



FACULTÉ DES SCIENCES ET GÉNIE

Stage en génie logiciel 1
GLO-2580
Été 2022
Baccalauréat en génie logiciel

Stagiaire développeur Fullstack
Direction des technologies de l'information (DTI)

Rapport de fin de stage

Destinataire

Département des stages en milieu pratique

6 septembre 2022

Frédéric Gosselin
536 783 793

Superviseur de stage : Laurence Dionne Bibaud
Adjointe au Chef de secteur, RGR

1 Résumé

Le présent rapport décrit le travail effectué par Frédéric Gosselin dans le cadre de son stage en informatique au sein de la Direction des technologies de l'information (DTI) à l'Université Laval, pendant l'été 2022, dans le secteur Recherche et Gestion de la Recherche (RGR), avec l'équipe de développement de l'infrastructure VALERIA. À titre de stagiaire en programmation backend et frontend, le but du travail est d'accomplir des fonctionnalités visant à mettre en place des produits pour effectuer la recherche.

Au courant de l'été, l'étudiant effectue des métriques d'utilisation en temps réel avec Keycloak des solutions VALERIA sous forme de documents Jupyter Notebook. Aussi, le stagiaire a pu mettre en place une infolettre faisant un lien entre les chaînes de courriels automatisées MailChimp et les bases de données centralisées PostgreSQL lors de l'inscription d'un nouveau membre ou directement sur le portail VALERIA. Il a également, construit des endpoints Rest (requêtes http) pour la gestion des collaborateurs dans le projet d'un chercheur. La partie frontend de ce dernier mandat, qui a été commencée par un développeur senior, sera poursuivie dans un emploi étudiant pour la session d'automne 2022.

Remerciements

J'aimerais remercier personnellement toute l'équipe de chez le projet VALERIA pour l'accueil qu'ils m'ont offert. Spécialement Kevin Arsenault et Pierre-Yves Langlois les développeurs analystes qui ont su m'encadrer lors de ce stage.

Copie envoyée à mon superviseur le 6 septembre 2022.

Table des matières

Introduction	5
Description du stage	6
Rôle et contributions du stagiaire	6
Objectifs du stage	6
Méthodologie	6
Tâches effectuées	7
Métriques d'utilisation VALERIA	7
Ajout du logo ulaval	8
Création d'une infolettre VALERIA	8
Gestion des collaborateurs	10
Comparaison avec les attentes du stagiaire avant le début de stage	11
Développement et renforcement des compétences	13
Technique et ingénierie	13
Communications	13
Réflexion sur la formation pratique	13
Réflexion sur la formation théorique	14
Bilan des acquis	14
Conclusion	14
Annexe A – Métriques d'utilisation VALERIA	16
Annexe B – Ajouts au portail principal et infolettre VALERIA	18
Annexe C – Gestion des collaborateurs backend	19
Annexe D – Logiciels	20

Liste des figures

Figure 0 – Interface Jupyter Notebook	8
Figure 1 – Patrons adapter et factory pour l'API MailChimp	9
Figure 2 – Diagramme d'états à l'inscription pour MailChimp	10
Figure 3 – Code obscur dans IDM	11
Figure 4 – Code clean dans IDM	11

2 Introduction

En tant que premier stage, ce rapport présente les travaux de Frédéric Gosselin, étudiant de deuxième année au baccalauréat en génie logiciel à l'Université Laval. Ces travaux s'échelonnent sur une période de 18 semaines pendant l'été 2022 et constituent une première expérience de travail dans son domaine d'étude.

Le stage a été réalisé au sein de la direction des technologies de l'information (DTI). Celle-ci œuvre dans les besoins d'affaires de l'Université Laval depuis 1980 et a subi plusieurs transformations organisationnelles depuis (notamment le passage du modèle Cascade au modèle Agile). Au niveau informatique, elle est composée de plus de 280 employés et est répartie en 5 secteurs : l'Enseignement, la Gestion des études, la Fondation technologique, l'Administration et Campus numérique et la Recherche et gestion de la recherche qui a accueilli l'étudiant. Ce dernier secteur a pour mission de « faciliter le travail des chercheurs et à répondre aux besoins spécifiques du domaine de la recherche grâce à des outils performants et innovants ». Les équipes de la Recherche et gestion de la recherche se situent au 7^{ème} étage du pavillon Louis-Jacques-Caseault situé au 1055, avenue du Séminaire.

Le stage consiste à l'ajout de fonctionnalités à l'infrastructure VALERIA. VALERIA offre plusieurs services comme, entre autres, du stockage S3, un accès gitlab, un environnement JupyterLab et un portail principal constitué d'un backend et d'un frontend. La plupart du travail a été réalisée sur JupyterLab ainsi que le portail principal. L'étudiant a été accueilli dans l'équipe chargée du maintien de VALERIA composée de 7 membres : 2 administrateurs RI (Ressources Informationnelles), 2 analystes RI, ainsi que de 3 développeurs analystes dont fait partie l'étudiant. Les membres se rencontrent tous les lundis, mardi et mercredi avec d'occasionnelles rencontres hebdomadaires entre les développeurs du même secteur. À son arrivée, l'étudiant a été jumelé avec Kevin, développeur senior de VALERIA, pour lui présenter ses différents modules, le familiariser avec les code reviews et lui inculquer la culture des tests unitaires dans ses programmes.

Le présent rapport présente une description du stage contenant le rôle du stagiaire, les objectifs de stage, les méthodes de travail et les tâches effectuées. Il poursuit avec une partie sur la réflexion pratique et théorique pré-stage et durant le stage, puis il fait un bilan des acquis.

3 Description du stage

3.1 Rôle et contributions du stagiaire

Le rôle principal occupé pendant le stage a été en tant que programmeur backend et frontend. Le stagiaire a travaillé sur une architecture d'aide à la recherche qui repose sur l'avis des utilisateurs pour donner de l'accès à des services ou développer de nouvelles fonctionnalités. C'est à la fin d'une itération que tous les membres du secteur vont chercher ces avis clients, mais les développeurs savent quelques jours à l'avance quels services sont plus utilisés que d'autres. Une première tâche était donc de fournir des statistiques d'utilisation sur les services principaux de l'architecture VALERIA. Par la suite, les tâches s'avéraient à être l'amélioration du portail VALERIA selon les demandes des clients. Ces tâches sont complétées et ont été analysées lors de rencontres hebdomadaires.

3.2 Objectifs du stage

Les objectifs du stage étaient de compléter les fonctionnalités au terme des épopées en lien dans Jira en respectant les délais prévus et les contrôles de qualité du code. Les défis relevés au cours du stage sont l'utilisation d'infrastructures complexes, travailler avec une équipe agile et interagir avec du code legacy. L'équipe de VALERIA devait pouvoir fournir des métriques d'utilisation de ses différents services mais aussi sur certaines plages données pour comparer certaines périodes tout en étant présentables. Il fallait donc faire un code sur une plateforme assez souple qui prenait ces aspects en charge. Pour ce qui est de l'amélioration du portail, une attention particulière a été portée au backend pour l'apprentissage de l'infrastructure logicielle Rest. L'étudiant devait aussi se familiariser avec le frontend.

3.3 Méthodologie

En arrivant la première semaine au pavillon Louis-Jacques-Caseault, un ordinateur portable est fourni au stagiaire. Parmi les logiciels préinstallés, on y compte le Cisco AnyConnect Security Mobile Client pour accéder au VPN du 7^{ème} étage et Microsoft Teams pour tout ce qui a trait aux communications d'équipe. Le stage est réalisé en formule hybride, l'étudiant peut se présenter à son lieu de travail avec une carte d'accès ou fonctionner par télétravail comme la plupart des autres employés. Comme il s'agit d'un stage principalement de programmation backend en java, l'environnement de déploiement intégré IntelliJ IDE est utilisé. L'équipe utilise la méthodologie Agile sur des Sprints de 3 semaines pour 3 rencontres par semaines (lundi, mardi et mercredi) où les membres discutent du travail qui a été fait et de leurs blocages, s'il y a lieu. De plus, les développeurs se rencontrent à tous les 2 semaines les mardis pour discuter de sujets tels que à quel point les architectures se doivent d'être complexes ou que signifie une bonne revue de code. Pour ce qui est de la réalisation des tâches, des tickets dans Jira sont émis. Une tâche doit correspondre à une branche dans git, avec un nom suivant le format « feature/mailchimp-add-user », par exemple. Pour la conception, des diagrammes de classes unitaires et du modèle, des diagrammes vus dans le cours de génie logiciel orienté objet, sont fournis par Kevin, programmeur senior de VALERIA et personne-ressource qui encadrerait le stagiaire. Avant de délivrer une tâche en production, il faut s'assurer que la fonctionnalité fait ce qu'on lui demande par l'entremise de tests unitaires, qu'elle passe les revues des codes par les autres développeurs et qu'elle compile dans Jenkins avec les bases de données de développement.

3.4 Tâches effectuées

3.4.1 Métriques d'utilisation VALERIA

La première tâche que s'est vu attribué le stagiaire a été la mise en place de statistiques d'utilisation sur les services de VALERIA utilisant Keycloak, un logiciel « d'authentification unique à travers la gestion par identité et par accès ». Entre autres, Keycloak enregistre au fur et à mesure les événements de connexion. On compte 4 services utilisant Keycloak, soit le GitLab, le navigateur pour stockage S3, le portail frontend VALERIA et une passerelle JupyterHub. JupyterHub fonctionne avec un système de Notebooks, où une page web peut rouler différents types de codes ou afficher du texte dans un navigateur comme Google Chrome. Le travail réalisé a été fait dans un seul Notebook Jupyter en python pour commencer les 2 premières semaines du stage. Une refonte a été faite la 3^{ème} semaine, pour un total de 10 scripts python .py et deux fichiers Jupyter Notebook .ipynb, un pour afficher les statistiques d'utilisation sous forme de graphe et un autre pour stocker ces données dans des fichiers CSV (comma separated values). La refonte vient du fait qu'on veut respecter les principes du clean code, mais que comme le script est utilitaire à l'interne pour les membres du même secteur qui, éventuellement, seraient tentés d'utiliser une version à peine modifiée du Notebook pour leurs autres logiciels, on se passe de tests unitaires, mais on garde un niveau de lisibilité au code. La librairie Keycloak étant un peu désuète sur l'environnement Jupyter, certaines fonctions présentent dans la documentation n'étaient pas disponibles, c'est pourquoi il a fallu faire plusieurs appels à la fonction `get_events` plusieurs fois pour un même intervalle parce qu'au final la fonction disponible pour obtenir les événements Keycloak retournait un nombre fixe d'événements. Les données sont retournées sous forme de tableaux de dictionnaire python avec des clés comme le `client_id` ou la date pour une case du tableau. Après plusieurs appels, ces tableaux sont fusionnés et les événements dont les `clients_id` sont les mêmes à des intervalles de 5 minutes sont supprimés : en utilisant le modulo sur la date on est capable de définir un créneau pour les événements. À ce point, les données peuvent être affichées dans des plots pythons ou rangées dans des tableaux de la librairie pandas, à utiliser dans les fichiers csv. Cependant, une étape de filtrage supplémentaire a été rajoutée pour donner suite à la demande d'un membre de l'équipe de recherche pour afficher les événements par individus uniques. Pour ce faire, on range les événements doublons ayant des `clients_id` et des `id_individus` similaires dans un ensemble python (Set). Il a été important d'être à l'aise avec ces structures de données imbriquées - des ensembles dans des tableaux et vice-versa - parce qu'il fallait par la suite procéder à un mappage des données à un domaine exprimé en jours, semaines ou mois. À ce point, les fonctions built-in de python comme `filter()` ou `map()` ont été très utiles ainsi que le site StackOverflow. Les événements recherchés étaient les types d'événements « LOGIN » dans Keycloak.

Le Notebook Jupyter fonctionne par la suite comme ceci : on fait l'appel d'une méthode `show_results` avec comme paramètres les dates de début et de fin ainsi que le type d'intervalle (jour, semaines ou mois).

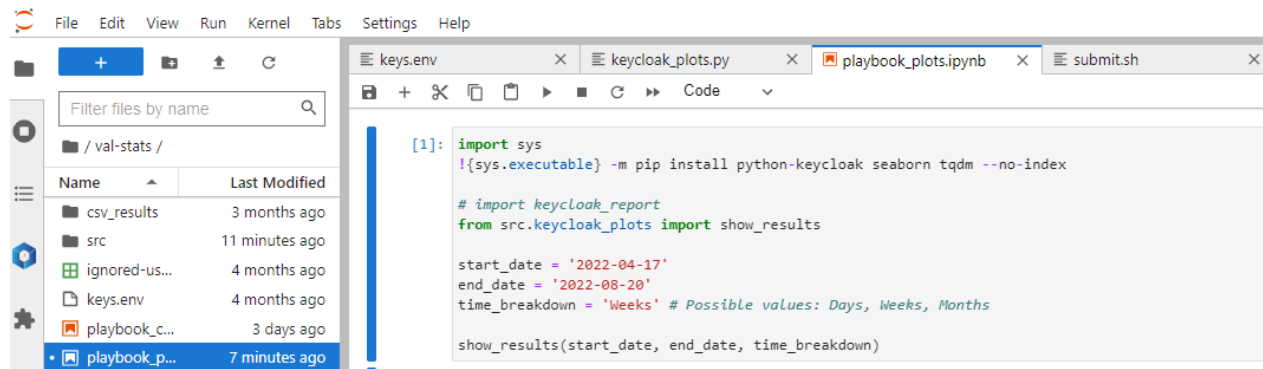


Figure 0 – interface Jupyter Notebook

Finalement, 4 graphes sont présentés avec pour acteurs chaque service : le nombre d'événements par intervalle sur la période donnée, le nombre total d'événements, les pourcentages d'événements et enfin le nombre d'événements par individus uniques par intervalle sur la période donnée. Par soucis de présentation, une barre de progression du script a été rajoutée avec la librairie python tqdm. Les métriques d'utilisations de VALERIA ont été présentées la 6^{ème} semaine en fin d'itération à tout le département de la recherche et gestion de la recherche.

3.4.2 Ajout du logo ulaval

Pour se familiariser avec le frontend, le stagiaire a utilisé des composantes React pour modifier l'en-tête du portail VALERIA. Le logo a aussi été ajouté au navigateur S3, un service séparé du portail donc il fallait émettre deux fonctionnalités dans Jira. Pendant l'ajout de cette fonctionnalité, le stagiaire a apporté des modifications au portail pour qu'il soit plus responsive, c'est-à-dire que le design soit adaptable sur tablette. Ainsi, un menu ouvrant à icône d'hamburger a été rajouté en temps réel selon la taille du site, toujours en utilisant des événements React.

3.4.3 Création d'une infolettre VALERIA

Avec les autres membres de l'équipe, il a été discuté de la création d'une infolettre pour fournir de la documentation et des nouveautés aux utilisateurs des services VALERIA. Sur la page d'inscription d'un compte VALERIA, un utilisateur peut cocher une case pour recevoir des communications. Un booléen est alors mis à jour sous l'onglets communications dans la base de données PostgreSQL. L'ajout d'une infolettre VALERIA vient du fait que les courriels, qui devraient être automatisés, étaient envoyés manuellement en collectant un fichier des adresses courriels dans VALERIA. Ce mandat se sépare donc en trois fonctionnalités : 2 endpoints Rest sur le backend, un pour l'ajout aux communications Mailchimp d'une adresse courriel et un autre pour l'ajout de l'idul ulaval ou du courriel aux communications Mailchimp à l'inscription, et un champ pour enregistrer son adresse courriel au niveau du frontend sur la page d'accueil du portail VALERIA, qui fait appel au premier endpoint Rest. En effet, Mailchimp est une compagnie américaine qui offre « une plate-forme de services marketing diversifiés pour les petites et moyennes entreprises ». Mailchimp fonctionne par requêtes Rest, donc des classes suivant les patrons de

conception factory et adapter sont ajoutées à une classe mère MailChimpApi : la factory en java se charge de construire une classe HttpAdapter faite maison avec les bonnes autorizations et HttpAdapter se charge d'effectuer les requêtes Rest à Mailchimp.

```
8 usages Goldenweiss
public void subscribe(String email, String name, String lastName, boolean subscribed) {
    // utiliser http client adapter pour faire les call à mail chimp.
    HttpClientAdapter mailchimpAdapter = httpClientFactory.get(email);

    JSONObject json = new JSONObject()
        .put("email_address", email)
        .put("status_if_new", subscribed ? "subscribed" : "unsubscribed")
        .put("status", subscribed ? "subscribed" : "unsubscribed");
    json.put("merge_fields", new JSONObject().put("FNAME", name == null ? "" : name).put("LNAME", lastName == null ? "" : lastName));

    HttpResponse response = null;
    try {
        response = mailchimpAdapter.put(json.toString());
    }
```

Figure 1 - patrons adapter et factory pour l'API Rest MailChimp

Parmi ces requêtes, on compte l'ajout d'un membre Mailchimp, sa suppression ou vérifier s'il existe. Une requête Rest, c'est une requête http contenant dans son entête un verbe sur le type d'action à effectuer (get, post put, delete, etc.), les autorisations (le mot Basic suivi de l'utilisateur et du mot de passe qu'on encode au format base64) et le corps de la requête au format XML ou Json. Pour les deux premières fonctionnalités, il suffit d'envoyer un objet json content les champs « subscribed », « LNAME » et « FNAME ». Par la suite, on vérifie que l'utilisateur a bien été ajouté en vérifiant le statut de code 200 et on retourne un CommunicationDTO (Data Transfer Object). Ce résultat est directement transmis au frontend, qui affichera alors un message de confirmation vert ou rouge, en cas de succès ou d'échec, respectivement. Pour ce qui est de l'inscription d'un nouvel utilisateur VALERIA, un Rest endpoint en java est nécessaire et on doit passer par celui-ci au lieu d'effectuer une requête Rest directement vers Mailchimp sur le portail parce qu'on doit par la suite synchroniser les adresses courriels inscrites, s'il y a lieu, avec la base de

données de VALERIA. Ci-dessous est un diagramme d'états détaillant cette fonctionnalité implémentée.

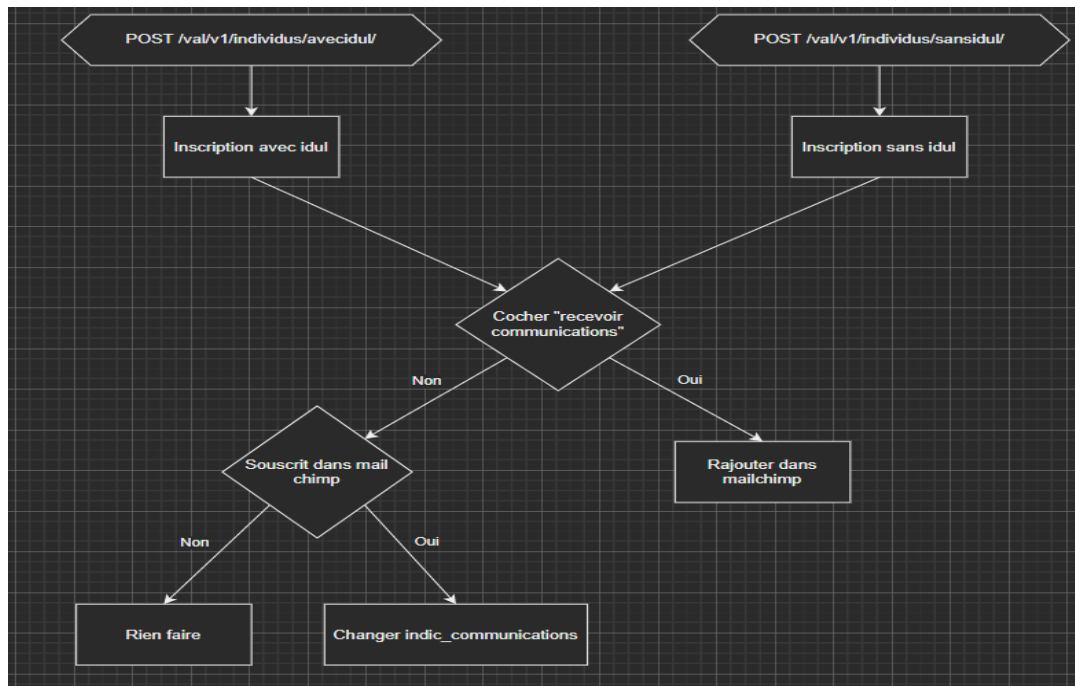


Figure 2 – diagramme d'états à l'inscription pour MailChimp

Deux classes d'opérations pour l'inscription doivent être implémentées dépendamment de si l'utilisateur s'inscrit avec son idul ulaval ou entre ses informations (contenant son adresse courriel). Il est à noter qu'en cas d'erreur à une étape dans l'inscription VALERIA il faut procéder à l'opération inverse si un membre est ajouté à Mailchimp. Le tout est géré dans une classe contenant une liste d'opérations : à l'inscription on fait appel à une factory pour placer les opérations qui pourraient être annulées (par héritage ces opérations possèdent des méthodes `apply` et `undo`). Par la suite, les tests unitaires sont réalisés avec la librairie Mockito. Toutes les méthodes et classes modifiées doivent être testées pour les exemples de comportements attendus.

3.4.4 Gestion des collaborateurs

Les utilisateurs de VALERIA ayant le statut de chercheur possèdent leur propre groupe (et peuvent faire partie de plusieurs groupes aussi) qui sont composés de collaborateurs. Ces groupes ou projets dans la base de données IDM permettent de partager les mêmes clés et accès à certains services pour un chercheur donné. Red Hat Identity Management (IDM) est un système d'identité, de politique et d'audit géré de manière centralisé dans un environnement Linux. Comme le langage python est le langage principalement supporté en tant qu'infrastructure logicielle pour les appels de fonctions, des requêtes http de style Json-Rpc sont envoyées avec comme paramètres des fonctions de mêmes noms en utilisant une librairie `JsonRpcHttpClient`. Ce dernier mandat est séparé en 3 fonctionnalités backend pour le stagiaire : afficher les collaborateurs, ajouter un collaborateur et supprimer un collaborateur. Afficher les

collaborateurs se réalise en invoquant la méthode `group_find` pour les collaborateurs internes et `group_show` pour les collaborateurs externes à l'université (sans idul). Le client `JsonRpcHttpClient` s'occupe de convertir les objets json en classes « built-in » de java. Ce travail a pu être réalisé en regardant la documentation de `Freelpa`, une version open source et gratuite d'IDM. Les fonctions retournent pour la plupart des objets `Map`, mais les valeurs de retour ne sont pas indiquées dans la documentation : il a fallu tester par essais-erreurs les fonctions voulues avec la base de données de développement. À un moment, une fonction comme exemple d'appel au service IDM était déjà présente dans le backend VALERIA, mais le code était obscur et n'était pas une bonne source d'aide dans le sens où l'objet `Map` était converti en `List` : on accédait désormais aux clés nommées avec des indices et si l'ordre alphabétique n'était plus respecté,

le code ne marchait plus.

```
try {
    Map<?, ?> query = invoke( methodName: "group_find", params);

    String gid = (String) ((ArrayList)((Map.Entry)((LinkedHashMap)((ArrayList)((Map.Entry)((LinkedHashMap)query)
        .entrySet().toArray()[2]).getValue()).get(0)).entrySet().toArray()[1]).getValue().get(0);
    return gid;
} catch (JsonRpcClientException exception) {...}
```

Figure 3 – code obscur dans implémentation idm

Une version propre et fonctionnelle a été par la suite programmée par le stagiaire.

```
4 usages Goldenweiss
@Override
public List<String> getGroupUsers(Project project) throws Throwable {
    Goldenweiss
    IpaClientParamsDTO params = new IpaClientParamsDTO() {
        public final String criteria = project.getName().toLowerCase();
        public final String cn = project.getName().toLowerCase();
        public final Boolean all = true;
    };

    try {
        Set<String> groupUsers = new HashSet<>();
        Map<?, ?> query = invoke( methodName: "group_find", params);
        List<Map<String, List<String>>> result = (List)query.get("result");
        List<String> users = result.get(0).get("member_user");
        if (result != null && users != null)
            groupUsers.addAll(users);
        return new ArrayList<>(groupUsers);
    } catch (JsonRpcClientException exception) {
```

Figure 4 – code clean dans l'implémentation idm

Les adresses courriels des collaborateurs ou iduls sont récupérés dans IDM puis on retourne les individus VALERIA qui y sont associés. Les individus sont cherchés dans la base de données PostgreSQL avec le patron DAO (Data Transfer Object) si présents. Une couche supplémentaire d'abstraction est rajoutée en utilisant le repository pattern pour encapsuler cette logique DAO. Cette méthodologie sera vu dans le cours GLO-4002 (qualité et métriques du logiciel) dont le stagiaire n'a pas encore fait dans son programme d'études. Une fois ce Rest endpoint en java fait et testé, un script utilitaire en python a été fait pour générer un fichier csv de tous les chercheurs et leur liste de collaborateurs sur une même ligne qui utilise

l’affichage des collaborateurs. Avec les autres membres de l’équipe, il a été discuté de l’ajout d’une page d’affichage des collaborateurs au frontend dans un future proche puisque le backend était terminé. Pour l’ajout ou la suppression d’un membre dans IDM, le stagiaire invoque les fonctions `group_add_member` et `group_remove_member`. En plus des tests unitaires, ces Rests endpoints doivent permettre de voir, ajouter ou supprimer des collaborateurs uniquement aux utilisateurs ayant les autorisations nécessaires. En effet, les requêtes http Rest peuvent être simulées avec le logiciel Postman par n’importe qui. Pour sécuriser les méthodes Rest, ont utilisera des annotations Spring Boot dans java qui appelleront certaines politiques et règles de sécurité, comme `ChercheurPolicy`, une classe vérifiant si l’appelant est un chercheur. On doit aussi vérifier si l’appelant est l’utilisateur présentement connecté au portail : cela se fait en regardant le token keycloak. Il a été très important d’utiliser les bonnes annotations. À un moment, les tests passaient le cas où les utilisateurs étaient présents dans IDM et présents dans la base de données VALERIA, mais que le service Spring Boot n’autorisait pas la communication du DAO parce que le service utilisant le DAO n’était pas transactionnel. Il fallait donc ajouter l’annotation `@Transactional`.

3.5 Comparaison avec les attentes du stagiaire avant le début du stage

Tout d’abord, le stagiaire ne s’attendait pas à obtenir un stage. Comme il s’agissait d’une première expérience de stage, les membres de l’équipe de Recherche et gestion de la recherche se sont montrés très accueillant avec le nouveau stagiaire et il s’est senti à son aise dès les premiers jours. Les tâches n’étaient pas simples, mais le stagiaire n’a pas éprouvé trop de difficultés, comme il s’agissait de savoir-pratique en programmation. En effet, il a pu faire ses preuves en présentant les métriques d’utilisations de VALERIA en fin d’itération vers la 6^{ème} semaine du stage. La charge de travail était élevée, mais le stagiaire s’attendait à avoir beaucoup de difficultés vues comment les cours de son programme sont difficiles pour lui. Même si l’étudiant ne comprends pas toute la théorie durant les réunions, ça ne l’empêche pas de faire sa part de travail et de poser des questions à sa personne-ressource, Kevin, un développeur analyste senior. L’étudiant s’attendait à ne recevoir aucune aide et de ne pas pouvoir jamais se vérifier, mais il a eu amplement de support avec les revues de code, les conseils de Kevin, les rencontres d’équipes et entre développeurs et même parfois des appels vidéo à des programmeurs plus expérimentés dans certains domaines, etc.

4 Développement et renforcement des compétences

4.1 Technique et Ingénierie

Le stagiaire a gagné de l’expérience pratique en programmation et a dû apprendre sur le tas. Du côté frontend, l’architecture VALERIA est codée en Javascript avec des composantes React et des feuilles de style CSS. Pour le backend, le langage de programmation utilisé est le Java. Les métriques d’utilisation de VALERIA ainsi que le script utilitaire pour générer un fichier des collaborateurs sont fait en Python. Les fonctionnalités ont été développées sur différents projets, sur d’uniques branches, en utilisant GitLab. Les tests unitaires backend sont réalisés avec la librairie Mockito en java. Parmi les logiciels utilisés qui étaient nouveaux pour l’étudiant, on compte :

- JupyterLab, un environnement web pour l’exécution de tâches dans plus de 40 langages de programmation, il sépare les scripts en répertoires ou cellules sur un même « Notebook ». Comme

service, VALERIA offre les langages Python et R, par exemple. Les statistiques d'utilisation ont été programmées dans un Jupyter Notebook.

- IntelliJ IDE, un environnement de développement intégré axé sur la programmation java.
- Visual Studio Code, un éditeur de code extensible qu'on utilise pour le frontend. Très convivial avec des projets GitLab.
- Docker, une plateforme permettant de lancer certaines applications dans des conteneurs logiciels. Démarre les bases de données en production ou en développement.
- PgAdmin 4, en complément avec Docker, permet d'éditer les bases de données PostgreSQL.
- Postman, une plateforme permettant de construire des API Rest en simulant des requêtes http.
- Jenkins, un outils open source de serveur d'automatisation.

Des diagrammes de conceptions sont partagés par Kevin, développeur senior, pour développer chaque fonctionnalité et le stagiaire s'assure de les suivre à la lettre où d'apporter des simplifications si nécessaire, comme ne pas vérifier pour la requête d'un chercheur qui veut voir ses collaborateurs dans un projet s'il s'agit de la même personne connectée, mais bien du même projet connecté dans Keycloak.

4.2 Communications

Les fils de discussions et les rencontres journalières (lundi, mardi et mercredi) sont effectués sur Microsoft Teams. Le stagiaire est un travailleur de peu de mots, mais cela ne l'empêche pas d'effectuer un travail de qualité. Durant les rencontres aux deux semaines entre développeurs, l'étudiant n'avait pas de nouvelles idées à apporter, mais il est très attentif. Pour la sixième semaine de stage, le stagiaire a eu l'occasion de faire ses preuves en présentant les métriques d'utilisations de VALERIA au département de la recherche et gestion de la recherche.

4.3 Réflexion sur la formation pratique

Dès la première semaine, l'étudiant a été jumelé avec un développeur d'expertise pour mettre en place son poste de travail et ses outils de développement. Par la suite, à la façon d'un cours en différé, l'étudiant a remis les travaux demandés dans les temps impartis comme un autodidacte. Certains cours du programme de génie logiciel se sont avérés être très utiles, comme le cours de génie logiciel orienté objet avec ses patrons de conceptions, ses diagrammes de classe et la construction d'une application en équipe. Cependant, les cours ne sont pas aussi intéressants qu'un stage, l'étudiant a eu une agréable expérience à sa grande surprise. Dans les premières semaines, l'étudiant demandait beaucoup d'aide à sa personne-ressource, puis il s'est habitué aux infrastructures complexes, un de ses objectifs. Il contacte maintenant Kevin, le développeur d'expertise, pour lui faire part des ses avancés ou s'il éprouve un blocage grave. Le stagiaire a aussi vu des bonnes pratiques de programmation, comme les repository pattern, une couche supplémentaire d'abstraction pour ne pas à avoir à toucher à la logique des data transfer objects (DAO).

4.4 Réflexion sur la formation théorique

Kevin, développeur analyste senior de l'infrastructure VALERIA, conseille fortement au stagiaire de suivre le cours Qualité et métriques du logiciel (GLO-4002) pour améliorer sa formation sur les bonnes pratiques de programmation et les tests unitaires parce que le stagiaire se retrouvait parfois avec des tests manquants. Les rencontres aux deux semaines avec les autres développeurs faisaient l'exploration d'architectures logicielles et étaient chargées en théorie, donc difficiles à suivre, le stagiaire propose donc de prendre des notes pour un éventuel prochain stage.

4.5 Bilan des acquis

Les points les plus importants sur lesquels le stagiaire a travaillé sont les tests unitaires. Ce stage a donc été une excellente pratique dans ceux-ci. Le stagiaire a pu aussi améliorer ses compétences en javascript et react pour le frontend. Les langages java et pythons lui étaient déjà familiers, mais il a pu parfaire ses connaissances avec les bibliothèques Spring Boot (Rest API) et Keycloak.

5 Conclusion

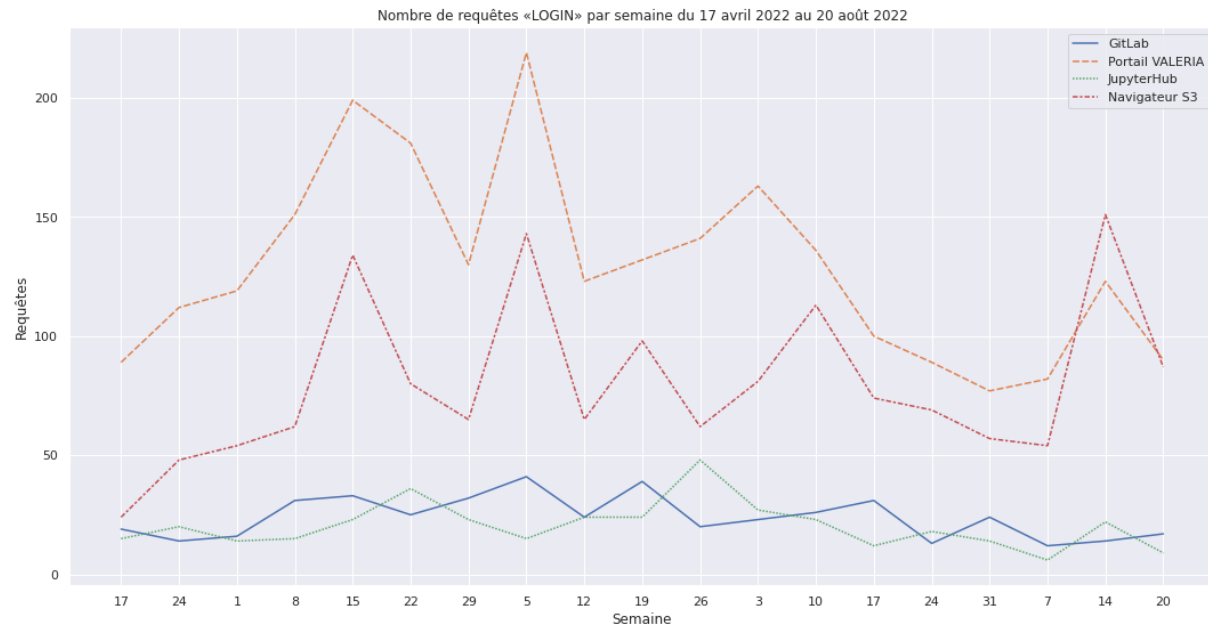
Pour conclure, l'étudiant a effectué des statistiques d'utilisations en temps réel sur certains services VALERIA, soit le GitLab, le navigateur S3, le portail principal ainsi que JupyterHub, en plus d'apporter des modifications au portail principal en implémentant une infolettre au backend et frontend sur la page d'accueil et dans la logique d'inscription. Il aussi implémenté des endpoints Rest pour la gestion des collaborateurs dans le projet d'un chercheur.

Après ce stage, il est clair que l'étudiant veut poursuivre dans son programme d'études, soit le programme de génie logiciel. Sous l'approbation de son superviseur, les autres membres de l'équipe se sont montrés très satisfaits du travail du stagiaire. C'est pourquoi le stagiaire poursuivra un emploi étudiant du même poste à la DTI en raison d'un jour par semaine. Il travaillera sur une interface pour la gestion des collaborateurs comme uniquement le backend avait été fait.

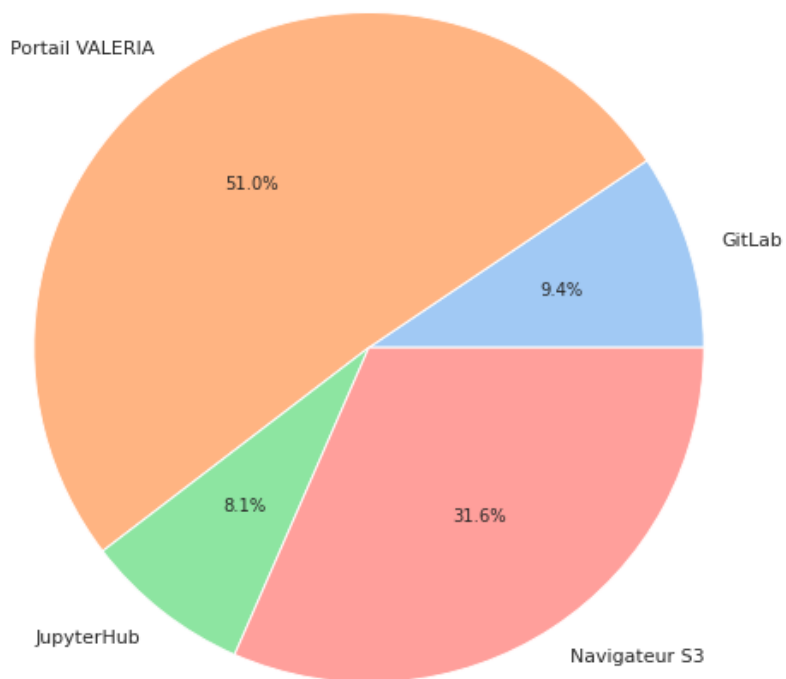
Ce stage se veut une première expérience de travail. Le stagiaire aimerait remercier la personne ressource avec laquelle il a été jumelé, soit Kevin, développeur analyste de VALERIA. Selon les principes de développement durable (code de déontologie des ingénieurs), le principe de la subsidiarité veut que tout échelon supérieur s'interdise de faire ce qu'un échelon inférieur pourrait faire. En ce sens, il s'est efforcé de ne pas lui donner les réponses pour des blocages mais de lui prodiguer des conseils directionnellement corrects.

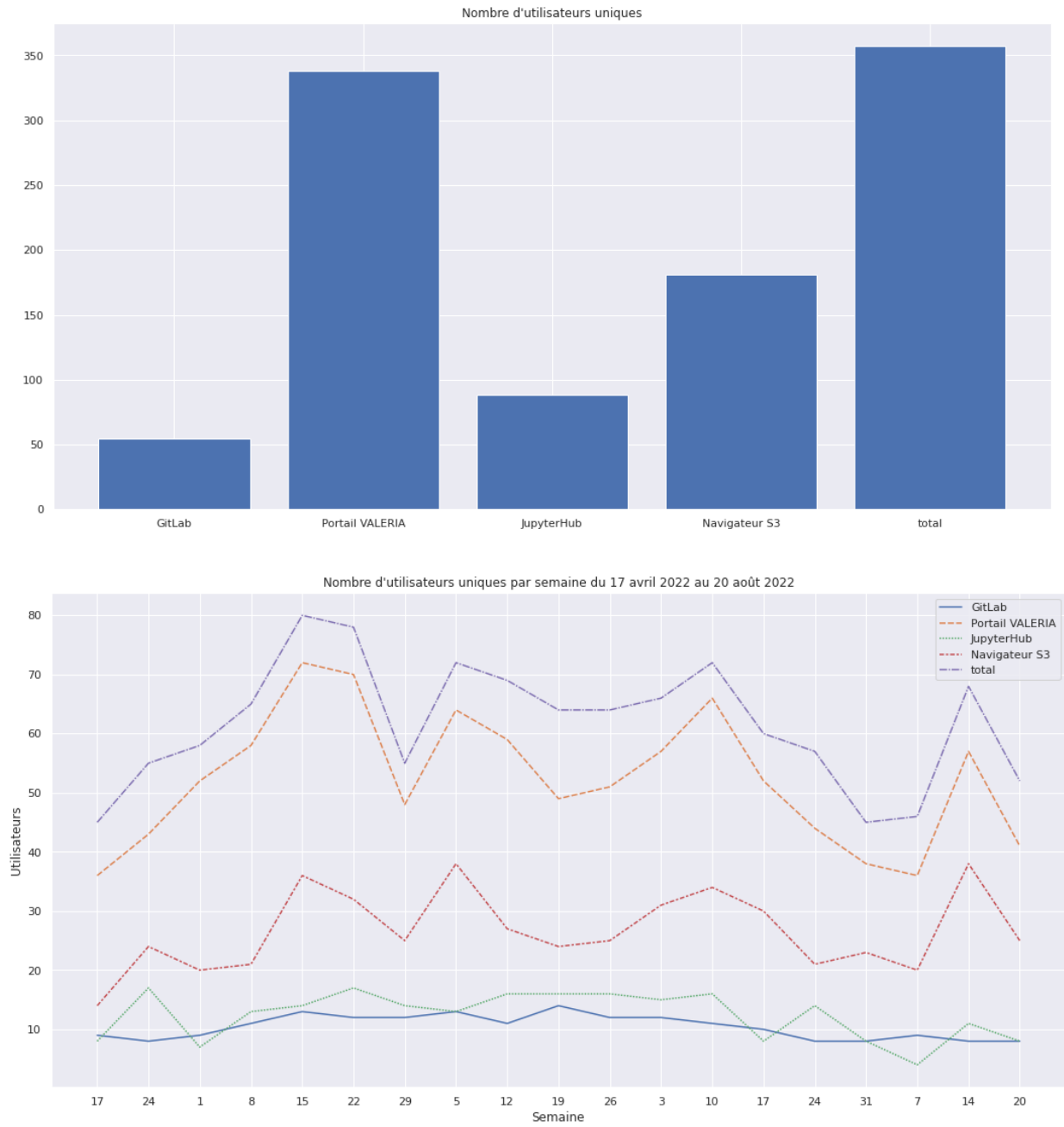
Annexe A - Métriques d'utilisation VALERIA

`show_results('2022-04-10 00:00:00', '2022-08-20', 'Weeks'):` 100% ██████████ 532/532 [00:08<00:00, 65.57it/s]
4821 requêtes LOGIN on été détectées après filtrage.



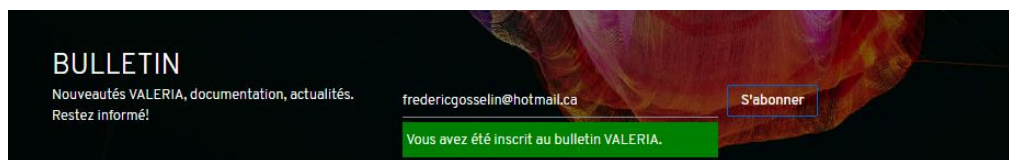
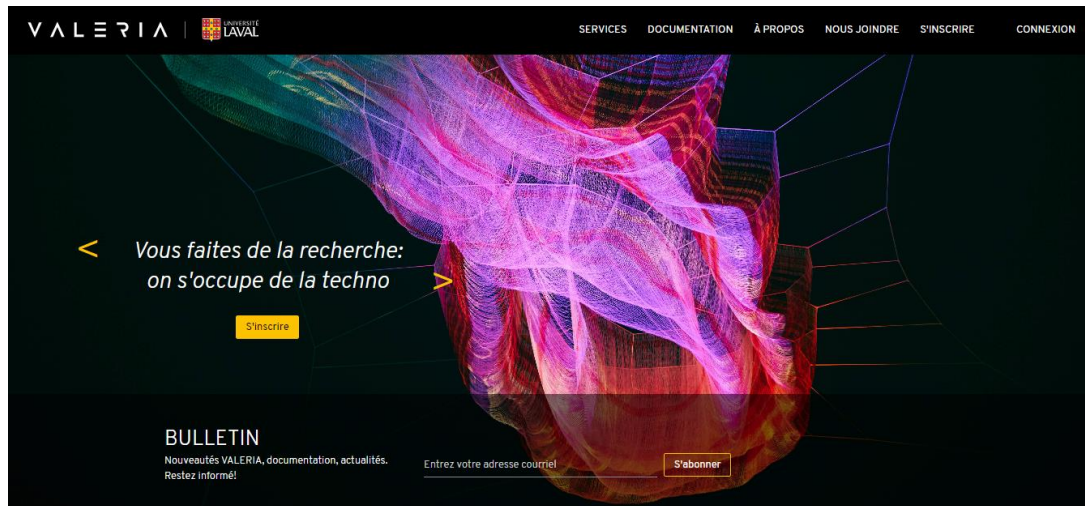
Pourcentages de requêtes «LOGIN» par semaine du 17 avril 2022 au 20 août 2022





Annexe B - Ajouts au portail principal et infolettre VALERIA

Branches git feature/mailchimp et feature/logo-ulaval



Branche git feature/new-user-mailchimp

Créer un nouveau compte avec votre adresse courriel

Prénom *	Nom *
<input type="text"/>	<input type="text"/>
Courriel *	Confirmation du courriel *
<input type="text"/>	<input type="text"/>
Téléphone	Poste téléphonique
<input type="text"/>	<input type="text"/>
10 chiffres sans espace ni trait d'union	
Mot de passe *	Confirmation du mot de passe *
<input type="password"/>	<input type="password"/>

☐ J'ai lu et j'accepte les conditions d'utilisation de VALERIA.

☒ Je souhaite recevoir des communications de VALERIA

☐ Je ne suis pas un robot

reCAPTCHA

Confidentialité - Conditions

Créer mon compte Annuler

feature/new-user-mailchimp

Exemple de test unitaire Mailchimp

```
@Test
public void subscribe_withAdresseCourrielInvalide_thenServiceExceptionThrown() throws IOException {
    // ARRANGE
    String email = "email@yopmail.com";
    String json = "{\"email_address\":\"email@yopmail.com\",\"status_if_new\":\"subscribed\",\"merge_fields\":{\"LNAME\":\"\",\"FNAME\":\"\"},\"status\":\"subscribed\"}";
    Mockito.when(httpClientFactory.get(email)).thenReturn(httpClientAdapter);
    HttpResponse httpResponse = HttpResponseTestFactory.get(400);
    Mockito.when(httpClientAdapter.put(json)).thenReturn(httpResponse);

    // ACT+ASSERT
    ServiceException exception = Assertions.assertThrows(ServiceException.class, () -> {
        mailChimpApi.subscribe(email, name: null, lastName: null, subscribed: true);
    });
    assertTrue(exception.getErreursAttributs().get(0).getCodesErreurs().containsKey(CodeMessageValidation.ERR_VAL0001));
}
```

Liste d'abonnements avec la compagnie américaine MailChimp

Audience
VALERIA

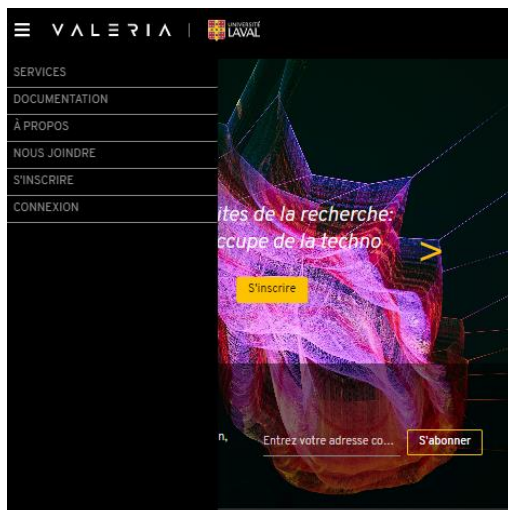
Your audience has 6 contacts. 2 of these are subscribers.

Overview **Manage contacts** Add contacts Signup forms Preferences center Settings Inbox Surveys

Toggle Columns Export Audience

	Email Address	First Name	Last Name	Address	Phone Number	Birthday	Tags	Email Marketing	Source	Contact R
<input type="checkbox"/>	frgos831@ulaval.ca	Eaewyn	Euubaer					Unsubscribed	API	★★★
<input type="checkbox"/>	frgos83@ulaval.ca	Maewyn	Neubaer					Unsubscribed	Admin Add	★★★
<input type="checkbox"/>	frederic.gosselin@naver3.com	Frédéric	Gosselin					Unsubscribed	API	★★★

Interface flexible « responsive » sur tablette



Annexe C - Gestion des collaborateurs backend

3 rests endpoints pour les branches git feature/project-get-members, feature/project-add-member, feature/project-remove-member

```

  Goldenweiss
  @Override
  @MethodeSecuriseePar(appliquerPar = ProjectUserDefaultRegleSecurite.class)
  @MethodeSecurisee public ProjectDTO getUserDefaultProject(@ProjectUserIsChercheurRegleSecurite ParamsGetProject params) {
      return serviceProject.getUserDefaultProject(params.getEmail());
  }

  Goldenweiss
  @Override
  @MethodeSecuriseePar(appliquerPar = ProjectAddMemberRegleSecurite.class)
  @MethodeSecurisee public void addMemberProject(@ProjectAddMemberIsProjectOwnerRegleSecurite ParamsAddMemberProject params) {
      serviceProject.addMemberProject(params);
  }

  Goldenweiss
  @Override
  @MethodeSecuriseePar(appliquerPar = ProjectRemoveMemberRegleSecurite.class)
  @MethodeSecurisee public void removeMemberProject(@ProjectRemoveMemberIsProjectOwnerRegleSecurite ParamsRemoveMemberProject params) {
      serviceProject.removeMemberProject(params);
  }

```

Utilisation d'un repository pattern

```

  3 usages  Goldenweiss
  @Override
  public void removeMemberProject(ParamsRemoveMemberProject params) {
      List<ReferencesAjouteesIndividuValeria> referencesAjouteesList = Arrays.asList(ReferencesAjouteesIndividuValeria.values());
      Set<String> referencesAjoutees = ReferencesAjoutees.getReferencesAjouteesString(referencesAjouteesList);

      Optional<IndividuValeriaDTO> individuValeriaDTO = individusRepository.getUserByEmail(params.getIndividuValeria().getAdresseCourriel(), referencesAjoutees);
      if (!individuValeriaDTO.isPresent()) {
          individuValeriaDTO = individusRepository.getUserByIdValeria(params.getIndividuValeria().getIdValeria(), referencesAjoutees);
          if (!individuValeriaDTO.isPresent())
              throw new ServiceException("Cannot find user for email", CodeMessageValidation.ERR_VAL0023);
      }

      IndividuValeriaDTO individuValeriaDTOGet = individuValeriaDTO.get();
      projectRepository.removeMemberFromProject(params.getProject(), individuValeriaDTOGet);
  }

```

Exemple de test unitaire

```

  Goldenweiss
  @Test
  public void estValide_estChercheurAUUnProjetNestPasConnecte_nEstPasValide() {
      // Étant donné
      String nameConnected = "ul-val-prj-def-connected";
      String nameOther = "ul-val-prj-def-notyourproject";
      ParamsAddMemberProject paramsAddMemberProject = ParamsAddMemberProjectObjectMother.get();
      paramsAddMemberProject.setProject(ProjectDTOObjectMother.get(nameOther));
      List<Project> projects = Arrays.asList(ProjectObjectMother.get(nameConnected));
      IndividuValeriaDTO individuValeriaDTOAuthentifier = paramsAddMemberProject.getIndividuValeria();
      Mockito.when(serviceIndividuAuthentifier.obtenirIndividuAuthentifier()).thenReturn(Optional.of(individuValeriaDTOAuthentifier));
      Mockito.when(chercheurPolicy.passesPolicy(individuValeriaDTOAuthentifier)).thenReturn(true);
      Mockito.when(projectRepository.getProjectForIdChercheur(individuValeriaDTOAuthentifier.getIdIndividuValeria())).thenReturn(projects);

      // Lorsque
      Boolean resultatEstValide = projectUserIsChercheurRegleSecurite.estValide(paramsAddMemberProject);

      // Alors
      assertFalse(resultatEstValide);
  }

```

Annexe D - Logiciels

Construction d'API avec Postman - enlever un collaborateur

The screenshot shows the Postman application interface. On the left, the 'My Workspace' sidebar lists various API collections under the 'valeria' workspace. The main panel displays a PUT request to the endpoint `[[val-services-url]]/v1/project/removemember`. The request body is a JSON object:

```
1 {
2   "project": {
3     "posixGroupName": "ul-val-prj-def-depic"
4   },
5   "individuValeria": {
6     "idValeria": "vdb5"
7   }
8 }
```

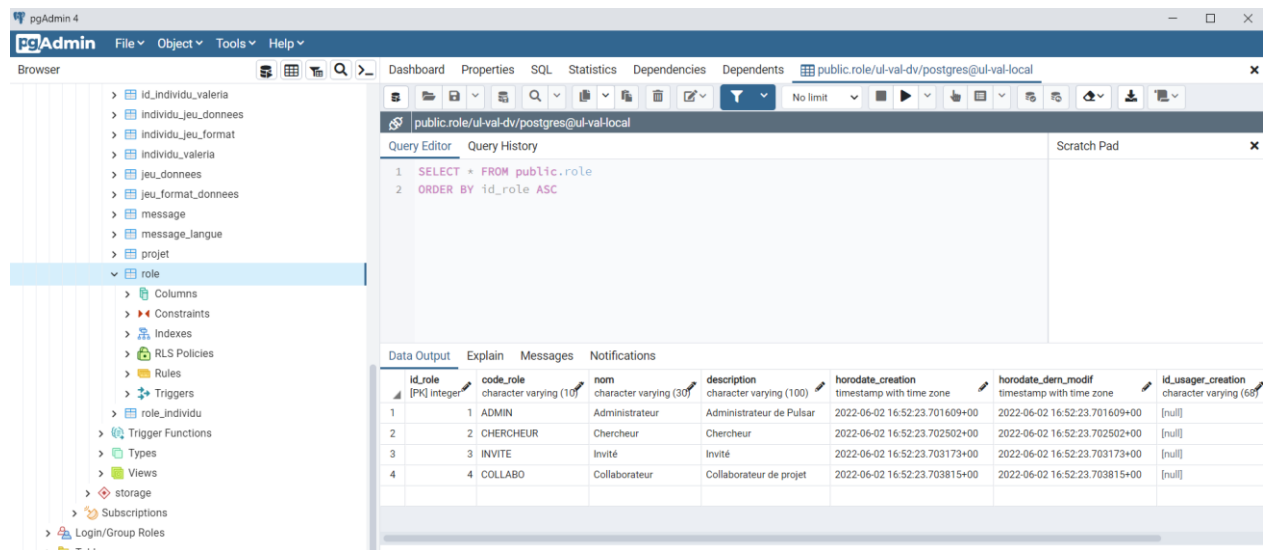
The 'Response' section is currently empty, showing a placeholder illustration of a person with a rocket. The bottom status bar indicates the request is online and provides options like 'Find and Replace' and 'Console'.

Déploiement en développement du frontend ou backend avec Jenkins - backend

The screenshot shows the Jenkins web interface for the pipeline 'VAL-S3-SERVICES-DEV'. The 'Stage View' displays the execution progress of the pipeline, which consists of six stages: Declarative: Checkout SCM, Préparation de l'environnement, Construction de l'image, Dépôt sur Artifactory, Dépôt sur Openshift, and Declarative: Post Actions. The pipeline is currently in a 'Completed' state, with the last build (#447) showing a successful status.

Stage	Declarative: Checkout SCM	Préparation de l'environnement	Construction de l'image	Dépôt sur Artifactory	Dépôt sur Openshift	Declarative: Post Actions
Average stage times:	2s	320ms	1min 16s	15s	31s	1s
#447 Sep 06 01:49	843ms	307ms	1min 12s	14s	34s	1s
#446 Sep 05 01:49	722ms	310ms	1min 8s	12s	31s	1s
#445 Sep 04 01:49	1s	315ms	1min 20s	15s	29s	1s

Accès aux bases de données (locales ou en production) avec PgAdmin4 – rôles VALERIA



The screenshot displays the PgAdmin 4 web interface. On the left, the 'role' table is selected in the browser pane. The central pane shows a SQL query in the Query Editor:

```
1 SELECT * FROM public.role
2 ORDER BY id_role ASC
```

Below the query editor, the 'Data Output' tab is active, showing the results of the query in a table format:

	id_role	code_role	nom	description	horodate_creation	horodate_dern_modif	id_usager_creation
1	1	ADMIN	Administrateur	Administrateur de Pulsar	2022-06-02 16:52:23.701609+00	2022-06-02 16:52:23.701609+00	[null]
2	2	CHERCHEUR	Chercheur	Chercheur	2022-06-02 16:52:23.702502+00	2022-06-02 16:52:23.702502+00	[null]
3	3	INVITE	Invité	Invité	2022-06-02 16:52:23.703173+00	2022-06-02 16:52:23.703173+00	[null]
4	4	COLLABO	Collaborateur	Collaborateur de projet	2022-06-02 16:52:23.703815+00	2022-06-02 16:52:23.703815+00	[null]