NewBank : Merchant webapp

Equipe B

Yahiaoui Imène - Gazzeh Sourour - Ben aissa Nadim

Al Achkar Badr

Mises à jour de l'incrément :

Pour cet incrément, nous avons :

- Mis à jour des fonctionnalités système recherchées dans notre architecture en mettant l'accent sur la partie variante, le sdk.
- Modifié très légèrement notre architecture et ajouté un composant entre notre SDK et notre Gateway.
- Ajout d'une partie supplémentaire “Limitation des risques” où nous expliquons comment notre architecture répond aux besoins.

Les modifications apportées aux explications écrites déjà fournies sont indiquées en rouge. Les nouveaux paragraphes sont ajoutés à la fin.

Introduction

Le présent rapport a pour but de mettre en lumière le sujet qui nous a été assigné, puis d'aborder la portée, les utilisateurs, les risques et les cas d'utilisation associés. De plus, nous avons élaboré un diagramme de composants qui sera présenté et expliqué.

Projet

La portée de ce projet englobe le développement et le déploiement d'une application web spécialement conçue pour les commerçants. Elle inclut l'intégration d'un kit de développement logiciel (SDK) pour faciliter les paiements par carte de débit et de crédit sur les sites web des commerçants. De plus, elle couvre la gestion des frais de transaction pour les transactions en ligne sans numéraire au sein du système bancaire.

Utilisateurs

1. Utilisateurs Principaux :
    - Commerçants : Ceux qui possèdent et exploitent des entreprises et qui utilisent l'application web pour gérer les transactions.
    - Clients : Individus effectuant des achats auprès des commerçants en utilisant des cartes de débit/crédit.
2. Utilisateurs Secondaires :
    - Administrateurs de la Banque : Responsables de la supervision du système bancaire en ligne sans numéraire.

- Administrateurs Système : superviser la suppression de comptes et de cartes, de gérer les données sensibles et d'assurer la sécurité en cas de perte ou de vol.
- Credit Card Network : ce réseau va nous solliciter afin d'autoriser une transaction chez nous (NewBank), notre système communiquera avec lui afin de lui demander de créer des cartes virtuelles.
- Banques externes: Les banques externes nous solliciteront pour les transferts des fonds.

Risques Possibles

- Préoccupations en matière de sécurité : Risques liés à la gestion des transactions financières. Possibilité de violation de la sécurité des données sensibles des utilisateurs, ce qui pourrait entraîner des conséquences graves pour la confidentialité et la confiance des clients.
- Intégration : Difficultés pour intégrer de manière transparente le SDK dans divers sites web de commerçants.
- Scalabilité : S'assurer que le système peut gérer un grand nombre de clients (jusqu'à 1 million) sans problèmes de performance.
- Haute Disponibilité : De même, soutenir la gestion en temps réel d'un grand nombre de transactions.
- Extensibilité : Comme notre système interne sera étendu en premier lieu par un sdk fourni par nous.
- Robustesse : Comme le kit de développement est fourni par nous et permet l'utilisation de nos services par de nombreuses entreprises, c'est à nous de garantir la robustesse contre les entrées et les essais erronés pour faire tomber notre service.

Cas d'Utilisation

- Enregistrement et Intégration des Commerçants :

Description : Un commerçant s'inscrit sur l'application web, fournissant les détails nécessaires à l'intégration au sdk.

- Gestion des Comptes Clients :

Description : Les clients peuvent créer et gérer leurs comptes, y compris les comptes d'épargne. Ils ont la capacité d'effectuer des opérations telles que l'ouverture de nouveaux comptes, la consultation des comptes existants.

- Consultation de Solde et Historique des Transactions Client :

Description : Les clients peuvent consulter le solde et l'historique des transactions de leur compte. Cette fonctionnalité permet d'obtenir des informations détaillées sur les mouvements associés au compte client.

- Virement entre Comptes :

Description : Les clients ont la possibilité de réaliser des virements entre leur compte bancaire et le compte associé à la carte de notre service.

- Traitement des Transactions avec Cartes de Débit/Crédit :

Description : Les commerçants initient des transactions en utilisant les détails des cartes de débit/crédit, qui sont traités de manière sécurisée par le système.

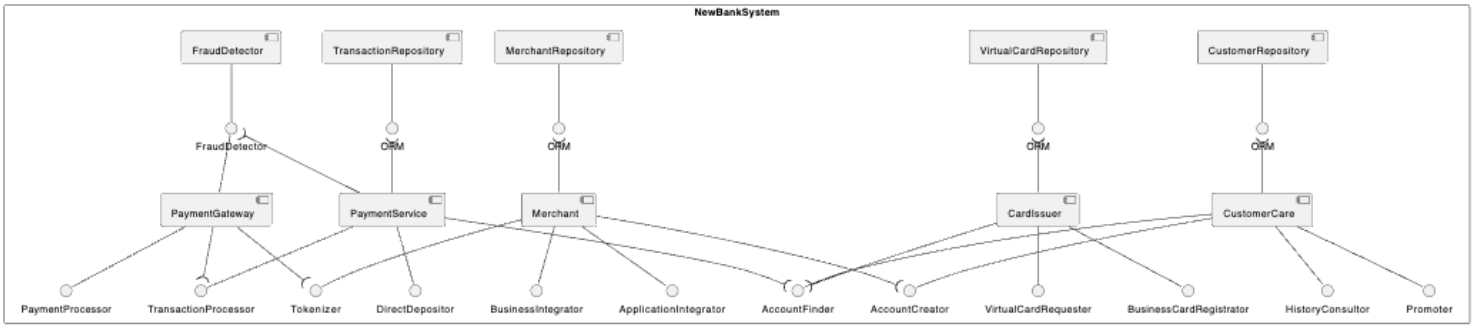
- Gestion des Frais de Transaction :

Description : Le système calcule et gère les frais de transaction associés à chaque transaction, assurant une facturation précise.

Diagramme de Composants :

Lien pour la version svg plus claire : architecture.svg

Pour démarrer notre conception architecturale, nous avons commencé par un diagramme de composants dans lequel nous avons essayé d'identifier les principaux composants métier qui entreront en jeu. Nous considérons ces composants comme les briques de notre compréhension de la logique métier que nous devrons mettre en œuvre.
C'est donc sur cette base que nos décisions en matière de conception architecturale émergeront.



Chaque composant est conçu pour jouer un rôle spécifique :

- CustomerCare : Ce composant gérera toute la logique liée à l'intégration des clients dans notre banque sans numéraire, de la création d'un compte à l'affichage de l'épargne et de l'historique des transactions.

- CardIssuer : Ce composant contiendra la logique d'émission de cartes virtuelles par la banque à ses clients, et permettra aux entreprises d'enregistrer différents types de cartes virtuelles qui pourraient leur servir à

fidéliser leurs clients, par exemple des cartes-cadeaux, des cartes de restaurant, etc.

- Merchant : Ce composant assurera l'intégration des entreprises dans notre banque sans numéraire. Les entreprises qui souhaitent utiliser notre sdk, par exemple, devront avoir leur application enregistrée dans notre système. De même, les entreprises disposent d'une panoplie d'options diverses en ce qui concerne leurs comptes.

- FraudDetector : Comme son nom l'indique, ce composant sera chargé de détecter les activités suspectes et les tentatives de fraude. Ce composant sera utilisé par d'autres composants dans divers scénarios.

- PaymentService : Ce composant gère la logique du service transactionnel de base fourni par la banque. Il traite les demandes de paiement, qu'il s'agisse d'un débit ou d'un crédit d'un compte, gère l'autorisation et la logique de base des transactions et Il utilise le composant de détection des fraudes FraudDetector dans son processus..

- PaymentGateway : Cette composante est la partie exceptionnelle de notre système. C'est la variante sur laquelle nous allons travailler et elle gère toute la logique du traitement des paiements effectués à l'aide de notre sdk. Il sera chargé de contacter le réseau CreditCardNetwork pour la validation des cartes, de déduire les frais de transaction et d'utiliser le mécanisme de tokenisation fourni lors de l'intégration du merchant pour authentifier et crypter ce trafic. Il utilisera également le détecteur de fraude.

- Les repositories représentent des composants responsables de la gestion et du stockage des données.

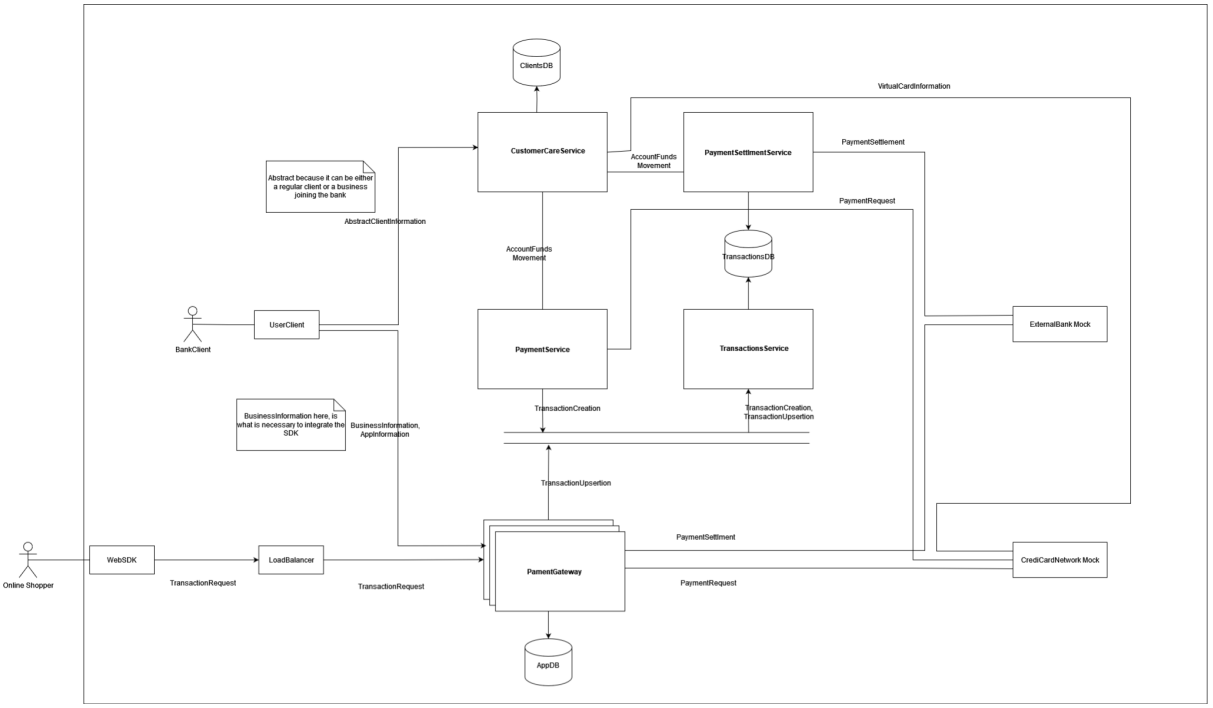
Hypothèses

- Les cartes virtuelles sont générées par les CCN, et par conséquent c’est le CCN qui se chargera de vérifier si un client possède une carte assignée à notre système NewBank pour nous appeler chez nous.
- La passerelle de paiement se charge de la demande d'autorisation et du déclenchement du règlement des fonds après la transaction.
- L'autorisation génère un jeton d'autorisation de la banque cliente récupéré par le CCN et conservé par la passerelle de paiement, puis utilisé pour les règlements de fonds.
- C'est lors de la demande de règlement, que les frais sont précisés pour nous être ensuite restitués.

- La demande de règlement des fonds est envoyée avec le jeton d'autorisation à la banque commerçant/acquéreur par la passerelle. Cette banque est chargée de demander le transfert de fonds à la banque du client à l'aide de ce jeton.

Proposition d'architecture du système :

Nous avons essayé de diviser le diagramme de composants défini précédemment en différents services. Voici le résultat de cette opération :
Lien svg plus clair : [diagramme-archi]



Pour notre première tentative de conception architecturale, nous avons choisi une architecture orientée services pour que chaque service délimite un domaine clair et ait une responsabilité claire pour répondre à des besoins métiers spécifiques et cohésifs de la banque en ligne.

Chaque service dispose d'un périmètre pour se développer indépendamment des autres services. Voici une description du rôle et des cas d'utilisation de chaque service ainsi que ses interfaces externes :

- CustomerCareService : Ce service sera chargé de tout ce qui concerne les relations avec les clients. Il créera des comptes d'utilisateurs, des comptes d'entreprises, gérera les mises à jour directes de leurs fonds et demandera des cartes de crédit virtuelles pour ces clients à un service externe : le réseau de cartes de crédit (mock).

  Interface externes :
  /POST /api/customers
  /GET /api/customers/:id
  /POST /api/customers/:id/virtualcard

  /PUT /api/customers/:id/movetosavings
  /PUT /api/customers/:id/addfunds
  /PUT /api/customers/:id/deducefunds
  /PUT /api/customers/:id/reservefunds

- PaymentService : Ce service est chargé d'autoriser la transaction réelle et la demande de paiement émanant d'une banque ou de notre réseau de cartes de crédit. Ce service effectue les contrôles de fraude nécessaires, par exemple, et les contrôles de fonds, car il demande les informations sur le client au service clientèle avant d'approuver la transaction. Il envoie également l'événement à une queue afin de créer la transaction et de la stocker dans la base de données par le service dédié qui écoute sur la queue.

  Interface externes :
  /POST /api/payments/authorize

- TransactionsService : Ce service s'occupera des insertions et des mises à jour de notre base de données de transactions. Il effectuera beaucoup d'écritures et sera purement utilisé pour nous permettre d'effectuer des autorisations plus rapides dans d'autres services sans se soucier de la latence d'écriture directe et concurrente dans la base de données par ses services (PaymentService et PaymentGateway).

  Interface externes :
  Il n'aura pas d'interface externe mais il va plutôt écouter sur la queue pour manipuler la db des transactions.

- PaymentSettlementService : Ce service s'occupera des mouvements de fonds et du règlement de toute transaction en cours qui a été approuvée. Il recevra les demandes de transfert de fonds de nos comptes clients vers d'autres comptes bancaires, et inversement, il procédera également à la

demande de règlement de fonds à partir d'autres comptes bancaires si ce sont nos clients qui recevront les fonds.

Interface externes :
/POST /api/settlements/deposit
/POST /api/settlements/settle

- PaymentGatewayService : Ce service a pour l'instant deux responsabilités. La première est d'intégrer une entreprise et son application, de générer le jeton et de permettre l'utilisation de notre sdk en l'utilisant. La seconde est le traitement des transactions effectuées par son intermédiaire, c'est-à-dire le calcul des frais, l'appel au CCN pour l'autorisation de paiement du client et le déclenchement du règlement des fonds après l'autorisation.
  Il n'est pas divisé en deux services, car la seconde responsabilité dépend étroitement de la première. Pour pouvoir procéder à une transaction, le service doit vérifier à chaque fois que le jeton est valide, que le commerçant est dans notre système, etc...

  Interface externes :
  /POST /api/gateway/businesses
  /GET /api/gateway/businesses/:id
  /POST /api/gateway/businesses/:id/application
  /POST /api/gateway/pay

- ExternalBank Mock : Ce service imitera un service bancaire typique : il autorisera les paiements et demandera les règlements de fonds qui sont déclenchés.

  Interface externes :
  /POST /api/bank/settle

- CreditCardNetwork Mock : Ce service simulera un CCN, vérifiant la validité d'une carte et renvoyant un jeton d'autorisation si le client est autorisé dans le cas d'une transaction approuvée, et émettant des cartes de crédit virtuelles.

  Interface externes :
  /POST /api/ccn/authorize

- **Loadbalancer** : Ce composant supplémentaire fonctionne seul et fournit une sorte de proxy inverse pour accéder à la passerelle de paiement. Ce composant est placé devant la passerelle et répartit la charge sur ses différentes instances qui sont redondantes. Il sera chargé de limiter l'utilisation en cas de demandes élevées et étranges de la part d'un certain demandeur. Il

réacheminera également les demandes vers une instance opérationnelle s'il détecte que l'autre instance en cours d'exécution a échoué.

Interface externes :
/POST /api/gateway/businesses
/GET /api/gateway/businesses/:id
/POST /api/gateway/businesses/:id/application
/POST /api/gateway/pay

Choix technologiques :

- Le développement des services se fera en Java Spring Boot pour deux raisons : la familiarité de la stack technologique pour l'équipe et la robustesse de java (multithreading, strong typing,..).
- La base de données de transactions sera en Cassandra pour permettre un flux d'écriture intensif.
- Les autres bases de données seront en Postgres vu qu'elles seront sollicitées autant en écriture qu'en lecture.
- La technologie de la file d'attente n'est pas tout à fait décidée, car le choix de la file d'attente elle-même doit encore être rediscuté. RabbitMQ semble être plus valable car il permet le retry et est plus intégré à l'écosystème Java.

Limitation des risques identifié

En commençant par l'extensibilité, nous avons utilisé REST pour que la fonctionnalité de notre passerelle soit facilement exposée par le biais d'un kit de développement logiciel.
L'exposition d'interfaces externes simples, conformes aux normes REST et fournissant des points d'accès à la passerelle basés sur des cas d'utilisation simples, facilite l'extension et l'intégration de SDK à la passerelle.

En ce qui concerne la robustesse, nous avons vu que, d'un point de vue fonctionnel, il s'agit de gérer les entrées erronées et les erreurs, qu'elles soient intentionnelles ou non. Mais d'un point de vue architectural, il est indispensable d'atténuer les problèmes tels que les pics de charge élevés et les essais d'effondrement du service. Soit en limitant l'utilisation, soit en la réacheminant vers d'autres instances moins débordées, soit en refusant simplement les demandes entrantes qui tentent d'affliger le système et d'arrêter le service.
Pour ce faire, nous avons ajouté un proxy inverse devant notre passerelle de paiement. Ce composant joue le rôle de gardien, limiteur de débit et équilibreur de charge.

Il convient de noter que pour parvenir à faire ce qui a été mentionné précédemment, nous allons faire recours à une redondance active-active de la passerelle de paiement derrière le proxy inverse.

SDK

Une clé api sera fournie à chaque client de l’SDK. Ce token JWT sera incorporé dans l'en-tête "Authorization" de chaque requête API envoyée au SDK. Si le token s'avère invalide, la gateway ne recevra aucune requête. En revanche, si le token JWT est valide, les transactions se déroulent sans accroc.

Concernant le cryptage des informations de carte, nous avons choisi d'employer l'algorithme RSA. Plus précisément, la gateway génère une paire de clés RSA, à savoir la clé privée et la clé publique. Le SDK utilise la clé publique pour chiffrer les données de carte avant de les transmettre à la passerelle. Ceci garantit une transmission sécurisée des informations sensibles.