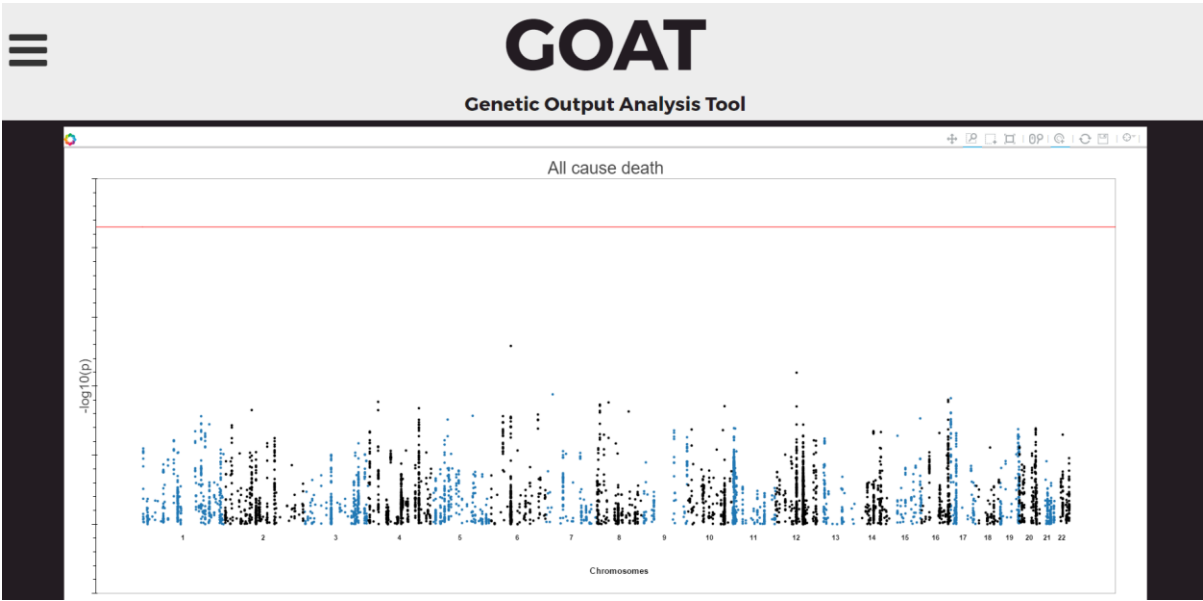


Figure-A VI-4 : Écran Interactive Manhattan

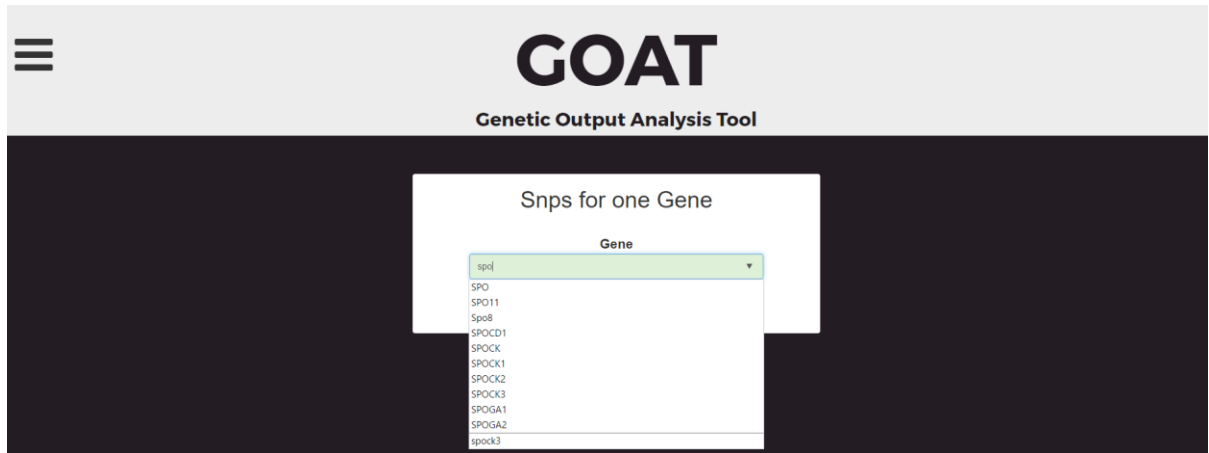


Figure-A VI-5: Snp For Gene Params

Snps for Gene : SPOCK3

Filter Results

Phenotypes

Unmet Renal Needs (uACR)

1 / 1

Filter Results

SETTINGS

rs_ID	Chr	P-value	GeneBefore	Gene	GeneAfter	Pos	experiment	info_assoc	Allele A	Allele B	cohort AA	cohort BB	Beta Assoc	maf	all_OR	Covariates	Phenotype	Risk Allele	Risk Af	Risk Allele Beta
rs4445346	4	0.00053406		SPOCK3		167889796	5	0.97834	G	T	2387.6	60.939	0.25872	0.13393	1.2753	age sex pc1 pc2	Unmet Renal Needs (uACR)	T	0.133935546	0.2587200105
rs1346376	4	0.00032658		SPOCK3		167892885	5	1	C	T	2398	61	0.26638	0.13232	1.2908	age sex pc1 pc2	Unmet Renal Needs (uACR)	T	0.132315922	0.266380012
rs62352380	4	0.00035188		SPOCK3		167939771	5	0.93947	G	A	2419.9	59.044	0.27624	0.12857	1.283	age sex pc1 pc2	Unmet Renal Needs (uACR)	A	0.1285654286	0.2762399912
rs62352385	4	0.00041309		SPOCK3		167971835	5	0.9302	G	T	2389.9	70.778	0.26654	0.13512	1.2741	age sex pc1 pc2	Unmet Renal Needs (uACR)	T	0.1351238826	0.2665399909
rs142394366	4	0.00084702		SPOCK3		167979851	5	0.93456	C	T	2254.1	89.91	0.23616	0.1995	1.233	age sex pc1 pc2	Unmet Renal Needs (uACR)	T	0.1995022554	0.2361599952
rs10050205	4	0.00081259		SPOCK3		167992004	5	0.938	T	C	2127.4	113.98	0.22458	0.18322	1.223	age sex pc1 pc2	Unmet Renal Needs (uACR)	C	0.1832183266	0.2245800048

Figure-A VI-6 : Snp For Gene

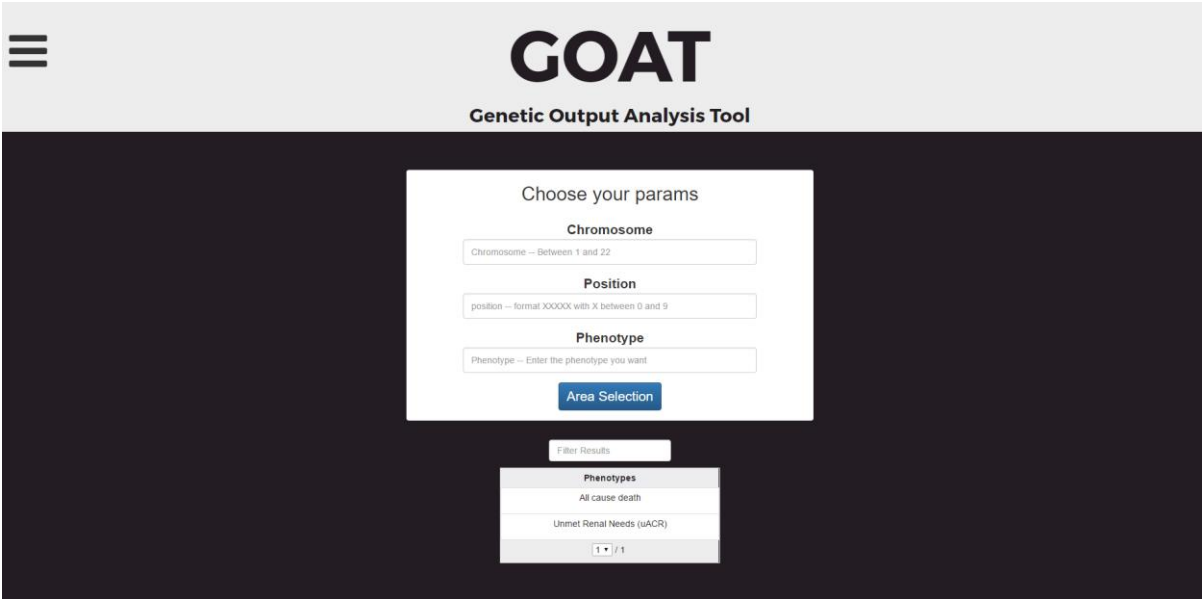


Figure-A VI-7 : Area Selection Params

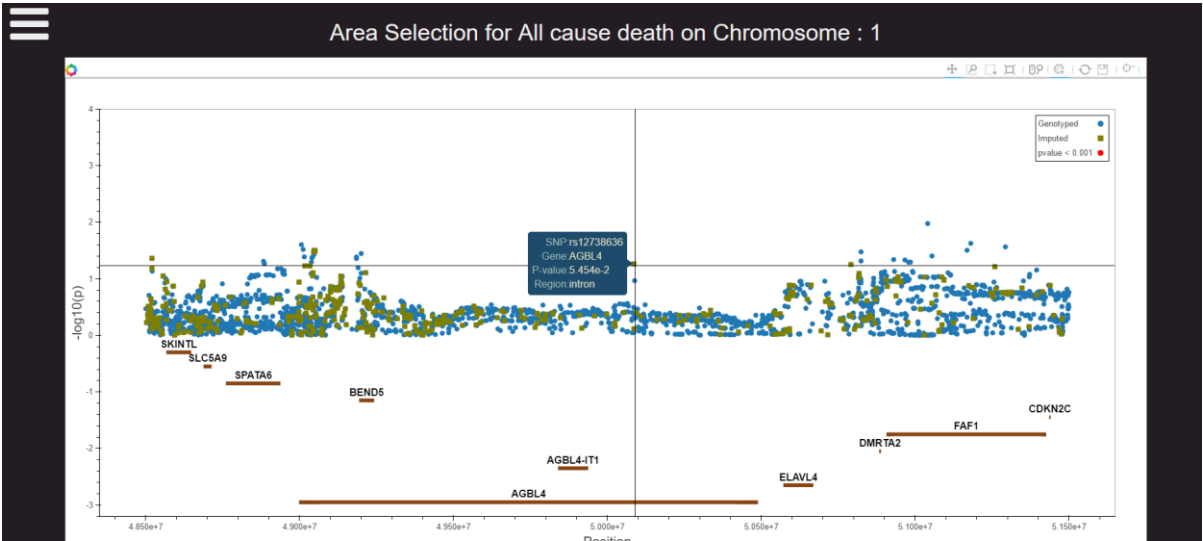


Figure-A VI-8: Area Selection 1

Filter Results

SETTINGS																		
rs_ID	Chr	Position	P-value	GeneBefore	Gene	GeneAfter	dbSNP	GWAS_Cat alog	GeneCards	Region	Imputed	Allele A	Allele B	Allele mineure	risk_allele	risk_af	risk_allele_ beta	Cohort
rs722656	1	48500746	0.33704	TRABD2B		SKINTL	dbSNP : rs722656	- rs_ID - gene before - gene - gene after	- gene before - gene - gene after	unknown		C	T	T	C	0.6320654333	0.074082	Advance_3449
rs303925	1	48503280	0.59437	TRABD2B		SKINTL	dbSNP : rs303925	- rs_ID - gene before - gene - gene after	- gene before - gene - gene after	unknown		C	T	T	C	0.5884314294	0.040017	Advance_3449
rs7554749	1	48506765	0.34365	TRABD2B		SKINTL	dbSNP : rs7554749	- rs_ID - gene before - gene - gene after	- gene before - gene - gene after	unknown		A	G	G	A	0.6350391418	0.07325	Advance_3449
rs6588642	1	48507037	0.44362	TRABD2B		SKINTL	dbSNP : rs6588642	- rs_ID - gene before - gene - gene after	- gene before - gene - gene after	unknown		C	G	G	C	0.6306175703	0.058956	Advance_3449
rs6691050	1	48507278	0.41167	TRABD2B		SKINTL	dbSNP : rs6691050	- rs_ID - gene before - gene - gene after	- gene before - gene - gene after	unknown		G	A	A	G	0.6308785155	0.063233	Advance_3449
rs1999593	1	48508854	0.34759	TRABD2B		SKINTL	dbSNP : rs1999593	- rs_ID - gene before - gene	- gene before - gene	unknown		G	C	C	G	0.6342381314	0.07251	Advance_3449

Figure-A VI-9: Area Selection