**Continuous Architecture in Practice: Software Architecture in the Age of Agility and DevOps**

**Додаток A**

**Вивчення проблеми**

У даному додатку описано конкретні приклади ілюстрації архітектурних методів.

Для прикладу обрано хмаро орієнтований додаток у сфері торгового фінансування. Торговельне фінансування - це сфера фінансового бізнесу, яка лежить в основі глобальної торгівлі фізичними товарами. Ідея торгового фінансування полягає у використанні низки фінансових інструментів для зменшення ризику неотримання платежів продавцями товарів, а також ризику неотримання товарів покупцями, які заплатили за них, але не отримали. Воно передбачає, що фінансові посередники (зазвичай банки) надають покупцям і продавцям гарантії того, що транзакція буде застрахована фінансовою установою за певних умов.

Повністю реалістичну систему торгового фінансування не описано, тому зосереджено дослідження на одній конкретній частині торгового фінансування - акредитивах - і надано загальний опис архітектури.

**Представляємо TFX**

Акредитив - це загальновживаний інструмент торгового фінансування, який зазвичай використовується компаніями, що беруть участь у міжнародній торгівлі, для забезпечення оплати товарів або послуг між покупцем і продавцем. Акредитиви стандартизовані за допомогою конвенцій, встановлених Міжнародною торговою палатою (в її керівництві UCP 600).

Крім покупця і продавця товару, в акредитивній угоді також беруть участь щонайменше два посередники, як правило, банки, один з яких діє від імені покупця, а інший - від імені продавця. Банки виступають гарантами в угоді, гарантуючи, що продавець отримає оплату за свій товар після його відвантаження (і що покупець отримає компенсацію, якщо він не отримає товар).

Історично, та й сьогодні, процеси торговельного фінансування є трудомісткими та документоорієнтованими, що передбачає ручну обробку великих, складних контрактних документів. У даному кейсі фіктивна фінтех-компанія побачила потенціал для підвищення ефективності та прозорості шляхом оцифрування значної частини процесу і планує створити онлайн-сервіс під назвою Trade Finance eXchange (TFX), який забезпечить цифрову платформу для випуску та обробки акредитивів. Стратегія компанії полягає в тому, щоб знайти велику міжнародну фінансову установу в якості свого першого клієнта, а потім продати доступ до платформи всім клієнтам цієї установи, які займаються торговим фінансуванням.

У довгостроковій перспективі планується запропонувати TFX як білу етикетку (ребрендингове) рішення для великих міжнародних банків, що займаються торговим фінансуванням. Отже, в короткостроковій перспективі це буде система для одного користувача, але з часом, якщо ця стратегія буде успішною, вона повинна буде перетворитися на багатокористувацьку систему.

Мета TFX - дозволити організаціям, які займаються випискою та використанням акредитивів у процесі торгового фінансування, взаємодіяти ефективно, прозоро, надійно та безпечно.

Система TFX будується як хмарна платформа "Програмне забезпечення як послуга" (SaaS). Для цілей даного прикладу використовується Amazon Web Services (AWS) як хмарного провайдера для системи. Обрано AWS через її широке розповсюдження в індустрії.

**Архітектурний опис**

У цьому додатку представлено загальний опис архітектури першої версії платформи TFX.

Опис архітектури представлено у вигляді набору видів, взятих з набору Розанського та Вудса (R&W), кожен з яких ілюструє окремий аспект архітектури. Погляди, що використовуються в цій моделі, є наступними:

* Функціональний: Функціональна структура системи під час виконання, що показує функціональні компоненти системи під час виконання, інтерфейси, які надає кожен компонент, а також залежності та взаємодію між елементами. (Функціональні компоненти - це ті, що виконують дії, пов'язані з проблемною областю, а не з інфраструктурою, такі як розгортання, оркестрування контейнерів та моніторинг).
* Інформація: Інформаційна структура системи, що показує ключові інформаційні об'єкти та їх взаємозв'язки, а також будь-які значущі моделі стану або життєвого циклу для ілюстрації того, як вони обробляються в системі.
* Розгортання: Визначає середовище виконання, в якому система буде працювати, включаючи інфраструктурну платформу, яка потрібна системі (обчислювальні потужності, сховище, мережа), вимоги до технічного середовища для кожного вузла (або типу вузла) в системі, а також відображення програмних елементів у середовище виконання, яке буде їх виконувати.

**Функціональний вигляд**

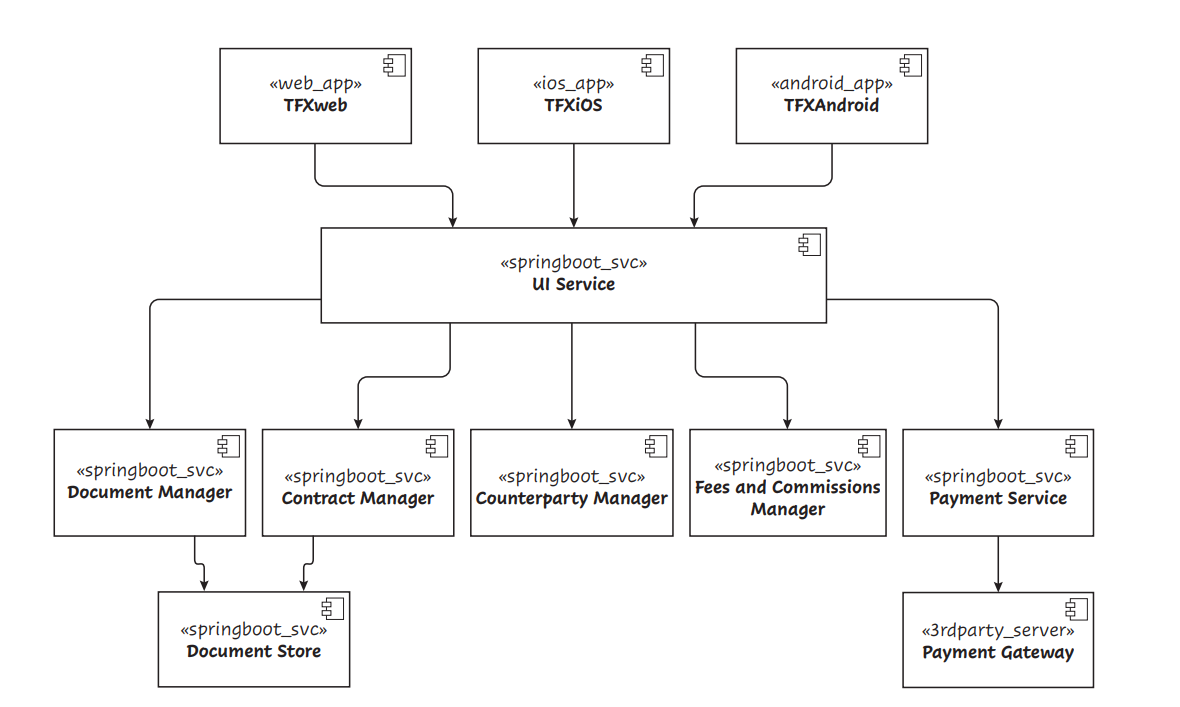
Функціональна структура системи проілюстрована діаграмою компонентів уніфікованої мови моделювання (UML) на рисунку A.1.

На рисунку A.1 використовується проста нотація у вигляді рамок і ліній, натхненну нотацією діаграми компонентів UML2.

- Прямокутні форми є функціональними компонентами.

- Лінії між компонентами показують, які компоненти викликають операції над іншими компонентами.

- Дрібніший текст всередині фігур, між дужками, - це стереотипи. Стереотипи використовуються для позначення типу кожного компонента. Ми використовуємо цей механізм, щоб вказати, як компонент буде реалізований технічно



*Рисунок A.1 Функціональна структура*

У таблиці неформально пояснено характеристики кожного компонента природною мовою. Також використоно термін API (інтерфейс прикладного програмування) для позначення міжкомпонентного інтерфейсу, що забезпечує виклик операцій, не обов'язково визначаючи, яка саме технологія API має бути використана для кожного з них.

|  |  |
| --- | --- |
| **Компонент: TFXWeb** | |
| Опис | Звичайний веб-інтерфейс користувача, призначений для запуску в настільному браузері; пишеться з використанням набору технологій JavaScript, HTML і каскадних таблиць стилів (CSS). |
| Обов'язки | Забезпечити орієнтований на завдання інтерфейс для запиту, створення, моніторингу, виконання та управління акредитивами та іншими артефактами акредитивів.  Надайте інтерфейс з урахуванням ролей для різних типів користувачів - співробітників банку та клієнтів. |
| Надані інтерфейси | Забезпечує людський інтерфейс через браузер. |
| Взаємодія | Часто звертається до служби інтерфейсу користувача, щоб отримати всю необхідну інформацію та виконати всі необхідні оновлення. |
| **Компоненти: TFXiOS та TFXAndroid** | |
| Опис | Нативні мобільні користувацькі інтерфейси, оптимізовані для використання на мобільних пристроях (як з маленьким, так і з великим екраном). |
| Обов'язки | Забезпечити орієнтований на завдання інтерфейс для запиту, створення, моніторингу, виконання та управління акредитивами та іншими артефактами акредитивів.  Надайте інтерфейс з урахуванням ролей для різних типів користувачів - співробітників банку та клієнтів банку. |
| Надані інтерфейси | Забезпечує людський інтерфейс через мобільний пристрій. |
| Взаємодія | Часто звертається до служби інтерфейсу користувача, щоб отримати всю необхідну інформацію та виконати всі необхідні оновлення. |
| **Компонент: Сервіс користувацького інтерфейсу** | |
| Опис | Надає набір API для підтримки потреб користувацьких інтерфейсів програми.  Діє як фасад для служб домену, тому є більш функціональним, ніж простий API-шлюз. |
| Обов'язки | Надайте набір доменних сервісів, спеціалізованих для потреб користувацьких інтерфейсів, щоб забезпечити отримання всіх доменних сутностей з системи та їх контрольоване оновлення. |
| Надані інтерфейси | Надає API до служб домену. |
| Взаємодія | Викликає служби домену для надання доступу до об'єктів домену та пов'язаних з ними операцій.  Активно викликається елементами TFXWeb, TFXiOS та TFXAndroid. |
| **Компонент: Менеджер документів** | |
| Опис | Сервіс, який надає можливість виконувати операції над великими, неструктурованими документами за допомогою інших елементів, зокрема користувацьких інтерфейсів. |
| **Компонент: Менеджер документів** | |
| Обов'язки | Приймаються файли у різних форматах (текст, JSON, PDF, JPEG).  Автоматично генерувати метадані з файлів.  Дозволяти прикріплювати метадані до файлів. Дозволяти запитувати набір документів для повернення списків відповідних документів, включно з індексами ключових слів природною мовою для нетекстових форматів, таких як PDF та JPEG.  Дозволити пошук певних файлів документів. |
| Надані інтерфейси | Надає API, що дозволяє здійснювати два аспекти управління документами:  1) імпорт та вилучення великих файлів документів у різних форматах за одну або кілька операцій;  2) управління набором документів за допомогою метаданих, що дозволяє тегувати, шукати та вилучати списки документів, що відповідають складним критеріям (наприклад, датам та тегам). |
| Взаємодія | Використовує сховище документів для зберігання, індексації та керування документами, що імпортуються. |
| **Компонент: Менеджер контрагентів** | |
| Опис | Керує та забезпечує контрольований доступ до інформації, що описує:  - Контрагенти - банки, а також покупці та продавці, яких підтримують банки.  - Банківські рахунки - банківські рахунки, що використовуються банками та їхніми клієнтами.  - Користувачі - інформація про співробітників банків та їхніх клієнтів, які є користувачами системи, що не може зберігатися в системах авторизації та автентифікації. |
| Обов'язки | Контрольоване створення, зміна та видалення контрагентів.  Контрольоване створення, модифікація та видалення банківських рахунків.  Контрольоване створення, модифікація та видалення користувачів.  Контрольоване створення, модифікація та видалення авторизацій користувачів.  Постійне зберігання та підтримка цілісності всієї цієї інформації.  Простий пошук інформації про контрагента, банківський рахунок, користувача та авторизацію.  Складні запити на інформацію про користувача та авторизацію.  Перевірка запитів на основі предикатів, пов'язаних з авторизацією (наприклад, "is-authorized-to-release"). |
| Надані інтерфейси | Надає API для використання внутрішніми елементами системи. |
| Взаємодія | Ні. |
| **Компонент: Контрактний менеджер** | |
| Опис | Надає можливість створювати та керувати об'єктами контрактів, що представляють акредитиви (L/C). |
| Обов'язки | Створення та управління об'єктами контрактного домену.  Управління станом об'єкта контракту (щоб він у будь-який момент часу перебував у дійсному, чітко визначеному стані життєвого циклу і переходив між станами лише відповідно до визначеної моделі стану).  Отримання необхідних дозволів на об'єкти контрактного домену (які є численними і можуть бути складними). |
| Надані інтерфейси | Надає API, що дозволяє створювати, керувати, затверджувати та запитувати об'єкти контрактів. |
| Взаємодія | Зберігання та пошук файлів документів (наприклад, PDF) залежить від Диспетчера документів.  Розрахунок комісійних залежить від менеджера по тарифам і комісіям, який розраховує комісії в рамках процесу створення акредитиву.  Залежить від менеджера по роботі з контрагентами щодо обліку операцій на клієнтських рахунках, що виникають у процесі погашення акредитивів. |
| **Компонент: Менеджер по зборам та комісійним винагородам** | |
| Опис | Забезпечує розрахункові операції, що дозволяють розраховувати збори та комісії на вимогу.  Дозволяє визначати та керувати правилами розрахунку комісійних та зборів. |
| Обов'язки | Забезпечити контрольоване створення, модифікацію та видалення правил тарифів і комісій.  Постійне зберігання та підтримання цілісності тарифів і комісійних правил.  Дозволяє розраховувати вартість зборів і комісій на вимогу. |
| Надані інтерфейси | Надає API для визначення правил комісійних та зборів.  Надає API, що дозволяє розраховувати збори та комісії для конкретних транзакцій. |
| Взаємодія | Ні. |
| **Компонент: Платіжний сервіс** | |
| Опис | Надає можливість створювати та керувати об'єктами платіжного домену. |
| Обов'язки | Дозволяє створити платіж, затвердити його, якщо потрібно, відправити в мережу та перевірити його стан. |
| Надані інтерфейси | Надає API для створення, управління та запиту платежів. |
| Взаємодія | Надсилає інструкції платіжному шлюзу. |
| **Компонент: Сховище документів** | |
| Опис | Сховище документів для неструктурованих даних документів. |
| Обов'язки | Забезпечити контрольоване створення, модифікацію та видалення неструктурованих даних документа з відповідними метаданими.  Забезпечити індексацію збережених документів.  Забезпечити тегування збережених документів.  Дозволити пошук документів за ідентифікатором.  Дозволяє ідентифікувати документ за метаданими та пошуком за тегами.  Дозволяє ідентифікувати документ за допомогою повнотекстового пошуку. |
| Надані інтерфейси | Надає API для зберігання, пошуку, запитів та управління документами. |
| Взаємодія | Ні. |
| **Компонент: Платіжна система Gatewa** | |
| Опис | Мережевий сервісний інтерфейс для забезпечення стандартизованого та спрощеного доступу до платіжних мереж через стандартний інтерфейс. |
| Обов'язки | Зведіть деталі доступу до платіжної мережі до стандартної моделі.  Забезпечити можливість для інших елементів системи виконувати стандартні операції платіжної мережі (оплата, скасування, звітність). |
| Надані інтерфейси | Надає API, який забезпечує простий і стандартизований доступ до операцій платіжної мережі в незалежний від платіжної мережі спосіб. |
| Взаємодія | Жодна з них не має внутрішнього доступу до однієї або декількох зовнішніх платіжних мереж через власні інтерфейси. |

**Сценарії використання**

Загальні взаємодії між елементами системи, які очікуються в архітектурі, виглядають наступним чином:

- Для всіх взаємодій з рештою системи користувацькі інтерфейси звертаються до служби інтерфейсу користувача (UI Service). Сервіс надає набір API, оптимізованих для потреб користувацьких інтерфейсів, і таким чином виступає в ролі фасаду для решти системи.

- Коли користувацьким інтерфейсам потрібно маніпулювати ключовими доменними сутностями в системі, вони викликають API сервісів інтерфейсу користувача, які, в свою чергу, викликають Менеджер контрагентів, Менеджер комісій, Менеджер документів, Менеджер договорів та Платіжний сервіс.

- Більш складні операції реалізуються у відповідному доменному сервісі (наприклад, Менеджер контрактів, який відповідає за створення та доставку акредитивного контракту, що є складним процесом).

- Враховуючи необхідність деяких довготривалих операцій, деякі служби домену повинні надавати асинхронні, а також прості транзакційні API-інтерфейси.

- Коли потрібно імпортувати та обробити великі документи (наприклад, обробка природної мови), Служба інтерфейсу надсилає запит до Диспетчера документів, який відповідає за це завдання.

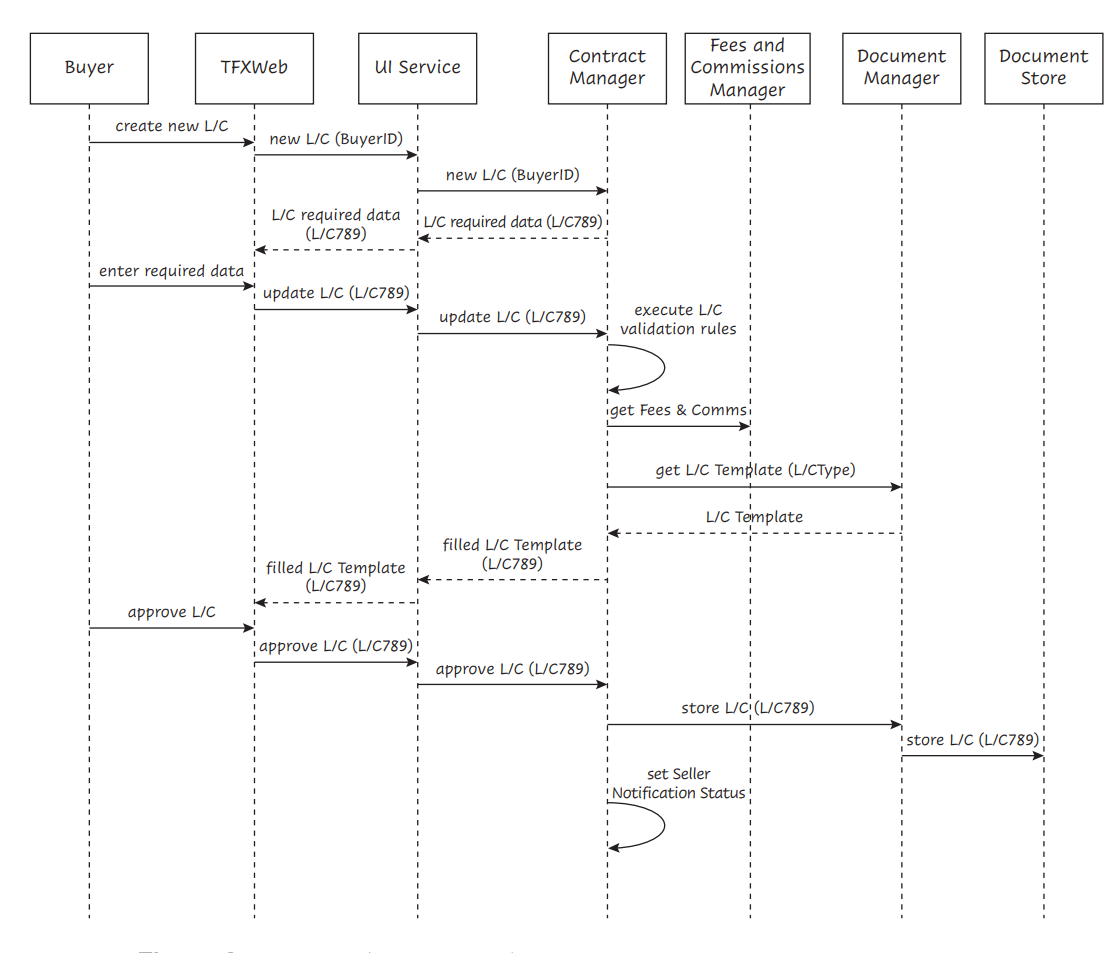
Ці операції за своєю суттю є асинхронними, і коли запит робиться через API Імпортеру документів, повертається ідентифікатор документа. Імпортер документів записує результати процесу в Диспетчер документів, ідентифікований за цим ідентифікатором, і користувач може запитувати статус документа, використовуючи API та ідентифікатор документа. Ідентифікатор документа також використовується для багатоетапних операцій, таких як завантаження документа по частинах через його розмір.

Дуже ймовірно, що система TFX також відкриє свої API для зовнішніх бізнес-партнерів. Це потенційно може призвести до різних потоків доходів і бізнес-планів.

Для наочного прикладу того, як система обробляє запит, розглянемо погодження видачі контракту на акредитив агентом, який є співробітником банку-емітента. Процес проілюстровано діаграмою взаємодії на рисунку А.2.

Для діаграм взаємодії в даному кейсі використано спрощену версію нотації UML. У верхній частині діаграми зображено елементи системи виконання, які взаємодіють між собою, а вертикальні лінії, що відходять від них вниз по сторінці, є їхніми "лініями життя", показуючи, де вони отримують та надсилають повідомлення, а час, що проходить між ними, відображається вертикальною відстанню. Суцільні стрілки - це певні зв'язки між елементами, а їхні мітки вказують на характер взаємодії. Пунктирні лінії - це відповіді елементів на запит або стимул. Для позначення типу взаємодії ми використали випадкові стереотипи.

Перший сценарій, проілюстрований діаграмою взаємодії на рисунку А.2, показує, як створюється новий акредитив з точки зору покупця.



*Рисунок A.2 Створення акредитивного сценарію-покупець*

Взаємодія на Рисунку A.2 виглядає наступним чином:

1. Покупець просить створити новий акредитив через TFXWeb.

2. Сервіс інтерфейсу користувача передає запит менеджеру контрактів з ідентифікатором покупця.

3. Контрактний менеджер визначає дані, необхідні для акредитива, на основі ідентифікатора покупця - це передбачає, що система може бути налаштована відповідно до характеристик покупця. Контрактний менеджер також присвоює запиту унікальний ідентифікатор (789).

4. Запитувані дані відображаються в інтерфейсі користувача, і покупець заповнює всі дані (наприклад, дані продавця, умови договору)

5. Покупець вносить всі необхідні дані, які потім передаються контрактному менеджеру

6. Менеджер контрактів виконує правила валідації, включаючи збори та комісії, через Менеджера зборів та комісій.

7. Менеджер контрактів запитує шаблон акредитива у Менеджера документів.

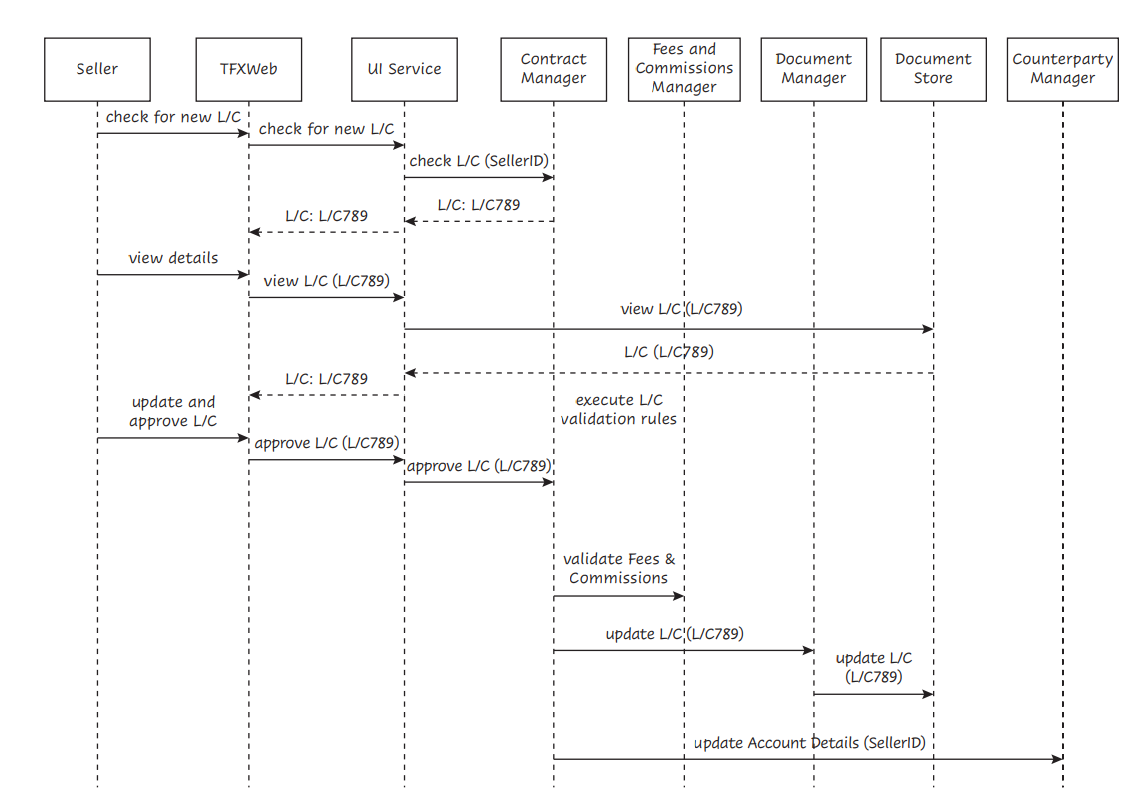
8. Шаблон акредитива, заповнений усіма необхідними даними, надається покупцю для остаточної перевірки.

9. Покупець затверджує акредитив, який передається Контрактному менеджеру.

10. Менеджер контрактів просить Диспетчера документів зберегти акредитив. Диспетчер документів витягує всі необхідні метадані та зберігає акредитив у сховищі документів.

11. Менеджер контрактів встановлює статус сповіщення продавця, щоб вказати, що для продавця відкрито акредитив

Той самий сценарій, проілюстрований з точки зору продавця, показаний на Рисунку А.3:



*Рисунок A.3 Створення сценарію акредитива-продавець*

1. Продавець просить TFX перевірити наявність нового акредитива через TFXWeb.

2. Служба інтерфейсу передає запит менеджеру контрактів з ідентифікатором продавця.

3. Контрактний менеджер знаходить новий акредитив, відкритий покупцем, і передає цю інформацію Продавцю.

4. Продавець просить показати повний документ акредитива.

5. Служба інтерфейсу користувача передає запит до Сховища документів з ідентифікатором акредитива.

6. Продавець затверджує акредитив, який Служба користувацького інтерфейсу передає Контрактному менеджеру.

7. Менеджер контрактів виконує правила валідації, включаючи валідацію зборів та комісій за допомогою Менеджера зборів та комісій. Для простоти ми не показуємо жодних винятків або циклів зворотного зв'язку, які були б присутні в більш складному сценарії; ми представляємо тут лише найпростіший сценарій "щасливого дня".

8. Менеджер контрактів просить Менеджера документів зберегти оновлений контракт з інформацією про схвалення продавця в Сховищі документів. Диспетчер документів оновлює відповідні метадані для контракту.

9. Менеджер контрактів просить Менеджера контрагентів оновити реквізити рахунку продавця.

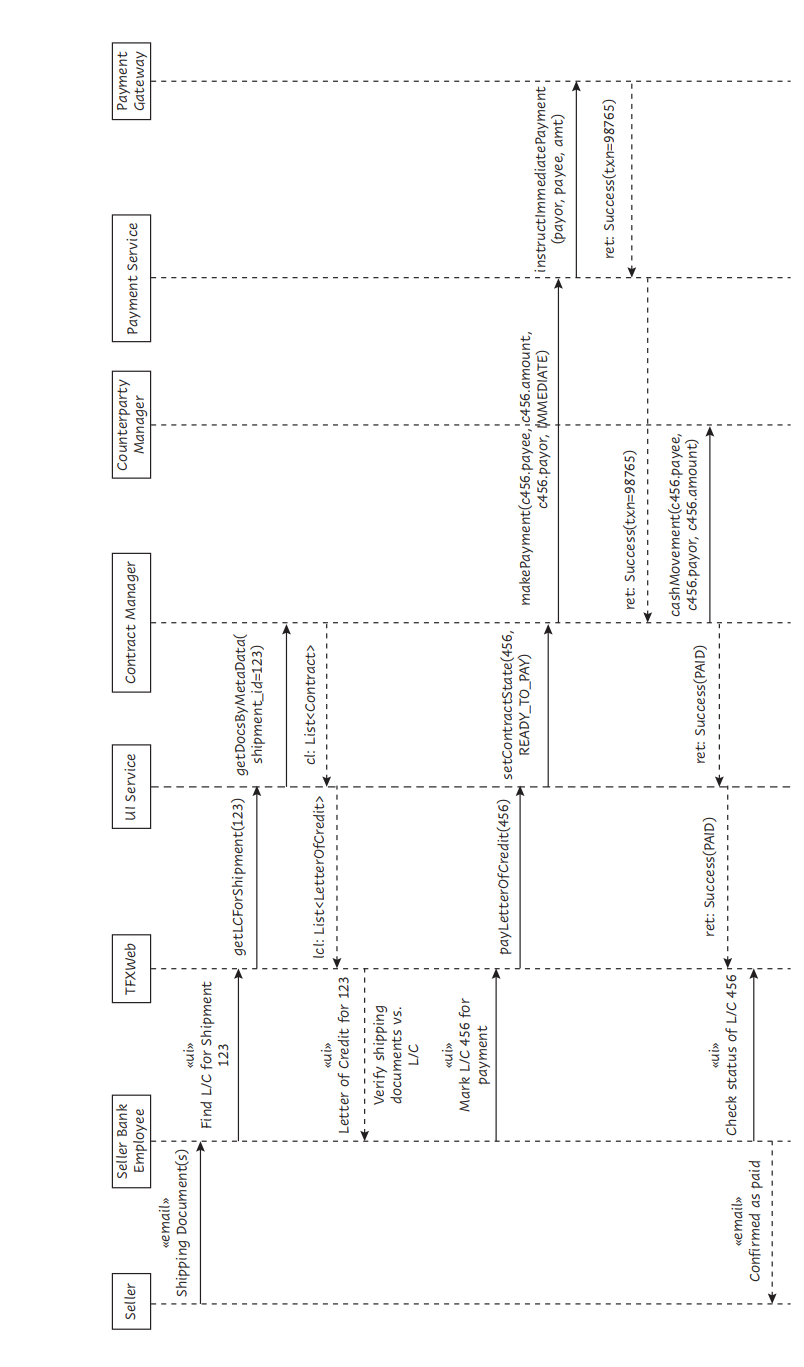
Другим прикладом є прибуття товару, що спричиняє платіж постачальнику відповідно до умов акредитивного договору. Цей процес проілюстровано на рисунку А.4:

1. Продавець повідомляє співробітника банку про відвантаження партії товарів, гарантованих за акредитивом, можливо, через коносамент електронною поштою.

2. Співробітник входить в інтерфейс користувача TFXWeb і шукає відповідний акредитив (можливо, використовуючи номер відправлення).

3. Інтерфейс TFXWeb викликає службу інтерфейсу користувача, щоб забезпечити простий інтерфейс для цієї операції.

4. Служба інтерфейсу викликає Менеджер контрактів, передаючи йому критерії пошуку, а Менеджер контрактів повертає інформацію про всі контракти, які відповідають критеріям.



*Рисунок A.4 Оплата акредитиву*

5. Інтерфейс TFXWeb відображає повернуті контракти і дозволяє користувачеві перевірити їх зміст. Користувач знаходить відповідний контракт і перевіряє умови, які вимагає акредитив для здійснення платежу.

6. Користувач підтверджує, що задоволений виконанням необхідних умов (перевіряючи надані документи і, можливо, уточнюючи деякі деталі телефоном або електронною поштою). Частини цього процесу мають потенціал для автоматизації (див. Розділ 8, "Архітектура програмного забезпечення та нові технології"), який ми тут не показуємо.

7. Задоволений, користувач обирає контракт в інтерфейсі користувача TFXWeb і позначає його як готовий до оплати.

8. Інтерфейс користувача TFXWeb викликає службу інтерфейсу користувача, яка звертається до менеджера контрактів, щоб повідомити про цю зміну статусу за допомогою виклику API.

9. Менеджер контрактів отримує деталі контракту і створює платіжну вимогу на переказ коштів з банківського рахунку на реквізити одержувача у зазначеній у контракті валюті та сумі, відповідно до реквізитів, вказаних у контракті. Потім він телефонує до платіжної служби, щоб доручити здійснити платіж, і встановлює статус контракту на "запит на платіж".

10. Платіжний сервіс отримує та зберігає платіжний запит. Потім він створює платіжну інструкцію для платіжного шлюзу і надсилає її на платіжний шлюз.

11. Платіжний шлюз намагається здійснити платіж і надсилає платіжному сервісу статус успіху або невдачі.

12. Якщо платіж пройшов успішно, Платіжний сервіс позначає платіж як завершений і телефонує до Контрактного сервісу, щоб повідомити про цю подію.

13. Менеджер контрактів оновлює рахунки платника та одержувача через Менеджера контрагентів, щоб вказати рух грошових коштів за транзакцією.

14. Служба контрактів позначає контракт як оплачений і завершений, і успіх повертається до служби інтерфейсу користувача, а потім до служби TFXWeb, де співробітник банку може побачити статус і повідомити продавця

Якщо платіжна інструкція не пройшла з тимчасовою помилкою, Платіжний сервіс повторить спробу кілька разів. Якщо платіжна інструкція не виконується з постійною помилкою або з декількома тимчасовими помилками, Платіжна система позначить запит як невдалий і зателефонує менеджеру контракту, щоб повідомити про цю подію. Після цього Менеджер контрактів позначить контракт на акредитив як такий, що перебуває в стані помилки і потребує оперативної уваги для виправлення платіжної інформації.

**Перегляд інформації**

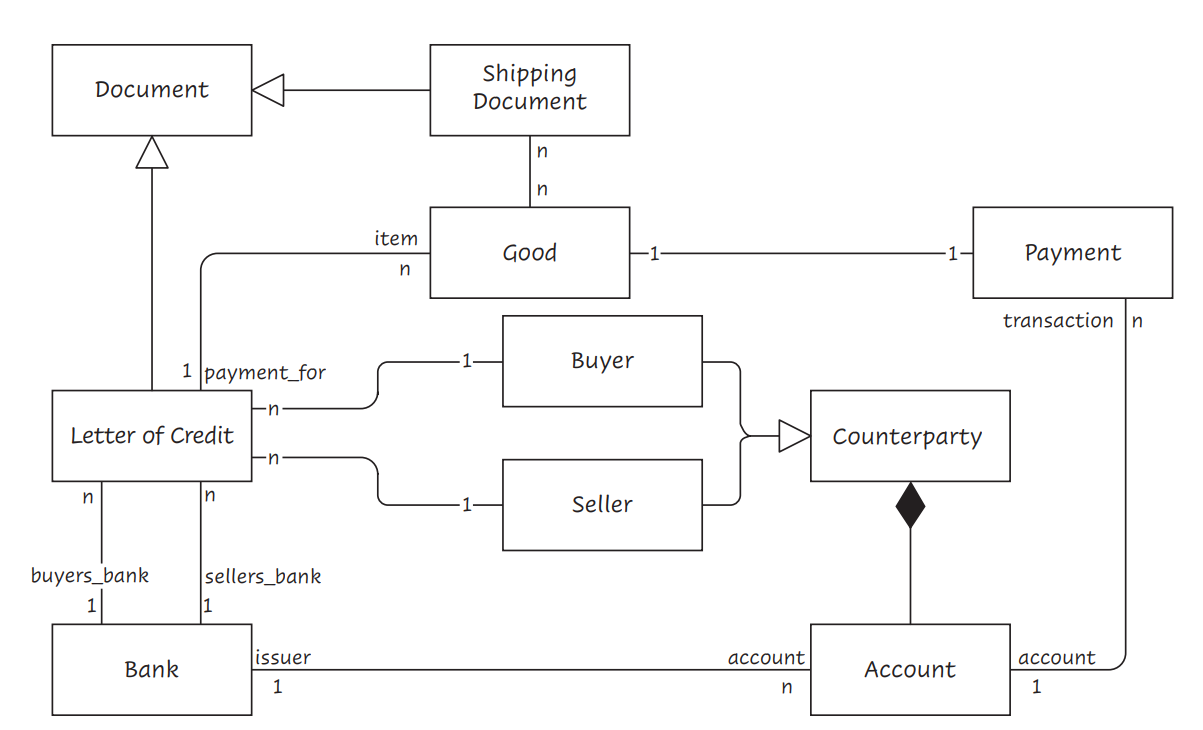
В інформаційному поданні описано ключові структури даних для системи і те, як вони пов'язані одна з одною. Модель даних для системи показана на Рисунку A.5.

Існує багато різних угод для представлення моделей даних, але ми взяли за основу UML2. Прямокутні фігури - це сутності (або "класи"); лінії вказують на зв'язки, а роль кожної сутності вказується на кінці зв'язку, де роль не очевидна, разом з кардинальністю цього кінця зв'язку (1 означає точно 1, а n - між 0 і n). Відношення з відкритим трикутником на одному кінці вказує на відношення узагальнення ("успадкування"), а заповнений ромб вказує на відношення агрегації ("стримування").

Ключовими елементами інформаційної структури в системі є наступні:

- Акредитив: Акредитив є центральною інформаційною одиницею в системі та причиною її існування. Це тип контракту, який усуває ризик того, що покупець певного товару (заявник) не заплатить продавцю товару (бенефіціару).

- Відвантажувальний документ: Документ, який використовується для підтвердження того, що товар було відвантажено, а отже, акредитив на цей товар може бути погашений. Зазвичай він має форму коносамента (або матеріальної накладної).



*Рисунок A.5 Модель представлення даних про інформацію*

- Документ: Узагальнення акредитива та пов'язаних з ним документів, що представляє загальні аспекти документа, які не є специфічними для конкретного типу документа.

- Оплата: Оплата здійснюється продавцю після пред'явлення та перевірки необхідних документів.

- Товар: Представлення товару, який є предметом торгівлі між продавцем і покупцем і який гарантується за допомогою акредитива.

- Покупець і Продавець: Представляють учасників контракту, обидва з яких є контрагентами, що мають клієнтські відносини з банком:

* Продавець - це контрагент, оплата якого гарантується контрактом. У міжнародній торгівлі це, як правило, експортер.
* Покупець - це контрагент, який вимагає укладення контракту, щоб продавець випустив товар до оплати. У міжнародній торгівлі це, як правило, імпортер.
* Банк покупця - це фінансова установа, яка укладає контракт і представляє покупця.
* Банк продавця - це фінансова установа, яка представляє інтереси продавця та консультує його після отримання кредиту.

- Банк: Представляє банки, які беруть участь у процесі акредитива, створюють та погашають акредитив, а також надають своїм клієнтам послуги з ведення рахунків та здійснення платежів.

- Рахунок: Банківські рахунки, пов'язані з контрагентами, що використовуються для здійснення транзакції, які можуть бути відкриті в різних установах, країнах і валютах

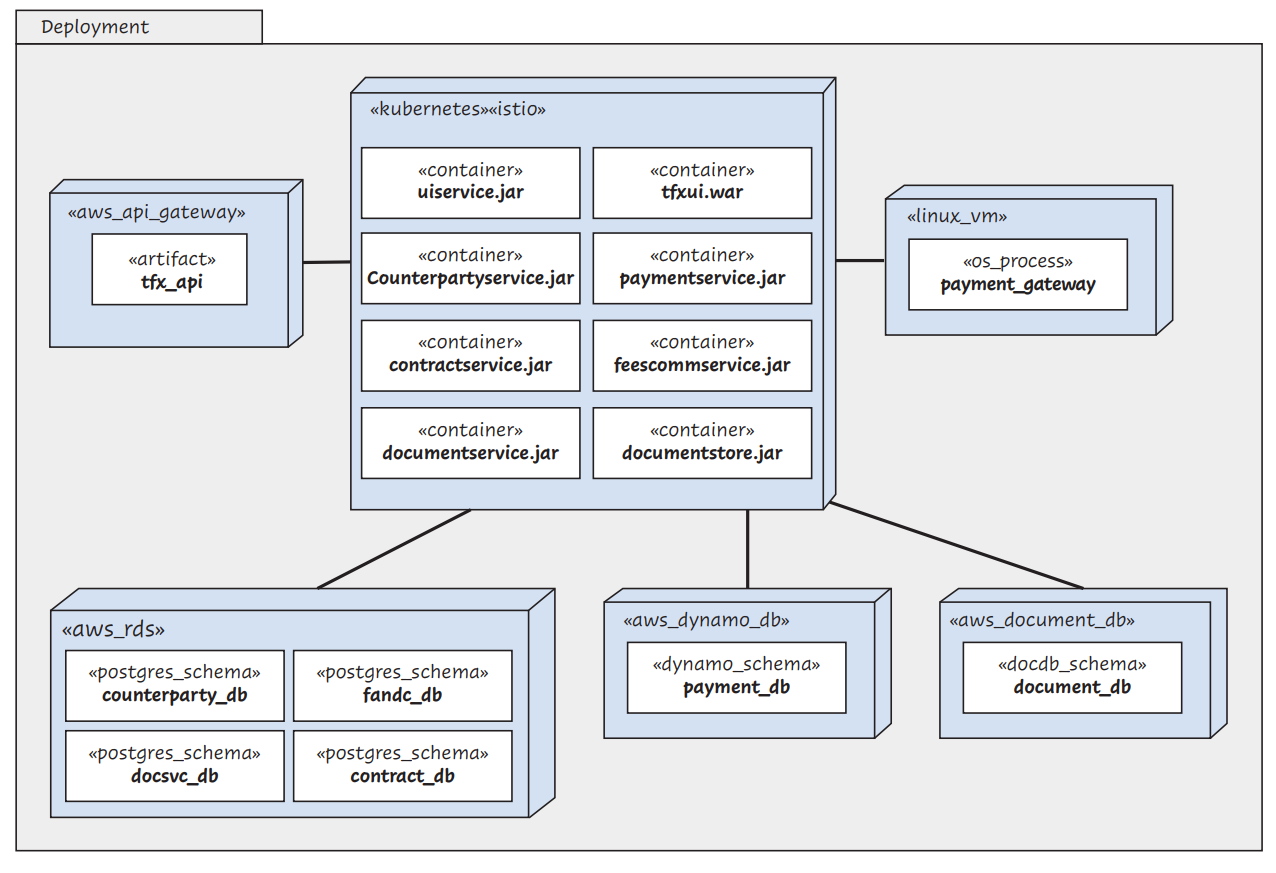
Це досить спрощене уявлення про інформаційну структуру, необхідну для підтримки реальної акредитивної операції, але воно відображає її суть і слугує нашим цілям, уникаючи при цьому надмірної складності.

Такі моделі допомагають пояснити суть предметної області людям, які не знайомі з нею, і є відправною точкою для навчання, а також контекстом, в який можна помістити більш детальні знання в міру їх засвоєння.

Така модель також може стати відправною точкою для створення більш детальної моделі предметної області та універсальної мови, необхідної для доменно-орієнтованого підходу до проектування. Як тільки почнеться реалізація, модель потрібно буде доопрацювати (наприклад, у цьому випадку "покупець" і "продавець" можуть стати ролями для "контрагента", а не самостійними сутностями, і потрібно буде додати відсутні елементи, такі як дані про комісійні та збори).

**Подання розгортання**

Розгортання системи в операційному середовищі описано на діаграмі розгортання UML на Рисунку A.6.



*Рисунок A.6 Подання розгортання*

Додаток вимагає від середовища розгортання наступні послуги:

- Контейнерне середовище виконання, таке як Kubernetes, та певний провайдер інфраструктури рівня послуг для виявлення сервісів, маршрутизації запитів, безпеки та відстеження запитів, наприклад, сервісна сітка Istio. На схемі це показано як компоненти середовища виконання, стереотипно позначені як "kubernetes" та "istio".

- Служба бази даних, що забезпечує зберігання реляційної бази даних SQL. На схемі це показано середовищем виконання, стереотипно позначеним як "aws\_rds" (AWS Relational Database Service).

- Сервіс бази даних, що забезпечує документно-орієнтоване зберігання баз даних. На схемі це показано середовищем виконання, стереотипно позначеним як "aws\_document\_db" (служба AWS DocumentDB).

- Сервіс бази даних, що забезпечує зберігання бази даних, орієнтованої на ключ-значення. На схемі це показано середовищем виконання, стереотипно позначеним як "aws\_dynamo\_db" (служба AWS DynamoDB).

- Віртуальні машини Linux, показані на схемі як середовище виконання, стереотипно позначені як "linux\_vm" (що відповідає сервісу віртуальних машин AWS EC2).

- Шлюз API для забезпечення безпеки та управління навантаженням для наших прикладних сервісів при доступі з Інтернету. На схемі це показано як середовище виконання, стереотипно позначене як "aws\_api\_gateway" (служба AWS API Gateway)

Більша частина системи розгорнута як набір контейнерних сервісів, що працюють у середовищі розгортання Kubernetes з використанням Istio для виконання ролі сервісної сітки та надання стандартизованих послуг інфраструктури рівня обслуговування, зокрема маршрутизації, виявлення та безпеки.

Бази даних для кожного з сервісів використовуватимуть керовані сервіси баз даних AWS, зокрема, Postgres в Amazon RDS, сервіс DocumentDB та сервіс баз даних DynamoDB.

Для запуску компонента "Платіжний шлюз" потрібні віртуальні машини Linux, оскільки це двійкове програмне забезпечення сторонніх розробників, яке зазвичай не запускається в контейнері, а засоби його встановлення та управління припускають, що воно запускається як звичайний процес операційної системи на машині з Linux.

Шлюз API AWS буде використовуватися для надання зовнішніх кінцевих точок API для підтримки мобільних додатків (і будь-яких майбутніх загальнодоступних API додатків). WebUI буде використовувати ті ж сервісні API, що і мобільні додатки, але отримуватиме доступ до сервісів напряму, а не через API-шлюз.

Єдиним компонентом, який повинен ініціювати вихідні мережеві запити, є "Платіжний шлюз", який повинен зв'язатися зі шлюзом платіжної мережі за допомогою HTTPS. Інші елементи не повинні надсилати запити за межі додатку. Таким чином, мережева взаємодія може бути обмежена вхідними запитами через API-шлюз і вихідними запитами від платіжного шлюзу.

**Архітектурні рішення**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Тип** | **Ім'я** | **ID** | **Короткий опис** | **Параметри** | **Обґрунтування** |
| Фундаментальний | Нативні мобільні додатки | FDN-1 | Інтерфейс користувача на мобільних пристроях буде реалізовано у вигляді нативних додатків для iOS та Android. | Варіант 1, Розробка нативних додатків  Варіант 2, Реалізація адаптивного дизайну через браузер | Кращий досвід кінцевого користувача. Краща інтеграція платформ. Однак існує дублювання зусиль для двох платформ і можлива неузгодженість між платформами. |
| Фундаментальний | Доменно-орієнтована структура | FDN-2 | Функціональна структура системи базується на структурі домену (контракти, збори, документи, платежі), а не на процесах або транзакціях (наприклад, створення акредитива, закриття акредитива, вивільнення платежів) | Варіант 1, Структура, орієнтована на домен  Варіант 2, Структура, орієнтована на процес/транзакцію | Зберігається стан системи розділений на окремі цілісні сховища, щоб мінімізувати зв'язок між ними. Сервіси системи узгоджені з чітко визначеними обов'язками. Сервіси системи можуть комбінуватися по-різному для різних процесів або транзакцій. Однак для багатьох процесів потрібно більше міжсервісної комунікації, ніж якби система була побудована навколо процесів. |
| Фундаментальний | Сервіс-орієнтована структура | FDN-3 | Фундаментальною структурною одиницею системи є сервіс, доступ до якого здійснюється через API (синхронно) або запит повідомлення (асинхронно). Вона не структурована навколо подій або будь-якого іншого механізму. | Варіант 1, традиційна монолітна структура Варіант 2, архітектура, керована подіями (EDA) Варіант 3, архітектура на основі сервісів | Сервіси добре зрозумілі більшості людей і досить прості у розробці. Моніторинг та налагодження набагато простіші, ніж з неявними зв'язками, такими як потоки подій, та з системами, керованими подіями. Набір сервісів легше, ніж моноліт, швидко розвивати і швидко випускати. Компроміс полягає в тому, що з часом з'являється багато міжсервісних залежностей, що ускладнює деякі типи еволюції (наприклад, заміну сервісу) |
| Фундаментальний | Незалежні послуги | FDN-4 | Система побудована як набір незалежних сервісів часу виконання, що взаємодіють лише через чітко визначені інтерфейси (а не через спільні бази даних чи монолітні колекції через локальні виклики процедур). Таким чином, вона побудована в дусі, якщо не в усіх деталях, патерну мікросервісів. | Варіант 1, Незалежні сервіси Варіант 2, Традиційна монолітна структура Варіант 3, Сервіси зі спільними базами даних Варіант 4, Архітектура, керована подіями | Варіант 1, Незалежні сервіси Варіант 2, Традиційна монолітна структура Варіант 3, Сервіси зі спільними базами даних Варіант 4, Архітектура, керована подіями. |
| Фундаментальний | RPC для взаємодії сервісів | FDN-5 | У поточній реалізації вся міжелементна взаємодія буде використовувати віддалені виклики процедур (RPC) (більшість з них - виклики JSON/HTTP у стилі REST), а не повідомлення команд і запитів для реалізації взаємодії запиту/відповіді через обмін повідомленнями. | Варіант 1, зв'язок на основі RPC Варіант 2, зв'язок на основі повідомлень | Початкове рішення - використовувати RPC, але компроміс полягає в тому, що підхід на основі RPC простіший у створенні, моніторингу та експлуатації і забезпечує меншу затримку за рахунок зниження масштабованості та гнучкості. Він також унеможливлює певні стратегії моніторингу та відновлення (наприклад, моніторинг довжини черги та повторне відтворення повідомлень). |
| Фундаментальний | Загальні хмарні сервіси | FDN-6 | Вирішуючи, як розгорнути додаток, команда вирішила використовувати AWS, але навмисно використовувала хмарні сервіси, які можна легко придбати і в інших хмарах (наприклад, Azure). | Варіант 1, специфічна для AWS архітектура з використанням специфічних для AWS сервісів Варіант 2, навмисне проектування, щоб уникнути специфічних для AWS сервісів | Наразі команда буде використовувати варіант 2, і це не є реальним компромісом, оскільки їм не потрібно нічого екзотичного від хмарних сервісів. Якщо в майбутньому вони захочуть використовувати щось специфічне для AWS (наприклад, Glue або Redshift), їм доведеться змінити це рішення або знайти незалежний від хмари спосіб досягти цього (наприклад, використовуючи Talend і Snowflake замість сервісів AWS) |
| Фундаментальний | Контейнеризація, Kubernetes та Istio | FDN-7 | Враховуючи широке розповсюдження контейнерів на основі Docker та системи оркестрування контейнерів Kubernetes, команда вирішила упакувати сервіси та веб-інтерфейс як контейнери, що працюють в Kubernetes. Вони також припускають, що для виявлення сервісів, міжсервісної безпеки, маршрутизації запитів і моніторингу використовується проміжне програмне забезпечення (наприклад, Istio). | Варіант 1, просте розгортання двійкових файлів на рівні операційної системи Варіант 2, використання Kubernetes та контейнерів Варіант 3, використання контейнерів та альтернатив, таких як DC/OS або Nomad | Команда буде використовувати варіант 2, хоча ці технології додають досить багато складнощів у розгортанні та експлуатації. Однак вони широко використовуються, і консультації доступні. Крім того, вони планують використовувати сервіс Amazon EKS замість того, щоб запускати K8S самостійно, що значно спрощує процес. Все одно доведеться вручну встановлювати Helm та Istio, налаштовувати та використовувати отримане середовище. Kubernetes настільки домінує на момент написання статті, що коли команда розглядала інші альтернативи, вони швидко вирішили прийняти його можливу вищу складність заради переваг масового впровадження |
| Дані | RDMS для транзакційних послуг | DAT-1 | Бази даних для Менеджера контрагентів, Менеджера контрактів та Менеджера комісійних будуть реляційними. | Варіант 1, Реляційна база даних Варіант 2, Сховище документів Варіант 3, Інші варіанти NoSQL | Відповідає вимогам атрибутів якості. Модель транзакцій ACID відповідає шаблонам доступу. У команді наявна база навичок. |
| Дані | Технологія баз даних для сховища документів | DAT-2 | Сховище документів використовуватиме сховище документів для свого рівня збереження. | Варіант 1, Реляційна база даних Варіант 2, Сховище документів Варіант 3, Інші варіанти NoSQL | Підтримує природну структуру документів і може бути розширена за потреби. Структура запиту (весь документ) підтримує очікувані шаблони. |
| Дані | Технологія баз даних для платіжного сервісу | DAT-3 | Платіжний сервіс використовуватиме сховище ключових значень для свого рівня збереження. | Варіант 1, сховище документів Варіант 2, сховище ключ-значення Варіант 3, реляційна база даних | Власне платіжний сервіс є зовнішнім по відношенню до платформи. Платіжний сервіс використовуватиме базову базу даних, яка виконуватиме функції журналу та аудиторського сліду. Вона може відповідати майбутнім вимогам масштабованості. |
| Дані | Хороша база даних відстеження | DAT-4 | База даних Good Tracking використовуватиме сховище документів для свого рівня збереження | Варіант 1, сховище документів Варіант 2, сховище ключ-значення Варіант 3, реляційна база даних | Зможе задовольнити вимоги до моделі даних, що змінюються. Підтримує простоту використання для розробників. Достатньо для задоволення відомих вимог до масштабованості та доступності. |
| Безпека | Зашифрований трафік | SEC-1 | Уникайте загроз конфіденційності для внутрішніх системних міжкомпонентних запитів, шифруючи міжкомпонентний зв'язок. | Варіант 1: Покладатися на безпеку кордонів середовища Варіант 2: Шифрувати весь міжкомпонентний трафік за допомогою захисту на транспортному рівні (TLS) Варіант 3: Шифрувати конфіденційну інформацію в параметрах запиту | Рішення TLS точка-точка (Варіант 2) відносно просте в реалізації. Покладатися на те, що зловмисники не зможуть дістатися до мережі, нерозумно, особливо на хмарній платформі. Шифрування чутливих параметрів є більш ефективним, ніж TLS, але залишає багато відкритих даних і є більш складним для безпечної реалізації |
| Безпека | Хмарне сховище таємниць | SEC-2 | Захищайте секрети, такі як облікові дані для входу в базу даних, за допомогою хмарного сховища секретів | Варіант 1: Приховати у файлах властивостей Варіант 2: Використовувати хмарний сервіс для зберігання секретних даних | Варіант 1 є простим, ненав'язливим і швидким у реалізації, але не дуже безпечним, якщо зловмисник отримає доступ до середовища. Варіант 2 є більш інтрузивним і вимагає більше роботи, але є необхідним з огляду на чутливість деяких даних. |
| Масштабованість | Механізм оновлення БД аналітики | SCA-1 | Коли база даних транзакцій TFX оновлюється, оновлення реплікуються в аналітичну базу даних, використовуючи механізм реплікації СУБД TFX. | Варіант 1, Використання процесу реплікації СУБД TFX Варіант 2, Використання шини подій для реплікації оновлень | Використання шини подій може збільшити затримки від мілісекунд до секунд на великих об'ємах через серіалізацію даних, передачу даних, десеріалізацію даних та обробку запису. Використання механізму реплікації СУБД TFX зменшило б затримку розповсюдження даних, оскільки в більшості випадків відбувалося б пересилання журналів бази даних, а не обробка подій. Це дуже важливо для узгодженості та масштабованості. Компроміс полягає в тому, що це збільшує зв'язок між компонентами. |
| Масштабованість | Реалізація розбиття таблиці TFX | SCA-2 | Не використовуйте розбиття таблиць для початкової реалізації TFX | Варіант 1: Впровадити розбиття таблиць як частину початкової інсталяції TFX Варіант 2: Відкласти впровадження, поки не будуть розглянуті інші підходи до багатокористувацької роботи, окрім початкової повної реплікації | Оскільки початкове розгортання TFX буде повною реплікацією, команда вирішила не застосовувати розбиття таблиць на даний момент. Це рішення відповідає принципу 3: "Відкладайте проектні рішення до тих пір, поки вони не стануть абсолютно необхідними". Однак команді потрібно буде переоцінити використання розбиття таблиць або навіть шардації, коли виникне необхідність переходу на інші підходи до багатокористувацької роботи, оскільки кількість банків, що використовують TFX, збільшиться. |
| Продуктивність | Визначення пріоритетності запитів | PRF-1 | Використовуйте синхронну модель зв'язку для високопріоритетних запитів, таких як платіжні операції з акредитивами. Використовуйте асинхронну модель з системою черг для низькопріоритетних запитів, таких як великі обсяги запитів. | Варіант 1: Використовувати синхронну модель для всіх запитів Варіант 2: Використовувати синхронну модель зв'язку для високопріоритетних запитів та асинхронну модель для низькопріоритетних Варіант 3: Використовувати асинхронну модель для всіх запитів | Використання синхронної моделі для всіх запитів простіше реалізувати, але це може призвести до проблем з продуктивністю. Впровадження асинхронної моделі для запитів з низьким пріоритетом вирішує ці проблеми. Це рішення є розвитком FDN-5. |
| Продуктивність | Зменшити накладні витрати | PRF-2 | Об'єднайте менеджер по роботі з контрагентами та менеджер по роботі з клієнтами в один сервіс. | Варіант 1: Тримайте менші компоненти окремо для можливості модифікації Варіант 2: Об'єднайте менші компоненти в більші для підвищення продуктивності та зменшення накладних витрат, спричинених міжкомпонентними зв'язками | Спочатку сервіс "Контрагенти" був розділений на два компоненти: "Менеджер контрагентів", який працює з основними даними про контрагента, та "Менеджер рахунків", який працює з рахунками контрагента. Це легко могло б стати життєздатною альтернативою. Однак рівень міжкомпонентної комунікації, створений за такого підходу, був би значним і, ймовірно, створив би вузькі місця в продуктивності. На початку процесу архітектурного проектування команда вирішила об'єднати обидва компоненти в службу контрагентів, щоб вирішити проблеми з продуктивністю. |
| Продуктивність | Кешування | PRF-3 | Розглянемо можливість використання кешу об'єктів бази даних для компонентів "Менеджер контрагентів", "Менеджер договорів" і "Менеджер комісій | Варіант 1: Не використовувати кеш об'єктів бази даних для спрощення реалізації TFX Варіант 2: Використовувати кеш об'єктів бази даних для певних компонентів TFX | Використання кешу об'єктів бази даних для певних компонентів TFX вирішує потенційні проблеми з продуктивністю, але ускладнює реалізацію цих компонентів, оскільки використання кешування створює проблеми з проектуванням і програмуванням. |
| Продуктивність | Не-SQL технології | PRF-4 | Використовуйте базу даних документів для Диспетчера документів і базу даних ключів-значень для Платіжного сервісу. | Варіант 1, Використання технології nonSQL для конкретних баз даних TFX Варіант 2, Використання технології SQL для баз даних TFX | Технологія баз даних SQL простіша у використанні і більш знайома розробникам TFX. Однак її використання для баз даних TFX, таких як Менеджер документів і Платіжна служба, може спричинити проблеми з продуктивністю. Це підтримує DAT-2 і DAT-3. |
| Стійкість | Розвантаження навантаження | RES-1 | TFX буде використовувати хмарний API-шлюз для моніторингу загального обсягу запитів і скидатиме навантаження, коли він перевищить заданий поріг. | Варіант 1: Використовувати розподілення навантаження через хмарний API-шлюз Варіант 2: Не використовувати розподілення навантаження Варіант 3: Вбудувати розподілення навантаження в додаток | Розподіл навантаження захистить додаток від перевантаження без внесення змін до програми, тому він є цінним. Вбудовування його в додаток є більш гнучким підходом, але вимагає значних інженерних зусиль. Шлюз API вважається достатньо хорошим для поточних вимог. Також виконує роль перегородки при розгортанні додатку |
| Стійкість | Вимикачі клієнтського інтерфейсу | ВДЕ-2 | Інтерфейси користувача TFX включатимуть шаблон автоматичного вимикача, щоб гарантувати, що в разі виникнення помилок вони будуть відступати. | Варіант 1, Використання автоматичного вимикача в клієнтах Варіант 2, Використання простої паузи, коли клієнт стикається з помилкою Варіант 3, Клієнт ігнорує помилки і повторює спроби | Проста пауза при помилці призведе до надмірної реакції на одну тимчасову помилку сервісу, але не забезпечить сервісу значного захисту в разі реального збою. Ігнорування помилок ризикує ускладнити відновлення сервісу, продовжуючи надсилати значний трафік запитів під час перезапуску. |
| Стійкість | Функції перевірки стану здоров'я | RES-3 | Усі компоненти середовища виконання TFX забезпечують стандартну операцію перевірки працездатності, яка повертає підтвердження (або не підтвердження) того, що компонент працює коректно | Варіант 1, Використовувати стандартну операцію перевірки стану Варіант 2, Використовувати зовнішній механізм для перевірки роботи компонентів Варіант 3, Покладатися на файли журналів | Зовнішній механізм порушує інкапсуляцію, і його потрібно змінювати щоразу, коли це робить компонент. Файли журналів виявилися недостатньо надійними для перевірки працездатності, оскільки вони є індикатором відставання. Файли журналів все ще потребують стандартизації між компонентами для використання з цією метою. |
| ML | Модель класифікації документів TFX | ML-1 | Для класифікації документів TFX використовуватиме готове багаторазове програмне забезпечення. | Варіант 1: Створіть нейронну мережу для TFX за допомогою таких мов, як R або Python Варіант 2: Використовуйте програмне забезпечення з відкритим вихідним кодом від експертів, наприклад, програмне забезпечення Стенфордської групи НЛП | З огляду на компроміси в архітектурі, пов'язані з варіантом 1, такі як підтримка, масштабованість, безпека та простота розгортання, використання готового багаторазового програмного забезпечення є кращим варіантом. Варіант 1 вимагатиме значних зусиль при створенні. |

**Інші архітектурні проблеми: Багатоквартирність**

План для TFX передбачає початкову версію, яка буде використовуватися першим клієнтом фінтеху, що її створив, - великою фінансовою установою, що займається торговим фінансуванням. Цей перший клієнт планує використовувати TFX для автоматизації процесів торгового фінансування, пов'язаних з випуском, управлінням і обробкою акредитивів для своїх клієнтів. Однак у майбутньому фінтех хотів би запропонувати платформу для процесів торгового фінансування іншим фінансовим установам, які пропонують подібні послуги.

Фінтех не хоче займатися продажем пакетного програмного забезпечення, а тому повинен пропонувати програмне забезпечення як SaaS для інших банків. Це передбачає певну форму мультиоренди, оскільки TFX потрібно буде одночасно розміщувати робочі навантаження декількох банків, робочі навантаження яких повинні бути повністю ізольовані одне від одного.

Є щонайменше три способи, якими команда може цього досягти:

- Повна реплікація: Вся система реплікується для кожного клієнта, кожен у своєму власному ізольованому хмарному середовищі. Це найпростіша форма багатокористувацької оренди, яка майже не потребує змін в основному коді програми, і для невеликої кількості клієнтів вона добре працює, усуваючи більшість ризиків безпеки, забезпечуючи ізоляцію ресурсів між клієнтами і дозволяючи незалежне оновлення та підтримку для кожного клієнта. Проблема з таким підходом виникає, коли потрібно підтримувати велику кількість клієнтів, що означає велику кількість (десятки або сотні) інсталяцій, кожну з яких потрібно контролювати і керувати незалежно. Їх також потрібно досить швидко оновлювати, щоб уникнути величезної операційної складності, пов'язаної з наявністю багатьох версій у виробництві одночасно (а іноді і швидко, наприклад, у разі випуску виправлень для системи безпеки).

- Реплікація баз даних: Передбачає наявність єдиного набору прикладних сервісів, але окремого набору баз даних для кожного клієнта. Таким чином, дані кожного клієнта фізично ізольовані, але всі вхідні запити обробляються єдиним набором прикладних сервісів. Такий підхід дає багатьом клієнтам значну впевненість у тому, що їхні дані не витечуть до інших клієнтів, і для великої кількості клієнтів є операційно простішим, ніж повна реплікація. Однак, він вимагає досить значних змін у частинах основного додатку, оскільки служби додатків повинні містити поточний ідентифікатор клієнта як частину контексту запиту, і його потрібно використовувати для вибору правильної бази даних для кожної операції. Такий підхід зменшує, але не усуває операційні накладні витрати при збільшенні кількості клієнтів. Єдиний набір прикладних сервісів значно зменшує складність моніторингу, оновлення та виправлень, але оновлення все одно може включати роботу з великою кількістю баз даних, коли велика кількість клієнтів перейшла на систему.

- Повне спільне використання: Найскладніший підхід з технічної точки зору, але найефективніший в операційному плані та в довгостроковій перспективі. Це підхід, який використовують гіперскалери, такі як Google, Microsoft, Amazon та Salesforce. Єдина система використовується для підтримки всіх клієнтів, причому всі елементи системи є спільними для них. Це означає змішані дані клієнтів у базах даних, а також спільні сервіси додатків. Величезною перевагою такого підходу (з точки зору оператора) є єдина система для моніторингу, експлуатації, виправлення та оновлення. Недоліком є те, що він вимагає складних механізмів у додатку, які гарантують, що конкретний запит завжди обробляється в правильному контексті безпеки, і тому ніколи не може отримати доступ до даних з неналежної сфери безпеки клієнта.

Таким чином, початкове розгортання буде повною реплікацією, оскільки немає клієнтів і потрібно якнайшвидше вийти на ринок для першого клієнта, але, ймовірно, доведеться застосувати один з більш складних підходів, коли система буде підтримувати значну кількість банків.