Université de Nice - Sophia Antipolis

Polytech Nice Sophia

Sciences Informatiques 5ème année

Architecture Logicielle

Rapport de conception

Alexis GARDIN, Théo FORAY, Laura LOPEZ, Nathan STROBBE

Équipe enseignante : Guilhem Molines et Philippe Collet

Table des matières

Chapitre 1

Périmètre du projet

Pour ce projet nous avons à réaliser une application de e-banking qui doit regrouper la plupart des fonctionnalités d'une banque en ligne. Celui-ci comporte deux composantes qui sont :

- Le datacenter de la banque qui est situé au Groënland
- Un site web accessible aux clients qui est situé en France

Nous avons donc défini notre périmètre en nous concentrant sur les fonctionnalités pour les particuliers.

Les fonctionnalités comme la gestion de compte, l'inscription et la connexion, la gestion de la carte et la consultation des données bancaires seront une priorité. De plus, nous accordons une attention particulière sur l'utilisation de notre site web qui doit être simple et intuitive.

En ce qui concerne la sécurité, pour le moment elle n'est pas une priorité, mais nous envisageons tout de même d'avoir un login possédant un mécanisme de sécurité plus avancée.

1.1 Personas

— Utilisateur : Personne n'ayant pas encore de compte utilisateur (donc pas de compte bancaire) étant présente sur le site. Potentiel futur client. Peut avoir accès aux parties du site web ne nécessitant pas d'authentification.

— Client : Personne ayant un compte utilisateur (donc au moins un compte bancaire chez CréditRama). Consulte ses comptes bancaires après s'être connecté, et effectue des opérations telles que des virements.

— Banque : A des droits administrateurs pour gérer notamment les stratégies en cas de découvert.

1.2 Fonctionnalités

Fonctionnalité	Description
Création de compte utilisateur	Un utilisateur peut se créer un compte pour se connecter à l'application, il aura un compte bancaire par défaut.
Connexion à l'application	Un utilisateur peut se connecter à l'application.
Consultation compte bancaire	Un client peut consulter l'historique des transactions effectuées sur un compte bancaire.
Ajout/suppression d'un bénéficiaire	Un client peut gérer ses bénéficiaires.
Ajout/suppression d'une carte bancaire	Un client peut ajouter une carte bancaire (carte de crédit ou de débit).
Gestion d'une carte bancaire	Un client peut gérer sa carte bancaire : déclarer un vol, activer le sans contact, changer le pin, connaître le pin.
Virement bancaire	Un client peut effectuer un virement bancaire vers un autre compte lui appartenant ou vers un bénéficiaire. Un virement peut être programmé ou exceptionnel.
Génération d'un RIB	Un client peut générer un RIB sous format pdf depuis l'application.
Débit d'une carte bancaire	Un utilisateur peut simuler un débit sur une carte bancaire permettant de simuler un achat Internet par exemple.
Notification	Un utilisateur peut être notifié lors d'évènements bancaires effectués sur son compte de plusieurs manières (email, sms ou fax).
Gestion du découvert	Un utilisateur peut consulter son compte et constater le découvert, il est également notifié lorsqu'il passe dans le rouge.

1.3 Extensions possibles

Les fonctionnalités précédemment décrites nous semblent être les plus importantes à implémenter, mais nous avons déjà réfléchis à plusieurs axes futurs. Notamment un effort conséquent qui devra être fait au niveau de la sécurité. Nous avons notamment pensé ajouter des *gateway* sur nos services basées sur un système d'API key, ce qui permettrait de les ouvrir vers l'extérieur.

Concernant les fonctionnalités, nous en avons déjà relevées quelques unes, par exemple, nous avons prévu que les découverts devront être gérés, ainsi que la génération d'autres pdf comme les relevés de comptes et autres factures possibles. D'autre part, un site e-banking ne peut être complet sans un support d'aide. Il peut s'agir d'un chatbot qui plus tard viendra en aide au client, d'un formulaire d'envoi de mail, d'aides écrites disponibles sur plusieurs pages du site, d'une F.A.Q, etc.

Enfin, une partie d'administration devra être envisagée pour ce site. Aujourd'hui, nous pré-

voyons de simplement réaliser la génération d'un tableau affichant l'état actuel de la base de données pour les administrateurs. Mais il serait intéressant pour eux d'avoir une vraie plateforme en ligne leur permettant de visualiser et gérer les clients.

Chapitre 2

Architecture du projet

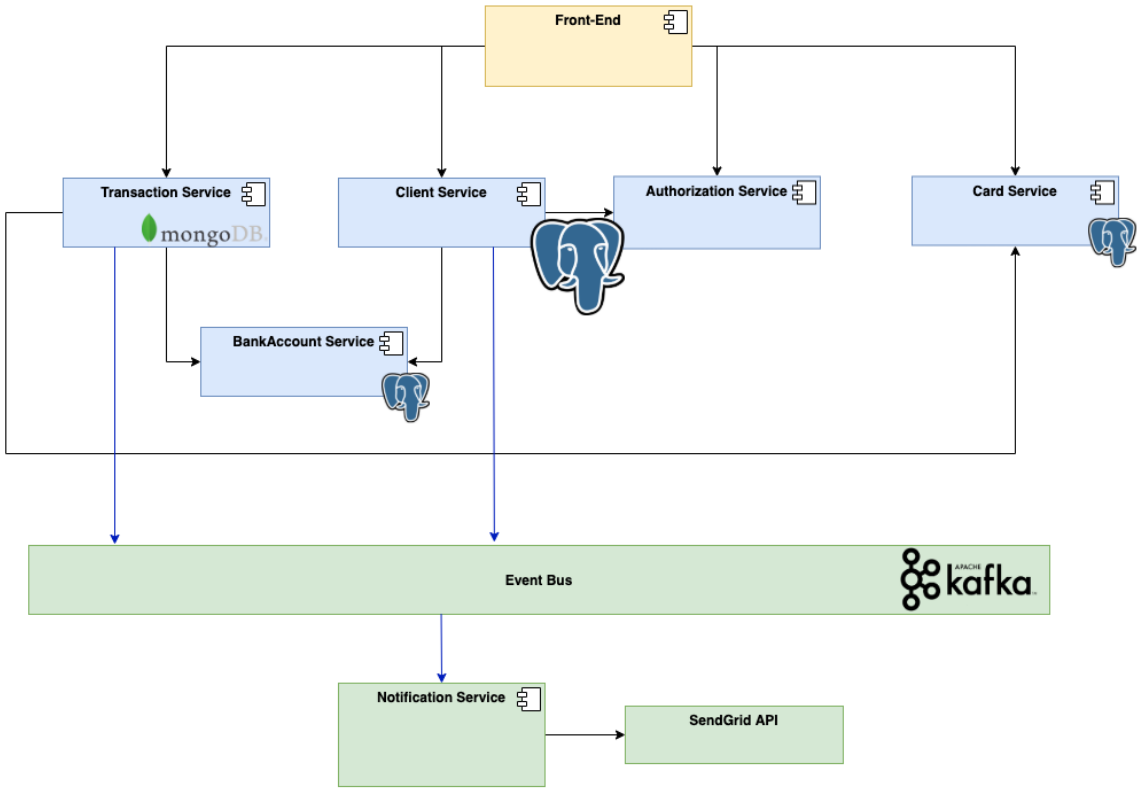

FIGURE 2.1 – Diagramme de composants - 14/01/2020

2.1 Présentation des services

2.1.1 Authorization Service

Le service d'authentification nous permet de sécuriser l'accès à nos services aux utilisateurs ayant les droits d'y accéder. Ce service permet d'utiliser le protocole **Oauth**, qui est largement utilisé dans l'industrie permettant de sécuriser les APIs avec différents types d'autorisations. Le type d'autorisation utilisé par ce service est « Password Credentials », avec ce type, les

identifiants (et donc le mot de passe) sont envoyés au client et ensuite au serveur d'autorisation. Il est donc impératif qu'il y ait une confiance absolue entre ces deux entités. De ce fait, il est principalement utilisé lorsque le client a été développé par la même autorité que celle fournissant le serveur d'autorisation.

2.1.2 Client Service

Le **Client Service** est le service dédié aux comptes utilisateurs. Il aura pour responsabilités la mise en base de données les nouveaux utilisateurs ainsi que l'ensemble de la logique métier liée aux utilisateurs. Ces derniers suivent une représentation classique d'un utilisateur lambda (avec un mot de passe, une adresse email, un nom de compte, ...). Cette représentation est définie dans le **Client Service** et l'**Authorization Service** qui utilisent tous les deux la même base de données. En effet, notre **Authorization Service** accède aux utilisateurs avec une vue abstraite d'un User lui permettant de gérer tout ce qui concerne les tokens et les rôles. Contrairement au **Client Service** qui lui va rajouter une spécialisation sur le User pour en faire un « Client », ce dernier possédant d'autres attributs, comme ses bénéficiaires ou ses comptes en banque.

2.1.3 Transaction Service

Le service Transaction est un point critique de notre architecture. Ce service est une abstraction d'une transaction bancaire dans le cadre d'un virement ou d'un paiement par carte de crédit. Il dialogue avec une base de données **MongoDB** qui nous permet de disposer de performances d'écriture plus intéressantes qu'une base de données SQL classique. En effet, nous voulons être capables de gérer un grand nombre de transactions et la vitesse d'écriture fournie par Mongo nous permet tout à fait de remplir cet objectif.

Pour répondre au besoin de consistance d'une transaction, nous avons utilisé le design pattern **SAGA** en nous aidant de la définition faites de ce pattern dans le livre « Microservices pattern »[1]. Ce pattern permet de maintenir la consistance de nos données à l'aide de plusieurs mécanismes. Nous avons choisi de réaliser un SAGA **orchestré** par le **Transaction Service** et qui, en cas d'erreur (par tout hasard dans le cas d'une base de donnée un peu capricieuse), arriverait à compenser ces erreurs. Le SAGA va déléguer l'opération de transfert au service BankAccount. L'ensemble des autres opérations concernant la transaction seront gérées directement dans le service Transaction. Nous avons réalisé une machine à état, représentant la logique métier du pattern implémenté, disponible ci-dessous :

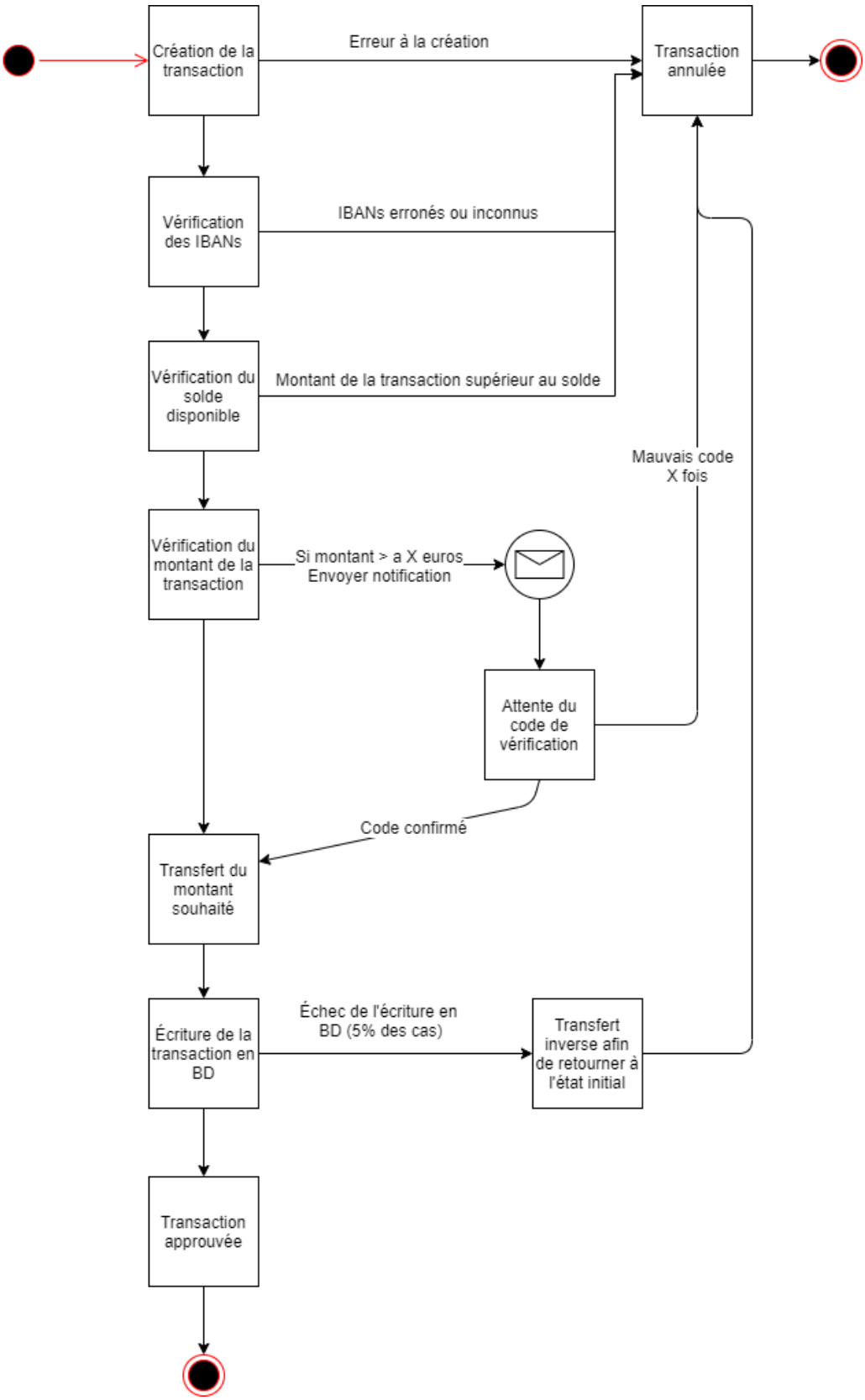

FIGURE 2.2 – Schéma sous forme de FSM représentant le cycle de vie d'une transaction CréditRama.

De plus, extraire la logique de transactions dans ce service permet de mieux séparer les responsabilités et répondre à la charge, une transaction étant probablement l'opération à la fois la plus critique et la plus opérée par l'utilisateur.

2.1.4 BankAccount Service

Le service **BankAccount** gère l'ensemble des opérations relatives aux comptes bancaires des clients de CréditRama. Il a été pensé afin d'abstraire la logique de compte bancaire de celle du client, ce dernier pouvant être associé à plusieurs comptes. Cette abstraction permet également de prévoir l'existence future de types de comptes différents tels que le compte entreprise, compte épargne, etc.

Ce service communique avec une base de données Postgres. Nous avons jugé qu'une base de données relationnelle répondait le mieux au besoin de consistance de données lié aux comptes bancaires.

Pour le moment le service des comptes a une architecture REST, néanmoins dans certains cas, par exemple la modification de la valeur des comptes, cela correspondrait davantage à un appel de procédure. Cet axe est encore sujet à discussion au sein du groupe.

2.1.5 Adaptations liées aux contraintes

Nous avons adapté notre architecture suite aux contraintes ajoutées pour cette seconde phase de projet.

2.1.5.1 Ajout d'un numéro d'autorisation

Afin de répondre à la contrainte de numéro d'autorisation permettant de sécuriser les transactions, nous avons créé le service **Transaction**. En effet, ce service attend un numéro unique lui permettant de vérifier la validité d'une transaction effectuée. Si ce numéro n'est pas présent, la transaction sera annulée (cf : 2.2).

2.1.5.2 Prévoir 5% d'échecs d'écriture en base de données

Pour cette seconde contrainte, nous allons simuler 5% d'échec des écritures en base de données sur la base **MongoDB** liée au service Transaction car il s'agit du transfert d'informations le plus critique de notre système. Comme indiqué précédemment dans la description du service (cf : 2.1.3), notre implémentation du pattern **SAGA** sur le service **Transaction** nous permettra de gérer ses échecs de manière fiable.

2.1.6 Choix architecturaux

Nous avons organisé notre architecture autour des fonctionnalités métier de l'application. Le datacenter étant situé au Groënland, nous avons décidé de mettre nos services dédiés aux comptes bancaires, aux cartes bancaires et aux transactions, au Groënland. Ces services ayant pour but de traiter des opérations critiques liées aux comptes en banque et aux transactions d'argent, nous avons jugé préférable qu'ils soient délocalisés au plus proche du datacenter avec lequel ils interagissent. L'importance de ses opérations engendre aussi une tolérance plus grande du temps d'attente de la part de l'utilisateur. En effet, nous avons pu constater suite à notre

utilisation d'applications bancaires que la vitalité de ces opérations impose une patience à laquelle nous sommes habitués en tant qu'utilisateurs.

D'autre part, nous prévoyons que le système d'authentification, l'interface graphique, le service de notification ainsi que la gestion des clients soient situés en France. La majorité sont des services qui ne nécessitent pas de communication avec les datacenter et privilégient une latence faible pour l'expérience utilisateur. De part sa communication avec le service d'authentification ainsi que sa charge estimée faible selon les cas d'utilisation principaux d'une application bancaire (dont la création d'un compte utilisateur ne fait pas partie) nous avons décidé de localiser ce service en France.

2.2 Scénarios

Suite aux changements dans le sujet, nous avons décidé de nous concentrer sur les défis apportés par ces modifications avant d'implémenter d'autres fonctionnalités. Les scénarios prévus sont donc réorientés pour répondre aux modifications, tout en gardant certaines fonctionnalités initiales (telles que : la gestion d'un type de carte bancaire, la possibilité pour un client d'avoir plusieurs comptes bancaires, le mail de découvert, et historique plus complet des transactions). Nous avons donc créé les User Stories correspondantes dans notre tableau Kanban sur le repository Github principal.

2.2.1 Scénario 1 (POC)

Alice se connecte, consulte le montant de son compte et souhaite faire un virement à son ami. Elle ajoute en bénéficiaire Bob, et lui fait un virement de 50€ pour son anniversaire. Bob se connecte et reçoit une notification par email lui indiquant un virement de 50€.

2.2.2 Scénario 2

Marcel se connecte à CréditRama. Il souhaite avoir un autre compte bancaire pour ses épargnes ainsi qu'une carte chez CréditRama. Marcel créé donc un nouveau compte bancaire et commande une carte bancaire et reçoit un mail d'information. Il effectue un virement sur ce nouveau compte pour mieux gérer son argent.

2.2.3 Scénario 3

Alexis veut s'acheter une belle voiture et souhaite transférer une grosse somme d'argent sur son compte en banque principal. Il effectue donc un virement sur CréditRama, la transaction s'effectue et est en mise en attente. Il va vérifier ses mails pour rentrer le code de confirmation dans l'application. Cependant, une erreur s'est produite pendant la vérification du code, Alexis est prévenu que la transaction a été annulée. Il recommence donc cette action d'effectuer un virement, confirmer par le mail et cette fois-ci la transaction a bien été traitée.

2.2.4 Scénario 4

Théo effectue un paiement par carte bancaire. Après la transaction validée, il reçoit un mail qui lui indique être à découvert. Il se connecte donc sur CréditRama, et va consulter l'historique de ses transactions pour comprendre comment il a pu dépenser tout cet argent.

2.3 Gestion du cycle de vie de la transaction

Le cycle de vie de la transaction est géré grâce au pattern SAGA directement implémenté dans le service Transaction.

2.3.1 Axon

Nous avons utilisé le *framework* Axon pour implémenter notre SAGA. L'outil permet notamment de faire de l'*event sourcing*, de mettre facilement en place une architecture CQRS et également de réaliser des SAGA. Nous avons fait le choix de l'utiliser car il s'intègre très facilement avec Spring et qu'il repose sur un système d'annotations avec lequel nous sommes familiers. Il nous a ainsi permis de mettre rapidement en place le pattern en gérant directement deux procédés essentiels du SAGA :

— les commandes, directement gérées en utilisant l'interface *CommandGateway*. Cette interface fournit une implémentation simplifiée du bus de commandes utilisé par Axon et permet de propager facilement les commandes depuis le SAGA. Les commandes sont ensuite gérées par l'aggrégat chargé du cycle de vie de la transaction directement dans le service Transaction, qui va effectuer les actions nécessaires à l'évolution de l'objet transaction.
— les évènements, appliqués par l'objet global *SagaLifeCycle*. Ces évènements sont réceptionnés par le SAGA qui propagera la prochaine commande en fonction afin de passer à l'étape suivante du cycle de vie, conformément à notre machine à état.

Le *framework* Axon repose sur l'utilisation de l'Axon Server. Il s'agit d'un serveur implémenté pour l'utilisation d'Axon permettant de stocker les évènements et les commandes. Ce serveur possède deux configurations différentes : "tracking" pour faire de l'*event sourcing* et *subscribing* à l'inverse permettant une gestion classique des évènements stockés. Dans notre cas, nous avons configuré le serveur en *subscribing*.

2.4 Roadmap

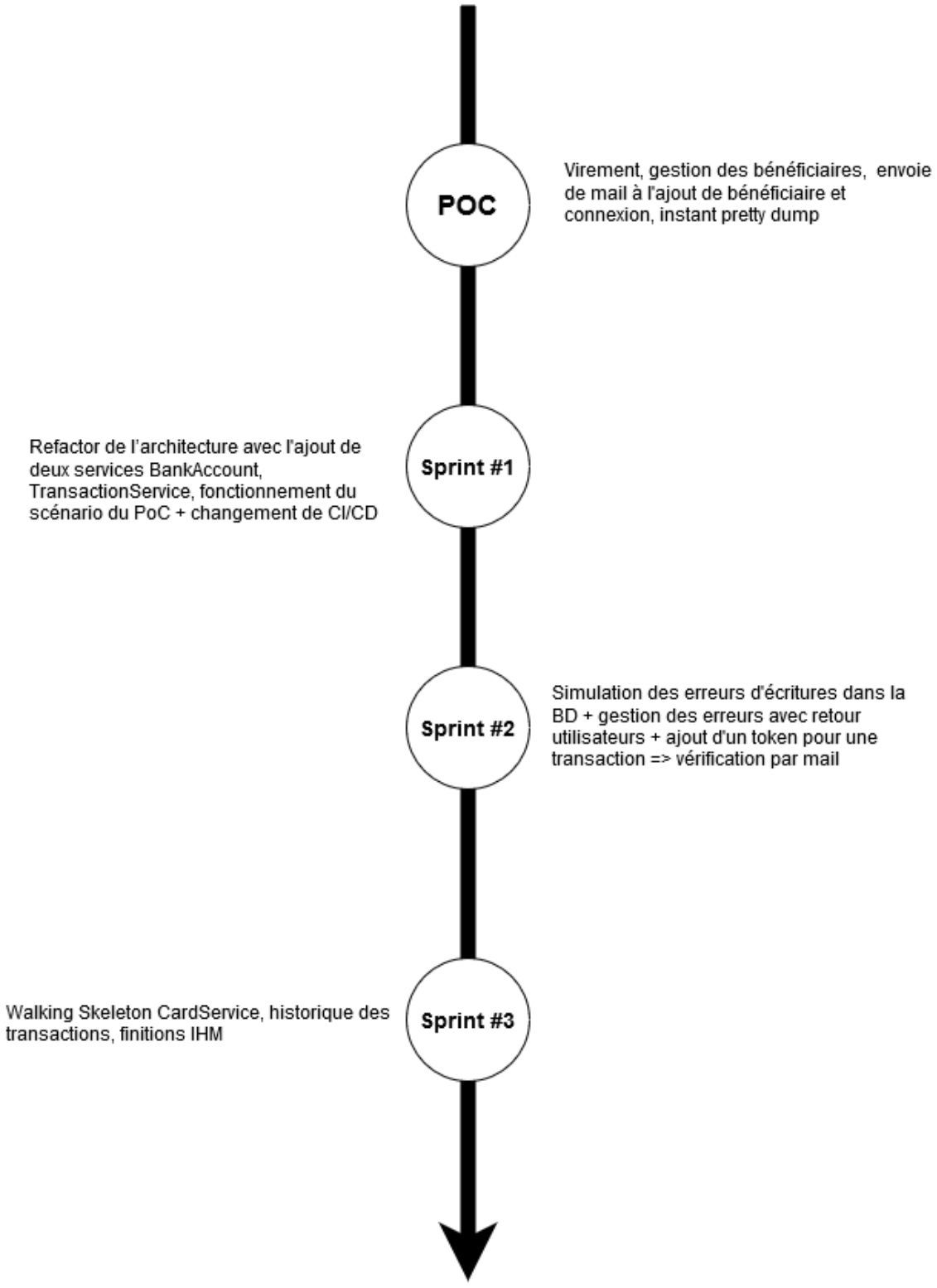

FIGURE 2.3 – Roadmap - 14/01/2020

2.5 Plan pour la période du PoC

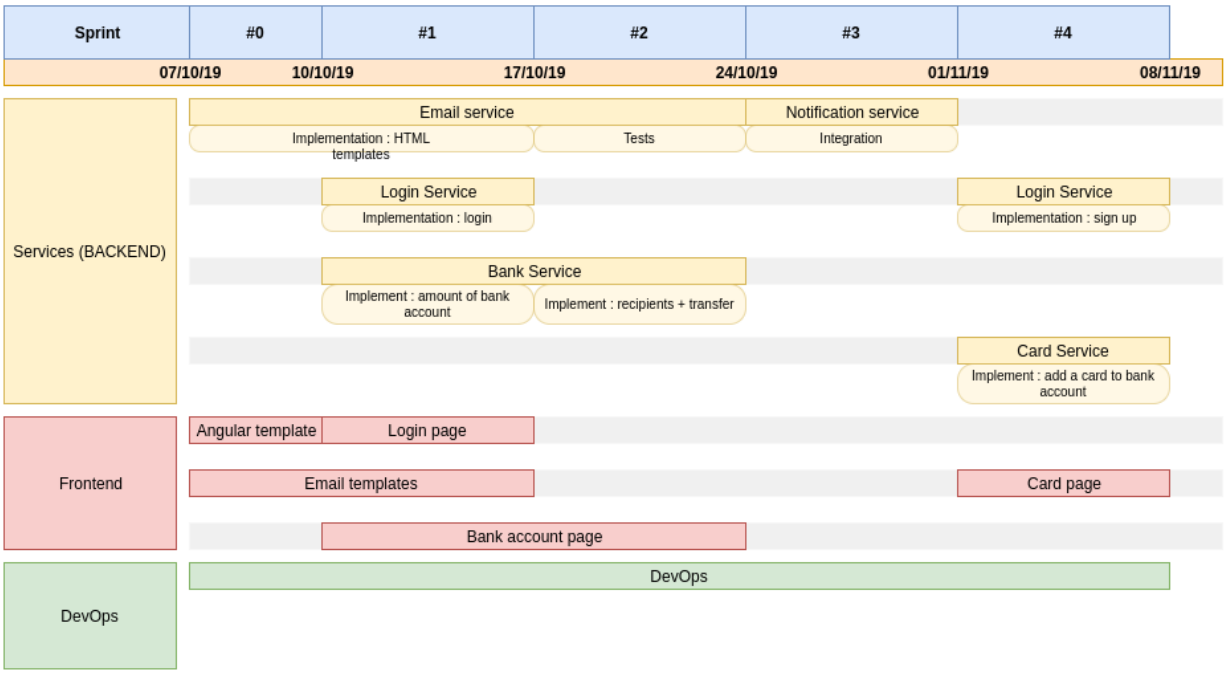

FIGURE 2.4 – Plan initial prévu - 09/10/2019

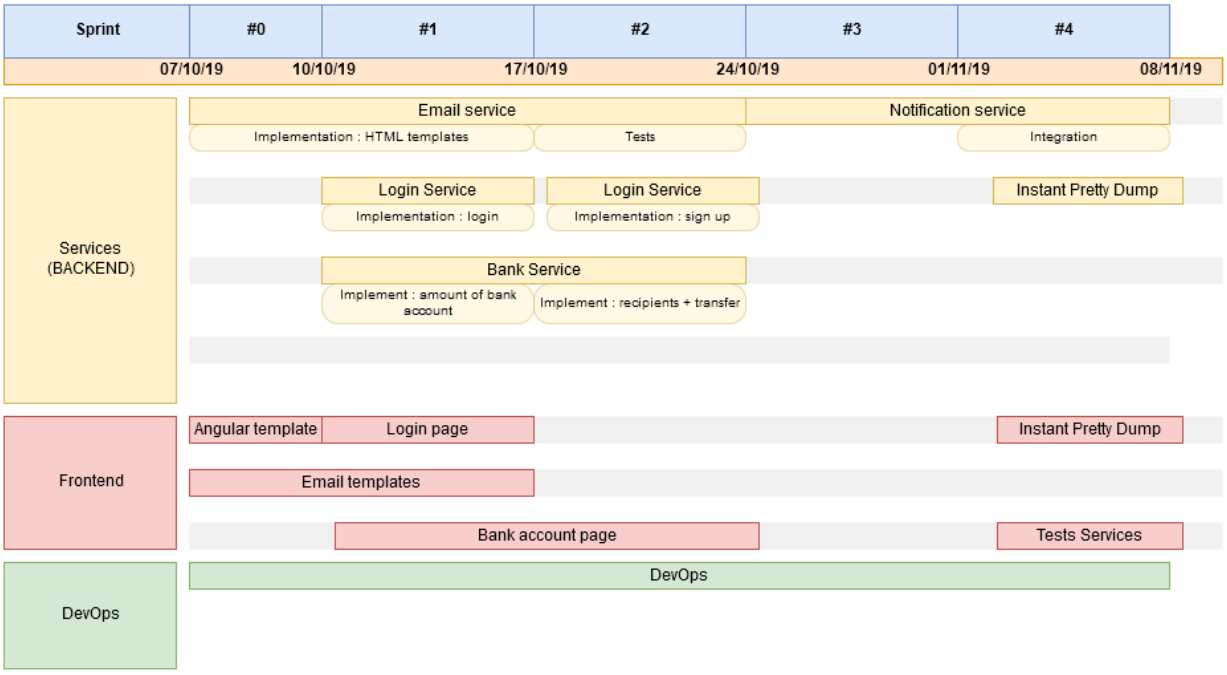

FIGURE 2.5 – Plan réel - 12/11/2019

Annexes

FIGURE 2.6 – Représentation d'une transaction sous format document (JSON)

Bibliographie

[1] Richardson, C. *Microservices Patterns With examples in Java.* Manning, 2018.