Ecole Publique d'ingénieures et d'ingénieurs en 3 ans

Rapport – Projet Industriel

BACK-OFFICE MONETIQUE

Publication le 5 5 février 2024,

Version 1

GUO David - @email
PICQUE Kylian - @email
SENG Thomas - @email
STEIMETZ Tangui - @email

Tuteur école : Joan REYNAUD
Tuteur entreprise : Ndiaga FAYE

ENSI CAEN
ÉCOLE PUBLIQUE D'INGÉNIEURS
CENTRE DE RECHERCHE

www.ensicaen.fr

TABLE DES MATIERES

TABLE DES FIGURES

Remerciements

Nous souhaitons exprimer notre profonde gratitude envers nos enseignants Monsieur Joan REYNAUD et Monsieur Ndiaga FAYE pour leur soutien tout au long de ce projet.

Nous tenons à remercier sincèrement Monsieur Joan REYNAUD pour sa disponibilité, son engagement continu ainsi que sa patience qui ont aussi contribué à l'avancement du projet. Son expertise et ses conseils ont été essentiels pour orienter notre effort et garantir le succès de notre travail.

Nous sommes également reconnaissants envers Monsieur Ndiaga FAYE pour son apport de connaissances techniques dans le domaine de la monétique, ainsi que pour ses conseils en termes d'implémentation. Ses indications auront été une aide précieuse dans la réalisation de notre projet.

BACK-OFFICE MONETIQUE - ANALYSE

1. Présentation générale du projet et du contexte.

1.1. Contexte.

Dans le cadre de la formation initiale en informatique de l'ENSICAEN, le 25 novembre 2023, les élèves de 3ème année ont dû choisir un projet industriel à réaliser. Ce projet avait pour objectif de diminuer la passerelle que les élèves ont entre l'école et le monde de l'entreprise. À la suite du Stand-Up reliant les groupes d'élèves aux différentes propositions des représentants entreprise, notre groupe s'est engagé auprès de M. Ndiaga FAYE.

M. FAYE est actuellement enseignant en monétique à l'ENSICAEN, et a pour volonté de varier ses travaux pratiques. C'est dans cet optique qu'il nous a demandé de réaliser un Back-Office monétique, réutilisable par les élèves et les enseignants.

Pour réaliser cette preuve de concept, nous avions à notre disposition, une fois par semaine, les locaux de l'ENSICAEN, ainsi que le matériel informatique présent sur place.

La date limite pour présenter nos livrables était fixée au 5 février 2024.

1.2. Réponse apportée.

Afin de réaliser le souhait de notre client M. FAYE, nous avons fait le choix d'implémenter une application web. Nous avons divisé cette application en deux parties : le Front-Office et le Back-Office.

Le Front-Office est une ***UI*** (***User Interface***), c'est-à-dire la partie visible par l'utilisateur. Cependant, pour un Back-Office monétique, il est indispensable d'avoir une logique poussée et de manipuler des données. C'est en réponse à cette problématique que nous avons liée à l'application Front-Office, un Back-Office qui serait sous forme d'**API** (***Application Programming Interface***) permettant la liaison entre la base de données où sont stockées les informations requissent, et le site parcouru par l'utilisateur.

1.3. Objectifs du projet.

L'implémentation entière d'un Back-Office monétique étant inconcevable dans le temps imparti, nous nous sommes restreints à la production de la partie émetteur de celui-ci.

De surcroît, par sa complexité et son ampleur, la totalité de la partie émetteur n'a pas pu être traitée. Notre équipe a dû se focaliser sur certains services réalisables, choisis sur la base de leurs dépendances, afin de produire un livrable exploitable.

Il a par ailleurs fallu adapter nos approches en fonction du rôle de l'utilisateur. Nous avons donc conçu nos services en accord avec les privilèges octroyés à chaque rôle.

2. Mise en place du projet

2.1. La méthode de travail.

Les besoins du client étant initialement vagues, nous avons opté pour une méthode agile. Celle employée se rapproche de la méthode Scrum. En effet, nous nous sommes appuyés sur les deux piliers de cette méthode :

- Le développement d'un logiciel de manière incrémentale en listant les tâches à effectuer ;
- La livraison fréquente d'un logiciel fonctionnel.

Par conséquent, à la fin de chaque itération, nous avions un logiciel de plus en plus robuste possédant de plus en plus de fonctionnalités et de services.

2.2. Identification des acteurs

La réalisation d'un projet débute par la détermination des différentes parties prenantes. Dans le cadre de notre projet, nous y distinguons notamment :

- Utilisateur : un personnel du membre du corps professoral ;
- *Product Owner* (responsable produit) : Tangui STEIMETZ ;
- Développeurs : David GUO, Kylian PICQUE, Thomas SENG, Tangui STEIMETZ.

2.3. La répartition des tâches.

Tout au long du projet, notre répartition des tâches a évolué. Celle-ci peut se décomposer en deux grandes parties : la répartition pré-soutenance, et celle post-soutenance de mi-parcours.

Durant la première phase, Thomas SENG a réalisé le Front-Office, tandis que Tangui STEIMETZ, Kylian PICQUE et David GUO ont produit la partie Back-Office. Néanmoins, nous faisions des réunions régulières pour connaître l'avancement de chacune des parties pour faire de l'intégration continu. Tangui STEIMETZ s'est occupé de concevoir toute l'architecture logicielle de la solution, en travaillant en collaboration avec David GUO qui a réalisé celle de la base de données. Kylian PICQUE a commencé à implémenter les points d'entrée et de sortie de l'*API*.

Puis, dans un second temps, Tangui STEIMETZ s'est occupé de la sécurité de l'application, de la gestion des différents rôles des utilisateurs de l'application, ainsi que la mise en place des

différents *end-points*, et David GUO s'est occupé de l'implémentation de la base de données. Kylian PICQUE a rejoint Thomas SENG pour récupérer les données du Back-Office et les afficher correctement sur le site web.

2.4. Outils utilisés.

La technologie qui gère la base de données est *PostgreSQL*, car il s'agit d'un système de gestion de base de données relationnel robuste et puissant, qui permet de manipuler en toute fiabilité d'importants volumes de données.

Pour le Back-Office, nous avons décidé de nous orienter vers une philosophie de micro-service, car il s'agissait d'une méthode que nous n'avions encore jamais appliqué durant notre apprentissage à l'ENSICAEN. De plus, les *frameworks Spring* et *Spring Boot* ont été retenu, car ce sont des technologies largement établies dans l'industrie. Ils nous ont paru plus judicieux que des solutions plus récentes, tel que *Quarkus*. Par ailleurs, le fait que Monsieur REYNAUD soit un développeur expérimenté en *Spring,* nous a mis en confiance dans l'apprentissage de cette technologie.

Pour le Back-Office, nous avons décidé de nous orienter vers une philosophie de micro-services, car il s'agissait d'une méthode que nous n'avions encore jamais appliqué durant notre apprentissage à l'ENSICAEN. De plus, les *frameworks Spring* et *Spring Boot* ont été retenu, car ce sont des technologies largement établies dans l'industrie. Ils nous ont paru plus judicieux que des solutions plus récentes, tel que *Quarkus*. Par ailleurs, Monsieur REYNAUD étant un développeur expérimenté en *Spring,* nous avons été confortés dans notre choix

Par soucis de compatibilité, le Back-Office, ainsi que la base de données, sont stockés dans des conteneurs *Docker*.

L'application web a été quant à elle conçu avec le *framework Angular*. Il nous a été recommandé par nos encadrants pour son efficacité dans la réalisation de notre site web, et pour sa popularité dans le milieu professionnel. Nous voulions avant tout un affichage des informations efficace et une navigation aisée. De plus, la réutilisabilité des composants a été un atout majeur dans la conception de certaines pages. Enfin, des librairies sont à disposition des développeurs, ce qui facilite d'autant plus la conception de nos propres composants.

Concernant la gestion des versions, nous avons utilisé *Git*, sur le *Gitlab* de l'école. Cela nous a permis de mettre en commun nos codes et d'incrémenter le projet à petit pas. *Gitlab* nous a aussi servi à noter toutes les « *issues* » et donc de pratiquer une partie de la méthodologie Agile.

Plusieurs ***IDE*** (***Integrated Development Environment***) ont été utilisés à travers le projet. *IntelliJ* et *WebStorm*, de la suite *JetBrains*, ont été nos principaux *IDE*, pour développer respectivement le Back-Office et le Front-Office. *Postman* quant à lui, a été utilisé pour tester les différents services de notre *API*.

Des logiciels de modélisation, tel quel *DBdiagram.io* ou bien *Draw.io*, ont aussi été employés pour illustrer notre architecture logicielle.

BACK-OFFICE MONETIQUE – REALISATION DE LA PARTIE EMETTEUR

3. Réalisation du Front-Office.

Tout le raisonnement lié à l'interface utilisateur (Cf. figure 1) permettant d'accéder aux services du Back-Office se trouvent dans cette partie.

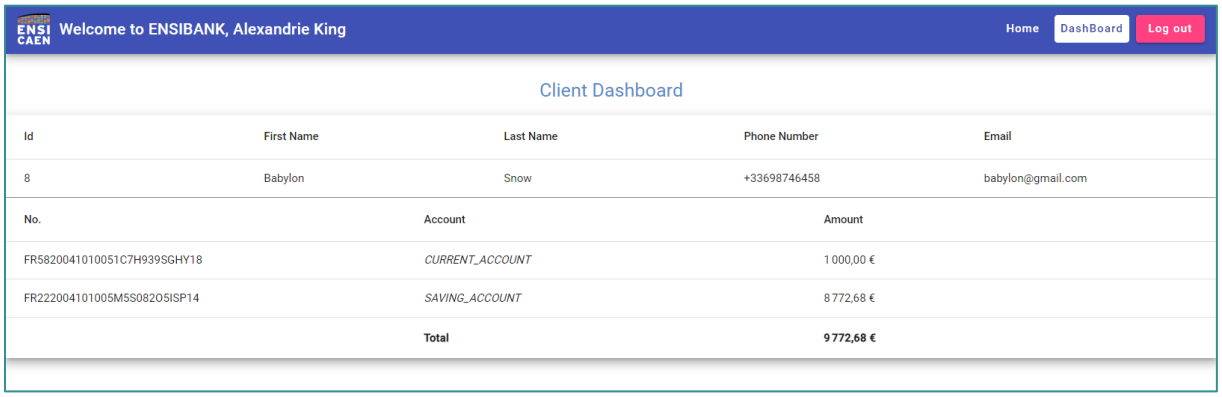

Figure 1 – Exemple d'interface utilisateur

3.1. Interface.

3.1.1. Présentation de l'application web.

L'application web se décline en plusieurs pages, permettant d'accéder aux différents services proposés par notre API.

Les différentes pages accessibles sont les suivantes :

- Page d'accueil ;
- Page de connexion au compte utilisateur ;
- Page d'enregistrement du compte utilisateur.

  Pour les Clients de la banque :
  - Tableau de bord des comptes bancaires ;
  - Affichage détaillé d'un compte bancaire.

  Pour les banquiers :
  - Tableau de bord des comptes *Client* et des produits carte ;
  - Tableau de bord des comptes bancaires d'un utilisateur *Client ;*
  - Affichage détaillé d'un compte bancaire d'un utilisateur *Client ;*
  - Affichage détaillé d'un produit carte.

  Pour les simulateurs :
  - Accès au simulateur de transactions.

3.1.2. Gestion des pages.

La navigation entre les différentes pages se fait à l'aide de boutons de navigation, apparaissant seulement si le compte utilisateur possède les droits requis pour accéder aux services liés.

Un même bouton redirige vers différents services selon le rôle de l'utilisateur, afin de respecter les privilèges accordés à chaque rôle. C'est notamment le cas pour le *tableau de bord*, qui amène une page propre au client de la banque ou au banquier.

Si la page nécessite de charger du contenu, un pop-up indiquant à l'utilisateur d'attendre s'affiche. La requête est alors envoyée au Back-Office et l'utilisateur peut accéder à la page une fois la réponse traitée.

3.2. Architecture générale.

3.2.1. Patrons de conception.

Pour la réalisation du Front-Office, nous nous sommes servis de différents patrons de conception. L'un des plus intéressants pour faire une interface graphique est l'**observateur**. Nous n'avons pas eu la nécessité de l'implémenter, car il est compris dans certaines fonctionnalité d'*Angular* : les fonctions observables qui requièrent une souscription. Ces fonctions notifient les composants, qui ont souscrits à celles-ci, qu'une information est arrivée. Un usage pertinent est celui des requêtes *HTTP*, que nous avons dû effectuer.

Un autre patron de conception qui nous est proposé par *Angular* est le **décorateur**. Au travers des propriétés *@Input* et *@Output*, qui permettent respectivement de recevoir et de transmettre dynamiquement des valeurs entre plusieurs composants imbriqués. Cette fonctionnalité est très importante pour permettre la communication entre les composants d'une page web en *Angular*. Ces outils nous ont été indispensables pour pouvoir transmettre le numéro de compte du *tableau de bord* aux détails du compte.

Afin de pouvoir administrer l'affichage des boutons en fonction du rôle de la personne connectée, nous avons implémenté une **chaine de responsabilité**. Nous avons fait ce choix, car nous postulons que le projet va être poursuivi prochainement. Dans cette optique, un outil adapté aux extensions de fonctionnalité était indispensable. Si les prochains groupes décident d'ajouter un bouton ou de modifier les rôles qui ont des droits, cela sera assez simple.

Pour pouvoir interagir efficacement avec l'**API**, nous avons réalisé et utilisé des **DTO**. Ils nous ont permis de sérialiser correctement les requêtes et réponses échangées avec l'**API**. L'inconvénient de ce patron est que chaque élément de notre **DTO** doit avoir la même nomenclature que la réponse, ce qui a pour conséquence de créer une dépendance entre requête et réponse.

3.2.2. Angular.

Comme évoqué précédemment, nous avons fait le choix d'utiliser le *framework Angular*. Celui-ci s'accompagne de nombreuses fonctionnalités qui sont déjà utilisables.

L'une des bibliothèques qui est efficace pour implémenter l'affichage de l'application est ***Angular Material***. En effet, nous l'avons utilisé à travers des éléments graphiques comme les tableaux, les boutons, les boîtes de dialogue (pop-up), ou encore les formulaires. L'avantage de cette décision est que les éléments sont rapidement ajoutés, et qu'une partie de design est déjà proposée. Nous avons par conséquence pu optimiser notre temps.

Une autre propriété d'*Angular*, qui nous a permis de gagner en efficacité, est la possibilité de créer des composants, formés d'une partie logique et d'une partie visuelle. Ils permettent de pouvoir utiliser facilement un schéma personnalisé, partagé par plusieurs pages. C'est par exemple le cas du *détail du compte bancaire* qui est utilisé dans deux pages différentes (pour le client et le banquier). Cela nous a permis de faire du code ***DRY*** et d'être plus performant.

Le dernier atout que nous avons exploité d'*Angular* est le fait de pouvoir faire des **services**. Cette fonctionnalité instancie une classe une seule fois et peut la partager à tout le projet (proche du singleton). Cela nous a permis de pouvoir faire du code utilisable dans plusieurs composants avec une certaine synchronisation des données. Grâce à cela, nous avons pu utiliser les patrons de conception précédemment évoqués.

3.3. Conformité des données

3.3.1. Vérification des entrées.

Lors de l'enregistrement et de la connexion, l'utilisateur est invité à entrer ses informations. Afin d'indiquer à l'utilisateur que ses informations sont manquantes ou ne respectent pas les formats imposés par le Back-Office, des indicateurs se révèlent pour mettre en évidence les champs mal renseignés (Cf. figure 2).

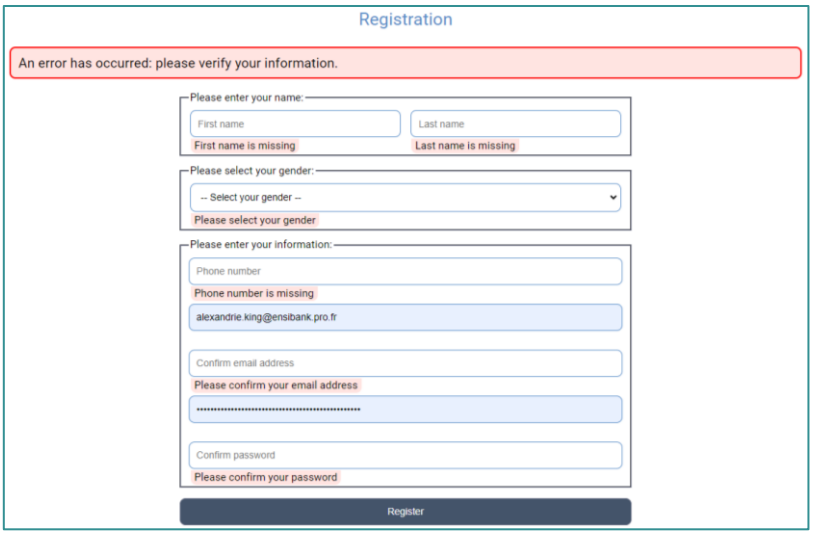

Figure 2 - Exemple d'affichage d'erreurs dues à des champs mal renseignés par l'utilisateur.

Des messages pop-up surviennent également après certaines requêtes (Cf. figure 3). Ils servent de *feed-back*, affichant les réponses émises par le Back-Office.

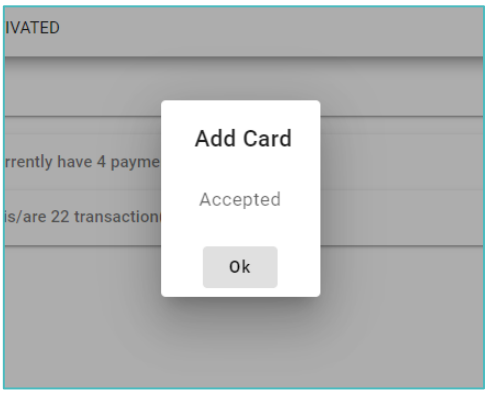

Figure 3 - Exemple de message pop-up affichant la réponse émise par le Back-Office.

3.3.2. Stockage des données.

La plupart des données sont stockées en tant que variables de classe grâce aux **DTO**. Cependant, certaines informations doivent être accessible en continu et depuis différents composants. Nous avons donc choisi de stocker dans les cookies le ***token JWT*** (***JSON Web Token***), à fournir au Back-Office lors de chaque requête en tant qu'utilisateur connecté, afin d'autoriser l'accès aux services. Les cookies recueillent également les informations de l'utilisateur afin d'accéder rapidement et à tout moment aux données importantes comme le rôle de l'utilisateur ou son identifiant dans la base de données.

En outre, nous avons besoin d'autres données qui peuvent s'avérer trop volumineuses pour être sauvegardées dans les cookies, comme la liste des comptes bancaires d'un *Client*. Nous avons fait le choix d'utiliser la mémoire de session par rapport au stockage local pour éviter que les données soient enregistrées sur le long terme. En effet, la mémoire de session supprime les données dès que la page est fermée contrairement au stockage local.

4. Réalisation du Back-Office.

Toute la logique liée à l'implémentation des différents services que doit remplir un Back-Office monétique se trouvent dans cette partie.

4.1. Sécurité des points d'accès et de la base de données.

La sécurité est le *driver* de la monétique. Il a été nécessaire de la prendre en compte tout au long de l'implémentation. Nous distinguons plusieurs types de sécurité au sein de l'API.

Concernant la sécurité des points d'entrée, nous nous posons les questions suivantes : qui sont les acteurs extérieurs et comment peuvent-ils communiquer avec un point d'entrée ? Pour répondre à la problématique du *qui*, comme mentionné précédemment, nous avons mis en place un système d'authentification qui s'exprime par la livraison d'un *token JWT*, devant être transmis dans les entêtes de chaque requête afin de la valider. Cela permet de vérifier qu'un utilisateur est connu par le système mais également de lui donner des accréditations particulières via un rôle donné (*Maître, Banquier, …)*. Nous pouvons ainsi bloquer l'accès à quiconque ne possédant pas les accréditations nécessaires.

De plus, grâce aux annotations *Spring* de validation et à la sérialisation des données, nous pouvons vérifier avec certitude que les arguments donnés sont formattés suivant les attentes du point d'accès.

Enfin, il faut également sécuriser les données sensibles au sein de la base de données. Ainsi, nous avons encodé les mots de passe des utilisateurs avec un algorithme robuste aux attaques (sel, temps constant, hash), empêchant toute possibilité d'attaque d'inversion de la fonction d'encodage.

Nous avons, de manière similaire, utilisé un système de chiffrement symétrique avec sel pour les données où la récupération de la valeur initiale était nécessaire, tel que l'*IBAN* (International *Bank Account Number*) et le *PAN* (*Primary Account Number*).

Dans les différents traitements, la donnée n'est jamais utilisée en clair, mais uniquement encodée ou chiffrée. Seule la classe de stockage initiale a donc la capacité de la comparer avec une autre.

Cependant, même si nous avons accès à une grande partie du développement sur la sécurité, il reste de nombreuses imperfections, tels que le *CVX2* qui est toujours stocké en clair (non conforme aux normes PCI-DSS) et l'historisation des actions effectuées par les utilisateurs. Il reste donc des axes d'amélioration.

4.2. Architecture générale de la base de données.

L'architecture générale d'une base de données est un élément crucial pour assurer une gestion efficace et cohérente des données. Nous avons abordé notre base de données sous différents angles :

- Le schéma « *User* » gère les informations utilisateur et rôle. Il inclut des tables pour stocker les détails du compte tel que le nom, le mot de passe et le courriel, et une table pour associer les différents utilisateurs aux différents rôles.

- Le schéma « *Account* » gère les diverses entités liées à la banque. Il comprend différentes tables pour stocker les détails d'un compte bancaire, suivre les transactions financières et la gestion des bénéficiaires. De plus, le schéma comprend des tables pour les services de prélèvement.

- Les schémas « *PaymentMethod* » et « *RequestPaymentMethod* » sont essentiels dans la mise à disposition des moyens de paiements pour le client. Le premier gère les moyens de paiements, et le second s'occupe de leur demande. Ils incluent des tables pour traiter ces deux services.

- Le schéma « *Card* » se concentre sur les différentes informations liées aux cartes. Nous y retrouvons notamment les différentes tables pour définir les produits cartes avec leurs politiques d'autorisation préférées, leurs logiciels, et les multiples contraintes liées à l'utilisation de la carte, comme le montant limite par transaction.

4.3. Architecture interne du Back-Office.

Le Back-Office est modélisé suivant une architecture classique 3-tiers :

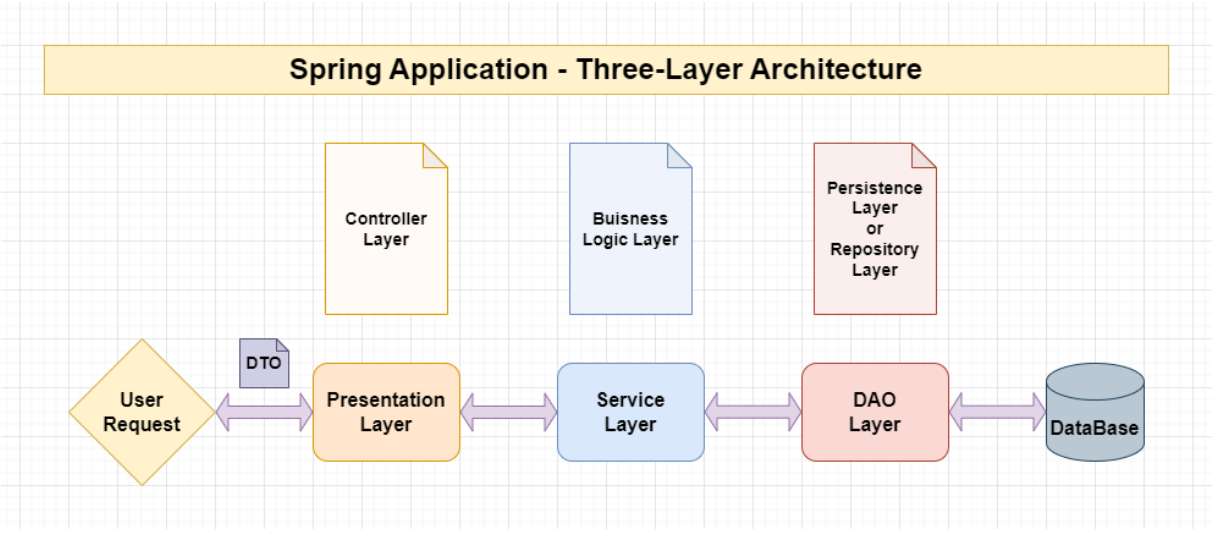

Figure 4 - Architecture 3-tiers

C'est une méthode courante dans les applications *Spring*, puisqu'elle permet de définir trois grandes parties (Cf. figure 4). Cette séparation permet d'avoir une meilleure scalabilité, une maintenabilité assurée et une plus grande facilité de développement, comparée à une architecture 2-tiers (qui était notre première idée), d'autant plus si le projet doit être repris par une autre équipe.

De plus, nous utilisons le patron **DTO** ainsi que leur ***Mapper*** respectifs, ajoutant une couche supplémentaire à l'architecture 3-tiers, et permettant ainsi :

- Une séparation des préoccupations, c'est-à-dire séparer les objets utilisés dans la logique métier et celles transmises aux utilisateurs ;
- Un meilleur contrôle sur les variables qui sont exposées aux utilisateurs, surtout dans le cas où l'accréditation de celui-ci a une importance quant à la visibilité ou l'ajout d'informations. Ce qui, par ailleurs, permet aussi d'éviter de surcharger le réseau en envoyant des données inutiles ;
- D'ajouter une dose supplémentaire de flexibilité puisque l'ajout d'un champ impacte très peu les services et contrôleurs.

Enfin, dans le cas où le processus de logique interne ne se déroule pas comme escompté :

- Une mauvaise accréditation ;
- Une mauvaise utilisation du point d'entrée ;
- Une erreur interne inattendue.

Nous avons implémenté notre propre gestionnaire d'erreurs, gérant les erreurs les plus courantes, notamment quand les arguments donnés sont incorrects, ainsi que des logs des erreurs brutes pour une meilleure compréhension (utile pour le débogage et la reprise du projet). Cependant, l'implémentation effectuée est loin d'être optimale et devrait être mise à jour de sorte à être **DRY**, plus flexible et explicite vis-à-vis des erreurs remontées. Actuellement, nous sommes à mi-chemin entre des exceptions automatiquement gérées par le gestionnaire, et d'autres qu'il faut capter et re-formatter. Cela est dû à un résidu d'une ancienne version du projet où nous étions beaucoup moins à l'aise avec le *framework*.

Pour résumer, nous pouvons définir l'architecture de communications de la sorte :

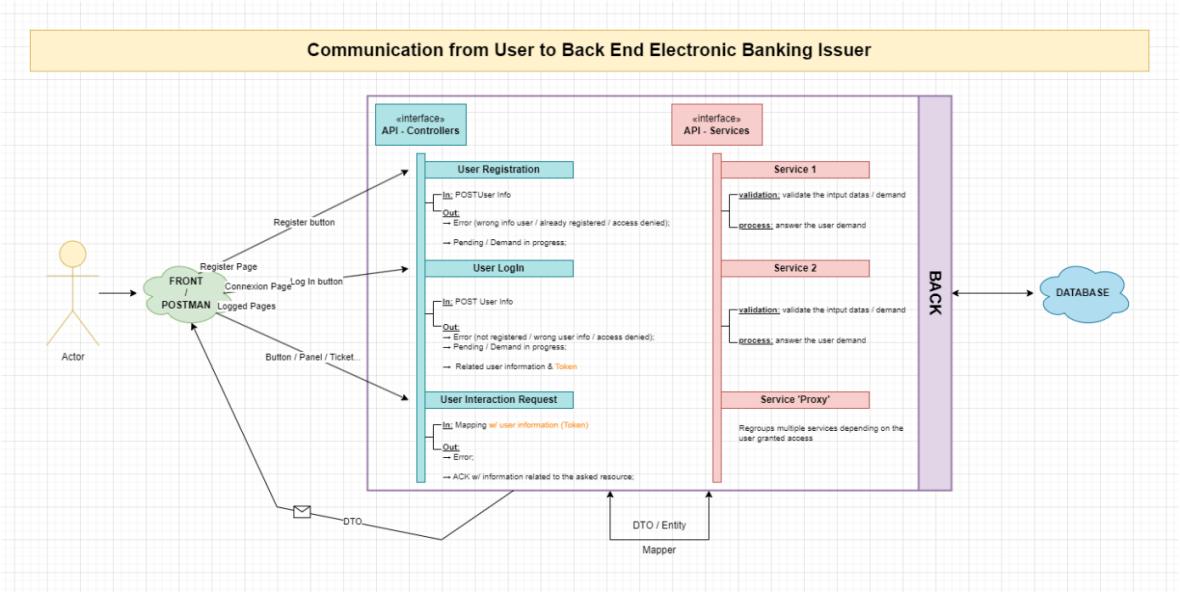

Figure 5 – Résumé de l'architecture de Back-Office

4.4. Services

Les différents services suivent la pensée d'implémentation suivante :

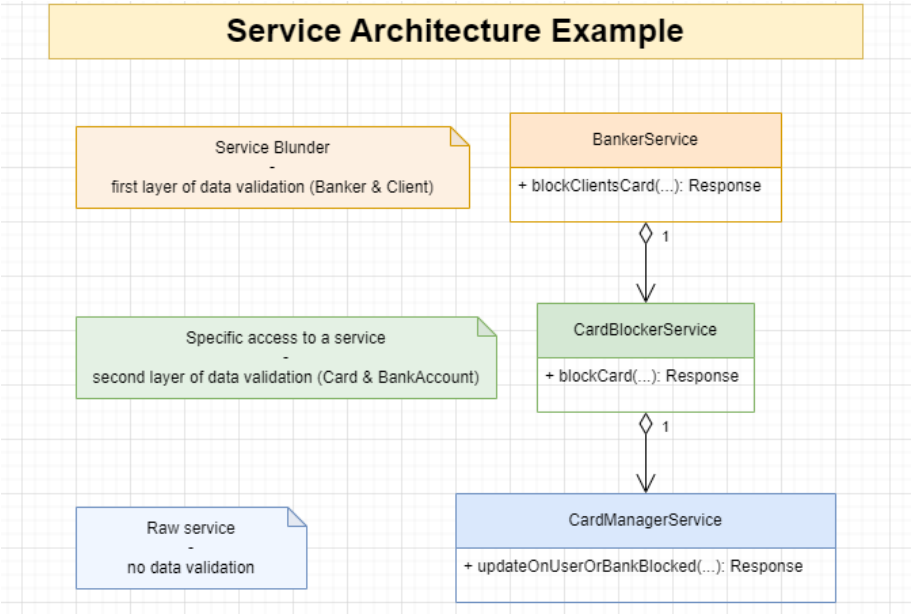

Figure 6 - Architecture des services avec l'exemple du blocage de carte via un utilisateur Banquier

L'objectif est d'avoir des services singuliers qui s'occupent d'une seule fonctionnalité / responsabilité avec comme but d'enregistrer les informations dans les tables, sans réel contrôle des données qui leur sont transmises (Cf. partie bleue de la figure 6, qui ne gère que les tables liées aux cartes.

Toutefois, cela permet l'obtention de services plus élaborés, qui regroupent un ou plusieurs services singuliers, pour appliquer un contrôle strict des données transmises suivant les accréditations accordées à l'utilisateur (Cf. les parties verte et orange de la figure 6, où nous

donnons la possibilité au *Banquier* de pouvoir bloquer des cartes bancaires actives de ses clients uniquement).

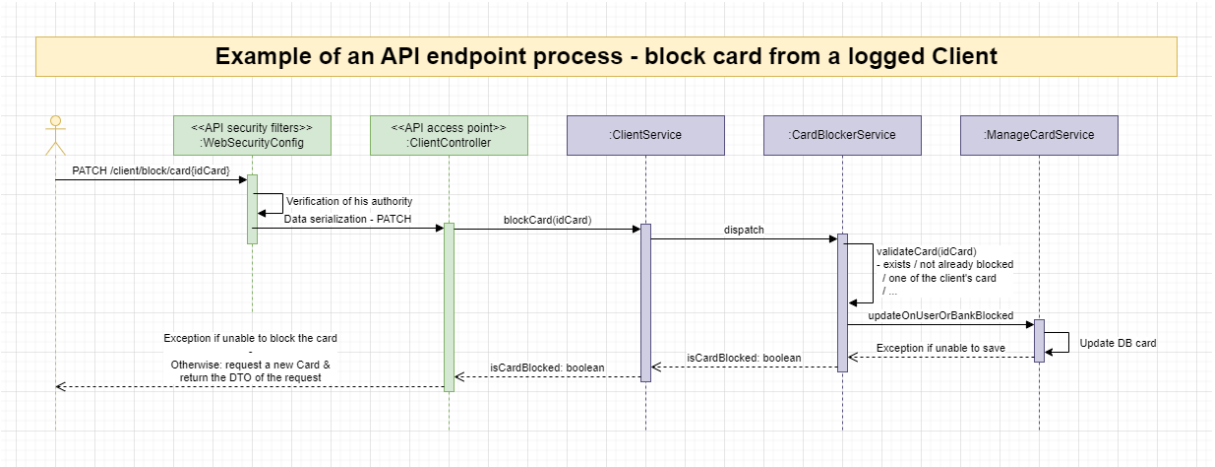

Figure 7 - Diagramme de séquence d'un end-point pour bloquer sa carte bancaire, sachant qu'il s'agit d'un client (vulgarisation)

Plus concrètement, la figure 7 expose une séquence illustrant la philosophie que nous avons adoptée pour l'implémentation de nos services.

Dans l'état actuel du Back-Office, nous avons pu remplir certaines fonctionnalités de la partie émetteur du Back-Office monétique, qui sont :

- **Gestion d'un compte client** : recevoir le détail de son compte, bloquer une de ses cartes, commander un chéquier ;
- **Gestion d'un compte banquier** : consulter le compte de ses clients, leur ajouter un nouveau compte bancaire, leur bloquer leurs cartes bancaires, leur commander de nouvelles cartes bancaires en spécifiant le produit carte associé et d'autres arguments ;
- **Gestion d'un compte comptable** : consulter les transactions qui ont impactées le Back-Office et constater leur caractère *On-Us* ou *Off-Us* ;
- **Gestion des connexions par le gestionnaire des utilisateurs** : consulter la liste des utilisateurs qui ont accès ou ont demandé l'accès aux ressources. Il peut donc valider ou rejeter une demande en lui assignant un rôle.
- **Collecte des transactions** : permet de collecter un jeu de transactions et les assigner aux différents comptes bancaires sans autre traitement. Les transactions sont seulement validées, aucune autre modification n'est appliquée (de type compensation ou traitement interne). Celui-ci va de pair avec un simulateur qui permet de générer des transactions aléatoires (utilisant soit des acteurs générés, soit des acteurs présents dans la base de données).

Les services qui sont actuellement disponibles peuvent être consultés sur la documentation *swagger*. Ceux-ci sont globalement complets, mais notre apprentissage de *Spring* fut rudimentaire. Afin de finir les services jugés comme nécessaire pour une première version, nous n'avons pas entièrement respecté cette philosophie de l'architecture des services précédemment exposée.

4.5. Développement opérations.

Au travers du projet, nous avons opté pour une infrastructure flexible, en utilisant le déploiement de *Docker Compose*, ainsi que les différents *Dockerfiles*. Cette approche permet de faciliter le déploiement des applications sur différentes plateformes, grâce à l'utilisation de conteneurs *Docker*.

Actuellement, le projet inclus deux conteneurs *Dockers* : un pour la base de données, l'autre pour le serveur Back-Office API. Cependant, nous n'avons pas réussi à implémenter celui pour le Front-Office.

5. Bilan sur le projet.

5.1. Points positives et améliorations.

5.1.1. Front-Office.

Le principal avantage du Front-Office est qu'il est pratiquement fonctionnel. En effet, il est possible d'avoir un usage normal du site web, avec des services opérationnels. Pour aller plus loin, celui-ci est également fluide et convivial.

Cependant, certaines imperfections sont à noter :

- **Impossibilité de se déconnecter** : dans le cas où l'utilisateur décide de consulter le *tableau de bord*, il peut rencontrer des difficultés à se déconnecter s'il rafraichit la page. Il s'agit d'une contrainte assez gênante pour l'utilisateur, et serait une des premières tâches qui devrait être traité par les prochaines équipes.

- **Refonte de code** : par manque de temps, nous avons été maladroit dans l'implémentation du Front-Office. Une refonte serait nécessaire pour supprimer des répétitions de code, et offrir une meilleure lisibilité, ainsi qu'une meilleure optimisation.

- **Documentation** : la majorité du travail réalisé est peu, voire pas documenté pour le Front-Office. Il serait donc indispensable de s'attarder sur cette mission.

- **Généralisation de l'utilisation de *Material*** : les pages d'enregistrement et de connexion ont été réalisées sans l'aide d'*Angular Material*, et sont donc visuellement en décalage avec le reste des composants.

- **Sécurisation** : les informations qui sont conservées dans les cookies et la mémoire de session sont en clair. Il serait préférable d'appliquer un chiffrement à ces données.

5.1.2. Back-Office

Comme nous avons pu l'expliquer dans les différentes parties, les refontes fréquentes du code nous ont permis de nous orienter vers l'architecture la plus extensible possible, et nous avons presque atteint cet idéal.

Nous pouvons également dire qu'en terme de sécurité, le Back-Office est en bonne voie pour être complétement sécurisé :

- Il manque notamment l'encodage du CVX, qui pour le moment a été pris en considération ;

- Il faudrait un meilleur contrôle des données envoyées. Par exemple, en ajoutant des annotations personnalisées comme nous avons commencé à le faire.

D'autre part, nous avons grandement orienté nos efforts sur la lourde migration du code vers l'utilisation des **DTO** et l'architecture 3-tiers, ce qui a impacté le nombre et la qualité des services. Nonobstant, cela rend les fonctionnalités du Back-Office majoritairement lisibles et ouvertes aux futures modifications.

5.1.3. Général

Le projet est globalement fonctionnel vis-à-vis des différents services que nous avons implémentés.

Cependant, un défaut notable de ce projet est le manque de tests unitaires. Cela est principalement dû au manque de temps, puisque nous avions commencé à les rédiger. Mais, voyant la date limite approchée, nous avons dû abdiquer.

En outre, nous avions dans un premier temps pour objectif de suivre une méthodologie fidèle au principe Kanban, mais pour des raisons similaires, nous avons dû la délaisser.

5.2. Retour d'expérience.

Un des principaux problèmes rencontrés est le réel manque de temps pour effectuer proprement et entièrement le projet :

Pour la première fois, nous étions confrontés à un vrai projet qui utilise une pluralité de *frameworks* et technologies différentes, d'où une alternance perpétuelle entre apprentissage et confection du projet. Pour cette raison, nous n'avons pas réussi à appliquer de « bonnes pratiques » de génie logicielle. Notamment, notre passage d'une architecture 2-tiers vers une architecture 3-tiers pour le Back-Office, ou encore la découverte des **DTO** à moins d'un mois du rendu, sont des exemples parfaits de difficultés auxquelles nous n'avons jamais été réellement confronté durant notre scolarité.

Nous voulions d'autre part souligner que ce projet a été une expérience très enrichissante pour chaque membre de l'équipe. Assurément, nous avons acquis des compétences concernant les différents *frameworks,* au même titre que notre compréhension du Back-Office monétique s'est améliorée.

6. Conclusion.

Pour conclure, cette expérience nous a permis de défier nos compétences en termes d'apprentissage, de flexibilité, ainsi que d'organisation. En moins de 16 séances, nous avons réalisé une application web et une API, fondant les bases d'un Back-Office monétique. Ce projet mérite d'être repris par de futurs équipes.

ENSI CAEN

ÉCOLE PUBLIQUE D'INGÉNIEURS
CENTRE DE RECHERCHE

Ecole Publique d'ingénieures et d'ingénieurs en 3 ans

6 boulevard Maréchal Juin, CS 45053
14050 CAEN cedex 04