



Rapport de stage

Rapport final

Développement d'une API de collecte d'informations
pour la construction d'un profil client : Intégration d'une
API partenaire

2022-2023

BUT Informatique

Réalisé par Antoine ARGAILLOT

Sous la supervision de Christophe TRIOLEYRE et de Nathalie PALLEJA

Remerciements

J'aimerais remercier l'ensemble de l'équipe du service informatique de INOVÉA qui m'ont accompagné tout le long de ce stage. J'aimerais notamment remercier Louis et Lucas avec qui j'ai travaillé en étroite collaboration sur plusieurs missions.

Je tiens également à remercier mon Tuteur Christophe TRIOLEYRE pour m'avoir donné la chance de travailler au sein du service, formé au fur et à mesure des tâches et pour toute son aide dans la réalisation de ce rapport et la préparation de la soutenance.

Enfin, je souhaite remercier Nathalie PALLEJA pour son aide en tant que tuteur sur la rédaction du rapport et ses conseils pour la soutenance.

Résumé et mots-clefs

Résumé

Ce stage a pour objectif d'intégrer deux API de collecte d'informations clientes entre elles afin de garder à jour les données. La première API est interne à INOVÉA et se nomme RIC tandis que la deuxième appartient à un partenaire externe Upsideo. Le but est de transmettre les informations du recueil RIC vers Upsideo après le remplissage initial, puis de mettre à jour le RIC automatiquement pour chaque modification côté Upsideo. Je travaillerais uniquement sur la partie back-end du RIC, l'API étant codé en C# avec le framework ASP.NET.

Mots-clefs : API, back-end, données, C#

Abstract

This internship's objective is to integrate two customer information gathering API between them to keep updated the data. The first API is domestic to INOVEA and is named RIC while the other belongs to an external partner Upsideo. The goal is to pass the information of the RIC on to Upsideo after the initial filling, then to automatically update the RIC for each modification on Upsideo's side. I will work only on the back-end part of the RIC, the API being written in C# with the framework ASP.NET.

Key words : API, data, back-end, C#

Sommaire

Table des matières

Introduction	1
1. Contexte du stage	2
1.1. Le groupe INOVÉA.....	2
1.2. Le service informatique	3
1.3. Les enjeux du stage	3
2. Analyse	4
2.1. Analyse du sujet	4
2.2. Analyse de l'environnement technique.....	5
2.3. Analyse de l'existant.....	6
2.4. Spécifications techniques	8
3. Rapport technique	10
3.1. Implémentation des dtos de Upsideo.....	10
3.2. Création des routes API Upsideo.....	11
3.3. Création des mappers	14
3.4. Route d'envoi vers Upsideo.....	15
3.5. Route de mise à jour du RIC.....	17
3.6. Validation	20
4. Méthodologie et organisation du projet.....	22
4.1. Gestion de l'équipe et des projets	22
4.2. Démarche personnelle	23
4.3. Projets annexes	24
4.3.1. CRUD Amazon.....	24
4.3.2. Serveur discord Unity.....	24
4.3.3. Connexion SSO Andil	25
Conclusion	26

Table des figures

Figure 1 : Diagramme de classe d'une route discord.....	5
Figure 2 : Capture d'écran de l'API RIC.....	7
Figure 3 : Diagramme Use Case du RIC	8
Figure 4 : Exemple d'objet Upsideo importé	11
Figure 5 : requête GET générique	12
Figure 6 : Exemple de requête GET.....	13
Figure 7 : Requête PUT/POST générique.....	13
Figure 8 : Exemple de mapper	14
Figure 9 : première implémentation de l'envoi	15
Figure 10 : Gestion des contacts dans service	16
Figure 11 : Code d'envoi d'un objet à Upsideo	17
Figure 12 : Code de mise à jour pour un objet RIC	17
Figure 13 : Code de mise à jour alternatif du RIC pour liste d'objets	18
Figure 14 : Mise à jour du RIC v2.....	19
Figure 15 : Exemple d'utilisation de POSTMAN pour l'envoi	20

Glossaire

API : Interface de programmation d'application (Application Programming Interface), ensemble des routes (services) mises à disposition par une application afin de transmettre et récupérer des informations.

Architecture MVCS : MVCS pour Modèle Vue Contrôleur Service, architecture de projet utilisée pour organiser les différentes parties du code (interface, logique métier, appel à la base de données...) dans différentes couches distinctes.

Back-end : Partie de l'application caché à l'utilisateur contenant l'ensemble du code métier ainsi que les appels aux serveurs et à la base de données.

CRM : Gestion de la Relation Client (Customer Relationship Management), système logiciel comprenant un ensemble de solutions afin de gérer les relations et interactions avec le client.

Dto : Data Transfer Object, objet format JSON qui sera transmis ou récupéré dans une route API.

Framework : Ensemble de composants logiciels permettant d'automatiser et de rajouter des fonctionnalités à un logiciel via une architecture préfète.

Front-end : Partie de l'application visible par l'utilisateur contenant toutes les interfaces.

Mapper : Méthode permettant de créer ou de modifier un objet avec les informations et valeurs d'un ou plusieurs objets passés en paramètre.

NuGet : Gestionnaire de paquets du Framework Microsoft .NET.

Swagger : Logiciel d'aide au développement d'API permettant de générer automatiquement de la documentation, du code ou encore des tests.

Introduction

Dans le secteur du conseil financier, il est important de disposer d'un maximum d'informations sur les objectifs des clients ainsi que de leurs situations. En effet, c'est en construisant un profil adéquat que le conseiller sera capable de fournir les solutions les plus adaptées et donc, d'aider son client.

Pour ce faire, plusieurs formulaires, recueils ou sondages doivent être fait auprès du client afin de soutirer le plus d'informations utiles tout en respectant ses droits sur les données personnelles.

C'est pour cela que le groupe INOVÉA fait appel à une solution d'un partenaire, Upsideo, « Juliette » sur laquelle les clients peuvent remplir leurs informations. Aujourd'hui, le groupe souhaite s'occuper lui-même du remplissage initial grâce à son propre recueil RIC pour simplifier la saisie et collecter les données. Mon rôle est d'intégrer les deux solutions entre-elles afin de réaliser l'envoi initial à Upsideo et de garder à jour les informations du RIC.

En premier lieu, je vais vous présenter le groupe INOVÉA et son fonctionnement. Ensuite, je vais procéder à l'analyse du projet avec l'environnement technique et les spécifications techniques en plus de l'analyse de l'existant. Et enfin, je passerais au rapport technique où je détaillerais mes différentes implémentations en justifiant mes choix et expliquant mes difficultés.

1. Contexte du stage

Ce stage est réalisé au sein du service informatique de INOVÉA dans le but de participer au développement des API* de l'entreprise, dont spécifiquement le RIC (pour Recueil d'Information Client). Dans un premier temps, nous allons présenter le groupe INOVÉA avec ses différents services et activités, ensuite nous nous concentrerons sur le service informatique et son organisation et enfin, nous nous attarderons sur les enjeux du stage avec les besoins de l'entreprise et les missions qui me seront données.

1.1. Le groupe INOVÉA

L'entreprise INOVÉA est une entreprise par Action Simplifiée fondée en 2017 spécialisée dans la gestion de patrimoine et le conseil financier. Son siège social est installé à Mauguio qui est son seul site. En tant que société holding, elle gère quatre autres sociétés : INOVÉA Epargne (société de courtage d'assurance) ; INOVÉA Finance (société de conseil et investissements financiers) ; INOVÉA Immobilier (société de transaction immobilière) et enfin INOVÉA Crédit (société de courtage en financement). Ensemble, elles forment le groupe INOVÉA comptant une cinquantaine d'employés et à cela s'ajoute tout un réseau de plusieurs centaines de consultants et conseillers indépendants à l'échelle nationale.

INOVÉA réalise son chiffre d'affaires en mettant en contact les clients et les conseillers du réseau afin de définir des plans d'investissement. Tout type de client pouvant contacter INOVÉA, les ambassadeurs vont leur proposer un questionnaire avant de trouver un consultant adapté. Ce consultant va ainsi pouvoir vendre au client une solution patrimoniale, parmi celles sélectionnées par les différentes filiales de INOVÉA, correspondant à ses objectifs et à son profil. Ce stage se concentre justement sur l'un des formulaires que les conseillers vont utiliser pour cerner le profil du client, le RIC (pour Recueil d'Information Client) qui va contenir plusieurs onglets où le client va renseigner ses informations personnelles, financières et ses objectifs.

1.2. Le service informatique

Le service informatique compte en tout onze personnes, dont deux développeurs Front-end*, quatre développeurs Back-end*, un chargé de support, un alternant et trois stagiaires. Son principal objectif est de développer et de maintenir les différents projets informatiques dont ont besoin à la fois l'entreprise et ses différents services ainsi que les consultants indépendants qui utilisent les solutions de INOVÉA. Parmi ces solutions, nous avons par exemple en interne, un CRM* afin de faciliter la visualisation de la base de données et de réaliser les tests pour les différents projets plus facilement. (un consultant externe, un web designer)

Concernant l'organisation de ce service, Eric SEYCHAL, le responsable et directeur des système d'informations (DSI), s'assure que les projets développés répondent aux besoins exprimés par les autres services. Il organise notamment des réunions quotidiennement pour voir les avancées et problèmes de chacun ainsi que pour mettre en place les dates de rendu. Pour communiquer, nous avons à disposition Microsoft Teams pour tout appel à distance, réunion ou encore pour envoyer des fichiers. De plus, le télétravail est autorisé et, en tant que stagiaire, j'y ai le droit deux jours par semaine. Enfin, le matériel est fourni par l'entreprise avec un ordinateur portable et un VPN pour se connecter au réseau et travailler à distance.

1.3. Les enjeux du stage

Sur la période où je réalise ce stage, le service informatique de INOVÉA travaille sur plusieurs projets en même temps allant de la création de nouvelles solutions à l'optimisation et la refonte d'anciennes. Par conséquent, INOVÉA est à la recherche de nouveaux développeurs pour aider à la réalisation de ces projets et étendre le service. Mon maitre de stage m'a donc assigné comme projet principal le RIC sur lequel je vais devoir intégrer l'API d'un partenaire Upsideo pour garder les données client à jour dans notre et leur base de données. En plus de ce projet, je réalise plusieurs tâches sur d'autres projets afin de me familiariser avec les différentes API, le langage C# et l'architecture du projet. L'objectif est de me former sur le développement C# sur ce stage afin de poursuivre sur une alternance en troisième année si j'arrive à accomplir mes tâches convenablement.

2. Analyse

Au sein de cette partie, nous allons procéder à l'explication du sujet de stage avec ses missions et ses contraintes. Pour ce faire, je vais diviser cette analyse en quatre sous-parties en commençant tout d'abord par l'analyse du sujet et des demandes initiales du stage. Ensuite, nous passerons à l'analyse de l'environnement technique ainsi que l'analyse de l'existant pour se rendre compte des outils et de l'architecture mises à disposition. Et enfin, nous verrons et détaillerons les tâches et missions du stage via le cahier des charges.

2.1. Analyse du sujet

Les conseillers appartenant au réseau de INOVÉA utilisent les solutions du service informatique pour créer des parcours répondant aux objectifs des clients.

Une des premières étapes pour constituer un parcours est de collecter des informations sur le client afin de connaître ses objectifs et de cerner son profil dans le but de trouver les solutions les plus appropriées. Pour ce faire, INOVÉA fait appel à un partenaire Upsideo qui met à disposition un site web où le client va remplir une multitude de champs par rapport à son identité, ses informations financières et familiales ou encore ses objectifs de vie.

INOVÉA souhaite aujourd'hui internaliser ce processus de collecte d'informations avec une nouvelle API interne nommée RIC afin de faciliter le remplissage d'information grâce à une interface plus ergonomique et de garder ces informations dans une base de données interne. L'API RIC n'est cependant utilisée que pour le remplissage initial du recueil, les informations seront donc stockées dans la base de données d'INOVÉA puis envoyées à l'API d'Upsideo qui sera toujours utilisé pour l'affichage et la modification.

La partie Front-end du RIC est en cours de développement par un alternant en suivant une maquette fournie par INOVÉA. Cette partie est donc sujet à des changements en fonction des décisions prises aux réunions et à l'évolution de la maquette.

Ainsi, les objectifs de mon stage sont d'intégrer l'API Upsideo à la solution RIC, c'est-à-dire mettre en place l'envoi des données de RIC à Upsideo en faisant

les associations nécessaires et faire l'inverse afin que chaque modification dans Upsideo soit effectuée dans la base de données de INOVÉA. Il sera question notamment d'utiliser les routes de Upsideo déjà mises en place pour la création, la récupération et la mise à jour des champs de Upsideo.

En plus de travailler sur le RIC, je dois également intervenir sur d'autres projets informatiques au niveau du Back-end pour apporter mon aide et me familiariser aux technologies utilisées.

2.2. Analyse de l'environnement technique

Pour réaliser ses différents projets, INOVÉA utilise C# avec le Framework* ASP.NET pour tout ce qui est développement Back-end tandis que le Front-end est codé en Javascript en utilisant le Framework Angular. En tant que développeur Back-end, je vais me concentrer sur l'architecture générale du projet en plus des différents outils et environnements accompagnant le développement.

En ce qui concerne l'architecture, les API sont organisés en utilisant une version légèrement modifiée de MVCS* avec les couches : Front ; Controller ; Business ; Service et Repository.

Pour illustrer les différentes couches utilisées, voici un diagramme de classe représentatif d'une route du RIC que j'ai développé :

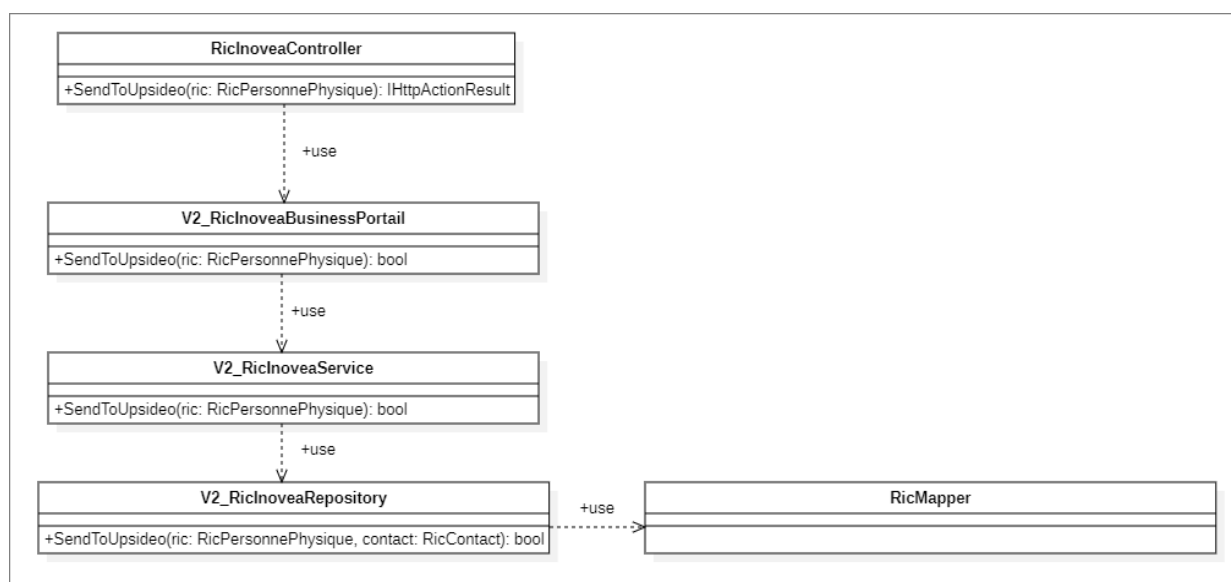


Figure 1 : Diagramme de classe d'une route discord

Comme nous pouvons le voir avec la méthode « GetRolesDiscord », tout part du Controller qui va appeler la méthode correspondante dans le Business qui lui est associé, de même le Business va faire pareil avec son Service qui va lui aussi appeler son Repository. Chaque couche a donc son propre rôle, le Repository gère tout ce qui est appel à la base de données, ici par exemple pour récupérer les objets Upsideo et les Dto du RIC ainsi que pour mettre à jour le RIC. Les couches Business et service concentrent le code métier, dans cet exemple, le Service va exécuter pour chaque contact du RIC la méthode SendToUpsideo du Repository. Enfin, nous avons le Controller qui va définir les paramètres de la route avec son nom et les paramètres nécessaires pour pouvoir lancer la requête sur l'API.

En plus de cette architecture, plusieurs éléments sont mis en place comme les mapper* qui servent à passer d'un objet à un autre selon le besoin ainsi que les Dto* pour pouvoir transmettre des données sous format JSON. Dans ce cas, j'utilise les mappers de RicMapper pour passer d'un Dto du RIC à un objet Upsideo et inversement avant de faire l'envoi ou la mise à jour.

Pour ce qui est des outils, l'ensemble du code est écrit en s'aidant de l'IDE de Microsoft « Visual Code » qui permet d'importer les différents paquets nécessaires via NuGet* ou encore d'aider au débogage. Le gestionnaire de version GitLab est également utilisé en parallèle pour créer les différentes branches et gérer l'avancement du projet. Comme nous utilisons spécifiquement la version communautaire de 2017 de Visual Code qui n'a pas git intégré, nous nous servons du client Git GitTower afin de réaliser les différentes commandes de git comme les commit. En outre, les différentes routes API sont testées grâce à la plateforme POSTMAN qui permet d'envoyer et de paramétrer des requêtes http et d'afficher leur résultat. Enfin, INOVEA utilise comme système de gestion de base de données « Microsoft SQL Server » avec l'outil « SQL Server management Studio » pour stocker et manipuler ses données.

2.3. Analyse de l'existant

Pour réaliser ce travail d'intégration, nous disposons de l'ensemble des routes API que Upsideo met à disposition (Elles sont regroupées et testables sur ce lien <https://API.preprod.upsideo.fr/swagger/index.html>). Ainsi, nous avons

juste besoin de créer les Dto correspondant aux objets Upsideo que le RIC prend en charge ainsi que les méthodes GET, PUT et POST pour chacun.

Le résultat de ces méthodes pourra être observé directement dans le site de Upsideo pour chaque client qui auront rempli le RIC ce qui facilitera les tests initiaux.

Concernant le RIC, une version stable et fonctionnelle a été développée par un alternant et comprend l'ensemble des onglets que j'aurais à intégrer à Upsideo. Les Dto du RIC ont notamment déjà été implémentés sur le site, mais il existe quelques erreurs avec certaines valeurs renvoyées qui ne correspondent pas à celle présente chez Upsideo ou qui n'ont pas été renvoyé par le formulaire. De plus, certains champs dans Upsideo manquent à la fois dans cette version du RIC et dans la maquette. Il conviendra donc de faire remonter ces problèmes afin de corriger cela dans la version finale. Vous pouvez voir un visuel des sites du RIC et de Upsideo sur les annexes 1 et 2 respectivement.

Après avoir rempli les différents formulaires du RIC, le client devra appuyer sur un bouton pour envoyer à Upsideo. C'est ici que sera utilisé ma route API qui se chargera d'appeler les différentes routes de Upsideo.

Voici un diagramme Use Case pour illustrer le fonctionnement :

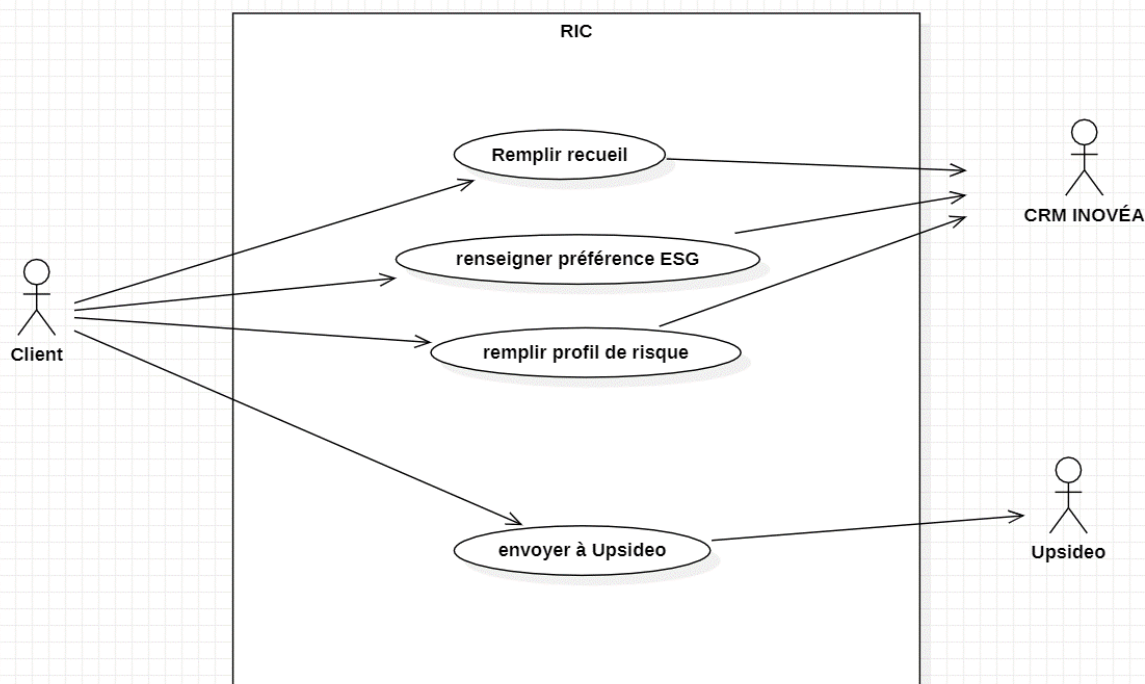


Figure 3 : Diagramme Use Case du RIC

2.4. Spécifications techniques

Dans le cadre du projet RIC, je n'ai pas eu de cahier des charges afin de rendre compte de mes tâches, j'ai donc, pour ce rapport, dû le rédiger moi-même avec les informations et missions que m'a donné mon maître de stage (cf. Annexe 3). J'ai jugé que ce projet pouvait être séparé en trois parties : l'analyse initiale des routes Upsideo et de la maquette du RIC ; l'Envoi des données du RIC vers Upsideo et la Mise à jour des données du RIC. Je vais par conséquent expliquer les besoins de chaque partie avec les tâches associées.

Tout d'abord, l'Analyse des routes Upsideo et de la maquette RIC qui correspond à la partie préparatoire du projet. En effet, si le RIC a été réalisé et pensé pour remplacer Upsideo et possède donc des champs et onglets similaires, beaucoup de différences existent entre les deux notamment au niveau de la disposition ou du nommage. Afin de faciliter les futurs mappings, un tableau Excel doit être réalisé avec les correspondances entre les onglets de RIC et les routes de Upsideo (cf. Annexe 4). En plus de ce tableau, l'ensemble des Dto de Upsideo doit être implémenté dans notre API pour pouvoir créer les multiples objets Upsideo et interagir avec leurs routes. Pour aller plus loin sur ces routes, il faut importer celles de création POST, de récupération GET et enfin de modification PUT dans Upsideo. Les routes de création et de modification serviront pour l'envoi vers Upsideo grâce aux objets Dto mentionnés précédemment tandis que celles de récupération seront utilisées pour l'inverse quand le recueil Upsideo sera modifié.

Après ces étapes préparatoires vient l'envoi des données vers Upsideo qui constitue la partie la plus importante du projet. Il faudra dans un premier temps créer tous les mappers nécessaires pour passer du Dto du RIC à son équivalent chez Upsideo. Toute la partie d'analyse précédente servira ici, mais les Dto du RIC peuvent être légèrement différentes de la maquette ce qui ralentit le travail. À la suite de cela, une route API devra être mise en place dans le but d'appeler toutes les routes de création de Upsideo. Une subtilité est que le recueil du conjoint, s'il existe, doit aussi être envoyé en même temps que celui du contact principal ce que je vais devoir en tenir compte dans mon implémentation.

Enfin, il ne restera plus qu'à effectuer le travail précédent dans le sens inverse avec la mise à jour des données du RIC. Cette partie sera plus rapide, car les correspondances auront déjà été trouvées/corrigées et il ne faudra que reprendre les mappers dans l'autre sens. Cependant, comme nous ne souhaitons pas supprimer ce qui se trouve déjà dans notre base de données, un Dto du RIC devra être récupéré et modifié par le mapper plutôt que d'en créer un nouveau directement. Ainsi, seules les routes de modification seront utilisées pour mettre à jour la base de données.

3. Rapport technique

Cette partie sera dédiée aux différents choix techniques et implémentations fait le long du projet sur le RIC en expliquant à chaque fois le fonctionnement et les éventuelles difficultés.

3.1. Implémentation des dtos de Upsideo

L'ensemble des objets que je vais intégrer aux RIC sont disponibles sur le swagger* de Upsideo qui détaille non seulement la structure de tous les objets avec l'ensemble des attributs, mais aussi toutes les routes API disponibles avec les différents paramètres et leurs retours. Voici un exemple d'objet Upsideo sur le swagger

```

PatrimoineDivers ▾ {
  id                string($uuid)
                    title: id
                    nullable: true
                    Entity identifier

  status            Status string
                    title: Status

                    'NotDefined' : NotDefined, 'Deleted' : Deleted, 'Updated' : Updated, 'Created' : Created
                    Enum:
                    > Array [ 4 ]

  timestamp         string
                    title: timestamp
                    nullable: true

                    Concurrency token. Ensure that you get an exception if a row you are updating has changed since you queried it.

  clientId          string($uuid)
                    title: clientId
                    nullable: true

  recueilId         string($uuid)
                    title: recueilId
                    nullable: true

  avecPatrimoineDivers boolean
                    title: avecPatrimoineDivers
                    nullable: true

  patrimoineDiversMontant number($double)
                    title: patrimoineDiversMontant
                    nullable: true

  patrimoineDiversDetail boolean
                    title: patrimoineDiversDetail
                    nullable: true
}

```

Ainsi, pour importer l'objet et créer le Dto correspondant, il suffit de reprendre rigoureusement ce qui est indiqué sur le swagger. Certains attributs nécessitent de créer des énumérations spécifiques, comme pour le « status » dans l'exemple. Comme les attributs id, timestamp et status sont générés à la création chez Upsideo, nous ne devons ni les préremplir ni les modifier sinon cela génère des erreurs à la création. Nous sommes par conséquent obligés de créer deux versions du Dto à chaque fois, une version pour l'envoi POST sans ces attributs

puis une version complète pour le GET et PUT qui hérite du POST. Voici l'adaptation de l'objet Upsideo précédent dans le code :

```
public class PatrimoineDiversUpsideo : PatrimoineDiversUpsideoPost
{
    [JsonProperty(PropertyName = "id")]
    public Guid? id { get; set; }

    [JsonProperty(PropertyName = "status")]
    public Status status { get; set; }

    [JsonProperty(PropertyName = "timestamp")]
    public string timestamp { get; set; }
}

public class PatrimoineDiversUpsideoPost
{
    [JsonProperty(PropertyName = "clientId")]
    public Guid? clientId { get; set; }

    [JsonProperty(PropertyName = "recueilId")]
    public Guid? recueilId { get; set; }

    [JsonProperty(PropertyName = "avecPatrimoineDivers")]
    public bool avecPatrimoineDivers { get; set; }

    [JsonProperty(PropertyName = "patrimoineDiversMontant")]
    public double patrimoineDiversMontant { get; set; }

    [JsonProperty(PropertyName = "patrimoineDiversDetail")]
    public bool patrimoineDiversDetail { get; set; }

    [JsonProperty(PropertyName = "notes")]
    public string notes { get; set; }

    [JsonProperty(PropertyName = "notesMimeType")]
    public string notesMimeType { get; set; }
}
```

Figure 4 : Exemple d'objet Upsideo importé

3.2. Création des routes API Upsideo

Upsideo possède plusieurs routes API permettant de faire le CRUD, celles-ci sont détaillées là encore dans le swagger (cf. Annexe 5) :

Nous avons besoin d'implémenter les routes POST et PUT pour l'envoi tandis que les GET serviront à la mise à jour du RIC. Pour pouvoir les appeler, nous utilisons la classe WebRequest qui permet de créer et d'envoyer des requêtes en définissant les différentes options (headers, méthodes, type de contenu ...) puis de récupérer le résultat. Avant mon arrivé, plusieurs méthodes avaient déjà été

réalisées pour appeler des routes Upsideo, mais elles reprenaient toutes la même structure en changeant seulement les valeurs et le type. Nous avons donc extrait la structure pour créer trois méthodes génériques, une pour le GET, une pour récupérer les notes et la dernière pour le PUT ET POST. Cela nous a permis de grandement alléger le nombre de lignes, d'éviter de dupliquer du code et enfin de simplifier le processus de création des méthodes d'appel.

Pour la première requête générique GET, qui va servir à récupérer l'objet, nous voulons pouvoir l'utiliser pour tous les objets Upsideo créés jusqu'ici. J'ai donc utilisé un paramètre de type générique qui permet de prendre en argument n'importe quel type spécifié par l'utilisateur lors de l'appel.

Voici la requête générique pour les GET :

```
private T getGenerique<T>(string id, string path, string typeApi) where T : new()
{
    try
    {
        T objet = JsonConvert.DeserializeObject<T>(GetUpsideo(id, path, typeApi));
        return objet;
    }
    catch (WebException)
    {
        return new T();
    }
}
```

Figure 5 : requête GET générique

Un autre problème pour les routes GET de Upsideo est que les champs de note, de précisions ou de descriptions ne sont pas renvoyés avec les autres attributs. Pour récupérer ces champs, Upsideo met à disposition des routes spécifiques renvoyant le string de la note correspondante. J'utilise donc une deuxième requête générique qui prend en paramètre l'identifiant d'un objet Upsideo et renvoi le string de l'attribut spécifié (cf. Annexe 6).

Pour coder une route GET complète finalement, il faut tout d'abord utiliser la première requête générique pour récupérer l'objet Upsideo et ensuite, assigner aux champs de notes les résultats des routes correspondantes.

Voici un exemple d'implémentation de GET pour illustrer :

```
public async Task<PatrimoineDiversUpsideo> GetPatrimoineDiversUpsideoRecueilId(string paramId)
{
    PatrimoineDiversUpsideo PatrimoineDivers = getGenerique<PatrimoineDiversUpsideo>(paramId, "PatrimoineDivers/RecueilId", _APIPARCOURS);
    PatrimoineDivers.notes = getGeneriqueNotes(PatrimoineDivers.id.ToString(), "PatrimoineDivers", _APIPARCOURS, "Notes");
    return PatrimoineDivers;
}
```

Figure 6 : Exemple de requête GET

Pour ce qui est de la méthode générique pour les POST et PUT, nous lui passons en paramètre un attribut « Method » pour choisir entre les deux puis nous lui passons le JSON de l'objet que nous souhaitons envoyer.

Voici la méthode généralisée pour les POST et PUT :

```
private string requestGenerique(string request, string json, string typeApi, string method)
{
    string stringResponse = string.Empty;

    WebRequest myWebRequest = HeadRequest(request, GetTokenUpsideo(typeApi), typeApi);
    myWebRequest.Method = method;
    myWebRequest.ContentType = "application/json; charset=utf-8";

    try
    {
        using (var streamWriter = new StreamWriter(myWebRequest.GetRequestStream()))
        {
            streamWriter.Write(json);
        }

        WebResponse response = myWebRequest.GetResponse();
        StreamReader reader = new StreamReader(response.GetResponseStream());
        stringResponse = reader.ReadToEnd();

        return stringResponse;
    }
    catch (WebException ex)
    {
        string retour = ex.Message;

        return retour;
    }
}
```

Figure 7 : Requête PUT/POST générique

3.3. Création des mappers

Une des étapes essentielles du projet est d'associer les objets du RIC et de Upsideo pour pouvoir ensuite envoyer à Upsideo ou mettre à jour dans le RIC les objets créés/modifiés.

Les méthodes « mappers » serviront donc à créer ou à modifier l'objet RIC/Upsideo avec les valeurs de l'objet passé en paramètre. De manière générale, pour tous les mappers de RIC vers Upsideo, l'objet Upsideo sera créé et rempli avec plusieurs attributs : le Dto du RIC correspondant ; les identifiants du recueil et du client Upsideo et enfin le contact du RIC. Pour ce qui est du Dto, il s'agit juste de trouver les attributs équivalents en fonction du nom ou des valeurs renvoyées. Pour le contact, il sera généralement utilisé pour tout ce qui est affiliation d'un bien ou d'une personne pour décider s'il appartient au couple ou un membre du couple en particulier. Et enfin, les identifiants vont remplir les champs de l'objet Upsideo du même nom. Il se peut qu'il y ait des problèmes de conversion entre les deux attributs, notamment avec les énumérations. Si la valeur renvoyée par le RIC est similaire à Upsideo, je peux faire un Enum.TryParse pour créer l'énumération à partir du string directement, sinon je dois coder un autre mapper. Voici ci-dessous un exemple d'un mapper typique :

```
public static IdentitePersonnePhysiqueUpsideoPost MapIdentitePersonnePhysiqueRicToUpsideo(this V3_ina_ric_recueil_identite_ph
{

    Enum.TryParse<Departement>(ric.departementNaissance, out Departement departement);
    Enum.TryParse<Pays>(ric.paysNaissance, out Pays pays);
    Enum.TryParse<CapaciteJuridique>(ric.capaciteJuridique, out CapaciteJuridique capaciteJuridique);
    IdentitePersonnePhysiqueUpsideoPost UpsideoPost = new IdentitePersonnePhysiqueUpsideoPost();

    UpsideoPost.departementDeNaissance = departement;
    UpsideoPost.paysDeNaissance = pays;
    UpsideoPost.villeDeNaissance = contact.LieuNaissance;
    UpsideoPost.codePostalDeNaissance = ric.codePostalNaissance;
    UpsideoPost.capaciteJuridique = capaciteJuridique;
    UpsideoPost.recueilId = recueilIdUpsideo;
    UpsideoPost.clientId = clientId;
    UpsideoPost.telephone = contact.TelephonePortable;
    UpsideoPost.telephoneProfessionnel = contact.TelephonePortable;

    return UpsideoPost;
}
```

Figure 8 : Exemple de mapper

Certains objets Upsideo nécessitent plusieurs Dto du RIC pour être remplis. Dans ce cas, il y a plusieurs mappers et je modifie un objet Upsideo passé en

paramètre plutôt que de le créer afin d’avoir une version complète à la suite de tous les mappings.

3.4. Route d’envoi vers Upsideo

La route API pour envoyer à Upsideo sera appelée par l’utilisateur dès qu’il aura confirmé son RIC et appuyé sur le bouton d’envoi. Elle prend en paramètre le dto global du RIC qui contient les dto des différents onglets. Celui-ci possède à la fois les informations du contact principal, mais aussi celui du conjoint s’il existe.

Pour pouvoir envoyer les informations du Dto RIC à Upsideo, cela se passe en trois étapes. Tout d’abord, il faut sélectionner le bon Dto RIC entre celui du contact principal et celui du conjoint. Ensuite, je dois créer le ou les objets Upsideo à partir du Dto RIC grâce aux mappers codés au préalable. Enfin, je peux procéder à l’envoi grâce aux routes POST de Upsideo.

La difficulté principale a été de trouver le moyen pour gérer le conjoint et envoyer ses données à lui aussi. Dans ma première implémentation, je parcours l’ensemble des objets du RIC en vérifiant s’il s’agit de l’identifiant du contact principal ou celui du conjoint. Ensuite, j’appelle le mapper avec les valeurs du contact correspondant avant de l’envoyer à Upsideo. Voici ce que cela donne dans l’extrait ci-dessous :

```
//FATCA

foreach (V3_ina_ric_recueil_fatcaDto fatca in ric.RicInformationsPersonnelles.RicRecueilFatcaDtos)
{
    if (fatca.contactId == contactId) {
        await upsideoRepository.PostFatcaUpsideo(fatca.MapFatcaRicToUpsideo(idUpsideo,recueilId));
    }
    else if (fatca.contactId == contactIdConjoint)
    {
        await upsideoRepository.PostFatcaUpsideo(fatca.MapFatcaRicToUpsideo(idUpsideoConjoint,recueilIdConjoint));
    }
}
```

Figure 9 : première implémentation de l’envoi

Cette implémentation pose plusieurs problèmes du fait de la duplication de code qui alourdit et complexifie la méthode, mais aussi pour la future maintenabilité de la solution. Si jamais à l’avenir, nous voulions rajouter des personnes ou enlever le conjoint, il faudrait modifier chaque boucle ce qui prendrait du temps et alourdirait davantage la méthode en cas d’ajout. De plus, si

nous rajoutions les informations sur les enfants par exemple, il y aurait un nombre variable de Dto ce qui serait impossible à traduire avec ce modèle.

Dans la deuxième implémentation de l'envoi, j'ai pris en compte ces éléments et j'ai décidé de gérer les contacts dans la couche service. Elle lance une boucle foreach sur les contacts dans laquelle j'appelle la méthode du Repository. Celle-ci prend désormais comme paramètre le contact qui servira à filtrer les Dto pour n'en traiter qu'un seul à la fois. Ainsi, l'envoi à Upsideo est fait pour l'ensemble des contacts présent dans le RIC, peu importe le nombre, et la méthode du Repository n'a plus aucune duplication. Le problème du conjoint ayant été réglé, la route remplit l'ensemble des spécifications demandées comme décrit dans l'apprentissage critique 21.01 « Élaborer et implémenter les spécifications fonctionnelles et non fonctionnelles à partir des exigences ». Voici à quoi ressemble la méthode dans le Service ci-dessous :

```
public async Task<bool> SendToUpsideo(RicPersonnePhysique ric)
{
    bool result = true;
    //Contact principal et conjoint s'il existe
    IEnumerable<RicContact> contacts = ric.RicInformationsPersonnelles.RicContacts.Take(2);
    foreach (RicContact contact in contacts)
    {
        result &= await _ricInoveaRepository.SendToUpsideo(ric, contact);
    }
    return result;
}
```

Figure 10 : Gestion des contacts dans service

Pour la couche Repository, j'ai également utilisé une LINQ qui permet de faire des requêtes de données sur des collections avec une syntaxe très similaire à SQL. À partir de la liste de Dto RIC, je tri en mettant dans la clause « Where » que le contactId doit être égale à celui passé en paramètre et ensuite, je génère les objets Upsideo en appliquant un mapper à l'objet RIC dans la clause « Select ». À la fin, j'obtiens une liste d'objets Upsideo que je vais pouvoir envoyer. Voici un exemple d'envoi ci-dessous :


```
//FATCA

FatcaUpsideoPost fatca = ric.RicInformationsPersonnelles.RicRecueilFatcaDtos
    .Where(f => f.contactId == contact.ContactId)
    .Select(f => f.MapFatcaRicToUpsideo(idUpsideo, recueilId)).FirstOrDefault();

await upsideoRepository.PostFatcaUpsideo(fatca);
```

Figure 11 : Code d'envoi d'un objet à Upsideo

Néanmoins, dans le cas où un objet RIC est lié à plusieurs objets Upsideo, j'enlève le Select et je fais les mappings manuellement pour chacun d'entre eux.

3.5. Route de mise à jour du RIC

Pour chaque modification du recueil chez Upsideo, une route API que j'ai développée sera appelée et se chargera de mettre à jour les données du RIC.

Une des premières difficultés pour concevoir cette route a été de savoir quelle approche j'allais prendre. J'ai eu le choix entre implémenter une version qui générerait un Dto global du RIC et une version où je récupérais le Dto RIC de chaque objet un par un. La première est plus simple à mettre en place et à comprendre, en plus de nécessiter moins de lignes. L'autre est plus modulaire, ce qui permet, avec des paramètres supplémentaires, de choisir les onglets qui sont à modifier et donc, enlever des appels à la base de données. Mon choix s'est finalement porté vers la seconde solution pour son potentiel d'optimisation, le nombre d'appel étant assez élevé si je devais en faire pour chaque objet.

Chaque recueil Upsideo, à la différence du RIC, ne concerne qu'un seul contact. J'ai par conséquent pris l'identifiant du recueil Upsideo et l'identifiant du contact RIC associé comme arguments de ma méthode afin de récupérer les différents objets via les GET mis en place. Ensuite, il suffit de faire la mise à jour du RIC avec une méthode PUT après avoir appliqué un mapper. Voici l'implémentation dans le code :

```
//FATCA

V3_ina_ric_recueil_fatcaDto fatcaRIC = (await GetRicFATCA(new V3_ina_ric_recueil_fatcaDto() { contactId = contactId })).FirstOrDefault();
FatcaUpsideo fatcaUpsideo = await upsideoRepository.GetFatcaUpsideoRecueilId(recueilIdUpsideo.ToString());
PutRicFATCA(fatcaRIC.MapFatcaUpsideoToRic(fatcaUpsideo));
```

Figure 12 : Code de mise à jour pour un objet RIC

C'est le cas le plus fréquent quand nous avons qu'un seul objet des deux côtés, mais pour ce qui est des onglets dans lesquels nous pouvons rajouter plusieurs objets comme les biens par exemple, il faut traiter cela différemment. En outre, nous n'avons pas moyen de connaître la correspondance entre le RIC et Upsideo et nous ne pouvons pas nous fier à l'ordre d'insertion car supprimer un objet le changerait. Ne voyant pas d'autres solutions immédiatement, j'ai décidé de remplacer complètement les valeurs du RIC par celles de Upsideo. Dans un premier temps, je supprime tous les objets du RIC d'un contact ou d'un recueil si c'est en commun. Puis dans un second temps, je crée les objets RIC avec les valeurs renvoyés par Upsideo avant de faire un POST pour l'envoyer à la base de données. Voici ce que cela donne :

```
//Biens d'Usage

List<V3_ina_ric_recueil_biens_usageDto> biensUsagesRIC = await GetRicRecueilBiensUsage(new V3_ina_ric_recueil_biens_usageDto() { recueilId = recueilIdRic });
biensUsagesRIC.ForEach(delegate (V3_ina_ric_recueil_biens_usageDto bien) {
    DeleteRicRecueilBiensUsage(bien);
});

List<ImmobilierDeJouissanceUpsideo> ImmobilierDeJouissanceUpsideo = await upsideoRepository.GetImmobilierDeJouissanceUpsideoRecueilId(recueilIdUpsideo.ToString());
foreach (ImmobilierDeJouissanceUpsideo immobilier in ImmobilierDeJouissanceUpsideo)
{
    await PostRicRecueilBiensUsage(immobilier.MapImmobilierDeJouissanceUpsideoToRic(recueilIdRic, contactId));
}
```

Figure 13 : Code de mise à jour alternatif du RIC pour liste d'objets

Après avoir réanalyser ma solution lors du refactoring, je me suis rendu compte que le nombre d'appel généré pouvait être réduit en adoptant une autre approche. A la place de tout supprimer, je commence par faire des PUT pour modifier le maximum d'objets du RIC puis je supprime les objets RIC en trop s'il en reste. Dans le cas où il y a plus d'objets Upsideo que dans le RIC, je fais des POST comme dans l'ancienne implémentation. Afin de coder cela, j'ai choisi de créer une pile contenant les objets du RIC avec la collection Stack de C#. Ainsi, je récupère les objets un par un dans la pile pour chaque PUT tout en les supprimant. Quand la pile est vide dans la boucle foreach, je passe à des POST, et si elle ne l'est pas une fois sortie du foreach, je lance une nouvelle boucle while pour supprimer le reste. Cela m'a permis de faire le lien avec l'apprentissage critique 22.01 « Choisir des structures de données complexes adaptées au problème », ayant choisi d'utiliser une pile à la place d'une liste normale car elle permet de faire la récupération et la suppression en une seule action via le dépilage.

Enfin, la dernière problématique de cette méthode était de trouver un moyen pour faire uniquement la mise à jour des onglets qui ont été modifié. De plus, Upsideo ne nous a pas donné d'indications sur quand ils comptaient appeler notre route, si c'était après la modification de chaque champ, d'un objet ou à la toute fin. La solution se doit donc d'être adaptable pour ne pas causer de problèmes chez eux. Mon choix s'est porté sur la création d'un Dto contenant les booléens de chaque objet de Upsideo que je vérifie avant chaque bout de code de mise à jour (cf. Annexe 7). Cela permet de garder une certaine flexibilité côté Upsideo, s'ils souhaitent mettre à jour un objet à la fois, ils ont juste à mettre le booléen correspondant à « true » dans le Dto. Dans le cas contraire où il voudrait le faire à la sortie du recueil, ils ont juste à modifier au fur et à mesure le Dto et faire l'appel à la fin.

Voici l'implémentation dans le code de cette deuxième implémentation :

```
//Personnes à Charges
if (modifications.personnesACharge)
{
    Stack<V3_ina_ric_recueil_foyerDto> foyerRIC = new Stack<V3_ina_ric_recueil_foyerDto>();
    await GetRicFoyer(new V3_ina_ric_recueil_foyerDto() { contactId = contactId });
    List<PersonneAChargeUpsideo> personneAChargesUpsideo = await upsideoRepository.GetPersonneAChargeUpsideoRecueilId(recueilIdUpsideo.ToString());
    foreach (PersonneAChargeUpsideo personneACharge in personneAChargesUpsideo)
    {
        if (foyerRIC.Count > 0) PutRicFoyer((foyerRIC.Pop()).MapPersonneAChargeUpsideoToRic(personneACharge));
        else await PostRicFoyer(personneACharge.MapPersonneAChargeUpsideoToRic(recueilIdRic, contactId));
    }
    while (foyerRIC.Count > 0)
    {
        DeleteRicFoyer(foyerRIC.Pop());
    }
}
```

Figure 14 : Mise à jour du RIC v2

3.6. Validation

Les tests de mes routes se sont effectués grâce à la plateforme POSTMAN sur lequel je renseigne les différents paramètres dont j'ai besoin pour tel client test avant de les lancer. Pour la route d'envoi par exemple, elle prend en paramètre un Dto du RIC que je vais donner en plaçant son JSON dans le body de la requête. Voici à quoi ressemble la requête sur POSTMAN :

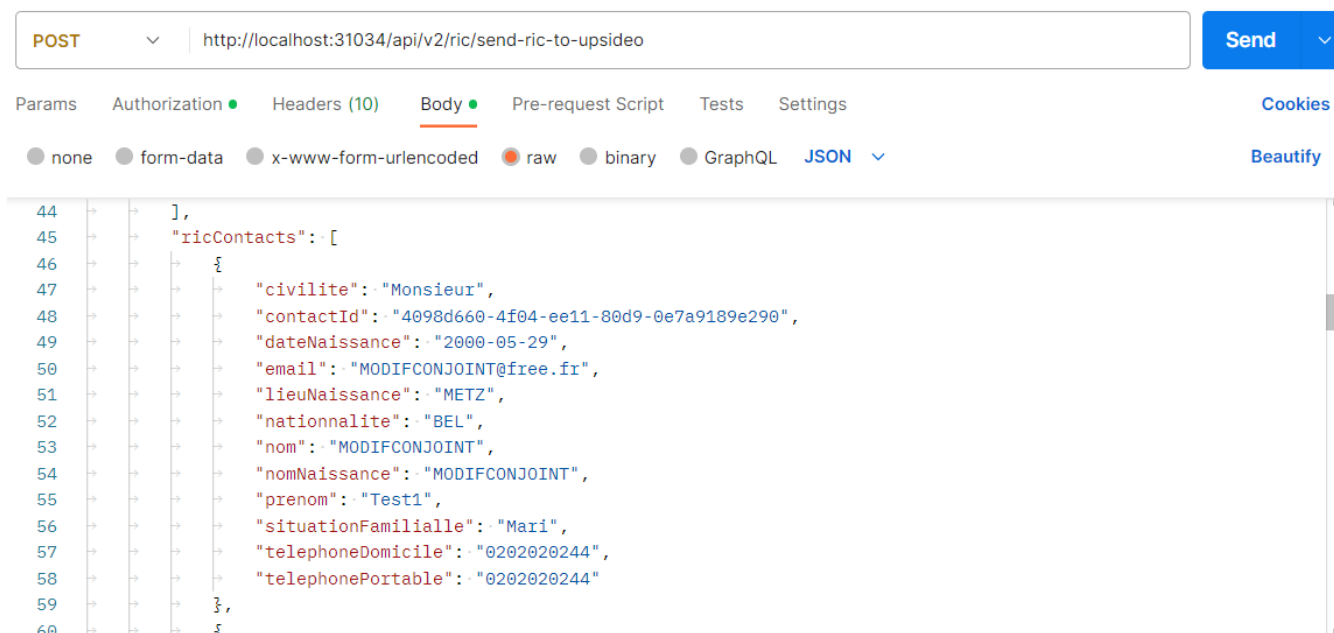
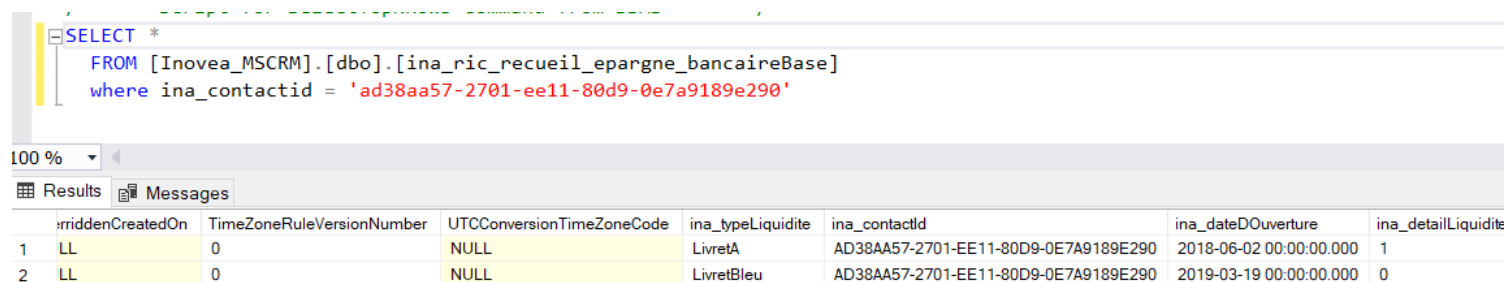


Figure 15 : Exemple d'utilisation de POSTMAN pour l'envoi

Ensuite, avec les points d'arrêt j'observe les lignes qui posent problèmes ou qui le pourraient à l'avenir. C'est particulièrement le cas avec les valeurs nulles auxquels j'ai dû rajouter plus de vérification pour que la route puisse fonctionner. Enfin, après une exécution réussie, je vérifie sur Juliette ou sur le RIC que les valeurs ont bien été transmises. Dans le cas où cela n'aurait pas marché, je test manuellement sur le swagger de Upsideo que le JSON que j'envoie soit correcte.

Pour la grande partie de ce stage, le Dto global du RIC devait être fait à la main en assemblant les Dto des différents onglets récupérés via les outils réseaux dans la partie requête. De plus, chaque client ne pouvant être testé qu'une fois, je dois en créer des nouveaux quand je veux essayer la route d'envoi.

Concernant la vérification dans la base de données, je visualise l'ensemble des tables grâce à l'outil Microsoft SQL Server Management Studio avec des requêtes SQL. Je l'utilise surtout pour la route de mise à jour où pour chaque modification qui ne semble pas fonctionner dans le site du RIC (lien avec la base non fait ou encore valeur non comprise) je regarde dans la base pour vérifier. SQL Server ne permettant pas de visualiser l'ensemble des données directement, je dois pour chaque table faire une requête simple où je prends en paramètre l'identifiant du contact afin de filtrer les résultats. Voici un exemple de requête et la table de résultat associé :



The screenshot shows the SQL Server Enterprise Manager interface. The SQL query editor at the top contains the following query:

```
SELECT *
FROM [Inovea_MSCRM].[dbo].[ina_ric_recueil_epargne_bancaireBase]
where ina_contactid = 'ad38aa57-2701-ee11-80d9-0e7a9189e290'
```

Below the query editor, the 'Results' tab is active, displaying the following data:

	HiddenCreatedOn	TimeZoneRuleVersionNumber	UTCConversionTimeZoneCode	ina_typeLiquidite	ina_contactId	ina_dateOuverture	ina_detailLiquidite
1	LL	0	NULL	LivretA	AD38AA57-2701-EE11-80D9-0E7A9189E290	2018-06-02 00:00:00.000	1
2	LL	0	NULL	LivretBleu	AD38AA57-2701-EE11-80D9-0E7A9189E290	2019-03-19 00:00:00.000	0

Il aurait été intéressant de faire des tests unitaires pour les mappers afin de vérifier que les objets renvoyés soient conformes à ce qu'il était prévu par rapport à ceux donnés en argument. Cependant, le nombre de mappers étant conséquent, en faire un ou plusieurs pour chacun ralentirait grandement le développement. J'ai donc préféré me contenter de tests fonctionnels où je peux tester l'ensemble du processus d'un coup.

4. Méthodologie et organisation du projet

J'expliquerais dans cette partie comment est organisé le travail au sein d'Inovéa. D'abord, nous verrons la gestion du service avec les différents outils utilisés. Ensuite, je détaillerai ma démarche de travail lors de ce stage. Et enfin, je présenterais quelques tâches annexes sur lesquelles j'ai pu travailler.

4.1. Gestion de l'équipe et des projets

Le service informatique d'Inovéa, étant tributaire des autres services du groupe, n'est pas sujet à des délais stricts ce qui permet une certaine liberté dans la conduite des différents projets. Cela se traduit par exemple par des demandes non rédigées et parfois un manque de cahier des charges précis.

Cependant, plusieurs éléments des méthodes agiles sont repris pour gérer l'équipe. M. SEYCHAL, qui est le DSI d'Inovéa et chef du service, organise tous les jours une réunion d'une trentaine de minutes à midi pour que chaque membre raconte ce qui a été fait, les problèmes ou incompréhensions rencontrés et les tâches à faire dans l'après-midi. J'ai ainsi pu aborder l'apprentissage critique 26.02 « Appliquer une démarche pour intégrer une équipe informatique au sein d'une organisation » et la composante essentielle « Élaborer, gérer et transmettre de l'information » ayant moi aussi participé à ces réunions pour rendre compte de mes progrès. Le télétravail étant possible dans le service, nous utilisons Teams pour communiquer à distance. Un autre élément des méthodes agiles repris est le système d'itération et d'User stories. Pour cela, l'outil Jira est utilisé pour gérer les tickets, la planification et l'état des tâches (cf. Annexe 8). Néanmoins, Jira et le système d'itération ne sont pas utilisés dans tous les projets, cela a été le cas pour le RIC où je n'ai pas interagi avec cet outil.

Pour organiser les différents travaux et gérer le versionnage, Inovéa utilise l'outil GitLab avec le client Git Tower. Le système de branches de Git est très important pour que chacun puisse travailler sur un même projet sans créer des problèmes de collision. Sur l'ensemble du projet RIC par exemple, je commitais sur la branche « devtu-ric-antoine » qui était formée à partir de la branche « devtu-ric-inovea » qui elle-même descend de la branche « devtu » (cf. Annexe 9). Tout cela est accompagné d'un processus de Merge request où Benoit VAÏSSE, un développeur back-end, va vérifier qu'il n'y a pas de problèmes dans le code avant

de le valider ou de le refuser. Une fois qu'un projet est assez avancé, il passe dans la branche « Homologation » dans laquelle il va être testé et validé par les différents product owner et testeurs. Une fois validé, le projet passe dans la branche « production » où il sera opérationnel et accessible par les autres services.

La cybersécurité est également une des problématiques essentielles chez INOVEA, les données des clients devant être protégées comme stipulé dans le RGPD. Ainsi, les employés sont régulièrement sensibilisés aux dangers tel que l'hameçonnage, les rançongiciels ou encore les bonnes pratiques à avoir via des cours en ligne distribués par Riot (cf. Annexe 10). De plus, des campagnes de tests sont lancées avec des faux mails d'hameçonnage pour préparer à des cas réels (cf. Annexe 11).

Enfin, conformément à l'apprentissage critique 26.01 « Comprendre la diversité, la structure et la dimension de l'informatique dans une organisation (ESN, DSI,...) », j'ai pu voir comment le service informatique joue un rôle transversal au sein du groupe via le développement de projets à destination à la fois des salariés, mais aussi du réseau. Le RIC est par exemple utilisé par le service crédit d'INOVEA et un autre projet de serveur discord, sur lequel j'ai travaillé, permet de regrouper l'ensemble des consultants du réseau entre eux.

4.2. Démarche personnelle

Tout au long du projet, mon maître de stage, Christophe TRIOLEYRE, m'a donné les différentes tâches et missions à réaliser. J'ai été surtout encadré, les deux premières semaines, sur plusieurs petits projets pour me montrer comment fonctionner l'API. Ensuite, j'ai commencé et passé le reste du stage à travailler sur l'intégration du RIC et de Upsideo. Pour ce qui est du RIC, j'ai eu beaucoup d'autonomie sur comment m'y prendre une fois les routes Upsideo importées, l'idée de faire des mappers à part plutôt que de le faire dans la route vient de moi notamment. J'ai dû cependant demander plusieurs fois pour des informations complémentaires sur comment fonctionne Upsideo, la base de données ou encore les valeurs renvoyées par le RIC afin de m'assurer qu'il n'y avait pas de problèmes ou pour m'aider dans mes choix de conception.

Ne connaissant pas C# initialement et ayant travaillé sur plusieurs bibliothèques externes, j'ai lu beaucoup de documentation pour savoir comment mettre en place telle solution et apprendre le fonctionnement. C'était le cas des LINQ qui sont spécifiques à C# et que j'ai appris en autodidacte et réutilisé le long du projet

pour simplifier le code. Pour des questions plus centrées sur l'API, le CRM et la base de données, je faisais appel aux autres développeurs pour me montrer le fonctionnement.

Concernant le suivi de mon stage, en lien avec l'apprentissage critique 25.04 « Définir et mettre en œuvre une démarche de suivi de projet », j'ai entretenu pendant les premières semaines un cahier de bord où je notais brièvement ce que j'avais fait dans la semaine en plus des outils utilisés (cf. Annexe 12). Ensuite, je me suis mis à noter puis rédiger sur le rapport directement le travail que j'ai effectué avec les démarches et difficultés afin de gagner du temps.

Enfin, je suis intervenu plusieurs fois pour aider les autres stagiaires du service, Louis et Antoine. Ayant travaillé sur le discord les premières semaines, j'ai proposé plusieurs idées à Louis pour qu'il puisse refactor son code et je lui ai expliqué le fonctionnement de l'API et de mes routes. J'ai par exemple proposé l'utilisation de deux dictionnaires pour les rôles discord, l'une avec les booléens et l'autre avec les identifiants correspondant dans le serveur afin de rajouter de manière plus optimisée les rôles à un utilisateur.

4.3. Projets annexes

4.3.1. CRUD Amazon

L'un des premiers projets que j'ai menés afin de me familiariser avec l'API d'Inovéa a été la création de plusieurs routes Amazon pour pouvoir interagir avec le serveur Amazon de Inovéa et faire le CRUD avec des fichiers en base64. Pour réaliser ces routes, j'ai utilisé la librairie d'Amazon AWS qui permet de créer les différentes requêtes et j'ai suivi la documentation. AWS n'ayant pas été importé sur l'API jusque-là, j'ai appris comment gérer les NuGet de Visual Code et ainsi, comment importer les différentes librairies dont j'ai besoin. Ce travail a ensuite été repris par Lucas dans le but de migrer les données de la base vers le serveur Amazon.

4.3.2. Serveur discord Unity

En collaboration avec un autre stagiaire, Louis CORNET, j'ai participé à la création d'un serveur discord nommé « Unity » pour le réseau de consultants du groupe INOVEA. Celui-ci inclut plusieurs fonctionnalités comme l'attribution initial

et la mise à jour automatique des rôles, les Webhooks pour avoir les dernières nouvelles ou encore la suppression automatique des membres ne faisant plus partie du réseau. J'avais pour mission de réaliser deux routes API, une pour récupérer et renvoyer les rôles d'un membre et une autre pour mettre à jour les rôles automatiquement après chaque modification, les deux étant ensuite appelés par le bot discord développé par Louis. J'ai dû plusieurs fois changer mon code afin de répondre à ses besoins ou problèmes ce qui est à mettre en lien avec la composante essentielle « Organiser son travail en relation avec celui de son équipe ». Pour réaliser les routes, j'ai dû faire des appels à la base de données en utilisant la classe `QueryExpression`. Cette classe a une approche objet pour concevoir des requête SQL, chaque condition, jointure et expression ayant des classes spécifiques plutôt que de mettre tout dans un string. J'ai également créé le Dto contenant les rôles avec les booléens associés afin que Louis puisse faire le traitement dans son bot. Après avoir réalisé ces routes, Louis a donc pu les utiliser pour implémenter les fonctionnalités du serveur et je suis intervenu plusieurs fois pour l'aider sur les routes API supplémentaires qu'il devait coder, notamment pour comprendre l'architecture de l'API ainsi qu'expliquer ce que j'avais implémenté. Grâce à ce travail de collaboration, j'ai pu développer l'apprentissage critique 26.03 « Mobiliser les compétences interpersonnelles pour travailler dans une équipe informatique ».

4.3.3. Connexion SSO Andil

Le dernier projet annexe sur lequel j'ai travaillé au moment du rapport a été la connexion SSO cliente à un service partenaire Andil. La connexion SSO permettant de passer d'un service à un autre de manière automatique une fois que les identifiants auront été rentrés une première fois sur la page principale. Ainsi, à partir de la clef secrète fournie par Andil et l'identifiant du client, je devais renvoyer l'URL d'accès du client. Un exemple de client PHP fonctionnel avait déjà été fourni, j'ai donc retranscrit celui-ci en C# et je l'ai placé dans une route renvoyant un Dto contenant l'URL. Comme les variables telles que la clef secrète et les différentes adresses de Andil auront des valeurs fixes, j'ai dû les déplacer dans le WebConfig dans lequel elles sont identifiées par une clef et accessible à n'importe quel point du projet.

Conclusion

Pendant ce stage, j'ai pu travailler sur l'intégration de l'API d'un partenaire Upsideo à notre solution RIC. J'ai importé les différentes routes de Upsideo, mis en place les mappers pour passer d'une API à une autre et enfin, j'ai créé les deux routes API pour l'envoi à Upsideo et la mise à jour du RIC.

Pour l'importation des routes Upsideo, j'ai repris la documentation de leur API et crée chaque objet que j'allais devoir intégrer. Grâce à deux méthodes génériques pour envoyer les requêtes GET, PUT et POST, j'ai ensuite importé toutes les routes de Upsideo et refactorisé celles qui avaient déjà été faites.

Après l'importation, j'ai codé l'ensemble des mappers nécessaires pour transmettre les informations entre les deux API. Deux documents Excel ont été créés en parallèle afin de voir les correspondances ainsi que les anomalies qui perturbent l'intégration.

Finalement, les deux routes d'intégration ont été codées en assemblant tout ce qui avait été fait précédemment en termes de Dto RIC, de routes Upsideo et de mappers. L'envoi à Upsideo sera appelé lorsque l'utilisateur appuiera sur le bouton correspondant à la suite du remplissage du recueil tandis que la mise à jour du RIC sera appelée automatiquement par Upsideo.

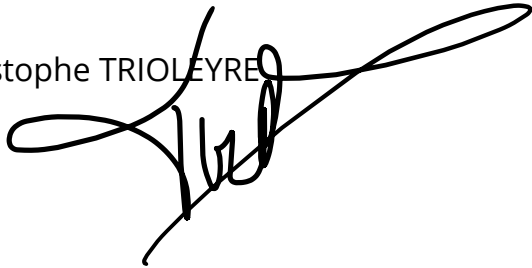
Tout le long du stage, j'ai également veillé à documenter mon code en indiquant l'objet Upsideo qui était traité et en organisant par onglet RIC pour plus de clarté.

Au moment où j'écris ce rapport, l'intégration est toujours en phase de développement et de test avec certaines valeurs qui nécessitent une clarification avec les techniciens de Upsideo. Néanmoins, le travail que j'ai fait fonctionne et permettra de mettre en production le RIC une fois que le front aura été terminé. Ce projet et ce stage de manière générale m'auront permis d'approfondir mes connaissances sur le fonctionnement des API et comment mettre en place des applications communicantes que ce soit en utilisant le format JSON ou en mettant en place des requêtes http. J'ai également pu apprendre le langage C#, que j'avais déjà abordé quelques fois auparavant, en abordant les méthodes asynchrones ainsi que les LINQ qui permettent de simplifier et de rendre plus lisible le code. J'ai également utilisé mes capacités d'analyse pour trouver les meilleures solutions

aux problèmes, c'était le cas par exemple pour le Discord où j'ai conseillé à Louis d'utiliser un dictionnaire pour optimiser son code. Enfin, j'ai mis en pratique de bonnes pratiques de développement, notamment à travers la lecture de documentation sur lequel j'ai passé beaucoup de temps pour mettre en place mes routes discord et AWS.

Visa du maitre de stage :

Christophe TRIOLLEYRE

A handwritten signature in black ink, appearing to be 'CTRIOLLEYRE', written over the printed name 'Christophe TRIOLLEYRE'.

Vu le 14/06/2023

Annexes

Annexe 1 : Visuel du RIC

FILIANSE
la banque de l'assurance

RIC de FRAICH Joe

Etat Civil

Coordonnées

Situation familiale

Composition du foyer

Situation professionnelle

Personne exposée

US Person

Etablissement bancaire

Informations personnelles

REVENUS ET FISCALITÉ

IMMOBILIER

EPARGNE

AUTRES BIENS

CRÉDIT(S)

Informations personnelles

État Civil

Date d'entrée en relation

Origine de la relation

jj/mm/aaaa

Mode d'entrée en relation

FRAICH Joe

☐ Madame

☒ Monsieur

Nom

Prénom

Suivant

Annexe 2 : Visuel de Upsideo

INOVÉA
la banque de l'assurance

FABRIEN_TEST PARENT_TEST

SUIVI

PROSPECTS

CLIENTS

BIBLIOTHÈQUE

SIMULATIONS

PARCOURS / RECUEIL / IDENTITÉ

Recueil

LCB/FT

Mission

Objectifs

Identité

Revenus & patrimoine

Crédit

Charges diverses

Préférences ESG

Profil de risque

Signature

IDENTITÉ

VIE PROFESSIONNELLE

SITUATION FAMILIALE

COMPOSITION DU FOYER

PEP

SOCIÉTÉ COTÉE

RENOMMÉE PUBLIQUE

RISQUES ACCRUS

FATCA

☐ Nouveau contact

Client depuis le

11/01/2024

Origine de la relation

Relation d'affaires

Mode d'entrée en relation

A distance

1

Annexe 3 : Cahier des charges du RIC

Cahier des charges - RIC
Analyse de la maquette et des routes Upsideo
Tableau de mapping : Réaliser un tableau sur excel avec la correspondance entre l'onglet RIC et la route Upsideo qui faudra appeler pour envoyer les informations.
Importation des objets Upsideo : Créer les objets Upsideo dans l'API RIC contenant les mêmes attributs que sur les routes afin d'envoyer/modifier.
Importation des routes Upsideo : Pour chaque objet Upsideo, faire les routes GET, PUT et POST correspondantes dans l'API.
Envoi des données vers Upsideo
Mappers de RIC vers Upsideo : Pour chaque objet Upsideo, créer un mapper en utilisant le ou les objets RIC correspondants et créer le Dto Upsideo avec les valeurs du RIC.
Route d'envoi de données : Réaliser la route API afin d'appeler les routes Upsideo de création avec les Dto Upsideo créés par les mapping.
Mise à jour des données du RIC
Mappers de Upsideo vers RIC : Pour chaque Dto Upsideo, créer un mapper pour modifier les valeurs du Dto RIC correspondant.
Route de Mise à jour : Réaliser la route API afin de récupérer les données du RIC et de Upsideo puis de mettre à jour la base de données de INOVÉA.

Annexe 4 : Document d'analyse RIC/Upsideo

Onglet RIC	API Upsideo	Routes
Informations Personnels		
Etat Civil	Organisation	Client
	RecueilIdentite	IdentitePersonnePhysique
	Parcours	Recueil
Coordonnées	Organisation	Client
	RecueilIdentite	IdentitePersonnePhysique
Situation familiale	RecueilIdentite	SituationFamiliale
Composition du foyer	RecueilIdentite	PersonneACharge
		Liberalite
		RepresentantLegal
Situation professionnelle	RecueilIdentite	VieProfessionnelle
Personne exposée	RecueilIdentite	MandatPolitique
		PEP
		RenommeePublique
		RisqueAccru
		SocieteCotee
US Person	RecueilIdentite	FATCA
Etablissement bancaire		Pas de correspondance
REVENUS & FISCALITE		
Revenus	RecueilRevenusEtPatrimoine	Revenu
		RevenuDetail
Impôt sur le revenu	ReceuilCharges	Charges
	RecueilIdentite	Fiscalite
IFI	RecueilIdentite	Fiscalite
Taxes	ReceuilCharges	Impot
IMMOBILIER		
Immobilier d'usage	ReceuilRevenusEtPatrimoine	ImmobilierDeJouissance
		PatrimoineImmobilier
Immobilier locatif	ReceuilRevenusEtPatrimoine	ImmobilierDeRapport
		PatrimoineImmobilier
SCI/SCPI	ReceuilRevenusEtPatrimoine	ImmobilierDeRapport
		PatrimoineImmobilier
EPARGNE		
Epargne bancaire	ReceuilRevenusEtPatrimoine	Liquidites
		PatrimoineFinancier
Assurance-vie	ReceuilRevenusEtPatrimoine	AssuranceVie
		PatrimoineFinancier

Epargne retraite / salariale	ReceuilRevenusEtPatrimoine	PlacementATerme
		PatrimoineFinancier
Investissements financiers	ReceuilRevenusEtPatrimoine	ValeurMobiliere
		PatrimoineFinancier
Prévoyance		
AUTRES BIENS		
Biens professionnels	ReceuilRevenusEtPatrimoine	BienProfessionnel
Biens meubles	ReceuilRevenusEtPatrimoine	Patrimoine Divers
CRÉDITS		
Crédit	ReceuilCharges	Credit
		CreditDetail

Annexe 5 : Swagger de Upsideo

PatrimoineDivers			^
GET	/api/PatrimoineDivers	list of patrimoineDivers, optionally with selection of fields and/or filtering	✓ 🔒
POST	/api/PatrimoineDivers	create a patrimoineDivers	✓ 🔒
PUT	/api/PatrimoineDivers	update a patrimoineDivers	✓ 🔒
GET	/api/PatrimoineDivers/{id}	find a patrimoineDivers by id	✓ 🔒
DELETE	/api/PatrimoineDivers/{id}	delete a patrimoineDivers	✓ 🔒
GET	/api/PatrimoineDivers/{id}/Notes	get Notes	✓ 🔒
GET	/api/PatrimoineDivers/RecueilId/{recueilId}	get patrimoineDivers by RecueilId	✓ 🔒

Annexe 6 : méthode générique pour les notes

```
private string getGeneriqueNotes(string id, string path, string typeApi, string precision)
{
    try
    {
        return JsonConvert.DeserializeObject<string>(GetUpsideo(id, path, typeApi, precision));
    }
    catch (WebException)
    {
        return "";
    }
}
```

Annexe 7 : Dto de mise à jour RIC

```
public class V3_Ric_update_dto
{
    public bool recueil { get; set; } = false;
    public bool identite { get; set; } = false;
    public bool vieProfessionnelle { get; set; } = false;
    public bool situationFamiliale { get; set; } = false;
    public bool personnesACharge { get; set; } = false;
    public bool liberalite { get; set; } = false;
    public bool pep { get; set; } = false;
    public bool renommeePublique { get; set; } = false;
    public bool risquesAccrus { get; set; } = false;
    public bool fatca { get; set; } = false;

    public bool fiscalite { get; set; } = false;
    public bool revenus { get; set; } = false;
    public bool charges { get; set; } = false;

    public bool immobilierDeRapport { get; set; } = false;
    public bool immobilierDUsage { get; set; } = false;

    public bool assuranceVie { get; set; } = false;
    public bool liquidite { get; set; } = false;
    public bool placementATerme { get; set; } = false;
    public bool valeurMobiliere { get; set; } = false;

    public bool biensProfessionnelles { get; set; } = false;
    public bool biensMeubles { get; set; } = false;

    public bool credit { get; set; } = false;
}
```

Annexe 8 : Jira du projet Discord

Jira Software Your work Projects Filters Dashboards Teams Apps Create

Discord-bot Software project

PLANNING Roadmap

Board

DEVELOPMENT Code

Project pages

Story Maps

You're in a team-managed project Learn more

Projects / Discord-bot

Tableau DB

Search AA Label

GROUP BY None Insights

TO DO 3 ISSUES

Créer un utilisateur Discord

DB-1

Envoi de mail a utilisateur

DB-9

FAQ

DB-10

+ Create issue

IN PROGRESS 2 ISSUES

Mettre en place les rôles et permissions discord.

DB-8

Supprimer un utilisateur

Automatiquement

DB-2

DONE 1 ISSUE

Supprimer un utilisateur

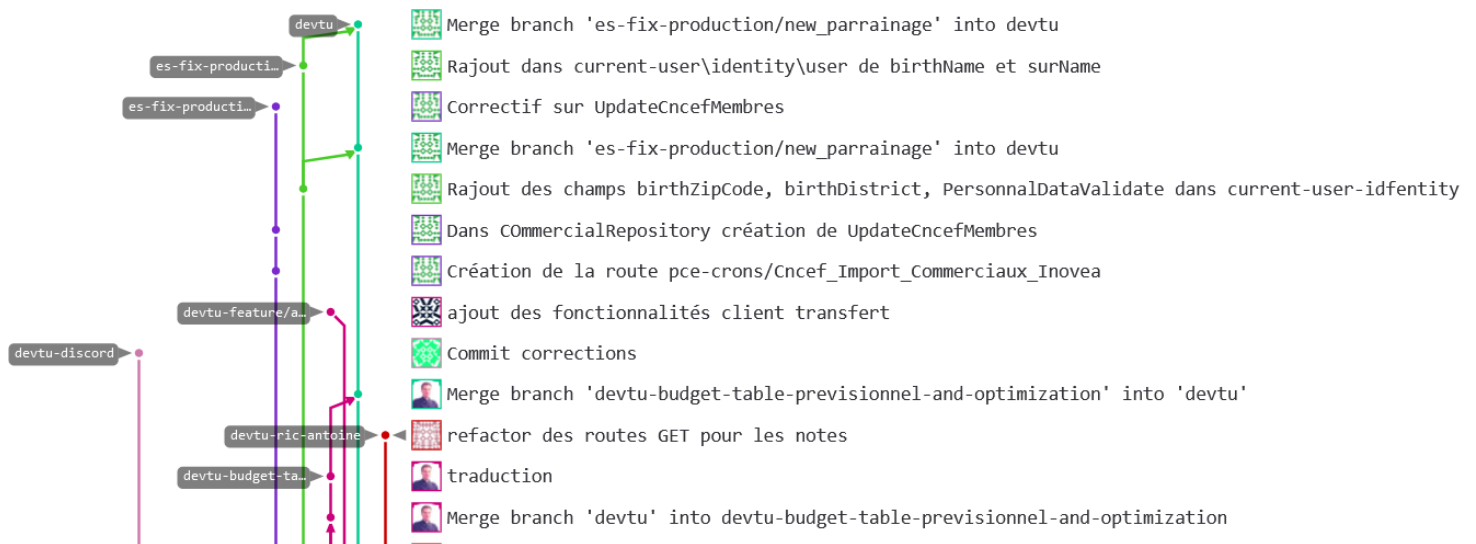
Manuellement

DB-11

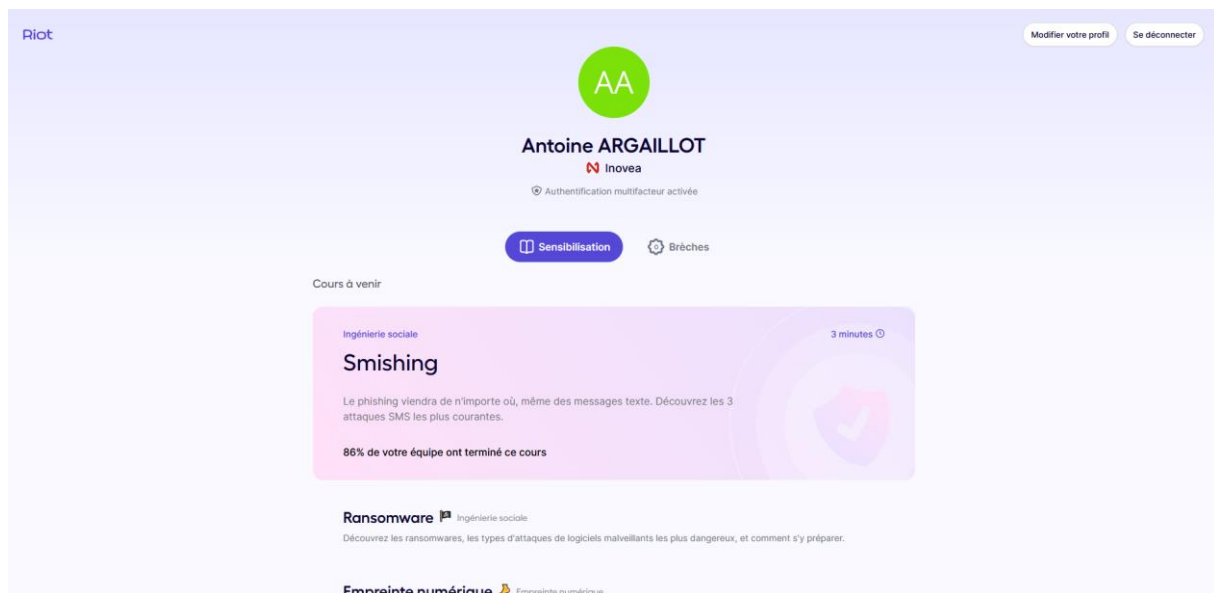
See all Done issues

Quickstart

Annexe 9 : Graphe du Git



Annexe 10 : Cours de cybersécurité Riot



Annexe 11 : Exemple de test pour la cybersécurité

De : hey@go.tryriot.com <hey@go.tryriot.com> de la part de Riot <hey@go.tryriot.com>

Envoyé : mardi 6 juin 2023 10:06

À : Christophe TRIOLEYRE <c.trioleyre@inovea-group.com>

Objet : Antoine ARGAILLOT a échoué à une attaque phishing

Bonjour Christophe TRIOLEYRE,

Antoine ARGAILLOT a échoué à une attaque phishing pour la 1re fois. L'attaque AA-2 a été envoyée par [Campagne trimestrielle pour tous les employés](#).

[Voir l'attaque](#)

Vous pouvez désactiver ces notifications dans [votre profil utilisateur](#).

Annexe 12 : Extrait du cahier de bord du stage

Semaine 2 :

Route discord :

- Changement de la route pour prendre en compte le profil discord
 - Si l'id discord n'est pas dans la base, un update est exécuté avec l'id de la requête
- > Création d'une requête bd sécurisée dans c# avec jointure

Route discord MAJ:

- idem, mail remplacé par profil et id
- début de la route pour le faire automatiquement après chaque mise à jour de jour

AWS :

- pb de package dans l'environnement argo cd

RIC:

- création des routes GET/POST/PUT dans le repository pour chaque élément de Upsideo demandé (création dto correspondant puis rajout de la méthode de requête)
- méthode pour récupérer les documents en fonction de l'id client et du type

Outils :

- Microsoft teams -> partage de fichier et réunion
- CRM pour modifier des informations, créer des utilisateurs et pouvoir trouver des cas de tests
- VPN pour travail à distance (mercredi et jeudi)

Bibliographie

- [1]
B. Wagner, « Basic LINQ Query Operations (C#) », 15 septembre 2021. [Online].
Disponible sur: <https://learn.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/LINQ/basic-LINQ-query-operations>. [Consulté le: 26 mai 2023]
- [2]
J. Douglas, « What is NuGet and what does it do? », 11 octobre 2022. [Online].
Disponible sur: <https://learn.microsoft.com/en-us/nuget/what-is-nuget>. [Consulté le: 28 mai 2023]
- [3]
S. Samadder, « Shallow Copy and Deep Copy in C# - GeeksforGeeks ». [Online].
Disponible sur: <https://www.geeksforgeeks.org/shallow-copy-and-deep-copy-in-c-sharp/>. [Consulté le: 22 mai 2023]
- [4]
« Swagger UI ». [Online]. Disponible sur:
<https://api.preprod.upsideo.fr/swagger/index.html>. [Consulté le: 9 mai 2023]
- [5]
« Discord Developer Portal — API Docs for Bots and Developers », *Discord Developer Portal*. [Online]. Disponible sur:
<https://discord.com/developers/docs/resources/application-role-connection-metadata#update-application-role-connection-metadata-records>. [Consulté le: 9 mai 2023]
- [6]
« Qu'est-ce que AWS kit SDK Mobile pour .NET et Xamarin ? - Kit SDK AWS Mobile ». [Online]. Disponible sur:
https://docs.aws.amazon.com/fr_fr/mobile/sdkforxamarin/developerguide/Welcome.html. [Consulté le: 3 mai 2023]
- [7]
« Qu'est-ce qu'un CRM ? | Oracle France ». [Online]. Disponible sur:
<https://www.oracle.com/fr/cx/what-is-crm/>. [Consulté le: 28 mai 2023]
- [8]
« Qu'est-ce qu'une API ? - api.gouv.fr ». [Online]. Disponible sur:
<https://api.gouv.fr/guides/api-definition>. [Consulté le: 28 mai 2023]