IUT Nantes
Pôle Sciences et technologie

BUT Informatique | 2023-2024

Rapport de SAE 4.01 - Automates et Langages

eq_02_00_blanchet-maxence_delval-esteban_lambert-alexis_rageau-nils_bruneau-rialland-baptiste

Membres :

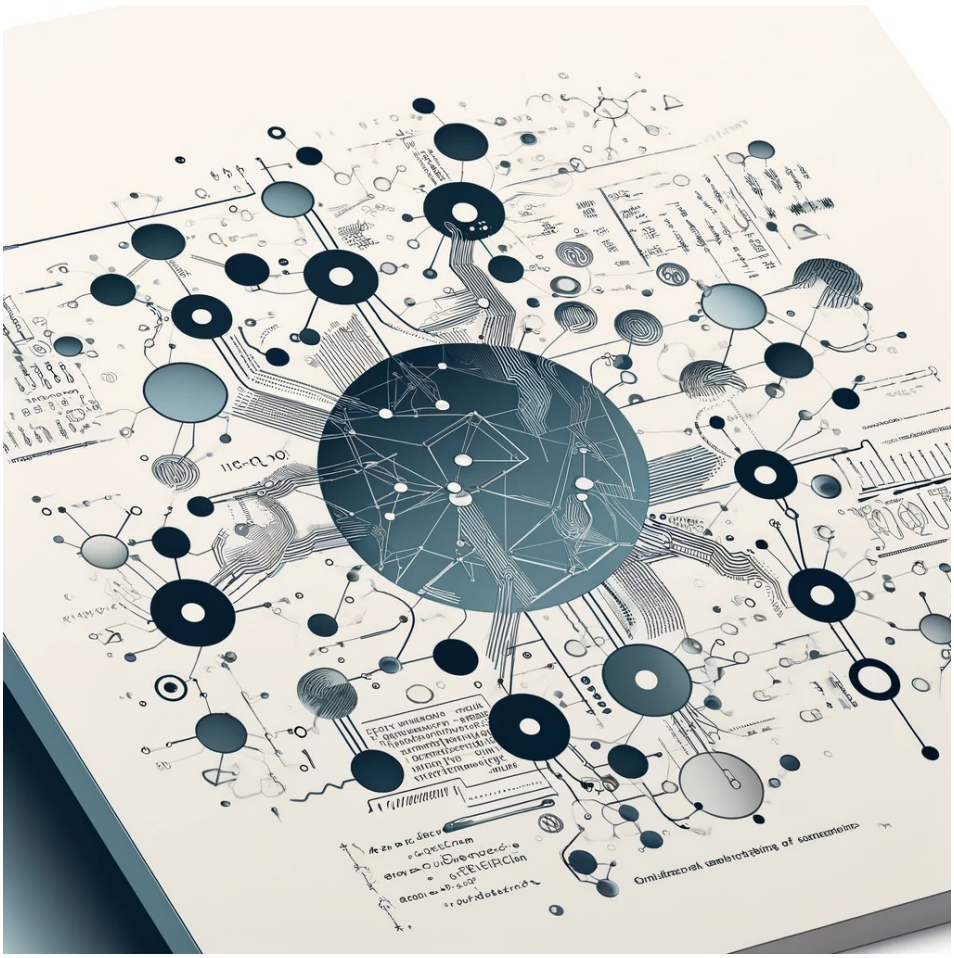
Lambert Alexis

Maxence Blanchet

Esteban Delval

Baptiste Bruneau Rialland

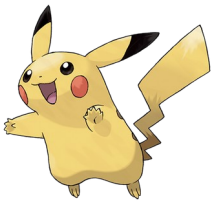
Nils Rageau



Introduction

Pour terminer notre 2ème année d'IUT, nous avons eu à réaliser pour notre projet de **SAE**, une application complexe. Pour ce faire nous avons développer des **API RESTful** en *NodeJS* sous forme de micro-services, un client web en *REACT* ainsi qu'une application mobile Android (nous nous focaliserons sur les API et le client web dans ce rapport).

Le thème de l'application n'était pas imposé. Nous avons donc choisis le thème des Pokémon, voici une petite explication en quoi elle consiste :



« L'univers des Pokémon est une franchise multimédia qui a débuté au Japon en 1996, centrée sur des créatures fictives appelées "Pokémon". Ces créatures vivent dans le monde Pokémon aux côtés des humains. Les humains, souvent appelés dresseurs, capturent et entraînent les Pokémon pour les faire combattre, généralement dans un but sportif ou compétitif. Chaque Pokémon a un ou plusieurs types (comme l'eau, le feu, ou la plante) qui déterminent ses forces et faiblesses face à d'autres Pokémon.

Les informations essentielles sur les Pokémon incluent leur nom, type, évolution, et un ensemble de capacités qu'ils peuvent utiliser en combat. Les Pokémon évoluent, c'est-à-dire qu'ils peuvent se transformer en une version plus forte d'eux-mêmes, souvent en atteignant un certain niveau ou en répondant à des conditions spécifiques.

L'univers Pokémon s'étend à travers des jeux vidéo, des séries télévisées, des films, des cartes à collectionner, et plus encore, permettant aux fans d'explorer cet univers de manières variées. Les jeux vidéo sont le cœur de la franchise, offrant une expérience immersive où les joueurs peuvent capturer, élever, et combattre avec des Pokémon dans le but de devenir le meilleur dresseur.

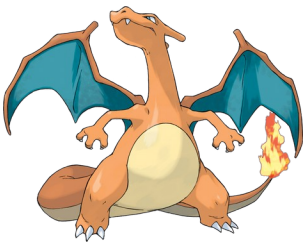
Au fil des années, l'univers Pokémon s'est enrichi de nouvelles régions inspirées de lieux réels, chacune avec ses propres espèces de Pokémon, ajoutant à la diversité et à la richesse de ce monde. L'interaction entre les dresseurs et leurs Pokémon est un thème central, soulignant l'importance de l'amitié, du respect, et du travail d'équipe.

En somme, l'univers des Pokémon est un monde vaste et captivant, où la collecte, l'élevage, et les combats de Pokémon forment la base d'aventures sans fin pour les dresseurs aspirants de tout âge. »

Source : explication brève de l'univers des Pokémon pour un adulte ne connaissant pas l'univers – ChatGPT 4

Notre application s'appelle « PokéManager », elle permet de consulter la liste des Pokémon, des objets, des capacités que peuvent apprendre les Pokémon. Le point clé de l'application se situe dans la partie interaction utilisateur, car ceux-ci peuvent gérer (ajouter, modifier et supprimer) des équipes qu'ils peuvent constituer à travers un espace profil qui leur est dédié. Une équipe est constituée de 6 Pokémon maximum et chaque Pokémon peut être personnalisé (attaques, objet qu'il porte, chromatique* ou non).

* Un Pokémon chromatique possède des couleurs différentes comparé à l'original. Il n'existe qu'une version d'un Pokémon chromatique.



Réalisation

Pour mettre en œuvre l'ensemble des fonctionnalités attendues, nous avons dû passer par une phase d'analyse complexe concernant l'implémentation. Le comportement d'un service est représenté par un automate, qui permet de facilement observer les différentes difficultés potentielles et le déroulement des actions. Dans ce rapport, nous ne traiterons qu'un nombre limité de fonctionnalité de chaque service développé.

Fonctionnalités

- f_1 Obtention de la liste des Pokémon
- f_2 Obtention de la liste des Capacités que peut apprendre un certain Pokémon
- f_3 Création d'un compte utilisateur
- f_4 Ajout d'une équipe de Pokémon

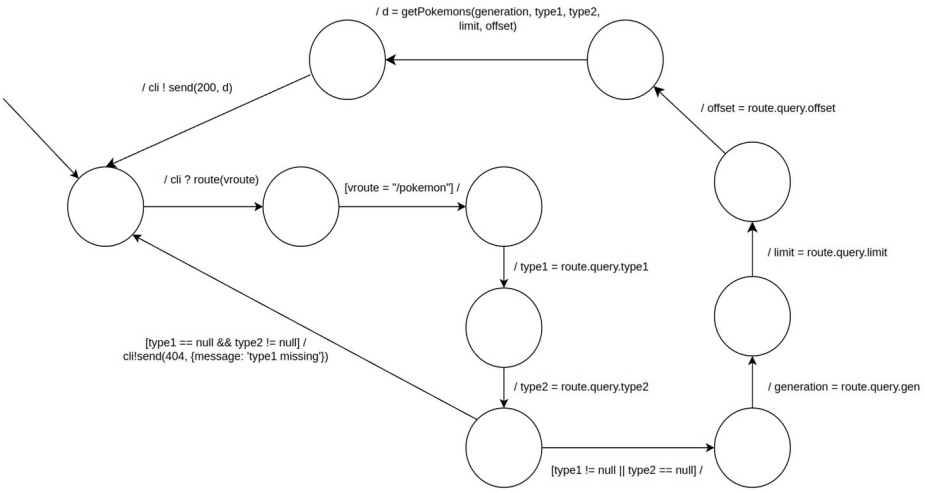
Dans les pages qui vont suivre, nous passerons en revue chaque fonctionnalité sous la forme d'automates. Un automate pour le fonctionnement serveur et un pour le client, accompagné d'une brève explication.

Note :

Lorsque l'on fera référence à la variable 'limit', cela représentera le nombre de document qu'une requête est censé renvoyer.

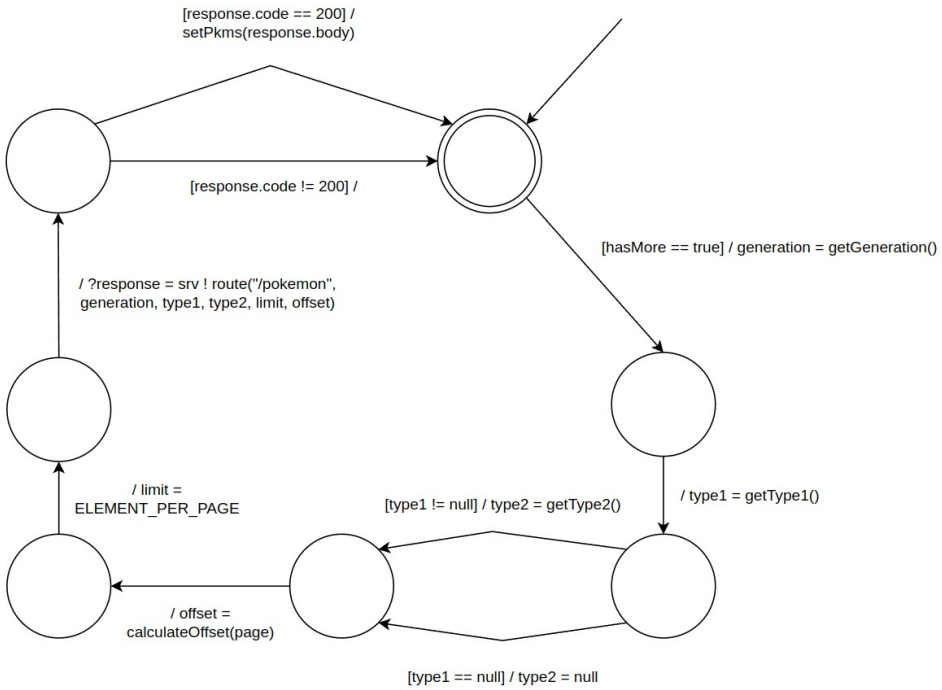
Lorsque l'on fera référence à la variable 'offset', cela représentera une exclusion des N premiers documents du résultat d'une requête.

f_1 : Obtention de la liste des Pokémon



Automate 1: Côté Serveur

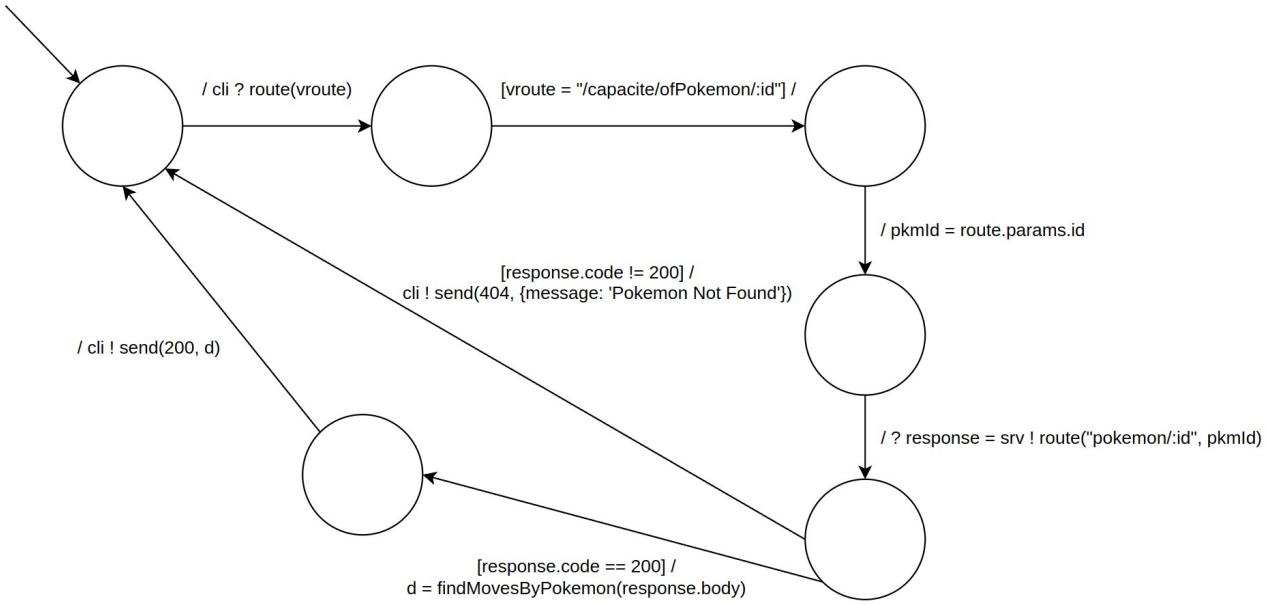
L'automate vérifie si la route demandée correspond à « /pokemon ». Si c'est le cas, il procède à l'extraction des paramètres de la requête. Il extrait type1, type2, generation, limit et offset. Si type1 est null et que type2 est non null, l'automate renvoie une réponse avec un statut 404 et le message "type1 missing". Sinon, une fois les opérations réussies et les données récupérées par la fonction getPokemons, une réponse avec un statut 200 et les données (d) est envoyée au client.



Automate 2: Côté Client

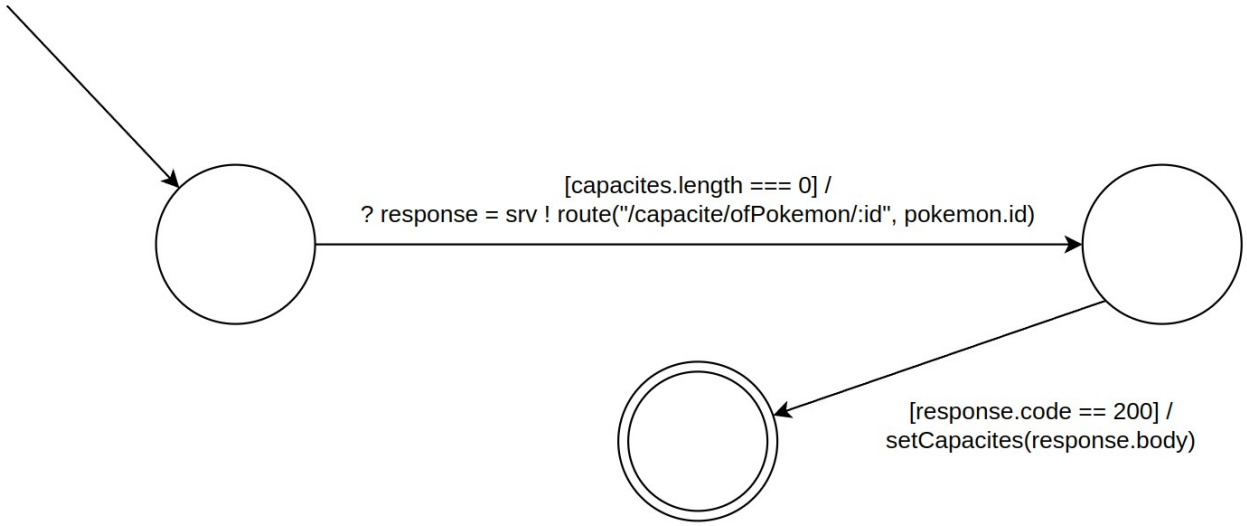
Le client a la possibilité de choisir plusieurs critères pour obtenir des Pokémon. Ici, l'automate récupère ces différents critères, et envoie une requête au serveur. L'état d'après traite la réponse, en vérifiant qu'elle est OK, si c'est le cas, on affiche les Pokémon sur l'interface utilisateur. Sinon, on ne fait rien (on considérera que la requête est erronée et qu'elle n'est pas réalisable à l'aide de l'interface). Une booléen pour charger plus de Pokémon est présente (hasMore). On récupère une nouvelle génération via getGeneration() et potentiellement des types via getType1() ou getType2(). Cela indique que l'utilisateur a la possibilité de changer les critères de recherche pour la génération ou les types avant de charger plus de données.

f_2 : Obtention de la liste des Capacités que peut apprendre un certain Pokémon



Automate 3: Côté Serveur

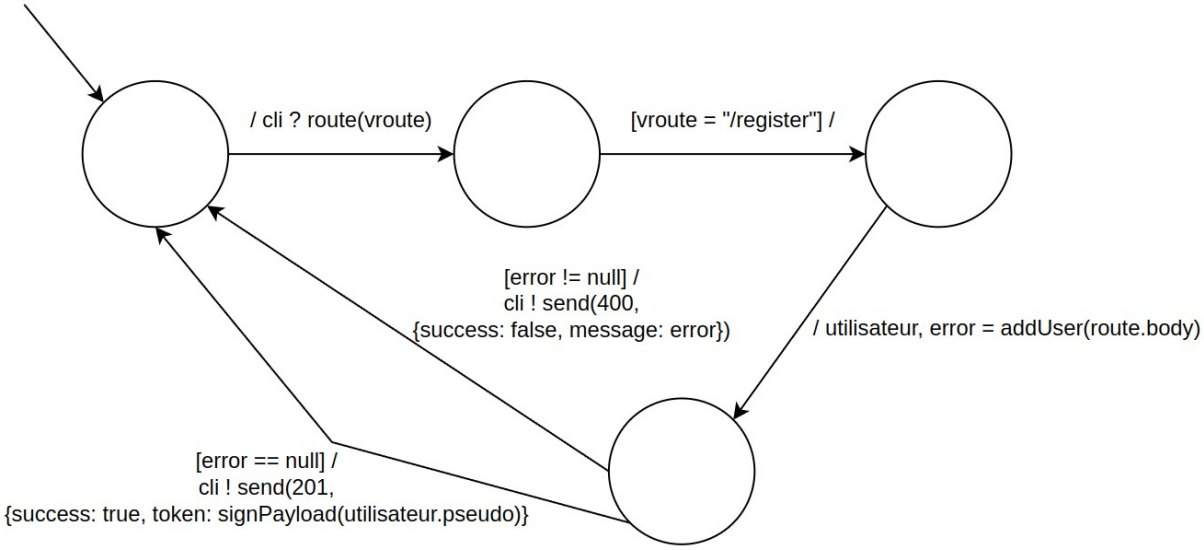
L'automate va d'abord vérifier si la route est "/capacite/ofPokemon/:id". Ensuite, il extrait le paramètre « id » de la route. Pour obtenir les capacités d'un Pokémon, il faut d'abord récupérer le Pokémon en question, pour ce faire une autre requête est adressé à "/pokemon/:id". Si le code de réponse est 404, le Pokémon n'a pas été trouvé. Sinon, on renvoie la liste des Pokémon pouvant apprendre la capacité.



Automate 4: Côté Client

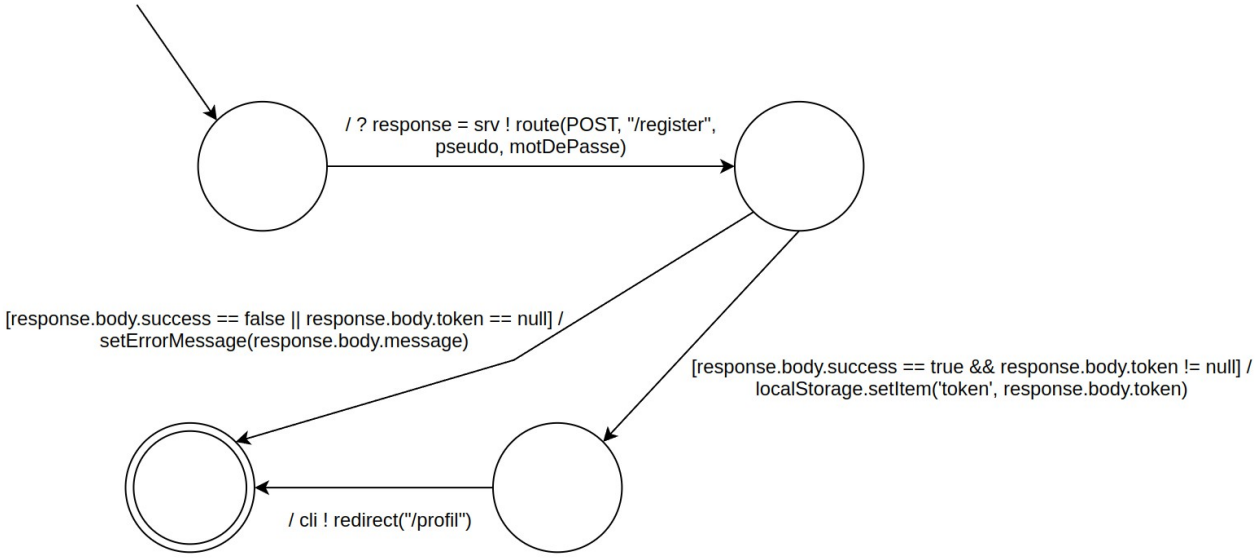
Le client peut ouvrir les détails d'un Pokémon, lorsqu'il le fait, il faut récupérer ses capacités. Pour ce faire, on va aller chercher sur la route "/capacite/ofPokemon/:id" les capacités du Pokémon en question. « capacites.length » correspond à la longueur de la liste contenant les capacités. Au départ, elle est vide. Une fois les capacités récupérées, elle se remplit. Cette condition assure qu'on ne récupère les données qu'une seule fois lors de l'ouverture des détails. « setCapacites » est une méthode permettant d'affecter les données.

f_3 : Création d'un compte utilisateur



Automate 5: Côté Serveur

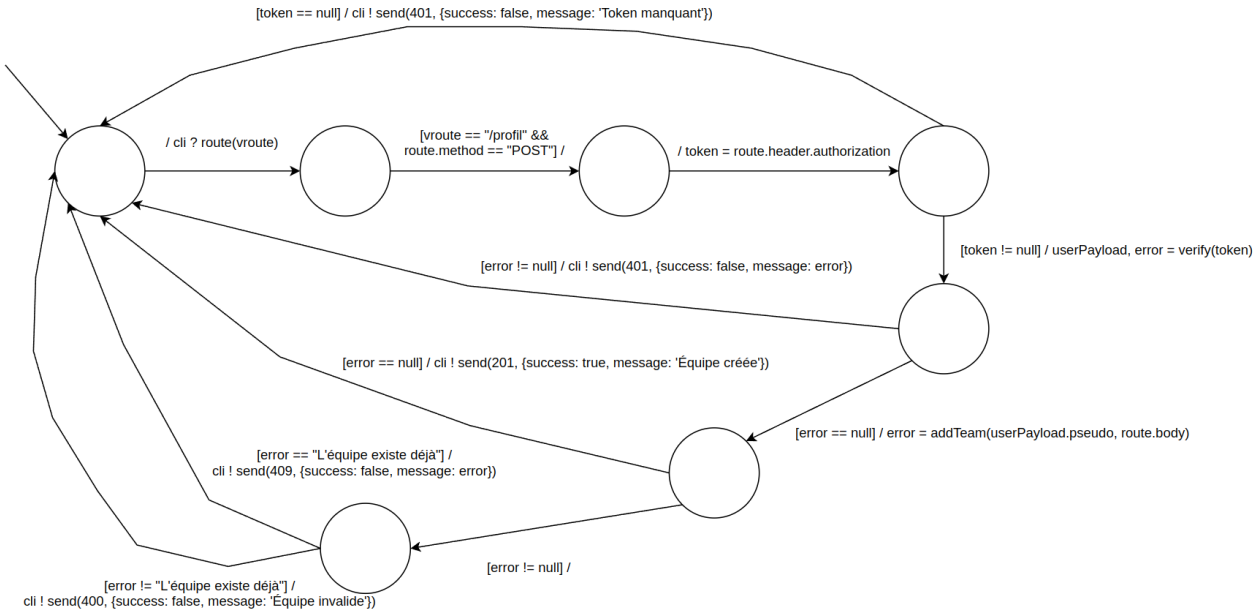
L'automate représente le processus de création d'un compte. On récupère les données de l'utilisateur depuis le corps de la requête. Si la création c'est bien passé, l'utilisateur reçois un jeton d'authentification, sinon, il reçoit un message d'erreur. L'attribut 'success' est là pour indiquer si la création a réussis. La méthode 'signPayload' permet de signer un pseudo pour en obtenir un jeton, cela encode le pseudo avec une clé secrète présente dans les variables d'environnement du serveur.



Automate 6: Côté Client

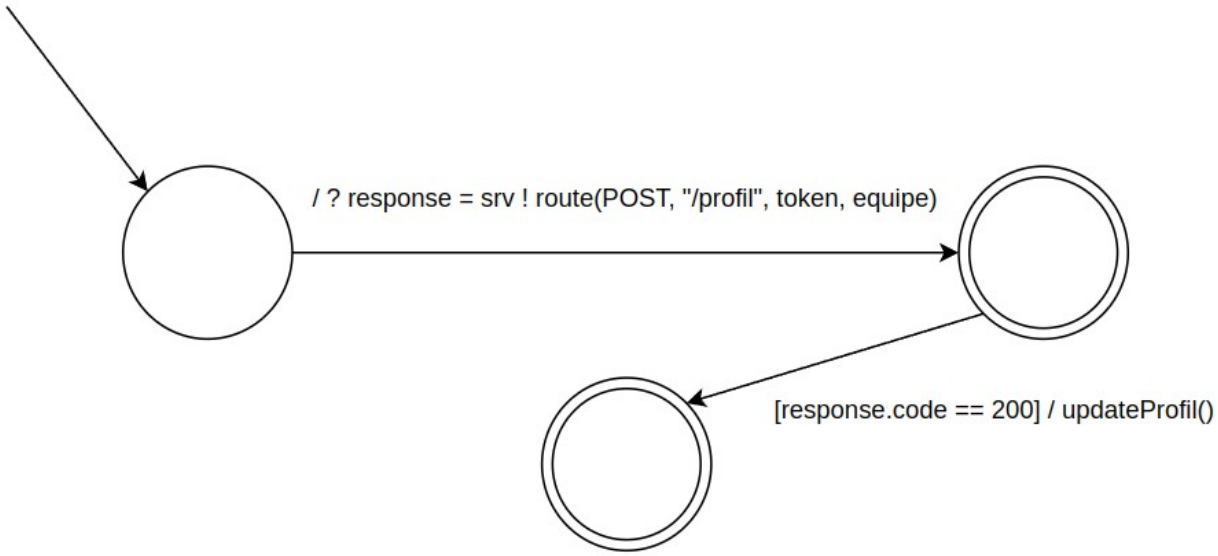
Le client envoie ses données dans le corps de la requête, si la réponse est un succès, l'utilisateur obtiens un jeton, qu'il stock dans le stockage web local à la clé 'token', il est ainsi redirigé vers la page de profil. Sinon, un message d'erreur s'affiche sur la page (affectée grâce à la méthode 'setErrorMessage')

f_4 : Ajout d'une équipe de Pokémon



Automate 7: Côté Serveur

Pour ajouter une équipe à un utilisateur, l'automate vérifie si le token est présent dans le header de la requête. S'il est présent, le token est vérifié. S'il est valide, alors on ajoute l'équipe (le corps de la requête route.body) grâce à la méthode 'addTeam'. La possibilité d'une erreur est traité.



Automate 8: Côté Client

Le client envoie l'équipe ('equipe') qu'il souhaite ajouter dans le corps de la requête. Il passe également le jeton d'authentification 'token'. Si la réponse est un succès, l'interface est mise à jour.

Si la réponse n'est pas valide, alors l'interface n'est pas mise à jour. Par manque de temps, nous n'avons pas pu implémenter une possibilité d'affichage d'erreur intelligent.

Modélisation

La création de ces automates a permis une formalisation rigoureuse des spécifications de notre projet en décrivant précisément les divers états du système ainsi que les transitions possibles. Ils ont également été utiles pour détecter des anomalies ou des contradictions potentielles, ce qui fait d'eux des outils précieux dans la phase de conception. Toutefois, contraints par le manque de temps, nous n'avons pas pu exploiter pleinement leur potentiel. Dans le contexte de délais serrés et d'un volume conséquent de travail, nous avons principalement utilisé les automates pour confirmer la validité de notre modèle et pour identifier d'éventuelles erreurs.

Conclusion

En conclusion, notre projet PokéManager a démontré l'importance des automates à états dans le développement de micro-services. En tant que modèle, ces automates ont permis de conceptualiser le fonctionnement de plusieurs processus de notre application, créant ainsi les fondements pour analyser les comportements attendus du système et déterminer les scénarios de test précis (même s'ils n'ont servi que de validateur comme expliqué précédemment). Comme mentionné par l'article de Red Hat sur les micro-services, « aucune connaissance n'est implicite lorsqu'il s'agit d'un service, les développeurs doivent avoir une base explicite et exhaustive depuis laquelle travailler »

Même si nous étions limités par le temps, les automates ont contribué à valider notre architecture et à nous assurer une documentation adéquate pour les prochaines itérations ou transitions. En fin de compte, cela a renforcé nos compétences de conception logicielle et notre sens de la modélisation pour garantir un développement continu de logiciels.

Les compétences que nous avons pu acquérir grâce à ce processus vont aussi loin que d'écrire simplement le code, elles incluent aussi le raisonnement critique qui est nécessaire pour créer une application robuste et entretenue. Notre facilité et capacité à formaliser des processus qui peuvent être compliqués et à prévoir le comportement des systèmes dans des circonstances différentes sont tous renforcés de manière significative. De plus, notre projet a servi de test pratique pour déterminer comment les principes fondamentaux des automates d'états finis se traduisent dans la construction d'applications modernes, car il est aligné sur les perspectives de l'industrie telles qu'elles sont présentées dans les articles de référence sur la modélisation des sagas et des flux métier.

Références :

- https://developers.redhat.com/articles/2021/11/23/how-design-state-machines-microservices#what_is_a_state_machine_
- https://levelup.gitconnected.com/modelling-saga-as-a-state-machine-cec381acc3ef
- https://www.quora.com/Can-we-create-Microservices-as-State-Machines
- Explication en détail Automate à état et contexte micro-services – ChatGPT 4