**RAPPORT DE STAGE**

Stagiaire en génie Logiciel - SimpleSurvey/OutsideSoft

Logo, company name

Description automatically generated

**SEG3901**

Faculté de génie

Université d’Ottawa

Gilles Djawa Ndjankeu

300094497

Période de stage : 2023/01/01 - 2023/04/30

Date de la soumission du rapport : 2023/xx/xx

# **Résumé**

# **TABLE DES MATIÈRES**

# **Liste des images**

# **Glossaire**

# **Introduction**

Durant ces 4 derniers mois, j’ai pu prendre part à la deuxième moitié de ma période de stage coop au sein de **SimpleSurvey**. Cette compagnie experte en sondage et 100% canadienne fondée en 2007 par M. David Houle est une incontournable dans le domaine du sondage et du vote électronique au Canada et s’assoie à la table des plus grands de la catégorie du fait de son ancienneté et de son expérience.

L’effectif à ce jour est constitué de 9 personnes, m’incluant, réparties au sein de différentes équipes à savoir l’équipe de développement, l’équipe de support technique et l’équipe financière/administrative. Comme toujours, le carde est idéale pour un stagiaire, et l’ambiance au sein de la compagnie reste chaleureuse et conviviale avec des employés qui n’hésitent jamais à partager au tour d’eux et au sein des équipes une belle dose de joie et de bonne humeur.

Lors de ces derniers mois, j’ai été de nouveau assigné à l’équipe de développement dont les membres ont su encadrer et superviser ma progression. Cette équipe est donc composé de 4 membres incluant deux analystes programmeurs, un ingénieur statisticien et moi-même, stagiaire en génie logiciel, dont les tâches ne diffèrent pas de la première partie de stage. J’ai donc été chargé de la programmation, du maintien et de la mise à niveau du logiciel de sondage, de soutenir l’équipe de développement dans ses tâches d’analyse et de programmation; de l’élaboration des jeux d’essais unitaires et de la pratique les tests unitaires requis pour la mise au point des programmes; d’assurer le support nécessaire lors des essais fonctionnels, des essais de système et de la mise en production en apportant les modifications requises aux programmes; de tester les mises à jour du logiciel et signaler les bogues et problèmes techniques; de contribuer à maintenir le guide de l'utilisateur à jour en fournissant à l'équipe chargée du guide, des informations sur les nouvelles fonctionnalités développées; et au besoin, d’effectuer toute autre tâche connexe.

Contrairement à la première phase du stage, cette deuxième période a été marquée par une évolution notable de ma productivité et de la qualité du travail fourni, ce qui m’a permis de me pencher sur des tâches et des projets plus complexes. La suite de ce rapport consistera donc à vous faire part de deux de ces projets à savoir la migration de la fonctionnalité de traduction de sondage du site de MS web Forms vers Blazor et la migration de la page d’information de projet de vue.js vers Blazor.

Ce rapport sera donc une rétrospective de la réalisation de tâches énoncées plus haut et fera étalage de leurs différents contours à travers des analyses détaillées de chacune des étapes réalisées de la programmation à la soumission et se conclura par une réflexion portée sur la globalité de cette première expérience professionnelle.

#### **Description des tâches**

Dans le cadre de ce stage, j’ai été assigné par l’équipe de développement à une sorte de projet personnel, celui de migrer le plus de page/fonctionnalité vers le Framework Blazor. Il s’agit là d’une initiative de la compagnie visant à faire évoluer et montrer la plateforme de sondage en ligne sous un tout nouveau jour. Jusqu’à très récemment, la plateforme était principalement constituée de pages écrites en Microsoft web forms (aspx), Vue.js ou Razor (cshtml) selon ce qui devait être fait sur la page ou selon la technologie en vogue au moment de sa création mais la nécessité de réunifier toutes les fonctionnalités sous une même technologie a fini par voir le jour et donc Blazor fut celle choisie.

Les avantages de l’utilisation de Blazor au lieu d’une des trois autres technologies sont nombreux :

* Blazor est Framework SPA (single-page application), ce qui permet une navigation quasi instantanée lorsqu’il est chargé et donc une meilleur expérience client.
* Le C# est bien plus apprécié que le Javascript en interne par les développeurs, ce qui offre une meilleure qualité du code fourni.
* Le code est partagé entre le frontend et le backend ce qui réduit les risques d’incohérence entre le côté client et le côté serveur.
* Blazor s’exécute du client sur le navigateur de l’utilisateur contrairement aux autres technologies qui sont côté serveur.
* Blazor regroupe la quasi-totalité des fonctionnalités des trois autres technologies telles que la création de composants réutilisables ou encore l’utilisation de modèles de programmation contrôlés par des évènements, dans une syntaxe plus fluide et facile de compréhension.

Ainsi, dans le cadre de ce projet, j’ai été confié les tâches de migrer la fonctionnalité de traduction de sondage de la plateforme et la page résumant les informations relatives à un sondage vers Blazor.

**Migrer la fonctionnalité de traduction de sondage de MS web forms vers Blazor**

En plus d’être développé autour de .Net Core qui est une technologie plus récente que .NET Framework (utilisé par les web forms), Blazor offre également une architecture plus moderne pour le développement d'applications web, basée sur des composants réutilisables et des mises à jour dynamiques, ce qui permet aux développeurs de créer des applications web plus dynamiques et réactives, et ceci à contrario des web forms qui quant à eux, sont basés sur un modèle d'événements de serveur pouvant entraîner des temps de réponse plus lents et une expérience utilisateur moins fluide.

La migration de cette fonctionnalité de traduction fût marquée par deux principales étapes à savoir : la migration de la page listant la liste des traductions disponibles pour un sondage puis, la migration de la page permettant la modification d’une traduction. Ces pages étant toutes deux des webs forms, ont une architecture assez simple : A picture containing text, diagram, line, plan

Description automatically generated

Côté serveur cette architecture comprend :

* Le code en arrière (code behind) : il s’agit de l’ensemble des fonctions, variables et attributs (en C#) assurant le fonctionnement de la vue côté client ainsi que la circulation des données depuis la base de données. Ce code va de pair avec la page aspx et porte généralement le même nom qu’elle, à l’exception des extensions qui diffèrent.
* La base de données : Elle contient les données affichées à l’utilisateur et enregistre ses modifications. Dans le cadre de notre application, la communication à la base de données se fait grâce à des procédures SQL préenregistrées (Stored procedure) qui réalisent une ou plusieurs tâches bien précises. Ainsi chaque table de la base données possède des procédures pour récupérer (Get) un certain type ou toutes les données qu’elle contient, mettre à jour des données, en sauvegarder ou même en supprimer. Pour que ces procédures SQL puissent être interprétées par le code C#, la bibliothèque Dapper est utilisée. Dapper est une bibliothèque open source pour .NET qui facilite l'accès à la base de données en permettant l'exécution de requêtes SQL et la cartographie des résultats dans des objets .NET. L’utilisation de cette bibliothèque est assez simple : supposons que nous avons une procédure stockée nommée "GetSurveysByUser" qui prend en entrée un paramètre @IdSurvey de type Guid (UniqueIdentifier) et qui renvoie une liste de sondages associés à l’utilisateur correspondant, le code permettant l’exécution de la procédure est le suivant : A screen shot of a computer

  Description automatically generated with low confidence

Dans cet exemple, une connexion à la base de données est créée à l'aide d'une chaîne de connexion stockée dans la variable "connectionString". Nous appelons ensuite la méthode "Query" de l'objet de connexion en spécifiant le nom de la procédure stockée ("GetSurveysByUser"), un objet anonyme contenant le paramètre "idSurvey" (dans cet exemple, un identifiant unique est généré grâce à la fonction NewGuid()), et le type de commande ("CommandType.StoredProcedure").La méthode "Query" renvoie une liste d'objets Survey, qui est stockée dans la variable "surveys". Nous pouvons ensuite utiliser cette liste comme bon nous semble.

Côté client, l’architecture est composée d’une page aspx servant de vue à l’utilisateur.

A picture containing text, screenshot, font

Description automatically generated

A screenshot of a computer program

Description automatically generated with medium confidence

Dans cette image on peut voir un exemple d’en-tête de page aspx. Il est possible de remarquer la directive Codebehind=”Translate.aspx.cs”. Cette directive référence la page contenant le code en arrière qui porte le même non que la page aspx à laquelle on rajoute l’extension « .cs ».

Cette migration consiste essentiellement à passer de cette architecture à une architecture Model-view-Controller (MVC). Dans cette nouvelle architecture, le modèle constitue les champs de données nécessaires au fonctionnement de la vue. Ces champs sont alimentés par la base de données et transmis à la vue par l’intermédiaire du contrôleur.

Diagram

Description automatically generated

La fonctionnalité de traduction de sondage de la plateforme permet à un utilisateur de traduire toutes les composantes (questions, titres, images, vidéos, etc…) de son sondage en plus de 30 langues données. Elle lui permet de faire les la traduction automatiquement ou manuellement. Ainsi, dans la nouvelle version de la fonctionnalité de traduction, la vue est une page Blazor.

Pour réaliser cette tâche, j’ai suivi des étapes bien déterminées :

* Il a tout d’abord fallu construire le « Model » : le Model contient les champs de données qui seront utilisés dans le frontend pour assurer le fonctionnement de la vue. Ces champs se font hydrater grâce à la base de données par l’intermédiaire du Service. Tout ceci signifie que dans un modèle de données, les champs sont juste créés mais pas instanciés. Ces champs seront accessibles à la vue mais y seront à la base vide puis hydraté grâce au service et au contrôleur. Il a fallu ici analyser la page aspx et les différentes bases de données liées à la traduction pour savoir quels modèles et champs créer. Pour mener à bien cette migration, 10 modèles ont été créés.

A screen shot of a computer program

Description automatically generated with low confidenceA screen shot of a computer program

Description automatically generated with low confidence

* L’étape suivante est la construction du service : Dans une architecture MVC (Modèle-Vue-Contrôleur), le service est une couche intermédiaire qui contient la logique d’affaire de l'application (Business logic). Cette couche est responsable de la coordination et de l'exécution des opérations. Elle utilise les modèles (modèles de données) pour accéder aux données et retourner les résultats au contrôleur, qui à son tour les affiche dans la vue (Blazor). Le service permet de séparer la logique d’affaire de la couche de présentation (vue) et de la couche d'accès aux données (modèles), rendant ainsi l'application plus modulaire, plus facile à maintenir et à tester. La couche de service peut également être utilisée pour encapsuler des opérations complexes impliquant plusieurs transactions ou plusieurs sources de données. À contrario dans aspx, la logique d’affaire est contenue directement dans le « Codebehind ». Cette étape n’est pas la plus compliquée car la plupart des fonctions à implémenter dans le service existent déjà dans le « codebehind » de la page aspx, il ne reste donc plus qu’à les adapter au nouvel environnement. Ci-dessous la fonction permettant de charger la liste de traduction existante et son équivalent :

A picture containing text, screenshot, software, multimedia software

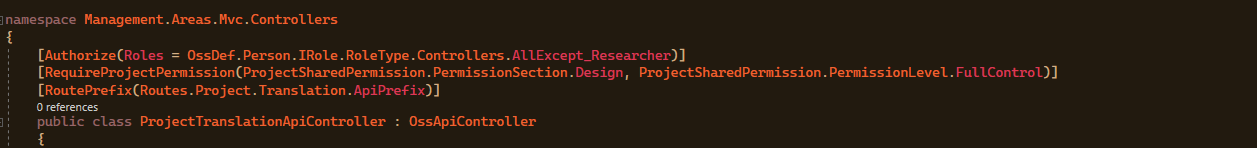
Description automatically generated

A screen shot of a computer code

Description automatically generated with low confidence

Il est possible de remarquer que l’appel à la stored procedure se fait comme dans l’example donné plus haut . Cette stored procedure nous retourne un objet de type DataTable qui ensuite peut être « casté » en IEnumerable grâce à l’extension AsEnumerable() de DataTable. Ce cast est nécessaire car la structure de données DataTable en elle-même est assez complexe à manipuler. Une fois la table casté en une collection de DataRow (ligne de la table) contenant les données (de type object), l’étape suivante est de « maper » ces données aux champs du modèle en les castant au type approprié grâce à la bibliothèque LINQ de C# comme suit A screen shot of a computer program

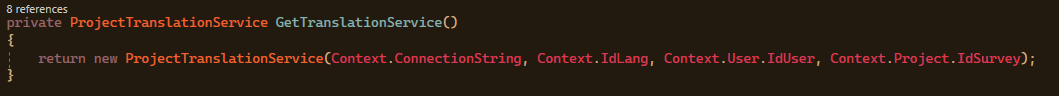
Description automatically generated with low confidence. Cette syntaxe peut être lue comme « Sélectionnes dans chaque ligne de ma table de données les champs IdTranslate, LangTo, DisplayName, DisplaySTatus, Status et passes leurs données aux champs correspondant de mon objet de type ProjectTranslationElement et convertie le tout en une liste ». Une fois les différentes fonctions nécessaires construites dans le service l’étape suivante est la mise en place du contrôleur.

* La mise en place du contrôleur elle aussi est assez simple une fois que le service est bien construit. Un contrôleur (controller) est une classe qui gère les interactions entre les vues (pages) et les modèles de données. Le contrôleur traite les demandes HTTP (comme les clics sur des boutons ou la soumission de formulaires) et renvoie des réponses appropriées (comme des pages HTML ou des données JSON). En somme, le contrôleur est une partie essentielle de l'architecture MVC (Modèle-Vue-Contrôleur) utilisée dans les applications web .NET. Dans l’application, le contrôleur est ce qui gère les permissions d’accès et dirige les requêtes http vers les bonnes ressources ; il est donc possible dans un contrôleur de de refuser l’accès à certaines données à un certain type d’utilisateur, ou si certaines conditions spécifiques ne sont pas remplies. 

Ces différentes autorisations peuvent être placées en en-tête du contrôleur ou en en-tête de chaque fonction. Dans notre cas c’est en en-tête du contrôleur. Seuls les utilisateurs ayant un rôle autre « chercheur » et les permissions de designer et de contrôle total peuvent avoir accès à la fonctionnalité de traduction.

A picture containing text, screenshot, software

Description automatically generated

L’image ci-dessus est un exemple de fonction dans le contrôleur faisant appel au service pour récupérer des données. L’appel au service se fait grâce à la méthode  qui retourne un objet ayant accès à toutes les fonctions du service. Une utilisation de cet objet se fait pour récupérer les données nécessaires au fonctionnement de la page listant la liste des traductions . Cette méthode retourne toutes les données nécessaires à la pageA screen shot of a computer program

Description automatically generated with low confidence. Cet objet est ensuite retourné sous forme de JSON grâce à un appel API, et sera interprété par la page Blazor0. À l’en tête de la fonction il est possible de noter les paramètres [HttpGet] et [Route(…)] : le premier sert à indiquer à l’API l’action a exécuter c’est-à-dire dans ce cas-ci faire un « Get » récupérer des données et le second permet au contrôleur de diriger l’appel API vers la bonne fonction.

Pour mener à bien cette migration 13 fonctions () accessible par API ont été créés.

* Une fois toutes les tâches liées au backend terminées, il ne reste plus qu’à construire la vue, la page Blazor. Blazor permet aux développeurs de créer des applications web à pages simples ou à pages multiples (pages imbriquées dans d’autres pages) en utilisant le langage C# et Razor syntax. Les applications Blazor peuvent être exécutées côté client dans le navigateur en utilisant WebAssembly ou côté serveur en utilisant SignalR. Blazor offre une expérience de développement moderne et productive avec une prise en charge de la syntaxe Razor, des liaisons de données, des événements, des composants réutilisables, ainsi que des outils de débogage et de test complets. Dans notre cas l’application est exécutée coté client car les interactions de l'utilisateur sont traitées localement sur le navigateur, ce qui permet une expérience utilisateur plus rapide et plus fluide; les mises à jour de l'interface utilisateur peuvent être effectuées sans avoir besoin de faire appel au serveur, ce qui réduit la latence et améliore les performances. En outre, Blazor côté client permet de réduire la charge sur le serveur, ce qui peut être particulièrement utile dans les environnements à forte charge tel que le nôtre. Cependant, il est important de noter que l'exécution côté client avec WebAssembly peut nécessiter une taille de téléchargement plus importante pour l'application et peut ne pas être compatible avec tous les navigateurs.

Pour ce qui est du code, les pages Blazor ont une structure qui se divise en deux parties : la partie code et la partie Razor. La partie code est là où est implémenté toute la logique nécessaire au fonctionnement de la page telle que les appels API, les fonctions permettant d’ordonner les données, ou encore des variables. La partie razor quant à elle est assez similaire à du HTML (qui est sa base) et se compose de « component » qui sont les constituant de la page. Ces components désignent des éléments de pages construits sur la base du HTML, faciles à utiliser et destinés à être réutilisés.

A picture containing text, font, screenshot, graphics

Description automatically generated

Le component ci-dessus permet la création d’un champ de texte et est l’équivalent de «  <label for="myInput">Mot de passe</label><input type="text" id="myInput" value="@report.PublickLinkPassword" style="display: block;" maxlenght="50" /> ». L’on constate que le code du component est plus cours, plus lisible et prend moins de temps et est intuitivement plus facile à comprendre en plus d’être réutilisable.

En ce qui concerne la tâche de migration de la page s’occupant de la fonctionnalité de traduction, j’ai commencé par coder la structure de la page avant de l’alimenter en données. Ce qui n’est pas réellement compliqué vu que j’ai pu m’inspirer du code aspx de la version web form de la page qui lui aussi est très similaire à du HTML. Une fois cela terminé, il ne me restait plus qu’à coder l’accès aux données c’est-à-dire les appels API. Dans notre application, un appel API ressemble à ceci : A screen shot of a computer

Description automatically generated with medium confidence

Le code ci-dessus code peut être disséqué de cette façon :

* `private async Task GetTranslationContextAsync()`: Il s'agit d'une méthode asynchrone qui renvoie une `Task`. Le mot-clé `async` indique que cette méthode utilise une programmation asynchrone.
* `await Api.Fluid`: `Api.Fluid` est l’appel au client Fluid API qui fournit des méthodes pour récupérer les données depuis le service.
* `.Get<ProjectTranslationContext>(url => url.ProjectTranslation.ApiGetProjectTranslation(IdSurvey))`: Cette ligne invoque la méthode `Get` sur `Api.Fluid` avec un type générique `ProjectTranslationContext`, indiquant le type attendu de la réponse. Le paramètre `url` est une fonction lambda qui construit une URL pour obtenir le contexte de traduction (l’ensemble des données nécessaires au fonctionnement de la page) du projet en fonction de la variable `IdSurvey` c’est-à-dire en fonction du sondage dont il est question.
* `.Data(SearchParams)`: Cette ligne invoque la méthode `Data`, qui est utilisée pour transmettre des données ou des paramètres supplémentaires à l'API. `SearchParams` est un objet qui contient les critères de recherche selon lesquels les traductions devraient être sélectionnées dans la base de données.
* `.LoadChanged(x => GettingData = x)`: Cette ligne invoque la méthode `LoadChanged`, qui définit la valeur de `GettingData` sur la valeur de `x`. `LoadChanged` est une méthode qui surveille les changements dans les données en cours de chargement depuis l'API.
* `.Success(data => Context = data)`: Cette ligne invoque la méthode `Success`, qui spécifie une fonction de rappel de succès qui sera invoquée lorsque l'appel API sera réussi. Le paramètre `data` dans la fonction de rappel représente les données de réponse de l'API, et elles sont assignées à la variable `Context`.
* `.SendAsync()`: Cette ligne invoque la méthode `SendAsync`, qui est responsable d'envoyer la requête API de manière asynchrone.

En résumé, ce code effectue un appel API asynchrone pour obtenir un contexte de traduction pour un projet, gérer les données de réponse et les affecter à une variable `Context` (contenant tous les champs de données nécessaires à la page) une fois que l'appel API est réussi. Bien évidemment cet appel API est beaucoup plus complexe mais il serait extrêmement long de le détailler et aussi cela toucherait des aspects sensibles de l’application.

Une fois les appels API construits et les données bien acheminées à la page il ne reste plus qu’à standardiser le code selon les pratiques de **SimpleSurvey** et à créer une soumission (Pull request) sur GitHub et attendre une évaluation du travail proposé par mon superviseur.

Cette tâche étant terminée, mon attention se porta sur la suivante, celle de migrer de la page d’information de projet de vue.js vers Blazor.

**Migrer la page d’information de projet de Vue.js vers Blazor.**

Cette tâche est assez similaire à la précédente pour ce qui est de son processus de résolution. La différence se situe au niveau du continu de la page à migrer. En effet elle contient des diagrammes d’activités retraçant les différentes interactions de l’utilisateur avec le sondage. Cette deuxième partie du rapport résumera donc les étapes que j’ai suivi pour migrer ces différents diagrammes de Vue.js vers Blazor.

Les différents diagrammes existants sur la plateforme **SimpleSurvey** sont basés sur la librairie Javascript HighCharts ; il s’agit une bibliothèque de diagrammes qui permet aux utilisateurs de créer des graphiques, des courbes et d'autres visualisations interactives et réactives pour le web. Elle propose une large gamme de types de graphiques (des graphiques en ligne, en aires, en barres, en colonnes, en camemberts, en nuages de points entre autres). Highcharts est compatible avec diverses technologies et plates-formes webdont Vue, et prend en charge une large gamme d'options de personnalisation (de couleurs, de polices, d’étiquettes, d’infobulles, d’animations, etc…).

Highcharts est largement utilisée dans le développement web et la visualisation de données pour présenter des données complexes de manière claire et significative. Il en va donc de soi que pour une compagnie telle que celle-ci travaillant dans le domaine du sondage interactif, cet outil soit un incontournable.

La migration des diagrammes de Vue.js vers Blazor intervient lors de la phase de migration du frontend, une fois que les modèles, le service ont bien été construit. Il est important de noter que les page Vue.js suivent elles aussi une architecture de type MVC (voir image MVC) donc nul besoin d’effectuer une migration du contrôleur. La raison de cette migration est la volonté de réunir les différentes pages et fonctionnalités de la plateforme sous une même technologie.

Pour revenir à la tâche, les tâches liées au backend (pareilles que la tâches précédente) étaient assez simple à exécuter donc passons au frontend et à la partie HighCharts. Implémenter HighCharts dans le contexte de Blazor requiert plusieurs étapes :

* Il faut tout d’abord ajoutez la bibliothèque JavaScript HighCharts au projet Blazor. Ceci se fait via le gestionnaire de packages NuGet de Visual Studio.
* Ensuite Dans le projet Blazor, créez un nouveau component (fichier Blazor (MyChart.razor par example)) qui hébergera le graphique Highcharts.
* Dans le code du component, il faut ajouter un élément (div) qui contiendra le graphique Highcharts. Par exemple <div id="chartContainer"></div>.
* Dans la section code du component, il est temps d’ajouter le code nécessaire pour générer le graphique Highcharts comme suit : A picture containing text, screenshot, software, font

  Description automatically generated Dans le code d'exemple ci-contre, la méthode **OnAfterRenderAsync** est utilisée pour appeler la fonction JavaScript **createChart**, qui générera le graphique Highcharts. L'interface **IJSRuntime** est injectée pour interagir avec le JavaScript dans C#.
* L’étape suivante est d’écrire la fonction createChart dans un fichier javascript du même nom que le fichier Blazor pour que les 2 soient automatiquement assemblés (MyChart.razor.js). En voici un exemple :

A screen shot of a computer screen

Description automatically generated with low confidence

Dans le code d'exemple ci-dessus, **createChart** utilise l'API Highcharts pour générer un graphique de ligne simple avec une série de données.

Ainsi grâce à mon apprentissage de ces différentes étapes il m’a été possible de migrer les différents diagrammes de la page de Vue.Js vers Blazor. J’ai pu passer de ce digramme montrant le nombre de tâche en rapport avec le sondage complété par jour A picture containing text, screenshot, software

Description automatically generated

À ce diagramme dans Blazor :

A screen shot of a computer program

Description automatically generated with low confidence

Bien sûr les étapes ont été plus approfondies mais il serait sensible d’aller en détails ici. Dans le but de mener à bien la tâche j’ai dû migrer un total de 5 diagrammes de Javascript vers Blazor.

À ce stade, la tâche étant terminée, il suffit de la soumettre sur GitHub pour revue. Il s’agit d’ici de l’un des projets les plus intéressant sur lequel j’ai pu travailler durant ce stage.

#### Résultats et réflexions

Dès le début de ce second stage au mois de janvier derniers, les différents objectifs d’apprentissage à atteindre étaient clairs. Ils étaient les mêmes que ceux du premier stage, et encore une fois m’ont servi de chemin à suivre tout le long. Ces objectifs sont au nombre de 5 et se présentent sous deux aspects, personnels et professionnels.

Pour ce stage, j’ai pris la décision de reconduire les mêmes objectifs car je ne les pensais pas achevés lors du premier ou autrement dit je trouvais qu’il y avait encore une marge de progression.

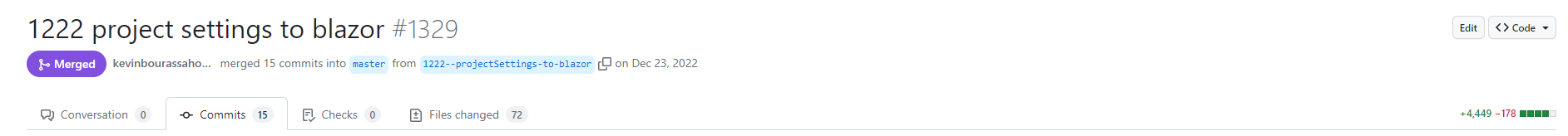
Sur le plan personnel, je me suis tout d’abord fixé l’objectif d’apprendre toute la structure du logiciel **SimpleSondage**, objectif allant de pair avec la UoCompétence « Curiosité intellectuelle et apprentissage continu ». En réalité, cet objectif en entier est pratiquement inatteignable pour un stagiaire, et pour vous donner un ordre d’idée cette tâche a pris 5 ans à mon superviseur. L’application **SimpleSurvey** comporte un total de 48 projets (applications) regroupées en 5 groupes à savoir les applications usuelles (il y’ en 15, contenant les applications rudimentaires telle que celle gérant les labels de la plateforme), les applications gérant les données (il y’en a 8 et parmi elles celle gérant la base de données et celle gérant les services), les applications API (au nombre de 8 qui assurent la communication client-serveur), les applications web (elles sont 12 et contiennent tout le frontend, blazor, apx et vue.js entre autres), et enfin les applications/services web (assurant le fonctionnement de l’application dans le cloud. Elles sont 5). De toutes ces applications et tout au long des 8 derniers mois je n’ai pu entrer en contact qu’avec 8 d’entre elles, pas par ce qu’elles ne m’étaient pas accessible mais parce qu’en ce laps de temps il ne m’a pas été possible de construire toutes les connaissances nécessaires pour les comprendre. Le mieux était donc de se focaliser sur des choses qu’il était possible de maitriser à l’instar de Blazor ou encore de l’architecture MVC, et je pense que j’y suis arrivé avec brio. Il est vrai que je ne maitrise pas la structure détaillée de l’application mais il est aussi vrai que J’en comprend l’architecture et tout ceci me permet de considérer cet objectif comme atteint.

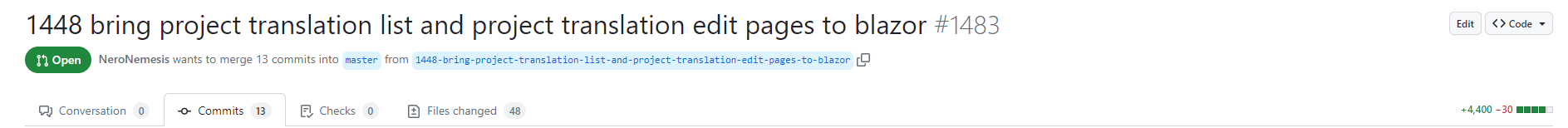
Mon deuxième objectif personnel était de connaitre les étapes à suivre dans le cadre de la migration vers de nouvelles technologies et la maintenance d'un logiciel existant (Par exemple migrer la partie code sans toutefois perdre des fonctionnalités au niveau de l'interface). Cet objectif est lié aux deux UoCompétences « Curiosité intellectuelle et apprentissage continu » et « Résolution de problème et créativité ». Durant la première moitié de mon stage, j’ai eu l’occasion d’effectuer des migrations de technologies c’est-à-dire aller d’un Framework web à un autre. Ces différentes tâches de migration ont éveillé en moi un intérêt particulier pour l’avancement, l’évolution, c’est-à-dire la recherche perpétuelle de l’excellence et du qualitatif. La qualité du code que j’ai fourni durant les 4 premiers mois de ce stage n’était pas la meilleure, mais j’ai eu ces 4 derniers mois pour m’améliorer et maitriser plus en profondeur les différents aspects à prendre en compte lors d’un changement de technologie. Pour illustrer mon propos, durant les 4 premiers mois je n’ai réussi à compléter que 4 tâches de migration mais durant la deuxième moitié du stage j’en ai fait plus de 8, ce qui démontre non seulement une évolution dans la productivité, mais aussi dans la maitrise du travail effectué. Ainsi, sur ces bases, cet objectif aussi peut être considéré comme plus qu’atteint.

Professionnellement parlant, les objectifs que je me suis fixés sont basés sur mon rendement car c’est l’un des aspects les plus importants pour définir la capacité à évoluer en équipe et donc dans le monde du travail. Ils sont au nombre de 3.

A screen shot of a computer program

Description automatically generated with low confidenceTout d’abord, le premier objectif était d’approfondir la qualité de rédaction de mon code par rapport aux 4 premiers mois de stage. Pour se faire, j’ai suivi en parallèle de mon stage plusieurs tutoriels et formations disponibles sur le web, tout en m’inspirant des pratiques de mes collègues. Cet objectif est lié à la UoCompétence « Curiosité intellectuelle et apprentissage continu ». En 8 mois de stage, les progrès sont palpables et clairement visible; j’ai pu passer d’un code comme celui-ci à un code comme celui-ci

Mon progrès se mesure aussi bien sûr lors de la révision des tâches que je soumets : par exemple pour la tâche de migrer la page de paramètre vers Blazor on passe de 178 corrections de mon code comme indiqué en bas à droite de l’image

à 30 corrections pour la migration de la fonctionnalité de traduction

Ainsi l’ensemble de ces arguments me permet de considérer cet objectif comme une réussite.

Deuxièmement, l’objectif était de terminer toutes les tâches qui m’étaient confiées en faisant preuve d’indépendance et de fiabilité. Durant la première phase de ce stage sur 13 tâches qui m’avaient été confiées, j’en avais terminé 12 dans les délais que je m’étais fixés. Ce résultat n’était pas mauvais et a eu pour conséquence de me voir être affecté à des tâches plus sensibles. Malgré tout j’ai gardé en tête le but de m’améliorer sur cet aspect et de faire mieux que lors des 4 premiers mois. Ainsi, sur les 16 tâches qui m’ont été confiées j’ai pu terminer toutes les 16 dans les délais que je m’étais imparti, avec un temps moyen de 2 semaines pour les tâches les plus compliquées. Il est important de souligné ici qu’aucun délai ne m’était imposé par mon superviseur, je me suis moi-même imposé ces délais pour continuer de maintenir un rendement élevé, une attitude qui va de pair avec la UoCompétance « Autonomie et résilience ». Étant donné ma résilience dans l’accomplissement de chacune de ces tâches, cet objectif lui aussi peut être considéré comme réussi.

Enfin, l’objectif numéro 3 était sur le plan communicationnel, à savoir développer un sens aigu du travail d'équipe pour offrir des feedbacks claires et précis sur mes avancées ou ce sur quoi je travaille; un objectif en phase avec les UoCompétences de « Communication » et de « Collaboration ». Il est certain qu’il est plus facile pour moi aujourd’hui de communiquer avec mes collèges qu’en début de stage; mes questions sont nettement plus claires et me permettent ainsi d’obtenir des réponses tout aussi claires, J’obtient les avis et le feedback de différents collègues par rapport aux tâches que j’accomplie et il m’est plus facile de faire des appels lorsque je bloque contre un mur. Une preuve de mon évolution sur cet aspect est l’absence de critique de mon superviseur sur ce plan, alors que cela avait été souligné lors de la première phase du stage. Ce qui me permet de considérer cet objectif comme atteint est ce sentiment d’appartenance et d’aisance que j’ai lorsque je pense à l’équipe mais aussi ma capacité à parler en détails de sujets techniques sans être mécompris.

L’atteinte de chacun de ces objectifs n’est pas anodine, c’est le résultat de bases d’apprentissage solides apprises tout le long de mon cursus universitaire. Chacun des cours proposés à l’université d’Ottawa dans le cadre du génie logiciel a pour but de maximiser nos et nous préparer pour le monde du travail. C’est ainsi que les cours de SEG2900 (Communication professionnel et Responsabilité) et SEG2911 (Pratique de la profession d'ingénieur logiciel) ont sur bâtir en moi le sens de l’excellence et du professionnalisme en situation d’emploi, et le cours de SEG2505 (Introduction au génie logiciel), m’a appris les normes et préceptes à suivre dans la rédaction, la soumission et la maintenance d’un code qualitatif. Ainsi chacune des connaissances que j’avais en débutant ce stage était le résultat de beaucoup de travail sur le plan théorique, et ces derniers mois m’ont permis de consolider ces connaissances en les mettant en pratique.

Ce stage de 8 mois que j'ai effectué chez **SimpleSurvey** a été une expérience enrichissante à bien des égards. Au-delà des compétences techniques que j'ai acquises, j'ai compris l'importance de la dévotion et du travail d’équipe pour le bien de ma carrière professionnelle. En effet, en prenant le temps de réfléchir sur mon rendement, j’ai pu rapidement identifier les aspects sur lesquels je devais évoluer, et ainsi m’appliquer en conséquence. Cette réflexion m'a également permis de prendre du recul sur mes ambitions professionnelles et de déterminer les étapes nécessaires pour atteindre mes objectifs à long terme. En écrivant ce rapport, j'ai pu établir des liens entre mon expérience chez **SimpleSurvey** et les valeurs et convictions qui me tiennent à cœur. Cette connexion a renforcé ma compréhension de ce que je veux accomplir en tant qu'ingénieur logiciel, et a guidé mes choix pour l'avenir. En fin de compte, je suis convaincu que la processus COOP est essentiel pour le développement de la carrière professionnelle de tout étudiant, et mon stage restera une ressource précieuse pour guider mes choix de carrière et m'aider à atteindre mes objectifs professionnels.

Je ne saurai terminer ce rapport sans transmettre mes sincères remercîments au bureau COOP pour son dévouement et son soutient perpétuel aux étudiant du début du placement jusqu’à la fin du stage. C’est aussi grâce à eux et leurs efforts non négligeables que tout ceci a été possible.

#### Conclusion

Au terme de ces 8 mois de stage chez SimpleSurvey, je peux dire que j'ai acquis une expérience professionnelle inestimable en génie logiciel. Les projets auxquels j'ai participé m'ont permis de mettre en pratique les connaissances théoriques acquises durant mon parcours universitaire et de développer de nouvelles compétences, notamment en matière de développement web et de travail en équipe. J'ai également appris à être autonome et à gérer efficacement mon temps pour respecter les échéances des projets.

Je suis reconnaissant envers mon superviseur Kevin, les membres de l’équipe de développement Mathieu et Cédric, M. David Houle et l'équipe de SimpleSurvey pour leur soutien et leurs conseils tout au long de mon stage. Leur expertise et leur professionnalisme ont été une source d'inspiration pour moi.

En termes d'amélioration, je pense que je pourrais encore travailler sur ma capacité à communiquer clairement et efficacement, ainsi que sur ma capacité à résoudre les problèmes de manière autonome. Je compte continuer à travailler sur ces compétences pour devenir un futur ingénieur logiciel encore plus performant.

Enfin, je tiens à remercier encore une fois le Bureau coop pour leur excellent travail dans le processus de placement et dans le suivi de mon stage. Leur soutien et leur encouragement ont été essentiels pour la réussite de mon expérience coopérative.

# **Références**

Ayant supervisé Gilles Djawa Ndjankeu, stagiaire coop, je, Kevin Bourassa-Houle, confirme que le rapport qui précède a été fait, à ma connaissance, au complet par cette personne. De plus, le rapport ne contient aucun renseignement confidentiel qui en interdirait la lecture par les membres du corps professoral.

Signature : \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Date : \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_