МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ УКРАЇНИ

ЧЕРКАСЬКИЙ ДЕРЖАВНИЙ БІЗНЕС-КОЛЕДЖ

***Кафедра комп’ютерної інженерії та інформаційних технологій***

Відділення інженерії програмного забезпечення

**Звіт з ІНДЗ ПР02**

з курсу «Архітектура програмного забезпечення»

Лісуна Івана Олександровича, Шаповала Максима Владиславовича, Вакуленка Іллі Вадимовича, Ткаченка Дмитра Олександровича, Ткаченка Іллі Віталійовича, Компанієць Юліанни Михайлівни

*ПІБ студентів*

студентів групи **2П-21**

Викладач Марченко С. В.

Черкаси-2023

**Система оцінювання**

|  |  |  |
| --- | --- | --- |
| **№** | **Тема** | **К-ть балів** |
| 1. | Завдання 1. Quality Attribute Workshop | 4 |
|  | **За практичну** | **4** |
| 2. | ІНДЗ-1. Додаток А | 2 |
| 3. | ІНДЗ-2. Quality Attribute Workshop для проєкту | 4 |
| 4. | ІНДЗ-3. UML/C4-модель проєкту | 4 |
|  | **Всього** | **14** |

1. ***(Додаток А)*** Розгляньте та *ретельно* законспектуйте в звіті додаток А з книги Continuous Architecture in Practice: Software Architecture in the Age of Agility and DevOps за авторством Murat Erder, Pierre Pureur, Eoin Woods (2021).

**Додаток А**

**Тематичне дослідження**

У цьому додатку описано тематичне дослідження, яке використовується в книзі для ілюстрації

архітектурних прийомів, представлених у кожному розділі. Основний компроміс полягає в тому, що

важко створити приклад, який би чітко ілюстрував кожну тезу, яку ми хочемо донести до читача

без того, щоб приклад не став надмірно складним і нереалістичним.

Однак ми вважаємо, що нам вдалося задовільно вирішити цю проблему.

Ми обрали для тематичного дослідження хмаро орієнтований додаток у сфері торговельного фінансування, оскільки

ми вважаємо, що він, ймовірно, має достатньо схожості з системами, над якими ви, ймовірно, працювали

над якими ви, ймовірно, працювали, щоб швидко зрозуміти, але є достатньо складним, щоб проілюструвати теми розділу. Додаток

додаток розроблений таким чином, щоб мати достатньо цікавих архітектурних особливостей, щоб ми могли

проілюструвати наші тези, не будучи при цьому штучними. Як згадувалося в Розділі 1, "Чому архітектура програмного забезпечення є більш важливою, ніж

"Чому архітектура програмного забезпечення є більш важливою, ніж будь-коли", цей кейс є повністю вигаданим. Будь-яка

схожість з реальними системами фінансування торгівлі є випадковою і ненавмисною.

Торговельне фінансування - це сфера фінансового бізнесу, яка лежить в основі глобальної торгівлі фізичними товарами.

фізичними товарами. Ідея торгового фінансування полягає у використанні низки фінансових інструментів для того, щоб

зменшити ризик неотримання платежів продавцями товарів, а одержувачами товарів - їх оплати,

але не отримати товар. Воно передбачає, що фінансові посередники (як правило, банки) надають гарантії покупцям і продавцям.

покупцям і продавцям гарантії того, що транзакція буде застрахована фінансовою установою за певних фінансова установа за певних умов.

У цій книзі ми не можемо повністю описати реалістичну систему торговельного фінансування, тому ми зосередилися на

наше тематичне дослідження на одній конкретній частині торгового фінансування - акредитивах - і надали

загальний опис архітектури, окремі частини якого ми доповнювали в міру використання кейсу

в основних розділах книги. У Розділі 1, "Чому архітектура програмного забезпечення

важливіша, ніж будь-коли", ми надаємо загальний огляд бізнес-сфери,а також ключові учасники та випадки використання системи. Цей додаток розширює цей опис настільки, що його можна використовувати як основу для решти книги.

опис достатньою мірою, щоб його можна було використовувати як основу для решти книги.

**Представляємо TFX**

Акредитив - це загальновживаний інструмент торгового фінансування, який зазвичай використовується

підприємствами, що беруть участь у міжнародній торгівлі, щоб гарантувати, що оплата товарів або послуг буде

між покупцем і продавцем. Акредитиви стандартизовані за допомогою конвенцій

встановленими Міжнародною торговою палатою (в її керівництві UCP 600).

Крім покупця і продавця товару, в акредитиві також беруть участь щонайменше два посередники, як правило, банки.

посередників, як правило, банків, 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 того, як він буде відвантажений.

(і що покупець отримає компенсацію, якщо він не отримає товар).

Історично, і навіть сьогодні, процеси торгового фінансування були трудомісткими

і орієнтованими на документи, що передбачають ручну обробку великих, складних контрактних

документів. У нашому кейсі фіктивна фінтех-компанія побачила потенціал

підвищити ефективність і прозорість шляхом оцифрування значної частини процесу і

планує створити онлайн-сервіс під назвою Trade Finance eXchange (TFX), який

цифрову платформу для випуску та обробки акредитивів. Стратегія компанії

стратегія компанії полягає в тому, щоб знайти велику міжнародну фінансову установу в якості свого першого клієнта, а потім продати доступ до платформи всім

а потім продати доступ до платформи всім клієнтам цієї установи, які користуються послугами торгового фінансування.

У довгостроковій перспективі ми плануємо запропонувати TFX як білу етикетку (ребрендингове рішення)

для великих міжнародних банків у сфері торгового фінансування. Отже, в короткостроковій перспективі

це буде система з одним орендарем, але з часом, якщо ця стратегія буде успішною, вона

потрібно буде розвиватися, щоб стати багатокористувацькою системою.

Мета TFX полягає в тому, щоб дозволити організаціям, які займаються випискою та використанням

акредитивами в процесі торгового фінансування, взаємодіяти в ефективний, прозорий, надійний і

безпечним способом.

Як зазначено в прикладах використання в Розділі 1 "Чому архітектура програмного забезпечення є більш важливою, ніж будь-коли", ключовим моментом є

важлива, ніж будь-коли", ключовими особливостями TFX є надання можливості міжнародним покупцям товарів запитувати акредитиви від своїх

товарів запитувати акредитиви у своїх банках і дозволяти продавцям товарів викуповувати акредитиви для

викуповувати акредитиви у своїх банках. Для банків платформа діє як простий і ефективний

механізм для написання, виконання та управління акредитивними контрактами з клієнтами.

Система TFX будується як хмарна платформа "Програмне забезпечення як послуга" (SaaS).

у вигляді хмарного програмного забезпечення як сервісу (SaaS). Для цілей нашого прикладу ми використовуємо Amazon Web Services (AWS) як

хмарного провайдера для системи, але хоча ми посилаємося на специфічні особливості AWS в

архітектурні особливості AWS, це лише приклад типу хмарної платформи, яку можна використовувати для таких систем, як TFX.

на якій, швидше за все, буде розміщена система, подібна до TFX, а не важливий аспект

тематичного дослідження. Ми обрали AWS через її широке розповсюдження в індустрії

і припускаємо, що багато технологів знайомі з її ключовими сервісами. Ми не рекомендуємо AWS або його конкретні сервіси, надаючи перевагу іншим хмарним провайдерам,

але ми використовуємо AWS та його сервіси, щоб додати реалістичності нашому кейсу.

**Архітектурний опис**

У цьому додатку представлено загальний опис архітектури першої версії

платформи TFX. У дусі звичайної промислової практики, ми надаємо "достатньо архітектури", щоб пояснити основні аспекти системи, не визначаючи при цьому

достатньо архітектури", щоб пояснити ключові аспекти системи, не визначаючи

деталей проектування, які краще зробити під час реалізації та описати у вигляді коду, а не моделі.

а не як модель. Важливо також пам'ятати, що це модель потенційної системи, і вона не відображає

це модель потенційної системи і не відображає систему, яку ми побудували. Тому існують компроміси

і рішення, які зазвичай були б необхідні для дотримання термінів поставки або для того, щоб задовольнити

певних особливостей продукту, які ми не змогли реалістично врахувати. Наприклад

Наприклад, хоча ми вважаємо, що відстеження технічної заборгованості є важливою архітектурною діяльністю, ми не можемо

ми вважаємо, що відстеження технічної заборгованості є важливою архітектурною діяльністю, ми можемо навести лише кілька гіпотетичних прикладів у цьому додатку.

Архітектурний опис також свідомо неповний. Деякі його аспекти

залишені як відкриті питання (описані далі в цьому додатку), які потім розглядаються в наступних розділах

у наступних розділах, щоб проілюструвати певний аспект процесу архітектурного проектування. Інші

аспекти просто залишаються невизначеними, оскільки вони не потрібні для підтримки змісту

книги, і тому ми опускаємо їх з міркувань економії місця.

Архітектурний опис представлений у вигляді набору видів, взятих з

Розанські та Вудса (R&W)1

кожен з яких ілюструє окремий аспект архітектури.

тики, кожен з яких ілюструє окремий аспект архітектури. Повний опис архітектурних точок зору можна знайти у Вікіпедії,

а набір R&W, зокрема, пояснюється в розділі "Архітектура програмних систем" та на

www.viewpoints-and-perspectives.info. Якщо коротко, то погляди, що використовуються в цій моделі, є наступними

наступним чином:

- Функціональний: Функціональна структура системи під час виконання, що показує функціональні компоненти системи під час виконання, інтерфейси

функціональні компоненти системи під час виконання, інтерфейси, які кожен компонент

інтерфейси, які надає кожен компонент, а також залежності та взаємодії між елементами. (Функціональні

компоненти - це ті, що виконують дії, пов'язані з проблемною областю, а не

а не пов'язані з інфраструктурою, такі як розгортання, оркестрування контейнерів та моніторинг).

(Функціональні компоненти - це ті, що виконують дії у проблемній області, а не пов'язані з інфраструктурою, такі як розгортання, оркестрування контейнерів та моніторинг).

- Інформація: Інформаційна структура системи, що показує ключові інформаційні об'єкти та взаємозв'язки між ними, а також будь-які значущі стани системи.

інформаційні об'єкти та їх взаємозв'язки, а також будь-які значущі стани або життєві цикли

моделі, що ілюструють, як вони обробляються в системі.

- Розгортання: Визначає середовище виконання, у якому має працювати система

працювати, включаючи інфраструктурну платформу, необхідну системі (обчислення, зберігання, мережа), технічні вимоги до

робочі), вимоги до технічного середовища для кожного вузла (або типу вузла) в системі, а також відображення елементів програмного забезпечення в середовище виконання.

системи, а також відображення програмних елементів у середовище виконання

яке буде їх виконувати. Як уже згадувалося, ми припускаємо, що TFX є хмарним додатком, що працює на AWS.

ням, що працює на AWS.

**Функціональний вигляд**

Функціональну структуру системи ілюструє діаграма компонентів уніфікованої мови моделювання

(UML) на Рисунку A.1 і пояснюється в наступному тексті.

На Рисунку A.1 ми використовуємо просту нотацію у вигляді клітинок та ліній, яка значною мірою натхненна нотацією діаграми компонентів

діаграми компонентів UML2.

- Прямокутні фігури - це функціональні компоненти.

- Лінії між компонентами показують, які компоненти викликають операції над

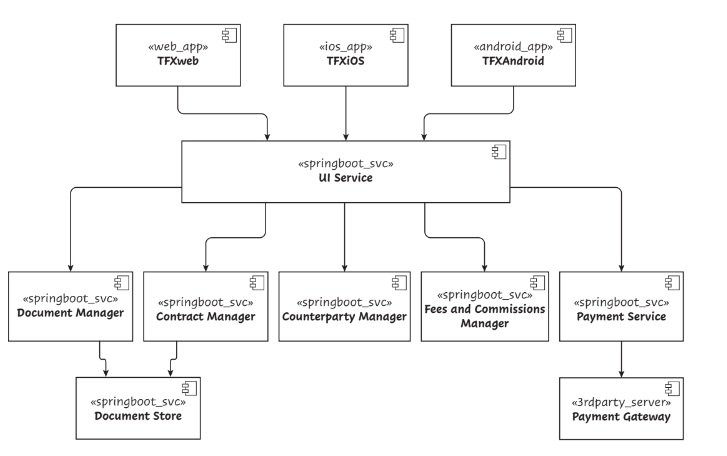
з іншими компонентами.

- Дрібніший текст всередині фігур, між дужками, - це стереотипи.

отипи. 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 A.1 ми неформально пояснюємо характеристики кожного компонента

природною мовою. Ми також використовуємо термін API (інтерфейс прикладного програмування)

для позначення міжкомпонентного інтерфейсу, що забезпечує виклик операцій,

не обов'язково визначаючи, яка саме технологія API буде використовуватися для кожного з них.

**Таблиця A.1** Опис функціональних компонентів

**Компонент: TFXWeb**

Опис Звичайний веб-інтерфейс користувача, призначений для запуску у десктопному браузері; пише

з використанням набору JavaScript, HTML та каскадних таблиць стилів (CSS)

на основі технологій JavaScript, HTML та каскадних таблиць стилів (CSS).

Обов'язки Забезпечення орієнтованого на завдання інтерфейсу, що дозволяє запитувати, створювати, контролювати, виконувати та управляти акредитивами,

виконання та управління акредитивами та іншими акредитивними документами

Забезпечення рольового інтерфейсу для різних типів користувачів банку та клієнтів

Надані інтерфейси Забезпечує людський інтерфейс через браузер

Взаємодія Інтенсивно викликає службу інтерфейсу користувача для отримання всієї необхідної інформації, а також для

зробити всі необхідні оновлення

**Компоненти: TFXiOS та TFXAndroid**

Опис Нативні мобільні користувацькі інтерфейси, оптимізовані для використання на мобільних пристроях (як з малим

так і з великим екраном)

Обов'язки Забезпечити орієнтований на завдання інтерфейс для запиту, створення, моніторингу, виконання та управління акредитивами та іншими артефактами акредитивів,

виконання та управління акредитивами та іншими акредитивними документами

Забезпечення рольового інтерфейсу для різних співробітників банку та користувачів-клієнтів банку типи

Надані інтерфейси Надає людський інтерфейс через мобільний пристрій

Взаємодія Інтенсивно викликає службу інтерфейсу користувача для отримання всієї необхідної інформації та внесення всіх необхідних оновлень

зробити всі необхідні оновлення

**Компонент: UI Service**

Опис Надає набір API для підтримки потреб користувацьких інтерфейсів програми

Виступає як фасад для служб домену, тому є більш функціональним, ніж простий API

шлюз

Обов'язки Надає набір доменних сервісів, спеціалізованих для потреб користувацьких інтерфейсів, щоб

дозволяє отримувати всі об'єкти домену з системи та керувати ними

оновлення

Надані інтерфейси Надає API до доменних сервісів

Взаємодії Викликає сервіси домену для надання доступу до сутностей домену та пов'язаних з ними операцій операцій

Активно викликається елементами TFXWeb, TFXiOS та TFXAndroid

**Компонент: Менеджер документів**

Опис Сервіс, що надає просту можливість виконувати операції над великими,

неструктурованими документами за допомогою інших елементів, зокрема користувацьких інтерфейсів

Обов'язки Приймання файлів у різних форматах (текст, JSON, PDF, JPEG)

Автоматично генерувати метадані з файлів

Дозволяти прикріплювати метадані до файлів

Дозволяти запитувати набір документів для отримання списків відповідних документів,

включно з індексами ключових слів природною мовою для нетекстових форматів, таких як PDF і JPEG

Дозволяє отримувати конкретні файли документів

Надані інтерфейси Надає API, який дозволяє здійснювати два аспекти управління документами: (1)

імпорт та вилучення великих файлів документів у різних форматах за одну або декілька операцій; (2)

операцій або операцій з декількома документами; (2) управління набором документів за допомогою метаданих

за допомогою метаданих, що дозволяє тегувати, шукати та знаходити списки документів

зіставлення складних критеріїв (наприклад, дат та тегів)

Взаємодія Використовує сховище документів для зберігання, індексування та керування документами, які імпортованими

**Компонент: Менеджер договорів**

Опис Надає можливість створювати та керувати об'єктами контрактів, які представляють акредитиви

акредитиви (L/Cs)

Обов'язки Створення та управління об'єктами домену контрактів

Управління станом об'єкта контракту (щоб він знаходився в дійсному, чітко визначеному стані життєвого циклу в будь-який момент часу)

визначеному стані життєвого циклу в будь-який час і переходив між станами тільки відповідно до

до визначеної моделі стану)

Обробка необхідних дозволів для об'єктів контрактної області (які є численними

і можуть бути складними)

Надані інтерфейси Надає API, що дозволяє створювати, керувати, затверджувати та запитувати об'єкти контрактів,

та запитувати об'єкти контрактів

Взаємодія Залежить від Менеджера документів для зберігання та отримання файлів документів (наприклад,PDF-файлів)

Залежить від Менеджера зборів і комісій для розрахунку зборів і комісій

комісій в процесі створення акредитиву

Залежить від Менеджера контрагентів для реєстрації транзакцій за рахунком клієнта

що виникають в процесі погашення акредитиву

**Компонент: Менеджер зборів і комісій**

Опис Забезпечує розрахункові операції, що дозволяють розраховувати збори та комісії

на вимогу

Дозволяє визначати та керувати правилами розрахунку комісійних та зборів

Обов'язки Забезпечує контрольоване створення, модифікацію та видалення тарифів і комісій

правил

Постійне зберігання та підтримка цілісності правил нарахування комісійних

Дозволяє розраховувати значення зборів і комісій на вимогу

Надані інтерфейси Надає API для визначення правил зборів і комісій

Надає API, що дозволяє розраховувати комісії та збори для певних транзакцій

транзакцій

Взаємодія Немає

**Компонент: Платіжний сервіс**

Опис Надає можливість створювати та керувати об'єктами платіжного домену

Обов'язки Дозволяє створювати платіж, затверджувати його, якщо потрібно, відправляти в мережу,

та запитувати 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 Забезпечувати контрольоване створення, модифікацію та видалення неструктурованих

неструктурованих даних документів з відповідними метаданими

Забезпечення індексування збережених документів

Забезпечення тегування збережених документів

Забезпечення пошуку документів за ідентифікатором

Ідентифікація документів за метаданими та пошук за тегами

Дозволяє ідентифікувати документ за допомогою повнотекстового пошуку

Надані інтерфейси Надає API для зберігання, пошуку, запитів та управління документами

Взаємодія Немає

**Компонент: Платіжний шлюз**

Опис Мережевий сервісний інтерфейс, що забезпечує стандартизований та спрощений доступ до

до платіжних мереж через стандартний інтерфейс

Обов'язки Зведення деталей доступу до платіжної мережі до стандартної моделі

Надавати можливість іншим елементам системи виконувати стандартні операції з платіжною мережею

мережеві операції (оплата, скасування, звітність)

Надані інтерфейси Надає API, що забезпечує простий та стандартизований доступ до платіжних

до операцій платіжної мережі у незалежний від платіжної мережі спосіб

Взаємодія Немає внутрішньої взаємодії - отримує доступ до однієї або декількох зовнішніх платіжних мереж через їхні

власні інтерфейси

**Сценарії використання**

Загальні взаємодії між елементами системи, що передбачаються в архітектурі, є наступними

наступним чином:

- Користувацькі інтерфейси викликають UI Service для всіх взаємодій з рештою

системи. Сервіс надає набір API, оптимізованих для потреб користувацьких інтерфейсів.

інтерфейсів і, таким чином, виступає в ролі фасаду для решти системи.

- Коли користувацьким інтерфейсам потрібно маніпулювати ключовими об'єктами домену в системі, вони викликають UI Service API.

вони викликають сервісні API інтерфейсу користувача, які, в свою чергу, викликають диспетчер контрагентів,

Менеджер комісій, Менеджер документів, Менеджер контрактів та

Платіжний сервіс.

- Більш складні операції реалізуються у відповідній службі домену

(наприклад, менеджер контрактів, який відповідає за створення та доставку

акредитивного контракту, що є складним процесом).

- Враховуючи необхідність виконання деяких довготривалих операцій, деякі служби домену повинні забезпечувати асинхронне виконання

служби домену повинні надавати асинхронні, а також прості транзакційні API-інтерфейси.

інтерфейси.

- Коли потрібно імпортувати та опрацьовувати великі документи (наприклад, обробка природних ландшафтів), то запит

(наприклад, обробка природних ландшафтів), то робиться запит від Служби інтерфейсу користувача до Диспетчера документів, який відповідає за цю задачу.

Document Manager, який відповідає за це завдання.

Ці операції за своєю суттю є асинхронними, і коли робиться запит

через API імпортера документів повертається ідентифікатор документа. Імпортер документів

Імпортер документів записує результати процесу в Диспетчер документів

Document Manager, ідентифікований за цим ідентифікатором, і користувач може запитати про стан документа, використовуючи

документа за допомогою API та ідентифікатора документа. Ідентифікатор документа також використовується

для багатоетапних операцій, таких як завантаження документа частинами через

його розміру.

Цілком ймовірно, що система TFX також надаватиме свої API зовнішнім бізнес-партнерам.

бізнес-партнерам. Це потенційно може призвести до різних потоків доходів і бізнес-планів.

бізнес-планів. Ми не розглядали цей аспект TFX, але найбільш вірогідним

підходом буде створення зовнішнього API-сервісу, що виступає в якості фасаду над доменними сервісами, подібно до

сервісів домену, подібно до сервісу UI для наших користувацьких інтерфейсів.

Щоб надати реальний приклад обробки запиту системою, розглянемо

схвалення видачі акредитиву агентом, який є співробітником банку-емітента

банку-емітента. Процес проілюстровано діаграмою взаємодії на рисунку А.2 та

та пояснюється в тексті.

Для діаграм взаємодії в нашому прикладі ми використовували спрощену версію

нотації UML. У верхній частині діаграми зображено елементи системи, що взаємодіють під час виконання

елементи системи, які взаємодіють, а вертикальні лінії, що проектуються вниз

вертикальні лінії, що відходять від них вниз, є їхніми "лініями життя", показуючи, де вони отримують та надсилають повідомлення.

надсилаються повідомлення, а плин часу відображається вертикальною відстанню. Суцільні стрілки

суцільні стрілки - це певні зв'язки між елементами, а їхні позначки вказують на характер взаємодії.

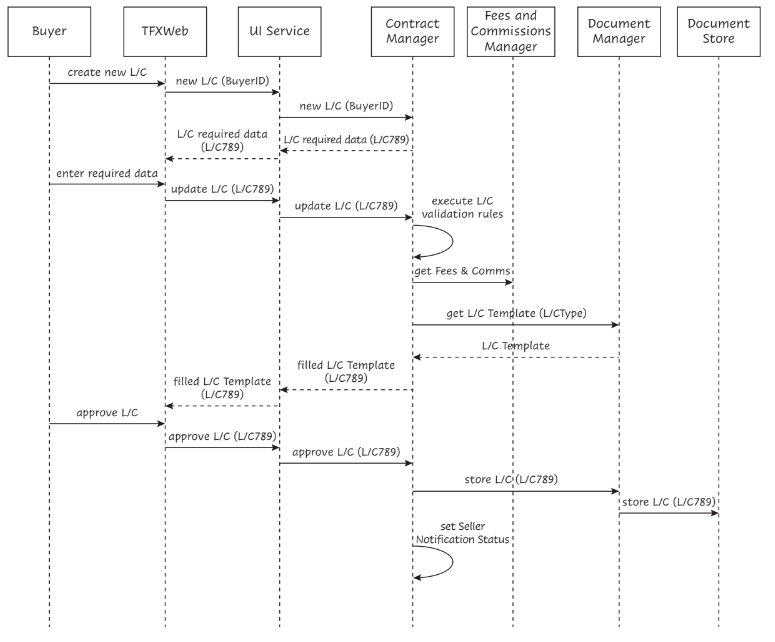
на характер взаємодії. Пунктирні лінії - це відповіді елементів на

запит або стимул. Для позначення типу взаємодії ми використали окремі стереотипи.

взаємодії.

Перший сценарій, проілюстрований діаграмою взаємодії на Рисунку А.2, показує, як

новий акредитив створюється з точки зору покупця.



**Рисунок A.2** Створення акредитивного сценарію-покупець

Взаємодія на Рисунку А.2 виглядає наступним чином:

1. Покупець просить створити новий акредитив через TFXWeb.

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

3. Менеджер контрактів визначає дані, необхідні для акредитива, на основі

BuyerID - це передбачає, що система може бути налаштована на основі характеристик покупця.

тики покупця. Контрактний менеджер також присвоює унікальний ідентифікатор (789)

запиту.

4. Запитувані дані відображаються в інтерфейсі користувача, і покупець заповнює всі

дані (наприклад, дані продавця, умови договору).

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

6. Контрактний менеджер виконує правила валідації, включаючи збори та комісійні, через Менеджера зборів та комісійних.

через Менеджера зборів та комісій через Менеджера зборів та комісій.

7. Менеджер контрактів запитує шаблон акредитива у Менеджера документів

Manager шаблон акредитива.

8. Заповнений шаблон акредитива з усіма необхідними даними передається покупцеві

для остаточної перевірки.

9. Покупець затверджує акредитив, який передається Контрактному менеджеру.

10. Контрактний менеджер просить менеджера по роботі з документами зберігати акредитив. Менеджер по роботі з документами

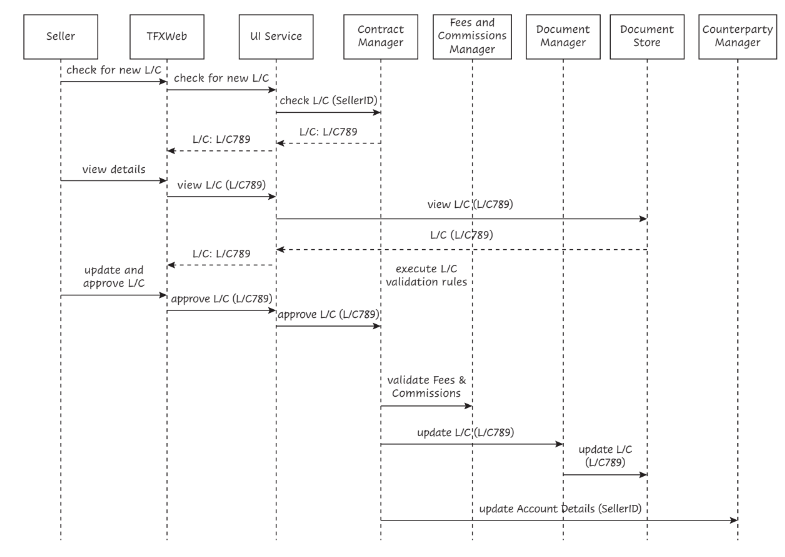
Менеджер документів витягує всі відповідні метадані та зберігає акредитив у

Сховище документів.

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

акредитив був створений для продавця.

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



**Рисунок A.3** Створення акредитивного сценарію – продавець

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

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

3. Менеджер контрактів знаходить новий акредитив, відкритий покупцем, і надає цю інформацію Продавцю.

інформація надається Продавцю.

4. Продавець запитує повний текст акредитива.

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

6. Продавець затверджує акредитив, який Служба інтерфейсу передає Контрактному

Менеджеру по контрактах.

7. Менеджер контрактів виконує правила валідації, включаючи валідацію зборів та

комісійних за допомогою Менеджера зборів і комісійних. Для спрощення ми не показуємо

ми не показуємо жодних винятків або циклів зворотного зв'язку, які могли б бути присутніми в більш складному сценарії; ми показуємо лише найпростіші винятки та

складнішому сценарії; ми представляємо тут лише найпростіший сценарій "щасливого дня".

8. Менеджер контрактів просить Менеджера документів зберігати оновлений контракт з інформацією про схвалення продавця в

тракту з інформацією про схвалення продавця у Сховищі документів. Менеджер документів

оновлює відповідні метадані для договору.

9. Менеджер контрактів просить Менеджера контрагентів оновити облікові

дані продавця.

Другий приклад - прибуття товару, що спричиняє платіж постачальника

відповідно до умов акредитивного контракту. Цей процес проілюстровано на рисунку А.4

і пояснюється в тексті.

1. Продавець повідомляє працівника банку про відвантаження партії товару, що гарантується акредитивом

товарів, гарантованих за акредитивом, можливо, за допомогою коносамента, який надсилається електронною поштою.

електронною поштою.

2. Співробітник заходить в інтерфейс користувача TFXWeb і шукає відповідний акредитив.

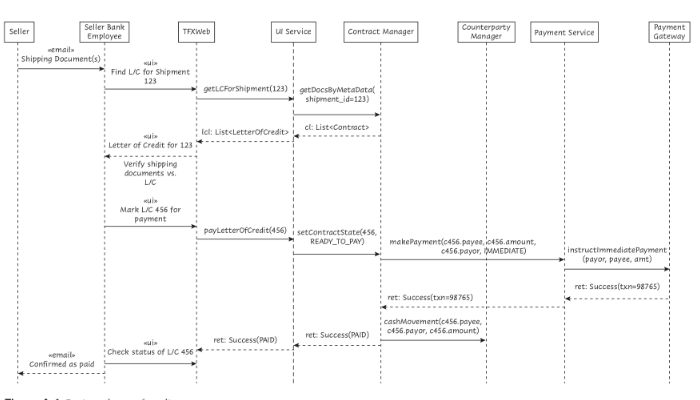
потрібний акредитив (можливо, використовуючи номер вантажу).

3. Інтерфейс TFXWeb викликає UI Service, щоб забезпечити простий інтерфейс для цієї операції.

інтерфейсу для виконання цієї операції.

4. Служба інтерфейсу викликає Менеджер контрактів, передаючи йому критерії пошуку, і

Менеджер контрактів повертає детальну інформацію про всі контракти, які відповідають критеріям.



5. Інтерфейс TFXWeb відображає повернуті контракти і дозволяє користувачеві

перевірити їх зміст. Користувач знаходить відповідний контракт і перевіряє умови, які вимагає акредитив.

умови, які вимагає акредитив для здійснення платежу.

6. Користувач підтверджує, що всі необхідні умови виконані (шляхом перевірки наданих документів). 6.

наданих документів 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"), які

ми тут не розглядаємо.

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 викликає менеджера контрактів, щоб повідомити про зміну статусу контракту.

аджер контракту, щоб повідомити про зміну статусу за допомогою виклику API.

9. Менеджер контрактів отримує деталі контракту і створює платіжний

запит на переказ валюти та суми з банківського рахунку на рахунок одержувача

реквізити, як зазначено в договорі. Потім він телефонує до платіжної служби, щоб доручити

здійснити платіж і встановлює статус контракту на "запит на платіж".

10. Платіжний сервіс отримує і зберігає платіжну вимогу. Потім він створює

платіжну інструкцію для Платіжного шлюзу та надсилає її на Платіжний

шлюзу.

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

статус успіху або невдачі назад до Платіжного сервісу.

12. Якщо платіж пройшов успішно, Платіжний сервіс позначає платіж як виконаний і телефонує до Контрактної служби, щоб повідомити про цю подію.

виконаним і телефонує до Контрактної служби, щоб повідомити про цю подію.

13. Контрактний менеджер оновлює рахунки платника та одержувача через Контрактного менеджера, щоб відобразити рух грошових коштів на рахунках платника та одержувача. 13.

контрагента, щоб відобразити рух грошових коштів за транзакцією.

14. Контрактна служба позначає контракт як оплачений і завершений, а повідомлення про успіх повертається до Служби інтерфейсу

повертається до Служби інтерфейсу користувача, а потім, у свою чергу, до служби TFXWeb, де співробітник банку може побачити статус і повідомити про це

де співробітник банку може побачити статус і повідомити продавця.

Якщо платіжна інструкція зазнає невдачі з тимчасовою помилкою, Платіжний сервіс

повторить спробу невелику кількість разів. Якщо платіжна інструкція не пройшла з перманентною помилкою або з кількома транзиторними помилками, платіжний сервіс повторить спробу ще кілька разів.

постійною помилкою або декількома тимчасовими помилками, Платіжний сервіс позначить запит як невдалий і зателефонує менеджеру контракту, щоб повідомити про це.

як невдалий і зателефонує Контрактному менеджеру, щоб повідомити про цю подію. Менеджер контрактів

потім позначить контракт на акредитив як такий, що знаходиться в стані помилки і потребує оперативної уваги для виправлення платіжної інформації.

для виправлення платіжної інформації.

**Перегляд інформації**

В інформаційному поданні ми описуємо ключові структури даних для системи і те, як

як вони пов'язані між собою. Модель даних для системи показана на Рисунку A.5.

Існує ряд 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 в кінці зв'язку, де роль не вказується.

зв'язку, де роль не очевидна, а також кардинальність цього кінця зв'язку (1 означає точно

кінця зв'язку (1 означає точно 1, а n - від 0 до n). Відношення

з відкритим трикутником на одному кінці вказує на відношення узагальнення

("наслідування"), а заповнений ромб вказує на агрегування ("стримування")

відношення.

Ключовими частинами інформаційної структури в системі є наступні:

- Акредитив: Акредитив є центральною інформаційною одиницею в системі

і є причиною її існування. Це тип контракту для усунення ризику, який несе покупець певного товару (заявник)

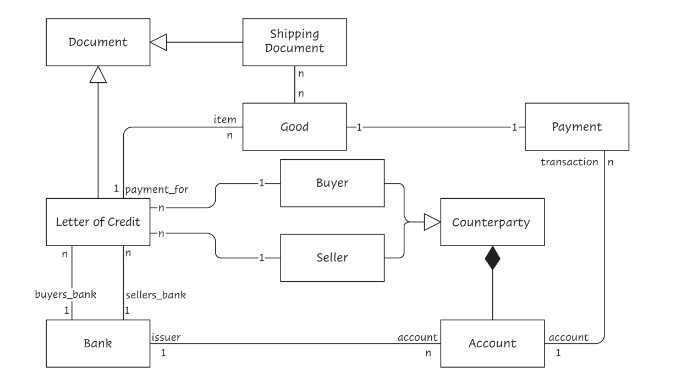
що покупець певного товару (заявник) не заплатить продавцю товару (бенефіціару).

бенефіціар).

- Відвантажувальний документ: Документ, який використовується для підтвердження того, що товари були відвантажені,

і, отже, акредитив на ці товари може бути погашений. Зазвичай він має форму

форму коносамента (або матеріальної накладної).



**Рисунок A.5** Модель представлення інформації-даних

- Документ: Узагальнення акредитива та пов'язаних з ним документів, що представляє

загальних аспектів документа, які не є специфічними для конкретного типу документа.

типу документа.

- Платіж: Платіж, здійснений продавцю після надання та перевірки

необхідних документів.

- Товар: Представлення товару, який є предметом торгівлі між продавцем і

покупцем і який гарантується акредитивом.

- Покупець і Продавець: Представляють учасників контракту, обидва з яких є

типи контрагентів, які мають клієнтські відносини з банком.

- Продавець - це контрагент, оплата якого гарантується

контрактом. Зазвичай це експортер у міжнародній торгівлі.

- Покупець - це контрагент, який запитує контракт для того, щоб

щоб продавець випустив товар до оплати. Зазвичай це імпортер у

у міжнародній торгівлі.

- Банк покупця - це фінансова установа, яка укладає контракт і

представляє покупця.

- Банк продавця - це фінансова установа, яка представляє продавця і

інформує продавця про отримання кредиту.

- Банк: Представляє банки, що беруть участь в акредитивному процесі, які створюють і погашають акредитив

акредитив і надають своїм клієнтам послуги з ведення рахунків і здійснення платежів.

- Рахунок: Банківські рахунки, пов'язані з контрагентами, які використовуються для здійснення транзакції.

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

та валютах

Ми усвідомлюємо, що це досить спрощене уявлення про інформаційну структуру

необхідної для підтримки реальної операції з акредитивом, але ми вважаємо, що вона відображає її суть

і служить нашим цілям, уникаючи при цьому надмірної складності.

Ми неодноразово виявляли, що на практиці подібні моделі є найбільш цінними

коли вони відображають суть домену, а не намагаються охопити всі

деталі, які може знати експерт у цій галузі. Такі моделі допомагають пояснити суть предметної області людям, які не є її експертами.

суть предметної області людям, які не знайомі з нею, і забезпечує відправну точку

для навчання і контекст, в який можна помістити більш детальні знання в міру їх засвоєння.

більш детальні знання, коли вони будуть засвоєні.

Така модель також може стати відправною точкою для створення більш детальної

моделі предметної області та універсальної мови, необхідної для доменно-орієнтованого проектування.

підходу. Як тільки почнеться реалізація, модель потрібно буде уточнити (наприклад,

у цьому випадку "покупець" і "продавець" можуть стати ролями для "контрагента", а не самостійними сутностями

ніж самостійними суб'єктами, а відсутні елементи, такі як дані про комісійні, повинні бути додані).

потрібно буде додати відсутні елементи, такі як дані про комісійні та винагороди). Однак ми виявили, що співпраця з

експертами в цій галузі для створення досить абстрактної моделі, подібної до цієї, є чудовим першим кроком у цьому процесі.

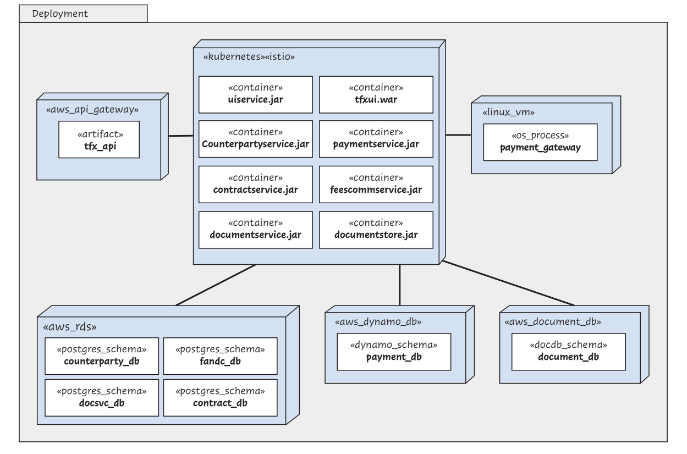
процесу, і така модель продовжує залишатися корисним орієнтиром для проекту

по мірі його розвитку.

**Подання розгортання.**

Розгортання системи у робочому середовищі описано на

UML діаграмі розгортання на рисунку A.6.



**Рисунок A.6** Подання розгортання

Ми припускаємо, що TFX буде розгорнуто на хмарній платформі AWS, хоча

ми подбали про те, щоб не покладатися на будь-яку дуже специфічну послугу, яку надає AWS, залишаючи

залишаючи можливість використання інших хмарних провайдерів або локального розгортання пізніше, якщо

якщо це буде потрібно.

Додаток вимагає від середовища розгортання наступних послуг:

- Контейнерне середовище виконання, таке як Kubernetes, і якийсь

провайдер інфраструктури рівня послуг для виявлення сервісів, маршрутизації запитів,

безпеки та відстеження запитів, наприклад, сервісна сітка Istio. Це показано

на схемі як компоненти середовища виконання, стереотипно позначені як

"kubernetes" та "istio".

- Сервіс бази даних, що забезпечує зберігання реляційної бази даних SQL. На діаграмі це показано

діаграмі середовищем виконання, стереотипно позначеним як "aws\_rds" (служба реляційних баз даних AWS

Relational Database Service).

- Служба бази даних, що забезпечує зберігання документно-орієнтованої бази даних. Це

показано на діаграмі середовищем виконання, стереотипно позначеним як "aws\_doc-

ument\_db" (служба AWS DocumentDB).

- Служба бази даних, що забезпечує зберігання бази даних, орієнтованої на ключ-значення. На схемі це

показано на діаграмі середовищем виконання, стереотипно позначеним як

"aws\_dynamo\_db" (служба AWS DynamoDB).

- ВМ Linux, показані на схемі як середовище виконання, стереотипно позначене

як "linux\_vm" (що відповідає сервісу віртуальних машин AWS EC2).

- Шлюз API для забезпечення безпеки та управління навантаженням для нашого додатку

сервісів при доступі з Інтернету. На діаграмі це показано як

середовище виконання, стереотипно позначене як "aws\_api\_gateway" (служба AWS API Gate-

шляховий сервіс AWS API).

Більша частина системи розгорнута як набір контейнерних сервісів, що працюють в

Kubernetes, використовуючи Istio для роботи в якості сервісної сітки та надання

стандартизовані сервіси інфраструктури рівня обслуговування, зокрема маршрутизацію, виявлення

та безпеку.

Бази даних для кожного з сервісів будуть використовувати керовані сервіси баз даних

AWS, зокрема Postgres в Amazon RDS, сервіс DocumentDB та

службу баз даних DynamoDB.

Для запуску компонента Платіжний шлюз потрібні віртуальні машини Linux, оскільки

це стороннє двійкове програмне забезпечення, яке зазвичай не запускається в контейнері, і його засоби встановлення та керування припускають, що він працює як звичайний процес операційної системи.

системний процес на комп'ютері з Linux.

Шлюз API AWS буде використовуватися для надання зовнішніх кінцевих точок API для підтримки мобільних додатків.

мобільних додатків (і будь-яких майбутніх публічних API додатків, які можуть знадобитися). WebUI

WebUI буде використовувати ті ж сервісні API, що і мобільні додатки, але матиме доступ до

сервісів безпосередньо, а не через API-шлюз.

Єдиним компонентом, який повинен ініціювати вихідні мережеві запити, є Платіжний шлюз.

ний шлюз, який повинен зв'язуватися зі шлюзом платіжної мережі за допомогою HTTPS.

Іншим елементам не потрібно надсилати запити за межі додатку. Таким чином, робота в мережі може бути обмежена

робота може бути обмежена вхідними запитами через API-шлюз і вихідними

запитами через платіжний шлюз.

Очевидно, що хмарна платформа і пов'язані з нею вимоги TFX будуть розвиватися з часом.

Зрозуміло, що хмарна платформа і пов'язані з нею вимоги TFX будуть змінюватися з часом, тому описані тут варіанти вибору будуть змінюватися в міру розвитку TFX і хмарної платформи.

Однак, основні варіанти розгортання системи на хмарній платформі навряд чи будуть кардинально відрізнятися від наведених тут.

навряд чи кардинально зміняться в осяжному майбутньому.

**Архітектурні рішення**

У Розділі 2 "Архітектура на практиці: Основні види діяльності" ми стверджуємо, що архітектурне рішення є одиницею роботи архітектора, тому ми не могли б не звернути увагу на те, що

архітектурні рішення є одиницею архітектурної роботи, тому було б упущенням з нашого боку не описати архітектурні рішення для нашого прикладу.

не описати архітектурні рішення для нашого кейсу. Однак, в рамках наявного простору, ми не можемо

ми не можемо перерахувати всі архітектурні рішення для побудови такої системи, як TFX.

системи на кшталт TFX. Тому ми представляємо вибірку архітектурних рішень у

Таблиці A.2 (див. стор. 288-295). Фундаментальні рішення (FDN) підтримують кейс

дослідження, як описано в цьому додатку. Додаткові рішення надають приклади

еволюції системи на основі тем, розглянутих у розділах з 3 по 8.

Ми не представили повну архітектуру системи TFX, засновану на всіх

архітектурних рішень. Це зроблено навмисно; наша мета - надати огляд

компромісів, на які доводиться йти командам.

Зрештою, ми очікуємо, що архітектурні рішення будуть більш детальними та

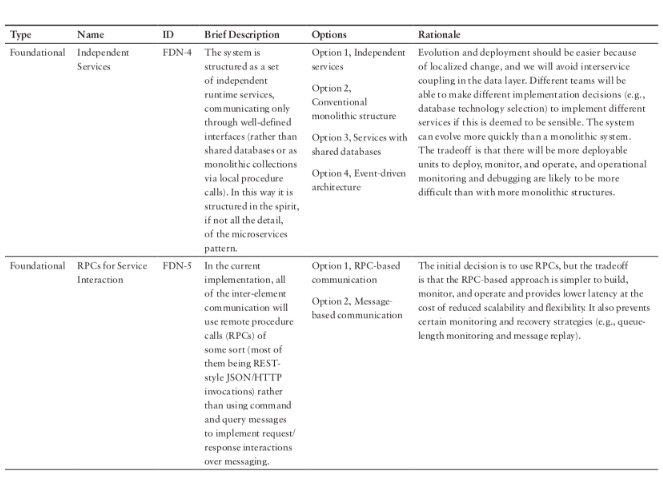
більш детальну інформацію та допоміжні матеріали в реальній системі. Зверніться до розділу 4 книги "Безперервна архітектура" (Continuous Architec

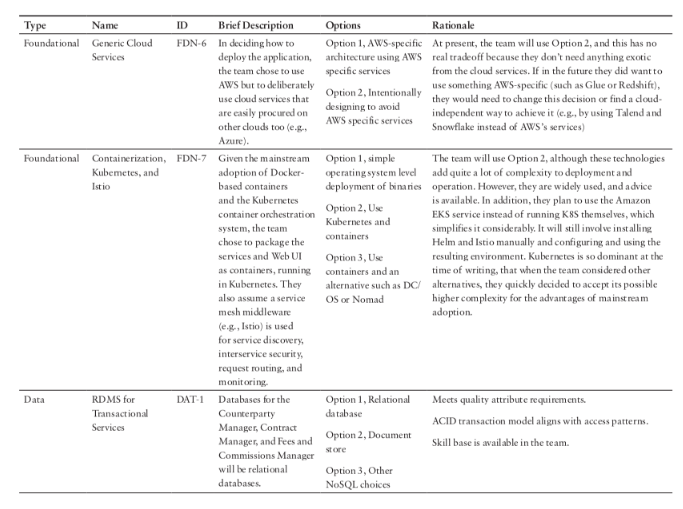
ture2

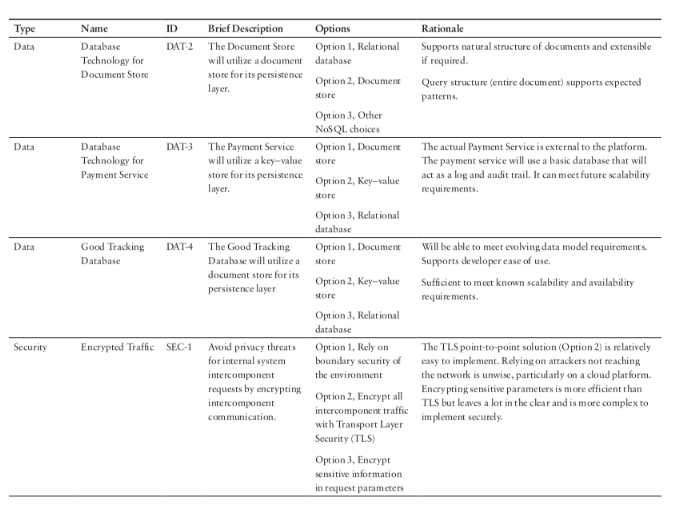
для більш повного огляду цього питання.

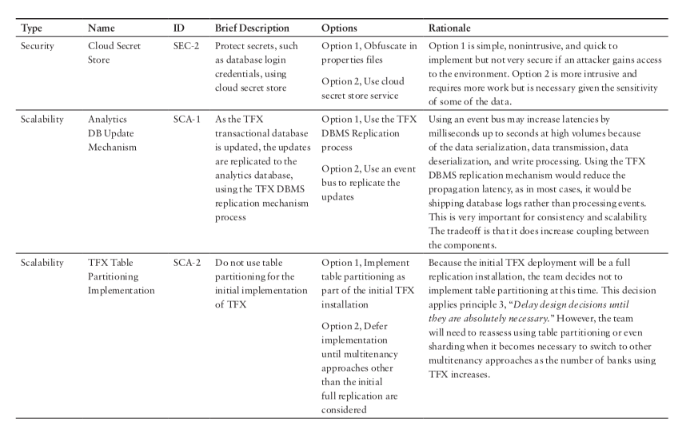
**Таблиця A.2 Журнал архітектурних рішень**

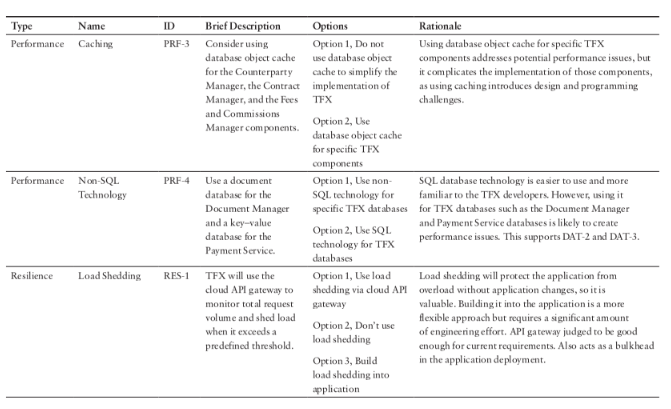
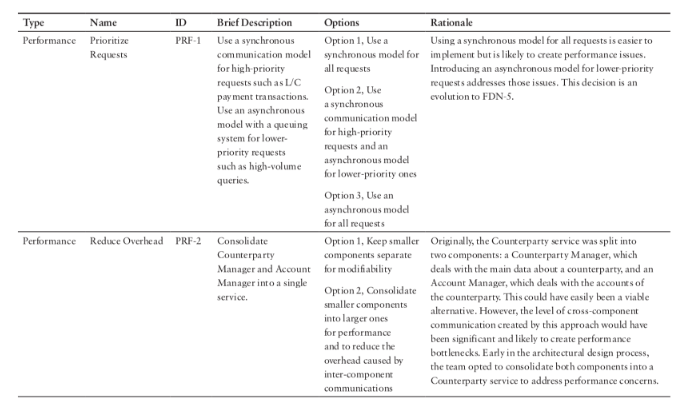
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Тип** | **Ім'я** | **ID** | **Короткий опис** | **Параметри** | **Обґрунтування** |
| **Фундаментальний** | **Native Mobile**  **Додаток\*** | **FDN-1** | **Інтерфейс користувача на мобільному пристрої" буде реалізовано у\* нативних додатках для iOS та Android.** | **Варіант 1, Розробка нативних додатків**  **Варіант 2. Реалізувати адаптивний дизайн через браузер** | **Кращий досвід кінцевого користувача. Краща інтеграція платформ. Однак існує дублювання зусиль для двох платформ і можлива неузгодженість між платформами.** |
| **Фундаментальний** | **Доменно-орієнтований**  **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, здійснення платежів).** | **Варіант 1. Доменно-орієнтована структура**  **Варіант 2. Орієнтована на транзакції процесу**  **структура** | **Зберігається стан системи розділений на окремі цілісні сховища, щоб мінімізувати зв'язок між ними.**  **Сервіси системи пов'язані між собою з чітко визначеними обов'язками. Сервіси системи можуть комбінуватися по-різному\* для різних процесів або транзакцій\*.**  **Однак для багатьох процесів потрібно більше міжсервісних комунікацій, ніж якби система була побудована навколо процесів.** |
| **Фундаментальний** | **Сервісний**  **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 :EDM** | **Сервіс\* дуги добре зрозумілий більшості людей і досить простий у проектуванні. Моніторинг налагодження дуги набагато простіший, ніж з неявним зв'язком\*, таким як потоки подій, і з системою, керованою подіями\*. Набір сервісів легше, ніж моноліт, швидко еволюціонувати і швидко випускати Компроміс полягає в тому, що з часом з'являється багато міжсервісних залежностей, що робить деякі типи еволюції більш складними (наприклад, заміна сервісу].** |
|  |  |  | **Варіант 3, архітектура на основі сенату** |

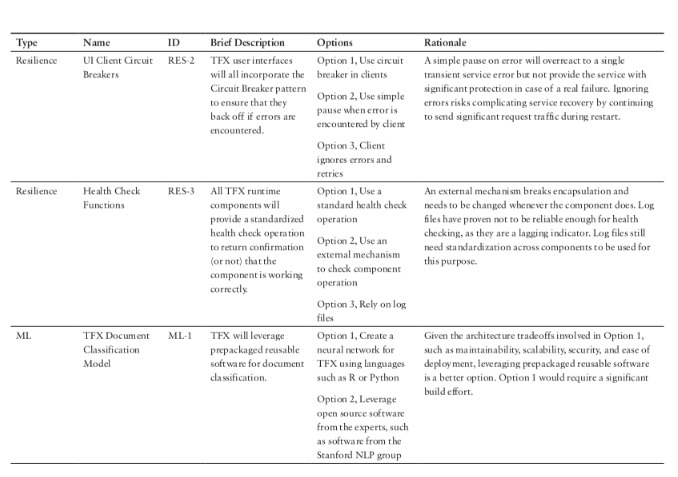












**Інші архітектурні проблеми**

Мультиоренда

План для TFX передбачає початкову версію, яка використовується першим клієнтом фін-технології, що її створила, - великою фінансовою установою, що займається торгівельним фінансуванням.

фінансиста, який її створив, - велика фінансова установа, що займається торговельним фінансуванням. Цей перший клієнт планує використовувати TFX для автоматизації торгового фінансування.

Цей перший клієнт планує використовувати TFX для автоматизації процесів торгового фінансування, пов'язаних з випуском, управлінням та обробкою акредитивів,

управлінням та обробкою акредитивів для своїх клієнтів. Однак, в майбутньому фінтех-компанія

фінтех хоче запропонувати платформу для процесів торгового фінансування іншим фінансовим

установам, які пропонують подібні послуги.

Фінтех не хоче займатися продажем готового 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

кількох банків, робочі навантаження яких мають бути повністю ізольовані одне від одного.

Існує щонайменше три способи, якими команда може досягти цього:

- Повна реплікація: Вся система реплікується для кожного клієнта, кожен у своєму

у власному ізольованому хмарному середовищі. Це найпростіша форма мультиоренди,

що не потребує майже ніяких змін в основному коді програми, і для невеликої кількості

клієнтів вона добре працює, усуваючи більшість ризиків безпеки, забезпечуючи ізоляцію ресурсів між клієнтами

ізоляцію ресурсів між клієнтами, а також незалежне оновлення та підтримку

для кожного клієнта. Проблема з цим підходом виникає, коли потрібно підтримувати велику кількість

клієнтів, що означає велику кількість (десятки або сотні) інсталяцій

інсталяцій, кожну з яких потрібно контролювати та керувати незалежно.

Їх також потрібно досить швидко оновлювати, щоб уникнути величезної операційної складності, пов'язаної з наявністю багатьох версій.

операційної складності, пов'язаної з одночасним запуском багатьох версій (і

іноді швидко для таких ситуацій, як виправлення безпеки).

- Реплікація бази даних: Передбачає наявність єдиного набору прикладних сервісів, але

але окремого набору баз даних для кожного клієнта. Таким чином, дані кожного клієнта фізично ізольовані

фізично ізольовані, але всі вхідні запити обробляються єдиним

набором прикладних сервісів. Такий підхід дає багатьом клієнтам впевненість у тому, що їхні дані будуть збережені.

що їхні дані не витечуть до інших клієнтів, а для великої кількості клієнтів

кількості клієнтів є операційно простішим, ніж повна реплікація. Однак

він вимагає досить значних змін у частинах основного додатку, оскільки сервіси додатків повинні

сервіси додатків повинні містити поточний ідентифікатор клієнта як частину контексту запиту, і це потрібно для того, щоб

контексті запиту, і це потрібно використовувати для вибору правильної бази даних для кожної

операції. Такий підхід зменшує, але не усуває операційне перевантаження, яке може виникнути при збільшенні кількості клієнтів.

перевантаження при збільшенні кількості клієнтів. Єдиний набір прикладних сервісів

значно зменшує складність моніторингу, оновлення та виправлень,

але оновлення все ще можуть 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

операційно і в довгостроковій перспективі. Це підхід, який використовується гіпер-скалерами, такими як Google.

масштабування, такі як Google, Microsoft, Amazon та Salesforce. Єдина система

використовується для підтримки всіх клієнтів, при цьому всі елементи системи є спільними

між ними. Це передбачає змішані дані клієнтів у базах даних, а також

спільні сервіси додатків. Величезною перевагою такого підходу (з точки зору

з точки зору оператора) є єдина система для моніторингу, експлуатації, виправлення та

модернізації. Недоліком є те, що він вимагає складних механізмів всередині

програми, щоб гарантувати, що конкретний запит завжди обробляється в

правильному контексті безпеки, а отже, ніколи не може отримати доступ до даних з неправильної сфери безпеки користувача.

користувацького контексту безпеки.

Підсумовуючи, наше початкове розгортання буде повною реплікацією, оскільки у нас

у нас немає клієнтів і нам потрібно якнайшвидше вийти на ринок для нашого першого клієнта,

але нам, ймовірно, доведеться реалізувати один з більш складних підходів

після того, як система буде підтримувати значну кількість банків.

***2. (Quality Attribute Workshop для проєкту)***На основі сформованих у попередніх завданнях навичок визначте тематику групового проєкту в межах підгрупи та підготуйте АТАМ-аналіз для обраного домену.

**Проект «DotaHub» (Вікіпедія):**

"Dotahub" - це онлайн-платформа, призначена для надання інформації про героїв та інші аспекти гри Dota 2. Проект розробляється з метою забезпечення гравців та шанувальників гри з усією необхідною статистикою, аналітикою та іншими ресурсами для поліпшення їхнього ігрового досвіду.

**Атрибут якості "Performance" у проекті "Dotahub":**

1. Швидкість завантаження інформації:

Завантаження статистики героїв: Забезпечення швидкого доступу до даних про кожного героя для гравців, які шукають стратегії та вдосконалюють свою гру.

2. Ефективність обробки та відображення даних:

Статистика гравців та матчів: Швидка обробка та відображення інформації про рейтинг, перемоги та інші показники для гравців, щоб вони могли визначати свій прогрес та порівнювати з іншими.

3. Зручний пошук та навігація:

Швидкий доступ до героїв та їхніх характеристик: Забезпечення ефективного інтерфейсу для швидкого пошуку та перегляду інформації про конкретних героїв.

4. Аналітика та стратегії:

Обробка та аналіз даних для гравців: Забезпечення інструментів для аналізу стратегій, побудови статистики героїв та рекомендацій гравцям.

5. Додаткові можливості:

Багатофункціональність: Забезпечення можливостей для гравців отримати інформацію щодо подій, турнірів та оновлень гри.

Спрямованість проекту:

Проект "Dotahub" зорієнтований на створення централізованої точки доступу для гравців гри Dota 2, де вони можуть легко знаходити та аналізувати важливі дані для покращення свого ігрового досвіду. За допомогою високої швидкості та ефективності, "Dotahub" стає незамінним ресурсом для всіх шанувальників гри Dota 2.

**Сценарії атрибуту якості "Performance" у проекті "DotaHub":**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **ID сценарію** | **DH-1** | **DH-2** | **DH-3** | **DH-4** | **DH-5** |
| **Атрибут якості** | Scalability | Scalability | Scalability | Scalability | Scalability |
| **Cценарій** | Користувач шукає героя та очікує негайного відображення його характеристик та інформації | Користувач шукає та переглядає інформацію про героїв з великою кількістю фотографій | Кілька користувачів одночасно шукають інформацію та переглядають героїв | Система автоматично визначає потребу в розширенні ресурсів під час великого потоку користувачів | Користувачі часто переглядають одни й ті ж героїв |
| **Бізнес-ціль** | Ефективне відображення даних | Швидке відображення великої кількості фотографій | Робота з багатьма користувачами одночасно | Автоматичне розширення ресурсів | Оптимізований кеш для частого перегляду |
| **Варіанти використання** | Тестування системи перед випуском | Тестування завантаження великої кількості фотографій | Тестування одночасного пошуку та перегляду героїв | Тестування автоматичного розширення ресурсів | Тестування ефективності кешу для частого перегляду |
| **Джерело стимулу** | Команда розробників | Команда розробників | Команда розробників | Автоматичне розширення | Команда розробників |
| **Cередовище** | Тестове середовище | Тестове середовище | Тестове середовище | Тестове середовище | Тестове середовище |
| **Відповідь** | Впровадження завершено | Впровадження завершено | Впровадження завершено | Впровадження завершено | Впровадження завершено |
| **Час реагування** | 1 день | 1 день | 1 день | 1 день | 1 день |
| **Проблеми** | Може трошки довше прогрузитись сторінка | Можливі проблеми з великими файлами | Може виникнути затримка при одночасному пошуку та перегляді | Може бути затримка при автоматичному розширенні ресурсів | Може бути невелика затримка при першому перегляді, потім швидко |
| **Артефакт** | Тестовий модуль | Тестовий модуль | Тестовий модуль | Тестовий модуль | Тестовий модуль |
| **Нотатки** | Забирає декілька годин на впровадження, не більше | Займає трохи більше часу для обробки великої кількості фотографій | Забирає декілька годин на впровадження, не більше | Забирає декілька годин на впровадження, не більше | Забирає декілька годин на впровадження, не більше |

**Атрибут якості "Reliability" у проекті "DotaHub":**

Атрибут якості "Reliability" (Надійність) в проекті "DotaHub" відіграє ключову роль у забезпеченні стабільності та безперебійності функціонування системи. Надійність визначається рівнем довіри, який користувачі мають до системи та її здатністю працювати без збоїв. Нижче наведено різні аспекти та сценарії, які стосуються атрибуту "Reliability" у контексті проекту "DotaHub":

1. Запобігання втраті даних користувачів: Надійність передбачає заходи для запобігання втраті даних користувачів, включаючи інформацію про гравців, матчі та інші дані, навіть у разі збоїв апаратної частини або програмного забезпечення.
2. Запобігання втраті функціональності в разі збоїв: Надійність системи включає механізми для автоматичного виявлення та виправлення збоїв, щоб забезпечити безперебійну роботу сервісу.
3. Забезпечення високої доступності системи: Надійна система повинна бути доступною для користувачів без значних перерв у роботі, особливо під час ключових моментів гри.
4. Відновлення після навантаження: Надійність також означає здатність системи відновлюватися після перевищення навантаження або після можливих атак.
5. Моніторинг і оповіщення про збої: Важливо мати системи моніторингу, які вчасно виявляють збої та надсилають повідомлення адміністраторам для швидкого реагування та виправлення.
6. Резервне збереження даних: Заходи для регулярного резервного збереження даних допомагають запобігти втраті інформації у випадку технічних проблем чи атак.

**Сценарії атрибуту якості "Reliability" у проекті "DotaHub":**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **ID** | **QR-1** | **QR-2** | **QR-3** | **QR-4** | **QR-5** | **QR-6** |
| **Сценарій** | Аварійне вимкнення сервера | Виявлення та автоматичне виправлення збоїв | Визначення часу доступності системи | Відновлення системи після DDoS-атаки | Виявлення збоїв та час їх виправлення | Здійснення регулярних резервних копій |
| **Бізнес-ціль** | Зменшення часу простою системи у разі критич-них помилок | Збільшення надійності та стійкості системи | Визначення часу доступу до системи протягом місяця | Відновлення роботи системи після DDoS-атаки | Виявлення та ефективне виправлення невдач роботи системи | Забезпечення можливості відновлення даних в разі потреби |
| **Варіанти викорис-тання** | Автоматичне відключення серверу | Виявлення та виправлення збоїв роботи системи | Моніторинг часу доступності системи | Реагування та відновлен-ня після DDoS-атаки | Заходи для виявлення та виправлення збоїв | Резервне копіювання та віднов-лення даних |
| **Атрибути якості** | Reliability | Reliability | Reliability | Reliability | Reliability | Reliability |
| **Стимул** | Серйозна помилка | Автоматичне виявлення | Системна статистика | DDoS-атака | Виявлення збоїв | Регулярне резервне копіювання |
| **Джерело стимулу** | Автоматична система | Система | Моніторинг системи | Зовнішня загроза | Моніторинг системи | Автоматич-не резерву-вання |
| **Середовище** | Критичний випадок | Робоче середовище | Продуктивне середовище | Уразливий сервер | Робоче середовище | Безпечне зберігання |
| **Відповідь** | Вимкнення сервера | Виправлення збоїв | Звіт про доступність системи | Автоматичне відновлення системи | Виправлення невдач | Відновлення даних |
| **Захід реагування** | Запуск аварійного резерву | Автоматичне виправлення | Аналіз часу доступності | Використання резервного сервера | Автоматичне відновлення | Відновлення інформації |
| **Проблеми** | Можливі проблеми з безпекою | Можливі проблеми при діагно-стуванні | Можливі проблеми з завантаженням | Можливі проблеми зі стійкістю | Можливі проблеми при аналізі збоїв | Можливі проблеми з пристроями зберігання |
| **Артефакт** | Аварійний резерв |  | Статистичний звіт | Резервний сервер | Документ з виправленнями | Резервний носій |
| **Нотатки** | Забезпечення безпеки в разі аварійного вимкнення |  | Визначення та аналіз доступності системи | Запуск резервного сервера після атаки | Відзначення виправлених збоїв | Актуаль-ність та доступність резервів |

**Атрибут якості " Security" у проекті "DotaHub":**

1. Продумана система ідентифікації та аутентифікації: "DotaHub" використовує надійні методи ідентифікації та аутентифікації користувачів, гарантуючи безпеку та впевненість у справжності кожного користувача.
2. Шифрування важливих даних: Всі конфіденційні дані, збережені в "DotaHub", піддаються потужному шифруванню, забезпечуючи їхню безпеку та недоступність для несанкціонованого доступу.
3. Захист від різновидів атак: "DotaHub" використовує передові техніки захисту від різних видів атак, таких як SQL-ін'єкції, щоб забезпечити цілісність та безпеку обробки даних.
4. Система моніторингу та аудиту безпеки: Регулярний моніторинг та аудит системи дозволяють "DotaHub" вчасно виявляти та ефективно реагувати на будь-які події, що можуть вказувати на можливі порушення безпеки.
5. Захист серверів від несанкціонованого доступу: Високі стандарти безпеки для серверів "DotaHub" гарантують їхню недоступність для несанкціонованих осіб та неперебільшену стійкість до зовнішніх загроз.
6. Захист від кросс-сайтового скриптування (XSS): "DotaHub" використовує ефективні методи для запобігання атакам XSS, забезпечуючи безпеку користувачів під час взаємодії з веб-інтерфейсом.

**Сценарії атрибуту якості " Security" у проекті "DotaHub":**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **ID сценарію** | **QS-1** | **QS-2** | **QS-3** | **QS-4** | **QS-5** | **QS-6** |
| **Cценарій** | Автентифікація та авторизація користувачів | Шифрування даних | Захист від атак | Моніторинг та аудит безпеки | Захист від несанкціонованого доступу до серверів | Захист від кросс-сайтового скриптування (XSS) |
| **Бізнес-ціль** | Забезпечення безпеки автентифікації та авторизації | Захист від несанкціонованого доступу до конфіденційних даних | Захист від різних видів атак, таких як SQL-ін'єкції, переповнення буфера тощо | Забезпечення системного моніторингу та аудиту для виявлення несправностей в безпеці | Забезпечення безпеки серверів від несанкціонованого доступу | Захист від XSS-атак та збереження цілісності даних |
| **Варіанти викорис-тання** | Тестування автентифікації та авторизації | Тестування шифрування даних | Тестування захисту від атак | Тестування моніторингу та аудиту безпеки | Тестування захисту від несанкціонованого доступу | Тестування захисту від XSS |
| **Атрибути якості** | Security | Security | Security | Security | Security | Security |
| **Стимул** | Підозра на несанкціонований доступ | Намагання доступу до зашифрованих даних | Намагання провести SQL-ін'єкцію | Несподіване змінення в налаштуваннях безпеки | Намагання несанкціонованого доступу до сервера | Введення скриптів в текстові поля |
| **Джерело стимулу** | Команда розробників | Команда розробників | Команда розробників | Команда розробників | Команда розробників | Команда розробників |
| **Середо-вище** | Тестове середовище | Тестове середовище | Тестове середовище | Тестове середовище | Тестове середовище | Тестове середовище |
| **Відповідь** | Впровадження завершено | Впровадження завершено | Впровадження завершено | Впровадження завершено | Впровадження завершено | Впровадження завершено |
| **Час реагу-вання** | 1 день | 1 день | 1 день | 1 день | 1 день | 1 день |
| **Пробле-ми** | Можливі проблеми з авторизацією | Можливі проблеми з дешифруванням | Можливі проблеми з виявленням атак | Можливі проблеми з точністю моніторингу | Можливі проблеми зі зміною налаштувань сервера | Можливі проблеми з виявленням XSS-уразливостей |
| **Артефакт** | Тестовий модуль | Тестовий модуль | Тестовий модуль | Тестовий модуль | Тестовий модуль | Тестовий модуль |
| **Нотатки** | Забирає кілька годин на впровадження, не більше | Забирає кілька годин на впровадження, не більше | Забирає кілька годин на впровадження, не більше | Забирає кілька годин на впровадження, не більше | Забирає кілька годин на впровадження, не більше | Забирає кілька годин на впровадження, не більше |

**Атрибут якості " Scalability" у проекті "DotaHub":**

1. Горизонтальна масштабованість: Здатність додавати нові сервери або вузли для обробки додаткового трафіку.
2. Вертикальна масштабованість: Здатність підвищувати потужність і ресурси на існуючих серверах.
3. Масштабованість баз даних: Здатність оптимізувати запити та розподіляти навантаження в базі даних.
4. Автоматичне масштабування: Система автоматично визначає потребу в розширенні ресурсів.
5. Масштабуваність серверів для високої доступності: Можливість розширення серверів для забезпечення високої доступності.
6. Скорочення ресурсів в періоди зменшеного навантаження: Система ефективно управляє ресурсами для оптимізації витрат.

**Сценарії атрибуту якості " Scalability" у проекті "DotaHub":**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **ID сценарію** | **QS-1** | **QS-2** | **QS-3** | **QS-4** | **QS-5** | **QS-6** |
| **Cценарій** | Збільшення кількості користувачів | Розширення функціональності | Збільшення обсягу даних | Зростання трафіку | Забезпечення горизонтального масштабування | Обробка паралельних запитів |
| **Бізнес-ціль** | Забезпечення ефективності при зростанні користувачів | Забезпечення розширення функціональності | Забезпечення оптимально-го доступу до даних | Забезпечення стабільності та швидкості завантажен-ня | Забезпечення можливості збільшення потужності системи | Забезпечення паралельної обробки запитів |
| **Варіанти викорис-тання** | Тестування збільшення кількості користувачів | Тестування розширення функціональності | Тестування збільшення обсягу даних | Тестування зростання трафіку | Тестування горизонталь-ного масштабува-ння системи | Тестування паралельної обробки запитів |
| **Атрибути якості** | Scalability | Scalability | Scalability | Scalability | Scalability | Scalability |
| **Стимул** | Збільшення кількості користувачів | Додавання нових функцій або модулів | Тестування збільшення обсягу даних | Зростання трафіку | Додавання нових серверів або вузлів для підтримки навантажен-ня | Перевірка паралельної обробки запитів |
| **Джерело стимулу** | Команда розробників | Команда розробників | Команда розробників | Команда розробників | Команда розробників | Команда розробників |
| **Середо-вище** | Тестове середовище | Тестове середовище | Тестове середовище | Тестове середовище | Тестове середовище | Тестове середовище |
| **Відповідь** | Впровадження завершено | Впровадження завершено | Впровадження завершено | Впровадження завершено | Впровадження завершено | Впровадження завершено |
| **Час реагу-вання** | 1 день | 1 день | 1 день | 1 день | 1 день | 1 день |
| **Пробле-ми** | Межі масштабування системи | Можливі проблеми з адаптацією до змін у функціоналі | Проблеми з обробкою великих обсягів даних | Проблеми із стабільністю та швидкіс-тю заванта-ження | Проблеми із додаванням нових серверів та вузлів | Проблеми з паралельною обробкою запитів |
| **Артефакт** | Тестовий модуль | Тестовий модуль | Тестовий модуль | Тестовий модуль | Тестовий модуль | Тестовий модуль |

**Utility-дерево:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **ID** | **| Атрибут** | **Сценарій** | **Imp** | **Com** |
| **QDH-1** | Зручний інтерфейс | Користувач шукає інформацію про гравців та героїв у Dota 2 | H | M |
| **QDH-2** | Швидкість завантаження | Користувач очікує швидкого завантаження статистики та фотографій | H | M |
| **QDH-3** | Можливість створення стратегій | Користувач може створювати та зберігати власні стратегії гри | H | M |
| **QDH-4** | Інтеракція з іншими гравцями | Можливість обміну думками та інформацією з іншими гравцями | H | M |
| **QDH-5** | Захист особистих даних | Дані користувачів повинні бути захищені від несанкціонованого доступу | H | M |
| **QDH-6** | Автоматичне відновлення | Автоматичне відновлення роботи системи після виявлення відмови | H | M |
| **QDH-7** | Шифрування даних | Захист інформації про гравців та героїв під час передачі та зберігання | H | M |
| **QDH-8** | Виявлення та виправлення збоїв | Автоматичне виявлення та виправлення системних збоїв для забезпечення надійності | H | M |
| **QDH-9** | Виявлення атак та аналіз подій | Захист від різних видів атак та ведення запису подій для подальшого аналіз у | H | M |
| **QDH-10** | Горизонтальна масштабованість | Здатність системи розширюватися горизонтально для обробки більшої кількості запитів | H | M |
| **QDH-11** | Вертикальна масштабованість | Можливість підвищення потужності і ресурсів на існуючих серверах | H | M |
| **QDH-12** | Оптимізація роботи з базою даних | Забезпечення ефективної роботи з базою даних для оптимізації швидкості та ресурсів | H | M |
| **QDH-13** | Автоматичне визначення потреби в розширенні | Система може автоматично визначати необхідність розширення ресурсів | H | M |
| **QDH-14** | Регулярні резервні копії | Здійснення регулярних резервних копій даних та їх відновлення у разі потреби | H | M |
| **QDH-15** | Масштабування серверів для високої доступності | Забезпечення можливості розширення | H | M |

***3. (UML/C4-модель проєкту)***Сформуйте архітектурний репозиторій для проєкту, включивши в нього технічні специфікації та діаграми, що описують високорівневу архітектуру програмного продукту.

**Діаграма класів:**

@startuml

class DotaHubSystem {

  - performance: Performance

  - reliability: Reliability

  - security: Security

  - scalability: Scalability

}

class Performance {

  + searchAndLoadInformation(): void

  + processAndDisplayData(): void

  + createAndSaveStrategies(): void

  + interactWithOtherPlayers(): void

  + protectPersonalData(): void

  + automaticRecovery(): void

}

class Reliability {

  + serverShutdown(): void

  + automaticFailureDetectionAndCorrection(): void

  + systemAvailability(): void

  + systemRecoveryAfterDDoS(): void

  + detectAndCorrectFailures(): void

  + regularDataBackupAndRestore(): void

}

class Security {

  + userAuthentication(): void

  + encryptDataTransmissionAndStorage(): void

  + defendAgainstAttacks(): void

  + detectSecurityViolations(): void

  + physicalAndLogicalProtection(): void

  + preventMaliciousScripts(): void

}

class Scalability {

  + horizontalScaling(): void

  + verticalScaling(): void

  + handleIncreasedDatabaseLoad(): void

  + automaticResourceScaling(): void

  + serverExpansionForHighAvailability(): void

  + releaseExcessResources(): void

}

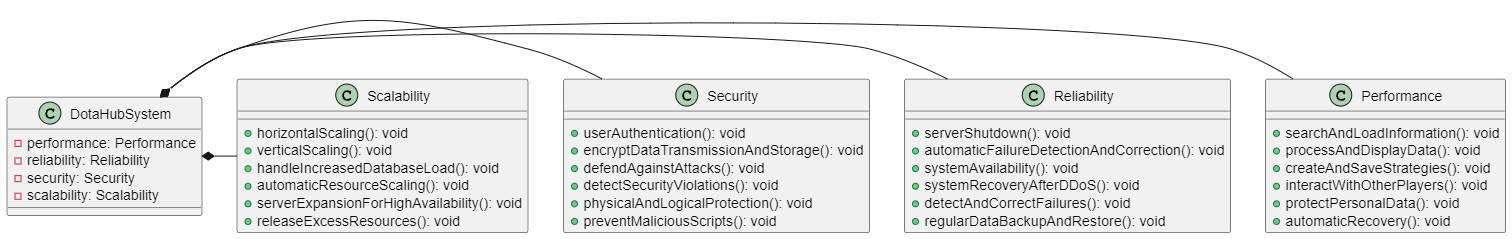
DotaHubSystem \*-right- Performance

DotaHubSystem \*-right- Reliability

DotaHubSystem \*-right- Security

DotaHubSystem \*-right- Scalability

@enduml

******

**Діаграма прецендентів:**

@startuml

left to right direction

actor Player

usecase "Search and Load Information" as UC1

usecase "Process and Display Data" as UC2

usecase "Create and Save Strategies" as UC3

usecase "Interact with Other Players" as UC4

usecase "Protect Personal Data" as UC5

usecase "Automatic Recovery" as UC6

actor Admin

usecase "Manage Heroes" as UC7

usecase "Manage Player Accounts" as UC8

usecase "Monitor Game Statistics" as UC9

usecase "Resolve Disputes" as UC10

actor System as DotaHubSystem

usecase "Server Shutdown" as UC11

usecase "Automatic Failure Detection and Correction" as UC12

usecase "System Availability" as UC13

usecase "System Recovery After DDoS" as UC14

usecase "Detect and Correct Failures" as UC15

usecase "Regular Data Backup and Restore" as UC16

actor Security

usecase "User Authentication" as UC17

usecase "Encrypt Data Transmission and Storage" as UC18

usecase "Defend Against Attacks" as UC19

usecase "Detect Security Violations" as UC20

usecase "Physical and Logical Protection" as UC21

usecase "Prevent Malicious Scripts" as UC22

actor Scalability

usecase "Horizontal Scaling" as UC23

usecase "Vertical Scaling" as UC24

usecase "Handle Increased Database Load" as UC25

usecase "Automatic Resource Scaling" as UC26

usecase "Server Expansion for High Availability" as UC27

usecase "Release Excess Resources" as UC28

Player --> UC1

Player --> UC2

Player --> UC3

Player --> UC4

Player --> UC5

Player --> UC6

Admin --> UC7

Admin --> UC8

Admin --> UC9

Admin --> UC10

DotaHubSystem --> UC11

DotaHubSystem --> UC12

DotaHubSystem --> UC13

DotaHubSystem --> UC14

DotaHubSystem --> UC15

DotaHubSystem --> UC16

Security --> UC17

Security --> UC18

Security --> UC19

Security --> UC20

Security --> UC21

Security --> UC22

Scalability --> UC23

Scalability --> UC24

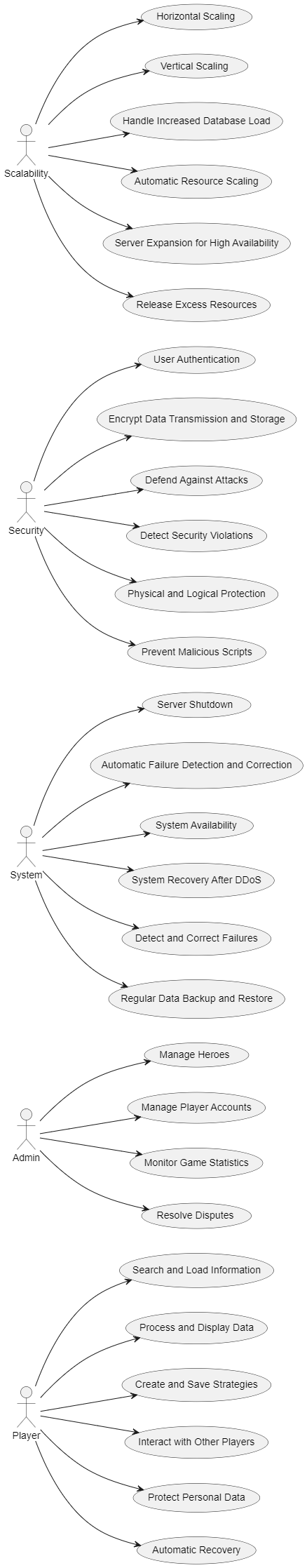
Scalability --> UC25

Scalability --> UC26

Scalability --> UC27

Scalability --> UC28

@enduml

****

**ER-діаграма:**

@startuml

entity Player {

  + playerID (PK)

  --

  username

  email

  password

  mmr

  playStyle

}

entity Hero {

  + heroID (PK)

  --

  name

  type

  strength

  agility

  intelligence

}

entity Match {

  + matchID (PK)

  --

  date

  duration

  result

}

entity Strategy {

  + strategyID (PK)

  --

  description

  tags

}

entity Team {

  + teamID (PK)

  --

  name

  logo

}

entity Admin {

  + adminID (PK)

  --

  username

  email

  password

}

Player --|> Team : "Joins"

Player --|> Strategy : "Creates"

Player --|> Match : "Participates In"

Hero <-- Match : "Picked In"

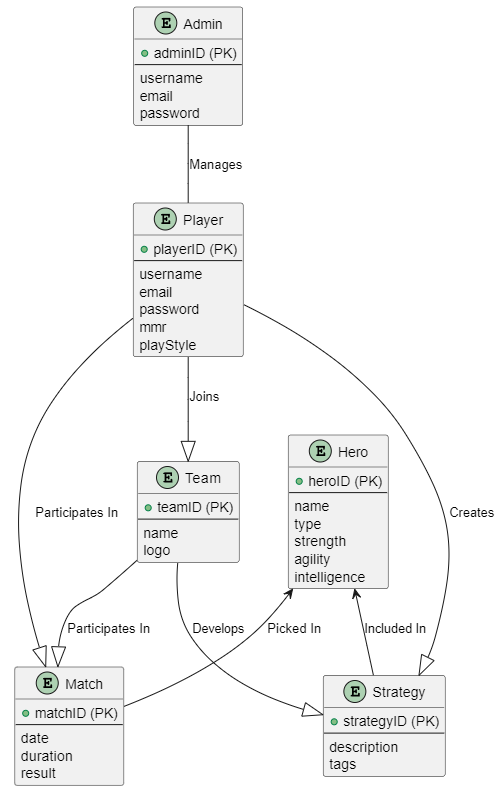
Hero <-- Strategy : "Included In"

Admin -- Player : "Manages"

Team --|> Match : "Participates In"

Team --|> Strategy : "Develops"

@enduml

****