КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БУДІВНИЦТВА І АРХІТЕКТУРИ

Факультет автоматизації і інформаційних технологій

Кафедра управління проєктами

ПОЯСНЮВАЛЬНА ЗАПИСКА
ДО АТЕСТАЦІЙНОЇ РОБОТИ
НА ЗДОБУТТЯ ОСВІТНЬОГО СТУПЕНЯ МАГІСТРА

на тему:

Управління проєктом розробки архітектури мікросервісу для отримання даних по клієнту

Рижук Віктор Віталійович
(прізвище, ім’я та по батькові студента повністю)

Київ 2022 р.

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БУДІВНИЦТВА І АРХІТЕКТУРИ

Факультет: Автоматизації і інформаційних технологій
Кафедра: Управління проектами
Освітній рівень: Магістр за освітньо-професійною програмою
Галузь знань: 12 Інформаційні технології
Спеціальність: 122. Комп'ютерні науки
Спеціалізація: Управління проектами

ЗАТВЕРДЖУЮ
Завідувач кафедри
Бушуєв С. Д.
"___" _____ 2022___ року

З А В Д А Н Н Я
ДО ВИКОНАННЯ АТЕСТАЦІЙНОЇ РОБОТИ НА ЗДОБУТТЯ ОСВІТНЬОГО СТУПЕНЯ МАГІСТРА

Рижук Віктор Віталійович

(прізвище, ім'я та по батькові студента)

1. Тема роботи: Управління проєктом розробки архітектури мікросервісу для отримання даних по клієнту
затверджена наказом ректора КНУБА № __1537/2__ від "__07_" жовтня 2022 року
2. Керівник роботи:
Бойко Євгенія Григорівна., к.т.н., доц

(прізвище, ім'я та по батькові, науковий ступінь, вчене звання)

3. Строк подання студентом роботи до захисту: 15.12.2022

4. Зміст пояснювальної записки (перелік питань, які слід розробити):

а) теоретичний розділ: теорія мікросервісної архітектури, створення та розгортання мікросервісів, CI/CD метод, комунікація між мікросервісами за допомогою REST API, клієнтська база;

б) дослідницько-аналітичний розділ: аналіз ринку схожих програмних продуктів, аналіз діяльності Креді Агріколь Банк, SWOT-аналіз, опис проєкту; статут проєкту.

в) рекомендаційний розділ: організаційна структура проєкту, матриця відповідальності, управління терміном проєкту, управління вартістю в проєкті, управління ресурсами в проєкті, управління якістю, управління ризиками;

г) дослідження з використанням комп’ютерних технологій: Microsoft Office Word для оформлення роботи, Power Point для створення презентації, Microsoft Office Project для створення моделі проєкту, Draw.io для таблиць, схем;

5. Графічний матеріал за розділами:

таблиці, малюнки, структура декомпозиції робіт проєкту, організаційна структура проєкту, календарно-мережевий графік робіт проєкту.

6. Календарний план виконання роботи:

Види робіт та їх зміст	Дата виконання
Збір матеріалів обраного напряму роботи	01.10.22 – 05.10.22
Опрацювання та аналіз матеріалів роботи	06.10.22 - 09.10.22
Вступ	10.10-20.10.22
Розділ 1. УПРАВЛІННЯ ПРОЄКТАМИ ТА АНАЛІЗ СЕРВІСНОЇ АРХІТЕКТУРИ	21.10- 31.10.22
Розділ 2. РОЗРОБКА СЕРВІСУ ДЛЯ ОТРИМАННЯ ДАНИХ ПО КЛІЄНТУ	01.11-11.11.22
Розділ 3. РЕАЛІЗАЦІЯ УПРАВЛІННЯ ПРОЄКТОМ РОЗРОБКИ АРХІТЕКТУРИ МІКРОСЕРВІСУ ДЛЯ ОТРИМАННЯ ДАНИХ ПО КЛІЄНТУ	12.11-22.11.22
Висновки	24.11.22
Остаточне оформлення роботи	23.11-30.11.22
Перевірка роботи на плагіат	01.12.22
Попередній захист роботи на кафедрі	06.12.22
Направлення роботи на рецензування	02.12.22

7. Консультанти розділів атестаційної випускної роботи

Розділ	Прізвище, ініціали та посада консультанта	Перевірив	
		дата	підпис
Розділ 1.			
Розділ 2.			
Розділ 3.			

8. Дата видачі завдання _____

Зав. кафедри _____ Бушуєв С.Д.
(підпис) (прізвище та ініціали)

Керівник _____ Бойко Є.Г.
(підпис) (прізвище та ініціали)

Студент _____ Рижук В.В.
(підпис) (прізвище та ініціали)

РЕЗЮМЕ (summary) ***до атестаційної випускної роботи студента:***			Рижук Віктор Віталійович
ЗВО	Київський національний університет будівництва і архітектури		
Тема	Управління проєктом розробки архітектури мікросервісу для отримання даних по клієнту		
Освітній ступінь	Магістр за освітньо-професійною програмою навчання		
Факультет	Автоматизації і інформаційних технологій		
Кафедра	Управління проектами		
Спеціальність	122. “Комп’ютерні науки”		
Спеціалізація	Управління проектами		
Керівник	Бойко Євгенія Григорівна, к.т.н., доц.		
Обсяг роботи:	*пояснювальна записка, сторінок*	*розділів*	*слайдів презентації*
	112	*3*	*13*
Розділ 1. УПРАВЛІННЯ ПРОЄКТАМИ ТА АНАЛІЗ СЕРВІСНОЇ АРХІТЕКТУРИ	У першому розділі було проводено аналіз та характеристику мікросервісної архитектури, плюси та мінуси цього підходу. Проаналізовано найпопулярніші та зручні засоби для розробки мікросервісів, їх розгортання та розробки. Проведено порівняння монолітної та мікросервісної архітектури. Описана комунікація між мікросервісами за допомогою REST API. Після проведення аналізу було визначено методи та засоби для розробки, розгортання, журнулювання та підтримки мікросервісної архітектури та в цілому - розробки проєкту.		
Розділ 2. РОЗРОБКА СЕРВІСУ ДЛЯ ОТРИМАННЯ ДАНИХ ПО КЛІЄНТУ	У другому розділі було проведено аналіз ринку схожих програмних продуктів, розглянуто питання що таке клієнтська база, для чого вона, та чим важлива. Розписано SWOT, проведено аналіз діяльності Креді Агріколь Банк. У розділі також було застосовано спосіб визначення ролей та обов’язків у групі за будь яким завданням, віхою або очікуваним результатом проекту . Дотримуючись принципів RACI, можна чітко розподіляти обов’язки та усувати плутанину, це зображено на матриці відповідальності(RACI).		

Розділ 3. РЕАЛІЗАЦІЯ УПРАВЛІННЯ ПРОЄКТОМ РОЗРОБКИ АРХІТЕКТУРИ МІКРОСЕРВІСУ ДЛЯ ОТРИМАННЯ ДАНИХ ПО КЛІЄНТУ	У третьому розділі було розроблено статут проєкта та ієрархічну структуру робіт та представлено графік робіт за проєктом, який створений за допомогою використанням програмного продукту MS Project. Розроблено план управління термінами, вартістю, ресурсами, якістю та ризиками. Також, проаналізовані зовнішні та внутрішні ризи, їх йомвірність та вплив, та в результаті способи їх уникнення чи пом'якшення.
Висновки по роботі:	В атестаційній роботі на здобуття освітнього ступеня магістра було проводено аналіз та характеристику мікросервісної архитектури, плюси та мінуси цього підходу. Проаналізовано найпопулярніші та зручні засоби для розробки мікросервісів, їх розгортання та розробки. Проведено порівняння монолітної та мікросервісної архітектури. Описана комунікація між мікросервісами за допомогою REST API, визначено методи та засоби для розробки, розгортання, журнулювання та підтримки мікросервісної архітектури та в цілому - розробки проєкту. Проведено аналіз ринку схожих програмних продуктів, розглянуто питання що таке клієнтська база, для чого вона, та чим важлива. Також які саме бувають клієнтські бази, це важливо, бо залежно від того, як компанія веде клієнтську базу, накопичує та аналізує інформацію. Розписано та представлено що таке SWOT. Визначено команду проєкту складається зі співробітників IT департаменту. Визначені можливі обмеження проєкту, описані управління якістю, ризиками, ресурсами.
Ключові слова: мікросервісна архітектура, управління проєктом, застосування інформаційних технологій, проєкт. **Keywords:** microservice architecture, project management, application of information technologies, project.	

Укладач: Рижук В.В.

Керівник: Бойко Є.Г.

"___" грудня 2022р.

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БУДІВНИЦТВА І АРХІТЕКТУРИ

Факультет автоматизації і інформаційних технологій

Кафедра управління проєктами

ЗАТВЕРДЖУЮ
Завідувач кафедри
Бушуєв С. Д.
“___” _____2022 року

ПОЯСНЮВАЛЬНА ЗАПИСКА
ДО АТЕСТАЦІЙНОЇ РОБОТИ
НА ЗДОБУТТЯ ОСВІТНЬОГО СТУПЕНЯ МАГІСТРА

Управління проєктом розробки архітектури мікросервісу для отримання даних по клієнту
(назва)

Виконав студент групи:
Рижук Віктор Віталійович
(прізвище, ім’я та по батькові повністю)

Спеціальність: 122. “Комп’ютерні науки”
Спеціалізація: Управління проєктами

Керівник: Бойко Євгенія Григорівна
(прізвище, ініціали,)
К.т.н., доцент
науковий ступінь, вчене звання

Рецензент:_____
(прізвище, ініціали,)

_____
науковий ступінь, вчене звання посада

Київ 2022 р.

ЗМІСТ

ВСТУП

Мікросервісна архітектура має багато переваг таких як масштабованість, легке обслуговування та часте розгортання. Але реалізувати ці шаблони складно. Однією з проблем, з якою можна зіткнутися, є керування базою даних мікросервісів. Архітектура мікросервісів — це тип архітектури додатків, де додаток розробляється як набір служб. Він надає основу для незалежної розробки, розгортання та підтримки діаграм і служб архітектури мікросервісів.

Таким чином, кожен окремий сервіс розробляється як незалежний блок, який підключається до інших сервісів. Оскільки блоки не перекриваються, набагато легше розробити кожну службу або виявити помилку. Крім того, коли одна служба дає збій, решта системи продовжує працювати. Це означає, що ви можете будувати своє рішення «цеглинка за цеглинкою», забезпечуючи водночас нескінченні можливості для масштабування свого бізнесу.

Тому децентралізація є основною характеристикою архітектури мікросервісів . Якщо ми порівняємо мікросервіси з монолітною архітектурою , ви побачите, що мікросервіси незалежні один від одного, і вам не потрібно буде рефакторинг всього рішення, коли ви збираєтеся його оновлювати або масштабувати

Архітектура мікросервісів дуже проста таким чином, коли кожна окрема послуга упакована в окремий мікросервіс, зрозуміти структуру вашого програмного забезпечення дуже легко. Ви бачите окремі блоки зі службами, підключеними один до одного.

Багаторазове використання. Поки сервіси живуть у своїх невеликих ящиках окремо один від одного, ви можете використовувати деякі частини коду багато разів, не побоюючись, що занадто багато копіювання завдасть шкоди вашому цифровому рішенню.

Швидка ізоляція несправності. Знову ж таки, зберігання служб в

автономних мікросервісах дозволяє командам розробників виявляти будь-які дефекти та ізолювати цей мікросервіс від решти системи, тим самим прискорюючи процес забезпечення якості та перевірки коду.

Свобода для самостійного розвитку. Оскільки кожен мікросервіс є автономним, це означає, що кілька мікросервісів можна розробляти одночасно, щоб скоротити час.

Але деякі недоліки також є.

Відсутність менеджменту. Коли кожна послуга розробляється окремо, дуже важко зібрати разом усі блоки з точки зору команд, стратегічного планування бізнесу тощо. Ви повинні мати можливість бачити цілісну картину свого рішення, що у випадку мікросервісів може бути важким.

Можливі перевантаження трафіку даних. Блоки з мікросервісами з'єднані один з одним різними протоколами, такими як HTTP. Отже, багато даних перекачується вперед і назад у програмній системі. Такі «пробки» можуть спричинити потребу в додатковій потужності сервера або додатковій роботі з підтримки клієнтів.

Послідовністю даних важко керувати. Знову ж таки, кожен блок має свою власну базу даних. Уся програма може мати дані в різних форматах, якими важко керувати та захищати разом. Доведеться докласти додаткових зусиль, щоб упорядкувати всі свої набори даних.

Об'єктом дослідження є мікросервісна архітектура як основна складова для підвищення ефективності розробки та реалізації проектів.

Предмет дослідження – реляційні та нереляційні бази даних, такі як Oracle, MongoDB, програмні забезпечення для розгортання, керування мікросервісами – Docker.

Мета і завдання магістерської роботи – розробити мікросервісну архітектуру для отримання інформації по клієнту для додатку.

Методи дослідження. Завдання вирішувалося на основі сучасних концепцій і методологій управління проектами. При дослідженні використані: методи управління проектами та програмами.

З метою проведення аналізу та постановки цілей використовуються методи SMART та SWOT аналізу. Для управління обмеженнями та часом у проєкті буде застосований метод побудови календарного графіку проєкту у програмі MS Project.

Для роботи з командою, проєкту будуть використані методи побудови організаційної структури проєкту, розподіл обов'язків та функції за допомогою побудови матриці відповідальності.

Використання різних методів для швидкої реалізації проєктів, попри свій успіх, мають і ризики. Тому будуть використані методи аналізу ризиків та розробки заходів з протидії.

Актуальність: розроблена архітектура на мікросервісі вирішить проблему багаторазового використання, швидкість розробки та підтримки, доступності до конкретних даних.

РОЗДІЛ 1. УПРАВЛІННЯ ПРОЄКТАМИ ТА АНАЛІЗ СЕРВІСНОЇ АРХІТЕКТУРИ

1.1 Управління проєктами

Щоб зрозуміти управління проектами, необхідно глибше розглянути, що являє собою проект. По суті, проекти - це тимчасові спроби створити цінність за допомогою унікальних продуктів, послуг і процесів. Деякі проекти створені для швидкого вирішення проблем. Інші вимагають подовжених часових рамок для отримання результатів, які не потребуватимуть серйозних покращень за межами запланованого технічного обслуговування (наприклад, дороги загального користування).

Звичайно, деякі проекти будуть сумішшю обох цих речей. Це стосується всього, починаючи від розробки нового програмного забезпечення і закінчуючи плануванням ліквідації наслідків стихійних лих. Тим не менш, це все дуже загальна інформація про те, що таке проект. Коли ми розбиваємо їх більш конкретно, ми бачимо, що проекти є об’єднанням завдань, заходів і результатів, які повинні бути ретельно структуровані та виконані для досягнення бажаного результату.

Перш ніж буде досягнуто результату, кожен аспект проекту повинен пройти через етапи ініціювання, планування та виконання. Цей процес відомий як життєвий цикл управління проектами, і він є джерелом життя успішних проектів. Крім того, цей цикл дозволяє керівникам проектів ретельно планувати кожне завдання та діяльність, щоб забезпечити найвищі шанси на успіх. Загалом, проект - це добре спланована робота, яка має певний початок і кінець.

Керівники проектів керують проектами.

Усі проекти - це тимчасова спроба створити цінність за допомогою унікального продукту, послуги чи результату. Усі проекти мають початок і кінець. У них є команда, бюджет, графік і набір очікувань, яким команда повинна відповідати. Кожен проект є унікальним і відрізняється від звичайних операцій -

поточної діяльності організації, - оскільки проекти завершуються після досягнення мети.

Зміна характеру роботи через технологічний прогрес, глобалізацію та інші фактори означає, що все частіше робота організовується навколо проектів, а команди об’єднуються на основі навичок, необхідних для виконання конкретних завдань.

Керівниками цих проектів є професіонали проекту - люди, яких навмисно чи через обставини попросили переконатися, що команда проекту досягає своїх цілей. Професіонали проекту використовують багато різних інструментів, методів і підходів для задоволення потреб проекту.

Деякі проекти потрібні для швидкого вирішення проблем із розумінням того, що протягом певного періоду часу будуть внесені покращення. Інші проекти мають довшу тривалість і/або створюють продукт або інший результат, який не потребуватиме значних покращень за межами запланованого технічного обслуговування, наприклад, шосе.

Ще інші будуть сумішшю обох цих типів проектів. Професіонали проекту використовують різноманітні навички та знання, щоб залучати та мотивувати інших для досягнення цілей проекту. Професіонали проектів мають вирішальне значення для успіху проектів, і вони дуже затребувані, щоб допомогти організаціям досягти їхніх цілей.

Управління проектами стимулює зміни. Протягом всієї історії людства управління проектами завжди практикувалося неформально, але воно почало формуватися як окрема професія в середині 20 століття, коли група далекоглядних людей із аерокосмічної, інженерної, фармацевтичної та телекомунікаційної галузей усвідомила, що світ змінюється. потрібні нові інструменти. Спонукані необхідністю вирішення питань планування та ресурсів, пов’язаних із дедалі складнішими проектами, вони зустрілися, щоб розпочати створення та стандартизацію інструментів для нової професії. А в 1969 році народився Інститут управління проектами (PMI).

Сьогодні ми живемо в проектній економіці , де проекти є рушійною силою

того, як виконується робота, реалізуються зміни та доставляється цінність. У проектній економіці всесвітнє зростання управління проектами доводить його цінність як:

- як визнана та стратегічна організаційна компетентність;
- як предмет для навчання і виховання;
- як кар’єрний шлях.

Нині загальновизнано, що базові знання з управління проектами можуть стати цінними для людей, які виконують різноманітні ролі в різноманітних справах. Навички управління проектами можуть допомогти молодому студенту, який працює над науковим проектом, досягти успіху, або керівнику компанії вирішити особисті суперечки. Ці навички можуть допомогти медсестрі оптимізувати зміни змін, щоб покращити час реагування пацієнтів у своїй палаті. Вони можуть допомогти IT-фахівцю розробити інноваційне програмне забезпечення в рекордно короткі терміни або допомогти урядовій установі покращити послуги, які вони надають, економнішим способом.

Ролі та обов’язки керівника проекту.

Як уже згадувалося, роль і навіть посада керівника проекту можуть дещо відрізнятися залежно від місця, але основи того, що керівник проекту робить для команди, досить послідовні (хоча деякі можуть бути менш формальними, ніж інші).

Роль керівника проекту передбачає багато завдань і обов’язків, зокрема:

1. Впровадження методології управління проектами
2. Планування та визначення обсягу
3. Встановлення та управління очікуваннями
4. Процес крафта
5. Створення проектних планів
6. Управління завданнями
7. Розподіл і планування ресурсів
8. Оцінка часу/вартості

9. Аналіз та управління ризиками та проблемами
10. Моніторинг та звітність про стан проекту
11. Забезпечення керівництва командою
12. Стратегія впливу
13. Полегшення спілкування та співпраці
14. Планування та проведення зустрічей

Це багато, щоб включити в одну посадову інструкцію, яка фактично не несе жодної операційної чи управлінської відповідальності за команду, що працює над проектами.

Менеджери проектів часто перебувають у важкому становищі, коли намагаються втілити щось у життя, не маючи повноважень, щоб справді висунути проблему. Щоб бути ефективними, ви повинні завоювати довіру та повагу своїх команд і мати підтримку вищого керівництва.

Управління проектами - це використання конкретних знань, навичок, інструментів і методів для надання людям чогось цінного. Розробка програмного забезпечення для вдосконалення бізнес-процесів, будівництво будівлі, надання допомоги після стихійного лиха, розширення продажів на новий географічний ринок - усе це приклади проектів.

1.2 Мікросервісна архітектура.

Мікросервіси стають все більш популярними за останні кілька років, оскільки організації використовують DevOps і процеси безперервного тестування, щоб стати більш гнучкими. Провідні онлайн-компанії, такі як Amazon, eBay, Netflix, PayPal, Twitter і Uber, відмовилися від монолітних архітектур і перейшли до мікросервісів.

Монолітна архітектура складається з програм, побудованих як великі автономні одиниці. Такі програми не можна легко змінити, оскільки вся система тісно взаємопов’язана. Навіть незначна зміна коду може вимагати створення та

розгортання абсолютно нової версії програмного забезпечення. Монолітні додатки також важко масштабувати, оскільки масштабування конкретної функції вимагатиме масштабування всієї програми.

Мікросервіс - це найпростіша одиниця, сервіс, який приймає вхідні запити для здійснення дії. Це може бути бекенд сервіс, який доступний цілодобово та без вихідних, або функція, яка викликається, коли відбувається подія. Простими словами, функція або набір функцій, доступних через певний API через мережу. Отже, це бекенд служба, розгорнута на сервері. У якомусь сенсі це монолітний додаток. Однак він не несе в собі всю функціональність системи, а лише меншу частинку логіки. На відміну від моноліту, отриманий додаток побудований як набір відносно невеликих незалежних служб, що називаються «мікросервісами», які комунікують через комп'ютерну мережу. Можна сказати, мікросервіси – це ті самі логічні модулі монолітного додатка, які розподілені через комп'ютерну мережу, замість того, щоб працювати в рамках одного процесу (пристрою).

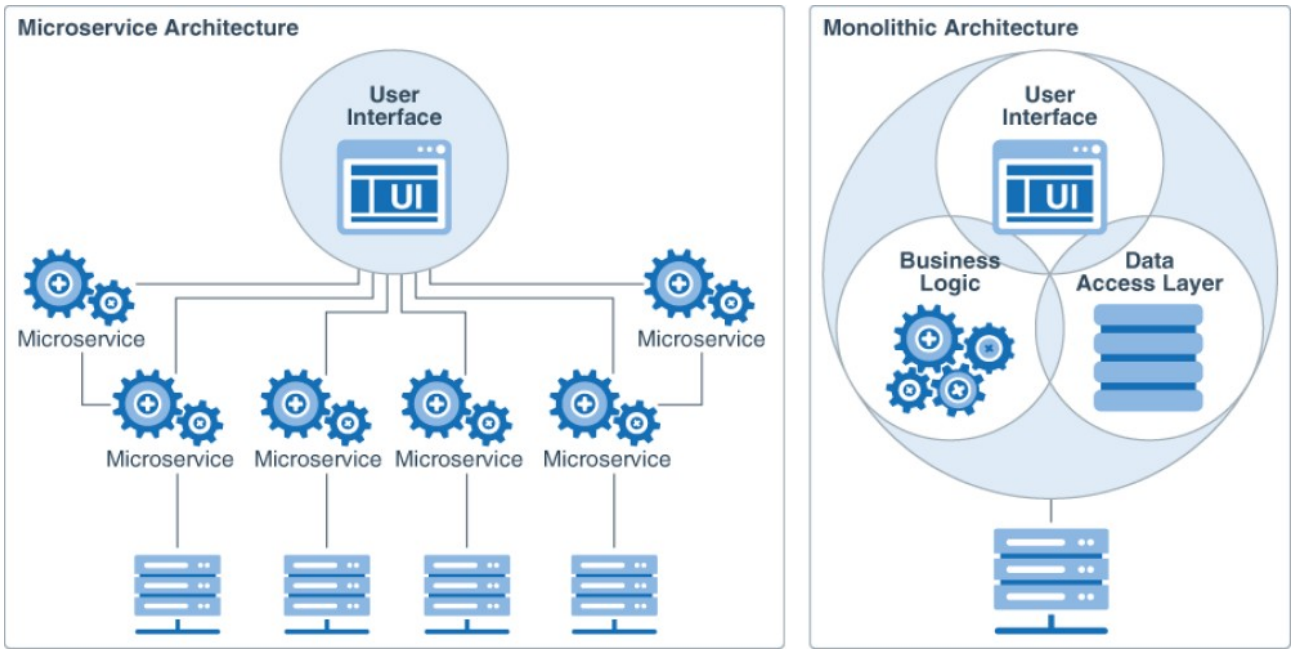


Рис 1.1 Мікросервісна та монолітна архітектура

Мікросервіси протистоять традиційній монолітній архітектурі. Монолітність означає, що складові продукту взаємопов'язані та взаємозалежні. Якщо один перестає працювати, всі інші також «відпадають».

Стрімкий розвиток IT-сфери висуває нові вимоги: розвиток технологій,

зміна потреб кінцевих користувачів – на все це потрібно швидко реагувати. Якщо в 2005 році розробляти продукт можна було кілька років, то зараз базову версію потрібно випустити за пару місяців.

У таких умовах архітектура мікросервісу перевершує моноліт:

Простіше змінити один із мікросервісів і негайно його впровадити, ніж змінити весь моноліт і перезапустити всю інфраструктуру;

Початківців розробників простіше залучити до роботи - для цього їм не потрібно вивчати всю систему, вони можуть працювати тільки на своїй частині;

Мікросервіси не залежать від жодної платформи, тому впроваджувати нові технології легше, ніж у моноліті.

Недоліки підходу.

Не все так ідеально. Мікросервіси накладають обмеження на розробку та підтримку продуктів:

Складність початкової розробки та створення інфраструктури. Розподілені системи складніше розробляти, оскільки необхідно передбачити незалежність одного мікросервісу від збою іншого компонента;

Розвиток розподілених систем накладає додаткові витрати на обмін даними між мікросервісами: потрібно правильно вибрати протоколи зв’язку між компонентами, щоб взаємодія була максимально ефективною;

Для розподіленої системи важко підтримувати сувору узгодженість: загальні частини системи потрібно або помістити в загальну бібліотеку, але потім, коли ця бібліотека зміниться, потрібно буде перезапустити всі залежні мікросервіси, або загальний код повинен бути перезапущений. зберігається в кожному з мікросервісів, але такий код суперечить принципу DRY (не повторюйтеся) і його важче підтримувати;

Інша структура команди розробників. Команда ділиться на групи, кожна з яких повністю розробляє один мікросервіс. Глава Amazon Джефф Безос пропонує оптимальний розмір команди: щоб їх можна було нагодувати двома піцами, тобто 7-9 осіб.

Виявляється, на ранніх етапах розробки результати, швидше за все, дають

монолітну конструкцію. З подальшим розвитком такої системи підтримувати її буде складніше. Мікросервіси вимагають від розробників великої уваги та майстерності. До випуску першого сервісу потрібно зробити багато: продумати інфраструктуру, зрозуміти протокол зв'язку, налаштувати конвеєри розгортання та змінити спосіб мислення в розробці програмного забезпечення. З появою першого готового сервісу новий функціонал з'явиться швидше, ніж з монолітною архітектурою.

Мікросервіси зазвичай структуровані ярусами. Не існує правил щодо того, скільки і яких ярусів має бути, однак, є найкращі практики. На діаграмі вище можна побачити найпоширеніші яруси, що використовуються в подібних архітектурах. Назви можуть відрізнятися, але структура дуже схожа на класичну багатоярусну архітектуру. Логічні яруси, які ми бачимо:

Front-End — Client Application, Static Content

Back-End — API Gateway, Experience Microservices, Domain Microservices

Data & Integration — Data Stores, Integration Interfaces / Connectors

Розглянемо кожен ярус і опишемо, за що він відповідає.

Логічні яруси.

Клієнтська програма.

У випадку веб-програми, клієнтом, як правило, є веб-браузер. Дехто вважає, що клієнт - це UI, але це не так. Браузер – це клієнт, який завантажує статичний контент із віддаленого сервера і рендерить UI всередині себе. І він же є клієнтом, що обробляє фізичні HTTP запити ща надходять до сервера. Важливо розрізняти клієнт та контент. Особливо, якщо ви розробляєте micro-app frontend.

Клієнт (веб-браузер) взаємодіє з Backend, щоб рендерити UI та викликати API.

Статичний контент.

Найкраща практика – розділити відповідальність шляхом роз'єднання логічних компонентів, щоб жоден компонент не відповідав за все. З цієї причини рекомендується подавати статичний контент через окремий серверний компонент. Це дозволяє відокремлювати статичний контент від API. Кілька важливих

принципів, на які потрібно звернути увагу, – це server-side rendering, кешування та контроль доступів.

API Gateway.

API Gateway – це центральні двері для всіх публічних API. Це єдина точка входу для так званих Experience APIs, доступних для клієнтських програм, і є важливою складовою найкращих практик управління API (API Management). По суті, він передає запити реальним бекенд сервісам та здатен приймати рішення. Частіше за все, API Gateway відповідає за такий функціонал, як:

1. Request handling
2. Upstream (або downstream) data transfer (або transformation)
3. Access control
4. Security policies
5. Throttling
6. Rate limiting
7. Analytics
8. Caching

Experience мікросервіси.

У неструктурованому розподіленому середовищі дуже легко втратити контроль над множиною мікросервісів. З цієї причини рекомендується контролювати мікросервіси, структуруючи їх за ярусами. Існують різні архітектурні шаблони, які можуть допомогти. Загальним для цих шаблонів є те, що всі вони зосереджені на experience.

Experience сервіси покладаються на доменні сервіси, що знаходяться на нижчому ярусі.

Уявіть набір мікросервісів, які надають API вищого рівня, орієнтовані на клієнтський досвід, повторно використовуючи низькорівневі гранулярні API, доступні в системі. Наприклад, різні продукти, що постачаються для веб- і мобільних клієнтів (Experience), можуть використовувати різний набір experience API, представлених незалежними мікросервісами за API шлюзом (API gateway).

Ці мікросервіси утворюють experience ярус, який логічно відокремлений

від інших ярусів. Він може бути керований та налаштований (масштабований, налаштований, захищений тощо) незалежно. Зверніться до архітектурних паттернів API-Led Connectivity та Backend For Frontend, щоб глибоко заглибитися у зв’язані концепції.

Доменні мікросервіси.

Доменні мікросервіси – це більш нижчий ярус, що відповідає за бізнес-логіку та від якого залежать experience сервіси.

Таким чином experience сервіси сфокусовані на користувачі та продукті, в той час, коли доменні мікросервіси відповідають за дані та ресурси, що знаходяться під їх управлінням. Тож, доменні мікросервіси постачають гранулярні API нижчого рівня, які дають можливість створювати різни experience сервіси, повторно використовуючи доступну доменну логіку.

В результаті маємо ситуацію, що додавання кожного нового experience сервісу не потребує додаткових витрат часу на повторну розробку доменної логіки.

Бази даних.

Best practice полягає в тому, що база даних повинна належати лише одному мікросервісу. Жодні інші сервіси не повинні мати можливості безпосереднього доступу до бази даних, лише через API сервіса-власника. Це дозволяє кожному сервісу керувати консистентністю та структурою бази даних, якою він володіє. Також це дозволяє обирати найбільш відповідну базу даних для конкретних цілей без будь-якої залежності від вимог системи в цілому. Наприклад, один сервіс може використовувати нормалізовану базу даних SQL, тоді як інший може отримати вигоду з бази даних NoSQL. Однак один мікросервіс може керувати кількома базами даних.

Сервіс може мати безпосередній доступ до бази даних, лише якщо він є власником. В іншому випадку він повинен використовувати API мікросервіса-власника і ніколи не мати безпосередній доступ до бази даних інших сервісів.

Спільне використання бази даних може потенційно призвести до анти-паттерна Distributed Monolith. Якщо вам потрібно надати загальний доступ до

даних, зверніться до таких концепцій централізованих сховищ, як :

DWH (Data Warehouse),

DFS (Distributed File System)

Data Lake.

Рекомендується застосовувати правильні схеми та підходи з самого початку, тому що потім все одно доведеться це зробити.

Інтеграційні інтерфейси та конектори.

Окрім внутрішніх джерел даних, програмі може знадобитися доступ до інших підсистем та сторонніх API. Корисна практика – розділяти інтеграційний ярус, запровадивши окремі сервіси-конектори.

Мікросервіси вирішують ці проблеми монолітної архітектури, використовуючи модульний підхід до розробки програмного забезпечення. Простіше кажучи, мікросервіси переосмислюють програми як комбінацію кількох окремих взаємопов'язаних служб. Кожна служба запускає спеціалізований процес і розгортається незалежно. За потреби служби можуть зберігати та обробляти дані за допомогою різних методів і можуть бути написані на інших мовах програмування.

Мікросервіси є одним із тих архітектурних шаблонів, які виникли у світі доменно-керованого проектування, безперервної доставки, платформи та автоматизації інфраструктури, масштабованих систем, поліглотного програмування та наполегливості.

Архітектура мікросервісів використовує той самий підхід і поширює його на слабко пов'язані служби, які можна розробляти, розгортати та підтримувати незалежно. Кожна з цих служб відповідає за окреме завдання та може спілкуватися з іншими службами через прості API для вирішення більшої складної бізнес-проблеми.

Ключові переваги архітектури мікросервісів.

Оскільки складові служби невеликі, вони можуть створюватися однією або декількома невеликими командами з самого початку, розділених межами сервісів, що полегшує масштабування зусиль щодо розробки, якщо це необхідно.

Після розробки ці служби також можна розгортати незалежно одна від одної, і, отже, їх легко визначити гарячі служби та масштабувати їх незалежно від цілої програми. Мікросервіси також пропонують покращену ізоляцію помилок, завдяки чому в разі помилки в одній службі не обов'язково припиняється функціонування всієї програми. Коли помилку виправлено, її можна буде розгорнути лише для відповідної служби замість повторного розгортання всієї програми.

Ще одна перевага, яку надає архітектура мікросервісів, полягає в тому, що полегшується вибір технологічного стеку (мови програмування, бази даних тощо), який найкраще підходить для необхідної функціональності (сервісу), замість того, щоб вимагати більш стандартизований, - універсальний підхід.

Характеристики архітектури мікросервісів

1. Розділити на численні компоненти.

Програмне забезпечення, створене з використанням архітектури мікросервісів, за визначенням розбивається на численні компонентні служби. Кожну службу можна створювати, розгортати й оновлювати незалежно без шкоди для цілісності програми. Усю програму можна збільшити, налаштувавши кілька конкретних служб, замість того, щоб знімати її та повторно розгортати.

2. Міцний і стійкий до поломок.

Додатку, створеному з використанням архітектури мікросервісів, нелегко вийти з ладу. Звичайно, окремі служби можуть виходити з ладу, що, безсумнівно, впливає на роботу. Зрештою, численні різноманітні та унікальні служби спілкуються одна з одною, щоб виконувати операції в середовищі мікросервісів, і збій обов'язково станеться в якийсь момент.

Однак у правильно налаштованому додатку на основі мікросервісів функція, яка зазнає простою, повинна мати можливість перенаправляти трафік від себе, дозволяючи підключеним службам продовжувати роботу. Також легко зменшити ризик збою, відстежуючи мікросервіси та якнайшвидше відновлюючи їх у разі збою.

3. Простий процес маршрутизації.

Мікросервіси складаються з інтелектуальних компонентів, здатних обробляти дані та застосовувати логіку. Ці компоненти з'єднані «тупими проводами», які передають інформацію від одного елемента до іншого.

Цей простий процес маршрутизації протилежний архітектурі, яка використовується в деяких інших корпоративних програмах. Наприклад, службова шина підприємства використовує складні системи для маршрутизації повідомлень, хореографії та застосування бізнес-правил. Однак мікросервіси просто отримують запити, обробляють їх і виробляють відповідний вихід для передачі запитуючому компоненту.

4. Децентралізовані операції.

Мікросервіси використовують численні платформи та технології. Це робить традиційні методи централізованого управління неефективними для роботи з архітектурою мікросервісів.

Децентралізоване управління краще підходить для мікросервісів, оскільки розробники по всьому світу створюють цінні інструменти для вирішення операційних проблем. Ці інструменти можуть навіть використовувати інші розробники, які стикаються з такими ж проблемами.

Подібним чином архітектура мікросервісів надає перевагу децентралізованому управлінню даними, оскільки кожна програма мікросервісу керує своєю унікальною базою даних. І навпаки, монолітні системи зазвичай працюють з використанням централізованої логічної бази даних для всіх програм.

5. Створено для сучасного бізнесу.

Архітектура мікросервісів створена, щоб зосередитися на виконанні вимог сучасного цифрового бізнесу. У традиційних монолітних архітектурах команди працюють над розробкою таких функцій, як інтерфейс користувача, технологічні рівні, бази даних і серверна логіка. Мікросервіси, з іншого боку, покладаються на багатофункціональні команди. Кожна команда бере на себе відповідальність за створення конкретних продуктів на основі окремих сервісів, які передають і приймають дані через шину повідомлень.

Визначення можливостей бізнесу та відповідних послуг вимагає високого

рівня розуміння бізнесу. Наприклад, бізнес-можливості програми для онлайн-покупок можуть включати наступне.

- Управління каталогом продукції
- Управління запасами
- Управління замовленнями
- Управління доставкою
- Керування користувачами
- Рекомендації щодо продуктів
- Управління оглядами продуктів

Після визначення бізнес-можливостей можна створювати необхідні послуги відповідно до кожної з цих ідентифікованих бізнес-можливостей.

Кожна служба може належати іншій команді, яка стає експертом у цій конкретній області та експертом у технологіях, які найкраще підходять для цих конкретних послуг. Це часто призводить до більш стабільних кордонів API та більш стабільних команд.

Внесення змін до існуючих API мікросервісів під час виробництва.

Ще одна поширена проблема, з якою зазвичай стикаються моделі мікросервісів, полягає в тому, щоб визначити, як внести зміни в існуючі API мікросервісів, коли інші використовують їх у виробництві. Внесення змін до API мікросервісу може порушити роботу мікросервісу, який залежить від нього.

Існують різні шляхи вирішення цієї проблеми.

Спочатку встановіть версію свого API, а коли потрібні зміни для API, розгорніть нову версію API, зберігаючи першу версію. Потім залежні служби можна оновити у власному темпі для використання новішої версії. Після того, як усі залежні служби перенесено на використання нової версії зміненого мікросервісу, його можна припинити.

Однією з проблем цього підходу є те, що стає важко підтримувати різні версії. Будь-які нові зміни або виправлення помилок необхідно вносити в обидві версії.

З цієї причини можна розглянути альтернативний підхід, у якому інша

кінцева точка реалізується в тій же службі, коли потрібні зміни. Коли нова кінцева точка буде повністю використана всіма службами, стару кінцеву точку можна буде видалити.

Очевидна перевага цього підходу полягає в тому, що сервіс легше підтримувати, оскільки завжди буде працювати лише одна версія API.

1.3 Створення та розгортання мікросервісу.

Після визначення меж обслуговування цих невеликих мікросер вони можуть бути розроблені однією або декількома невеликими командами з використанням технологій, які найкраще підходять для кожної мети. Наприклад, можна створити службу користувача на Java з базою даних MySQL і службу рекомендацій продуктів за допомогою Scala/Spark.

Після розробки конвеєри CI/CD можна налаштувати з будь-яким із доступних серверів CI (Jenkins, TeamCity, Go тощо) для запуску автоматизованих тестів і незалежного розгортання цих служб у різних середовищах (інтеграція, контроль якості, постановка, виробництво). тощо).

CI/CD — це метод частої доставки додатків клієнтам шляхом впровадження автоматизації на етапах розробки додатків . Основні концепції CI/CD — безперервна інтеграція, безперервна доставка та безперервне розгортання. CI/CD — це рішення проблем, які інтеграція нового коду може спричинити для команд розробки та операцій (відомих також як «інтеграційне пекло»).

Зокрема, CI/CD запроваджує постійну автоматизацію та постійний моніторинг протягом життєвого циклу додатків , від етапів інтеграції та тестування до доставки та розгортання . У сукупності ці підключені практики часто називають « конвеєром CI/CD » і підтримуються командами розробки та операцій, які працюють разом у гнучкий спосіб за допомогою підходу DevOps або розробки надійності сайту (SRE).

Безперервна інтеграція.

У сучасній розробці додатків мета полягає в тому, щоб кілька розробників одночасно працювали над різними функціями однієї програми. Однак, якщо організація налаштована на об’єднання всього розгалуженого вихідного коду в один день (відомий як «день злиття»), результат може бути стомлюючою, ручною та трудомісткою роботою. Це тому, що коли розробник, який працює ізольовано, вносить зміни в програму, існує ймовірність, що вони конфліктуватимуть з різними змінами, які одночасно вносяться іншими розробниками. Ця проблема може посилитися, якщо кожен розробник налаштував власне локальне інтегроване середовище розробки (IDE) , а не команда домовилася про одну хмарну IDE.

Безперервна інтеграція (CI) допомагає розробникам частіше — іноді навіть щодня — об’єднувати зміни коду назад у спільну гілку або «ствол». Коли зміни, внесені розробником у програму, об’єднані, ці зміни перевіряються шляхом автоматичного створення програми та виконання різних рівнів автоматизованого тестування, як правило, модульних і інтеграційних тестів, щоб переконатися, що зміни не порушили роботу програми. Це означає тестування всього, від класів і функцій до різних модулів, які складають цілу програму. Якщо автоматичне тестування виявляє конфлікт між новим і існуючим кодом, CI полегшує швидке та часте виправлення цих помилок.

Постійна доставка.

Після автоматизації збірок і тестування модулів та інтеграції в CI безперервна доставка автоматизує випуск перевіреного коду в репозиторій. Отже, щоб мати ефективний безперервний процес доставки, важливо, щоб CI вже був вбудований у ваш конвеєр розробки. Метою безперервної доставки є мати кодову базу, яка завжди готова до розгортання у виробничому середовищі.

У безперервній доставці кожен етап — від злиття змін коду до доставки готових збірок — передбачає автоматизацію тестування та автоматизацію випуску коду. Наприкінці цього процесу операційна команда може швидко та легко розгорнути робочу програму.

Безперервне розгортання.

Останнім етапом зрілого конвеєра CI/CD є безперервне розгортання. Як розширення безперервної доставки, яка автоматизує випуск готової для виробництва збірки в сховищі коду, безперервне розгортання автоматизує випуск програми для виробництва. Оскільки на стадії конвеєра перед виробництвом немає ручних воріт, безперервне розгортання значною мірою залежить від добре розробленої автоматизації тестування.

На практиці безперервне розгортання означає, що зміни, внесені розробником у хмарну програму, можуть запрацювати протягом кількох хвилин після її написання (за умови, що вона пройшла автоматизоване тестування). Це значно полегшує постійне отримання та врахування відгуків користувачів. У сукупності всі ці пов’язані практики CI/CD роблять розгортання програми менш ризикованим, завдяки чому легше випускати зміни до програм невеликими частинами, а не всі відразу. Однак також є багато початкових інвестицій, оскільки автоматизовані тести потрібно буде написати, щоб врахувати різні етапи тестування та випуску в конвеєрі CI/CD.

Які поширені інструменти CI/CD?

Інструменти CI/CD можуть допомогти команді автоматизувати розробку, розгортання та тестування. Деякі інструменти спеціально обробляють інтеграцію (CI), деякі керують розробкою та розгортанням (CD), тоді як інші спеціалізуються на постійному тестуванні або пов’язаних функціях.

Одним із найвідоміших інструментів з відкритим кодом для CI/CD є сервер автоматизації Jenkins. Jenkins розроблено для роботи з чим завгодно, від простого CI-сервера до повноцінного концентратора CD.

Розробляючи служби, ретельно визначте їх і подумайте про те, що буде відкрито, які протоколи використовуватимуться для взаємодії зі службою тощо.

Дуже важливо приховати будь-яку складність і деталі реалізації послуги та відкрити лише те, що потрібно клієнтам послуги. Якщо розкриваються непотрібні деталі, пізніше буде дуже складно змінити послугу, оскільки буде потрібно багато копіткої роботи, щоб визначити, хто покладається на різні частини служби. Крім

того, при самостійному розгортанні служби втрачається значна гнучкість.

Для вирішення цієх проблему на схемі зображено взаємодія між мікросервісами та базою даних.

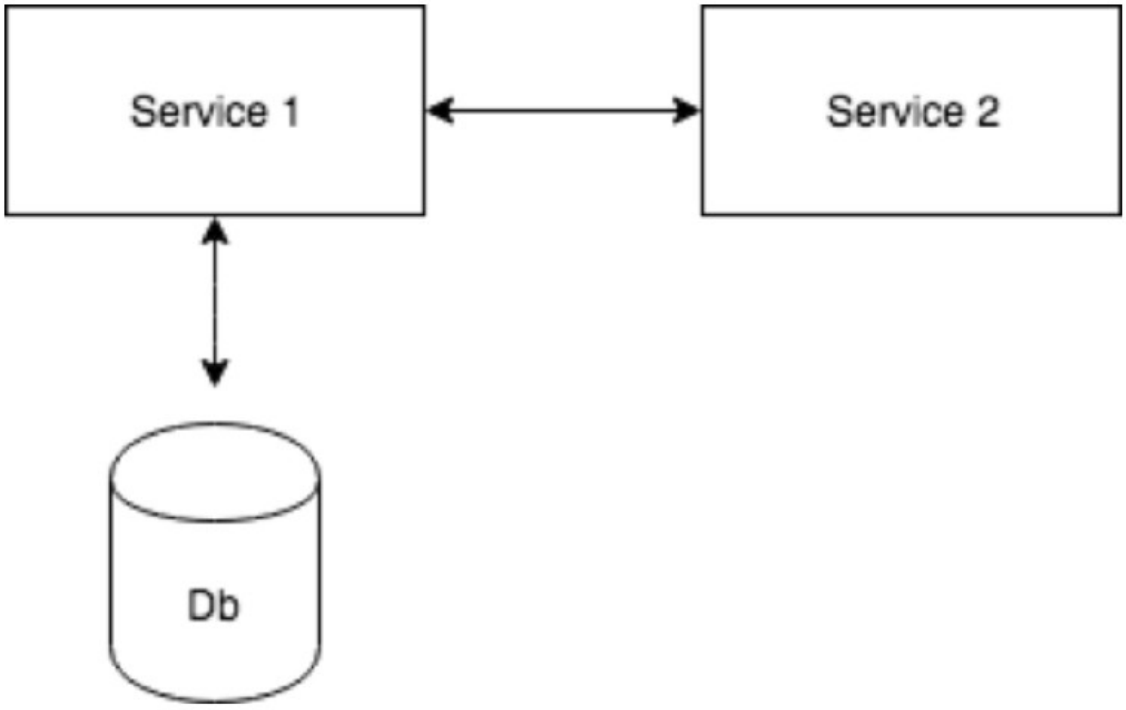


Рис 1.2 Схема взаємодії мікросервісу

Служба 2 повинна мати доступ до служби 1 і уникати переходу безпосередньо до бази даних, таким чином зберігаючи максимальну гнучкість для різноманітних змін схеми, які можуть знадобитися. Не варто хвилюватися про інші частини системи, якщо ви переконаєтеся, що тести відкритих API проходять успішно.

Є організації, які досягли успіху з мікросервісами та наслідували модель, за якою команди, які створюють сервіси, піклуються про все, що пов’язано з цим сервісом. Саме вони розробляють, розгортають, обслуговують і підтримують його. Немає окремих груп підтримки чи обслуговування.

Ще один спосіб досягти того ж - мати внутрішню модель з відкритим кодом. Застосовуючи такий підхід, розробник, якому потрібні зміни в сервісі, може перевірити код, попрацювати над функцією та надіслати PR сам, замість того, щоб чекати, поки власник сервісу візьме та попрацює над необхідними

змінами.

Щоб ця модель працювала належним чином, потрібна відповідна технічна документація, а також інструкції з налаштування та вказівки для кожної служби, щоб будь-кому було легко підібрати послугу та працювати з нею.

Ще одна прихована перевага цього підходу полягає в тому, що він зосереджує розробників на написанні високоякісного коду, оскільки вони знають, що на нього дивитимуться інші.

Для будь-якого API, від якого залежить, важливо писати контракти, керовані споживачами. Це зроблено для того, щоб нові зміни в цьому API не порушили ваш API.

У Consumer Driven Contracts кожен API споживача фіксує їхні очікування щодо постачальника в окремому контракті. Усі ці контракти передаються постачальнику, щоб вони могли зрозуміти зобов'язання, які вони повинні виконувати для кожного окремого клієнта.

Перед розгортанням і перед внесенням будь-яких змін до API необхідно повністю пройти контракти, керовані споживачами. Це також допомагає постачальнику знати, які послуги залежать від нього та як інші послуги залежать від нього.

Коли справа доходить до розгортання незалежних мікросервісів, існує дві поширені моделі.

Кілька мікросервісів для кожної операційної системи

По- перше, можна розгорнути кілька мікросервісів на операційну систему . За допомогою цієї моделі час економиться на автоматизації певних речей, наприклад, хост для кожної служби не потрібно надавати.

Недоліком цього підходу є те, що він обмежує можливість самостійно змінювати та масштабувати послуги. Це також створює труднощі в управлінні залежностями. Наприклад, усі служби на одному хості повинні використовувати однакову версію Java, якщо вони написані на Java. Крім того, ці незалежні служби можуть викликати небажані побічні ефекти для інших запущених служб, які можуть бути дуже важкою проблемою для відтворення та вирішення.

Один мікросервіс на операційну систему.

Через зазначену вище проблему друга модель, де розгортається один мікросервіс на операційну систему , є кращим вибором.

За допомогою цієї моделі служба більш ізольована, а отже, легше самостійно керувати залежностями та масштабувати служби. Але ви можете запитати себе: «Хіба це не дорого»? Ну не дуже.

Традиційним рішенням для вирішення цієї проблеми є використання гіпервізорів, за допомогою яких кілька віртуальних машин розміщуються на одному хості. Такий підхід до рішення може бути неефективним, оскільки сам процес гіпервізора споживає певні ресурси, і, звісно, чим більше віртуальних машин надано, тим більше ресурсів споживатиметься. І саме тут модель контейнера отримує хорошу тягу та є кращою. Docker є однією з реалізацій цієї моделі.

Внесення змін до існуючих API мікросервісів під час виробництва.

Ще одна поширена проблема, з якою зазвичай стикаються моделі мікросервісів, полягає в тому, щоб визначити, як внести зміни в існуючі API мікросервісів, коли інші використовують їх у виробництві. Внесення змін до API мікросервісу може порушити роботу мікросервісу, який залежить від нього.

Існують різні шляхи вирішення цієї проблеми.

Спочатку встановіть версію свого API, а коли потрібні зміни для API, розгорніть нову версію API, зберігаючи першу версію. Потім залежні служби можна оновити у власному темпі для використання новішої версії. Після того, як усі залежні служби перенесено на використання нової версії зміненого мікросервісу, його можна припинити.

Однією з проблем цього підходу є те, що стає важко підтримувати різні версії. Будь-які нові зміни або виправлення помилок необхідно вносити в обидві версії.

З цієї причини можна розглянути альтернативний підхід, у якому інша кінцева точка реалізується в тій же службі, коли потрібні зміни. Коли нова кінцева точка буде повністю використана всіма службами, стару кінцеву точку

можна буде видалити.

Очевидна перевага цього підходу полягає в тому, що сервіс легше підтримувати, оскільки завжди буде працювати лише одна версія API.

Важливо розуміти, що мікросервіси не є стійкими за замовчуванням. Будуть збої в службах. Збої можуть статися через збої в залежних службах. Крім того, збої можуть виникати з різних причин, таких як помилки в коді, тайм-аути мережі тощо.

Для архітектури мікросервісів критично важливо переконатися, що вся система не зазнає впливу або вийде з ладу, якщо в окремій частині системи є помилки.

Існують шаблони, такі як перегородка та вимикач, які можуть допомогти вам досягти кращої стійкості.

Перегородка.

Патерн Bulkhead ізолює елементи програми в пули, щоб у разі відмови одного з них інші продовжували працювати. Візерунок назвали перегородкою, тому що він нагадує розділені перегородки корпусу корабля. Якщо корпус корабля пошкоджений, тільки пошкоджена частина заповнюється водою, що запобігає затопленню судна.

Автоматичний вимикач.

Шаблон Circuit Breaker обертає виклик захищеної функції в об’єкт автоматичного вимикача, який відстежує збої. Коли відмова перетинає порогове значення, автоматичний вимикач спрацьовує, і всі подальші виклики автоматичного вимикача повертаються з помилкою, захищений виклик взагалі не виконується протягом певного налаштованого часу очікування.

Після закінчення часу очікування автоматичний вимикач дозволяє проходити деякі виклики, і якщо вони вдаються, автоматичний вимикач повертається до нормального стану. Протягом періоду виходу з ладу автоматичного вимикача користувачі можуть отримувати сповіщення про те, що певна частина системи зламана, а решту системи все ще можна використовувати.

Моніторинг і журналювання.

Мікросервіси є розподіленими за своєю природою, тому моніторинг і журналювання окремих сервісів може бути проблемою. Важко переглядати та співставляти журнали кожного екземпляра служби та з’ясовувати окремі помилки. Як і у монолітних програмах, немає єдиного місця для моніторингу мікросервісів.

Агрегація журналів.

Щоб вирішити такі проблеми, бажаним підходом є використання централізованої служби журналювання, яка збирає журнали з кожного екземпляра служби. Користувачі можуть шукати ці журнали з одного централізованого місця та налаштовувати сповіщення, коли з’являються певні повідомлення.

Стандартні інструменти доступні та широко використовуються різними підприємствами. ELK Stack є найбільш часто використовуваним рішенням, де демон журналювання, Logstash , збирає та об’єднує журнали, які можна шукати через інформаційну панель Kibana, індексовану Elasticsearch.

Агрегація статистики.

Подібно до агрегації журналів, статистичні дані, такі як використання процесора та пам’яті, також можна використовувати та зберігати централізовано. Такі інструменти, як Graphite, чудово справляються з надсиланням до центрального сховища та ефективним збереженням.

Коли один із нижчестоящих сервісів не в змозі обробляти запити, має бути спосіб ініціювати сповіщення, і саме тут впровадження API перевірки працездатності в кожному сервісі стає важливим — вони повертають інформацію про працездатність системи.

Клієнт перевірки працездатності, який може бути службою моніторингу або балансувальником навантаження, викликає кінцеву точку для періодичної перевірки працездатності екземпляра служби через певний проміжок часу. Навіть якщо всі низхідні служби справні, все одно може виникнути проблема зі зв’язком між службами. Такі інструменти, як проект Netflix Hystrix , дають змогу ідентифікувати такі типи проблем.

Найпростішим способом комунікації між мікросервісами є синхронні

HTTP запити. При цьому один сервіс просто викликає інший сервіс, часто використовуючи REST API. Перший сервіс викликає другий, чекає поки той обробить запит, отримує відповідь та на основі цієї відповіді виконує подальшу логіку застосунку.

Рис 1.3 Комунікація мікросервісів

Даний спосіб спілкування легко імплементувати, бо він є досить сталим, історично показав свою надійність, та більшість розробників з ним знайомі. Але разом з тим, у нього є ряд недоліків.

По-перше, можливі перебої в роботі мережі, результатом яких будуть тайм-аути, коли перший сервіс не може дочекатися відповіді від другого. У такому випадку перший сервіс виконає логіку щодо повернення своїх даних у початковий стан. Втім, другий сервіс все-таки може в цей час оброблювати запит та зробити зміни на своїй стороні, що у висновку призведе до неконсистентного стану всієї системи.

По-друге, сервіси стають сильно зв’язними. Перший сервіс не може працювати, якщо другий не працює. Звісно, можна застосувати механізм “Retry”, коли сервіс повторює запит декілька разів, проте це вочевидь призведе до затримок роботи системи. Більше того, сервіси мають обмінюватись певним агрегатом даних, який відноситься тільки до одного з цих сервісів. Тобто перший сервіс знає більше, ніж йому потрібно за принципом єдиної відповідальності.

Тож даний спосіб комунікації має певні недоліки, які неодмінно з’являться в ході розробки застосунку. Але він найкраще підходить для випадків, коли сервіси мають синхронно спілкуватись у форматі запит-відповідь для подальшої обробки складного запиту від користувача системи.

ВИСНОВОК ДО РОЗДІЛУ 1.

Перш ніж починати розробку та ствоерння проекту необхідно було ознайомитись з теоретичною частиною управління проектом, розглянуто ролі та обов’язки керівника проекту, зокрема:

1. Впровадження методології управління проектами
2. Планування та визначення обсягу
3. Встановлення та управління очікуваннями
4. Процес крафта
5. Створення проектних планів
6. Управління завданнями
7. Розподіл і планування ресурсів
8. Оцінка часу/вартості
9. Аналіз та управління ризиками та проблемами
10. Моніторинг та звітність про стан проекту
11. Забезпечення керівництва командою
12. Стратегія впливу
13. Полегшення спілкування та співпраці
14. Планування та проведення зустрічей

У даному розділі проводено аналіз та характеристику мікросервісної архитектури, плюси та мінуси цього підходу.

Проаналізовано найпопулярніші та зручні засоби для розробки мікросервісів, їх розгортання та розробки.

Проведено порівняння монолітної та мікросервісної архітектури, де були визначені наступні переваги другої, мікросервіси вирішують проблеми монолітної

архітектури, використовуючи модульний підхід до розробки програмного забезпечення. Простіше кажучи, мікросервіси переосмислюють програми як комбінацію кількох окремих взаємопов’язаних служб. Кожна служба запускає спеціалізований процес і розгортається незалежно. За потреби служби можуть зберігати та обробляти дані за допомогою різних методів і можуть бути написані на інших мовах програмування.

Описана комунікація між мікросервісами за допомогою REST API.

Після проведення аналізу було визначено методи та засоби для розробки, розгортання, журнулювання та підтримки мікросервісної архітектури та в цілому - розробки проєкту.

2. РОЗРОБКА СЕРВІСУ ДЛЯ ОТРИМАННЯ ДАНИХ ПО КЛІЄНТУ

2.1 Аналіз ринку схожих програмних продуктів

Сьогодні компанії використовують багато різних способів пошуку клієнтів: від холодних дзвінків через довідники, розміщення оголошень і роздачі листівок до масштабних акцій і рекламних кампаній в Інтернеті.

Після виконаної роботи компанія починає отримувати перші відгуки та запити від потенційних клієнтів, які можуть принести прибуток у короткостроковій перспективі. Потім виникає необхідність створити клієнтську базу і вибрати програму або сервіс для управління клієнтами.

Як показує практика, кожна компанія, яка працює зі 100 клієнтами, вже потребує створення клієнтської бази. Якщо кількість клієнтів більше тисячі, такий інструмент потрібен терміново. Тут допоможуть сервіси для підтримки клієнтської бази.

Що таке клієнтська база - простий лід-лист чи потужний інструмент для розвитку компанії та спосіб збільшення продажів? Звичайно, другий! Але ефективність такого засобу безпосередньо залежить від того, чи вмієте ви з ним працювати. Збір номерів клієнтів навіть з додатковою інформацією – це дорога в нікуди, якщо немає подальшого розвитку. Просто витрачайте дорогоцінний час і ресурси. Як працювати з базою даних, що робити, якщо її ще немає, які існують сервіси і як аналізувати дані, ми розповімо в статті.

Що таке клієнтська база

Клієнтська база - це впорядкована система даних з контактною інформацією. Так, з одного боку, це можна назвати звичайним списком, який ні на що не впливає. Але не все так просто. Клієнтська база - це хроніка подій і детальна історія взаємовідносин з клієнтами. Допомагає цілеспрямована робота з кожним контактом: ви точно знаєте, кому і в який момент зробити вигідну пропозицію, яка призведе до продажу.

Мова йде навіть не про повторні покупки постійних клієнтів (а це, як правило, є основним джерелом прибутку компаній), а про загальний системний індивідуальний підхід. Що це означає? Уявімо таку ситуацію: телефонують у відділ продажів забудовника. Жінка хоче купити квартиру, але під час розмови з'ясовується, що вона розлучається з чоловіком, офіційно не працює, а грошей на перший внесок по іпотеці у неї немає. Що думає менеджер? Наприклад, він зробить висновок, що продажів від цього дзвінка не буде і про нього можна забути. Або він може внести всі дані в базу даних, а через деякий час знову зв'язатися з потенційним клієнтом і дізнатися, чи змінилися умови проживання і поговорити про нові пільгові іпотечні програми. Таке нагадування зіграє на руку компанії, навіть якщо не призведе відразу до продажу.

Чому клієнтська база така важлива?

Клієнтська база та грамотна робота з нею забезпечують конкурентні переваги компанії:

Якість зібраної інформації впливає на якість обслуговування. Ви, наприклад, знаєте, коли у клієнта день народження, чи є у нього діти тощо, що означає, що ви можете відстежувати запити та створювати персоналізовані пропозиції, як-от святковий промокод або знижка до дня захисту дітей.

Задоволені клієнти можуть рекомендувати ваші послуги своїм знайомим і тим самим збільшувати базу.

На основі бази даних можна сегментувати клієнтів: наприклад, розділити їх на звичайних і преміальних.

Будь-які маркетингові кампанії легше організувати, якщо знати, які канали комунікації віддають перевагу різні групи клієнтів (дзвінки, електронні листи, смс-повідомлення).

Завдяки клієнтській базі ви можете відстежити шлях через воронку продажів і залучити до транзакції того, хто поділився з вами контактною інформацією, або нагадати собі про давно забутих клієнтів. Як відомо, набагато вигідніше поповнити базу контактами постійних покупців, а не кидати всі ресурси на залучення нових «одноразових» клієнтів.

База даних містить різноманітну інформацію про споживачів, а значить, можна робити прогнози та оцінювати перспективи розвитку.

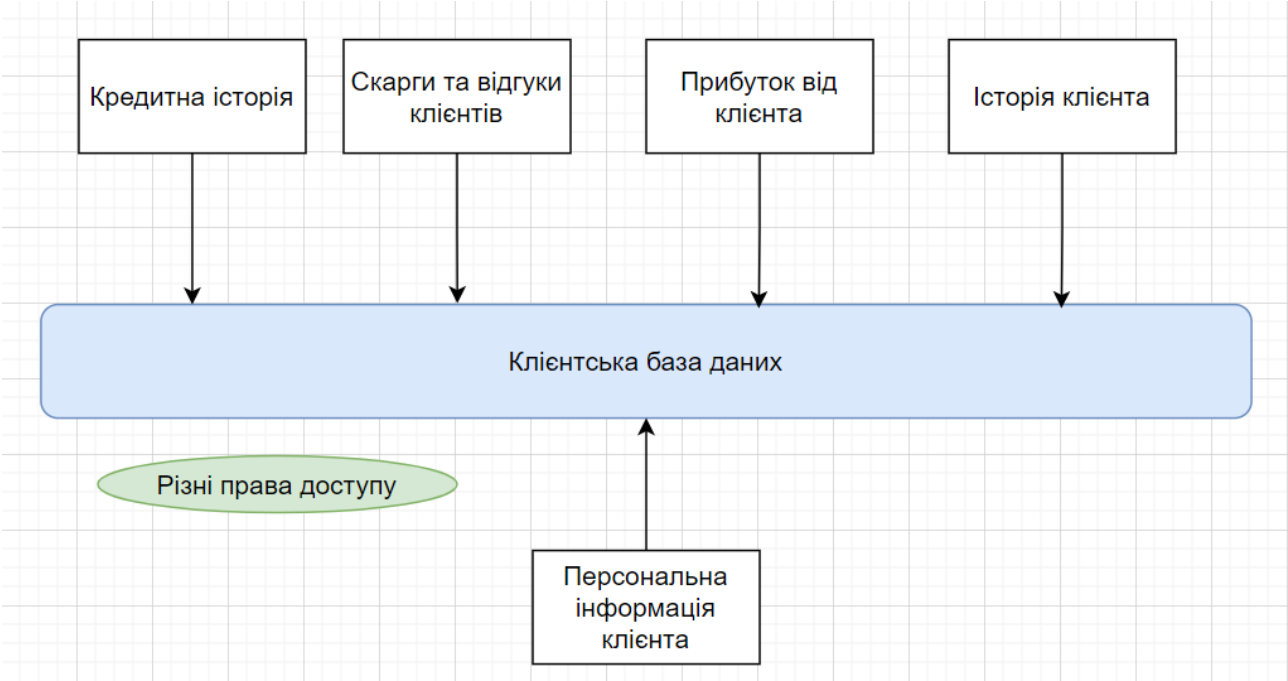


Рис 2.1 Клієнтська база даних

Які бувають клієнтські бази:

Залежно від того, як компанія веде клієнтську базу, накопичує та аналізує інформацію, розрізняють кілька типів баз даних.

База «мертвих» клієнтів - є клієнти, які не користуються послугами компанії, але залишаються зареєстрованими. Співробітники вже витратили час і ресурси на залучення ліда, а просто видаляти контакти нераціонально: цілком можливо, що «мертві» клієнти звернуться до вас знову в майбутньому.

Потенційна клієнтська база - це контакти, які ще не стали покупцями, але з ними можна і потрібно працювати.

Активна клієнтська база - складається з постійних покупців, перевірена часом. Тобто новачків, хоч і досить активних, сюди відносити не варто.

Працююча клієнтська база − Тут клієнти, які зробили одну покупку. Далі важливо продовжувати роботу і закріплювати перший отриманий результат.

База постійних клієнтів - це, як правило, ті, хто вже купував більше трьох разів, а значить, потрібно відстежувати відгуки, помічати закономірності при

здійсненні покупки.

База даних перевірених часом партнерів - містить дані про організацію або осіб, з якими встановлені надійні міцні відносини.

VIP-партнери - це особливі клієнти, які розподіляються за певними критеріями на розсуд компанії.

Ключові клієнти - ця категорія включає попередню групу та перевірених клієнтів, які працюють у компанії протягом тривалого часу.

Послуги з ведення клієнтської бази.

Важко уявити ефективну роботу з базою даних без автоматизації бізнес-процесів. Для цього зручно використовувати спеціальні програми або сервіси. Поговоримо про найпопулярніші.

Excel - дійсно універсальна програма, в якій можливо практично все! Найпростіший варіант – створити базу даних у вигляді таблиці, яку легко адаптувати під будь-які потреби бізнесу та вказати необхідні деталі. Які переваги? Це безкоштовно, багато формул, які зручно рахувати, можливість створити шаблон. Які недоліки: незручно працювати в команді - немає багатокористувацького доступу, дані легко скопіювати або втратити, важко з аналітикою.

Access — це програмне забезпечення від Microsoft, яке начебто схоже за функціями на Excel, але насправді воно складніше та надійніше. Тут дані більш захищені: кожен менеджер має доступ тільки до своєї таблиці; залежно від типу в базах даних можуть працювати один або декілька користувачів. Однак у програми є свої суттєві недоліки: вона більш вимоглива до операційної системи, і при збоях в локальній мережі є ймовірність появи помилок.

Google Sheets – Google Sheets добре підходить для малого бізнесу: безкоштовний, зручний, без додаткових функцій, які спочатку не потрібні, послуга безкоштовна, обмін дозволяє працювати всій команді та бачити зміни від кожного користувача. Ще один приємний варіант: ви можете налаштувати розсилку електронної пошти клієнтам за допомогою спеціального плагіна. Які недоліки: важко автоматизувати деякі процеси, відсутність розширених

можливостей для великого бізнесу, незручний інтерфейс для мобільних пристроїв і планшетів.

CRM-системи (Customer Relationship Management) — це спеціально розроблені сервіси для автоматизації процесів, контролю продажів і аналітики. Це найбільш зручний, надійний і комфортний спосіб ведення бази даних для великих компаній. У CRM є те, чого немає в інших сервісах: підключення до соціальних мереж, сайт, налаштування пошти, телефонії, створення автоскриптів, управління завданнями та призначення відповідальних осіб, відображення проміжних етапів у завданнях, сегментація клієнтів.

Всі рутинні дії автоматизовані, є аналітика, вся історія взаємодії з клієнтом відображена в хронологічному порядку, не потрібно запам'ятовувати кому, коли і з якої причини потрібно дзвонити - програма сама нагадає, а також самостійно реалізує деякі функції (наприклад, надішле повідомлення в соцмережах, обробить заявку з сайту і так далі). Однією з головних переваг є візуальна аналітика, яка дозволить оцінити діяльність менеджера, всього відділу продажів, рекламну кампанію.

Що таке сегментація і для чого вона потрібна при роботі з базою даних

Сегментація - це поділ покупців на різні категорії за певними критеріями, наприклад, за обсягом покупок, географічним розташуванням, уподобаннями щодо товарів тощо. Очевидно, що за допомогою такого інструменту працювати з базою набагато зручніше та ефективніше: наприклад, пропонувати спеціальні послуги для клієнтів з дітьми або створювати знижки залежно від платоспроможності клієнта.

Сегментація таким чином також економить витрати на маркетинг: персоналізовані пропозиції для кожної групи матимуть більший ефект, ніж загальна рекламна розсилка по всій базі.

Як зберегти клієнтську базу

Щоб було зручніше працювати з базою даних, потрібно продумати структуру, вибудувати інформацію в правильному порядку. У довідковій інформації необхідно вказати контактну інформацію - організацію, графік роботи,

номер телефону або будь-який засіб зв'язку, імена представників, їх посади, зручний час для дзвінка і так далі.

Потім ви можете розділити клієнтів на категорії, наприклад, щоб вказати лояльність або ступінь інтересу до пропозиції за допомогою кольорів. Це робить відстеження воронки продажів для кожного контакту простішим і зрозумілішим. Ще один варіант створення категорій — географія: регіональні клієнти, із сусідніх країн тощо. Інша класифікація передбачає поділ контактів за родом діяльності, що дозволяє відстежувати підвищений інтерес до будь-якої сфери діяльності.

У розділі, присвяченому бізнес-процесам, можна відобразити етапи транзакції - який статус замовлення, де воно зараз знаходиться, які товари найчастіше замовляє клієнт.

Що ще важливо:

визначити оптимальний час для спілкування з клієнтом (головне не перестаратися і не втомитися) і відобразити його в базі даних;

суворо стежити за тим, щоб інші менеджери не спілкувалися з одним клієнтом;

Максимально уточнюйте всю інформацію та вчасно вносьте зміни.

Як аналізувати дані.

База клієнтів дає чудовий матеріал для аналізу ефективності діяльності компанії, маркетингової стратегії та роботи відділу продажів. Але перед цим потрібно визначити свої цілі. Що тобі потрібно? Визначити найбільш вигідних клієнтів, дізнатися частоту їх покупок або відстежити реакцію на маркетингову пропозицію? Залежно від мети вибирають різні способи. Поговоримо про два популярних види аналізу.

RFM аналіз (Recency Frequency Monetary) - цей тип аналізу підходить для сегментації та прогнозування дій клієнтів у майбутньому. Як зрозуміло з назви, оцінюються три показники: новизна (Recency), частота покупок (Frequency), дохід (Monetary). Аналіз RFM базується на переконанні, що клієнт, який витратив більше грошей на продукт чи послугу або зробив покупку нещодавно, більш схильний до взаємодії з компанією.

ABC-аналіз — це найпоширеніший тип аналітики, який ранжує клієнтів за важливістю. Виходячи з принципу Парето (коли 20% зусиль дають 80% результату), компанія ділить клієнтську базу на найцінніших (A), проміжних (B), що представляють найменшу цінність.

2.2 SWOT аналіз проєкту

SWOT розшифровується як: S Strength, W eakness, O pportunity, T Threat. SWOT-аналіз допоможе вам визначити сильні та слабкі сторони (SW) вашої організації, а також ширші можливості та загрози (OT). Розвиток більш повного усвідомлення ситуації допомагає як у стратегічному плануванні, так і в прийнятті рішень.

Метод SWOT спочатку був розроблений для бізнесу та промисловості, але він однаково корисний у роботі з охорони здоров'я та розвитку громади, освіти та навіть для особистого зростання.

SWOT - не єдиний метод оцінювання, який ви можете використовувати. Порівняйте його з іншими інструментами оцінювання в інструментарії спільноти, щоб визначити, чи це правильний підхід для вашої ситуації. Сильними сторонами цього методу є його простота та застосування на різних рівнях роботи.

КОЛИ ВИ ВИКОРИСТОВУЄТЕ SWOT?

SWOT-аналіз може запропонувати корисні перспективи на будь-якому етапі роботи.

Ви можете використовувати SWOT, щоб:

1. Дослідити можливості для нових зусиль або вирішення проблем.
2. Приймайте рішення про найкращий шлях для вашої ініціативи. Визначення ваших можливостей для успіху в контексті загроз успіху може прояснити напрямки та вибір.
3. Визначте, де можливі зміни. Якщо ви перебуваєте на переломному або поворотному етапі, аналіз ваших сильних і слабких сторін може виявити пріоритети, а також можливості.

4. Відкоригуйте та уточніть плани в середині курсу. Нова можливість може відкрити ширші шляхи, тоді як нова загроза може закрити шлях, який колись існував.

SWOT також пропонує простий спосіб повідомити про вашу ініціативу чи програму та чудовий спосіб упорядкувати інформацію, яку ви зібрали під час досліджень або опитувань.

ПЕРЕЛІК ВАШИХ ВНУТРІШНІХ ФАКТОРІВ: СИЛЬНІ ТА СЛАБКІ СТОРОНИ (S, W).

Внутрішні фактори включають ваші ресурси та досвід. Загальні області, які слід враховувати:

- Людські ресурси - персонал, волонтери, члени правління, цільова аудиторія.
- Фізичні ресурси - ваше місцезнаходження, будівля, обладнання.
- Фінансові - гранти, фінансові агентства, інші джерела доходу.
- Діяльність і процеси - програми, які ви запускаєте, системи, які ви використовуєте.
- Минулий досвід - будівельний матеріал для навчання та успіху, ваша репутація в суспільстві.

Не будьте занадто скромними, перераховуючи свої переваги. Якщо вам важко назвати їх, почніть із простого перерахування ваших характеристик (наприклад, ми невеликі, ми пов’язані з околицями). Деякі з них, ймовірно, будуть сильними сторонами.

Хоча сильні та слабкі сторони вашої організації є вашими внутрішніми якостями, не ігноруйте точку зору людей поза вашою групою. Визначте сильні та слабкі сторони як з власної точки зору, так і з точки зору інших, у тому числі тих, кому ви служите або з якими маєте справу. Чи бачать інші проблеми чи активи, яких ви не бачите?

Як ви отримуєте інформацію про те, як сторонні сприймають ваші сильні та слабкі сторони? Можливо, ви вже знаєте, якщо слухали тих, кому служите.

Якщо ні, можливо, саме час зібрати таку інформацію. Дивіться відповідні розділи, щоб отримати ідеї щодо проведення фокус-груп , опитувань користувачів і сеансів прослуховування .

ПЕРЕЛІК ЗОВНІШНІХ ФАКТОРІВ: МОЖЛИВОСТІ ТА ЗАГРОЗИ (O, T).

Накиньте широку сітку для зовнішньої частини оцінки. Жодна організація, група, програма чи сусідство не застраховані від зовнішніх подій і сил. Складаючи цю частину свого SWOT-списку, подумайте про свої зв'язки, як на краще, так і на гірше.

- Сили та факти, які ваша група не контролює, включають:
- Майбутні тенденції у вашій галузі чи культурі
- Економіка – місцева, національна чи міжнародна
- Джерела фінансування - фонди, донори, законодавчі органи
- Демографія - зміни у віці, расі, статі, культурі тих, кому ви служите, або у вашій місцевості
- Фізичне оточення (ваша будівля знаходиться в районі міста, що розвивається? автобусна компанія скорочує маршрути?)
- Законодавство (нові федеральні вимоги ускладнюють чи полегшують вашу роботу?)

Місцеві, національні чи міжнародні події

SWOT-аналіз (див. рис. 2.2):

Рис 2.2 SWOT-аналіз

2.3 Аналіз діяльності Креді Агріколь Банк

Креді Агріколь Банк – найстаріший іноземний банк в Україні; працює на ринку з 1993 року та надає весь спектр банківських послуг, є стратегічним партнером агробізнесу та одним із лідерів ринку автокредитування. Банк входить до міжнародної Credit Agricole Group (Франція) , європейського лідера банківського страхування та управління активами та основного партнера французької економіки.

Креді Агріколь у ТОП-5 найнадійніших і найкомфортніших банків України та в ТОП-10 банків за розміром активів. Понад 265 000 активних клієнтів-фізичних осіб, 2 700 міжнародних та корпоративних компаній, понад 7 000 суб’єктів малого та середнього бізнесу підтверджують надійність та відмінну ділову репутацію банку.

Понад десять років Креді Агріколь є стратегічним партнером агробізнесу . Credit Agricole продовжує надавати фінансування за підписаними угодами навіть

у воєнний час. Понад 90% коштів спрямовується на фінансування потреб аграрних компаній в обігових коштах. Кожен четвертий клієнт-аграрій працює в Креді Агріколь більше 10 років.

Під час війни в Україні співробітники Credit Agricole працюють щодня та щоденно обслуговують клієнтів, банк виділив майже 15 млн грн на допомогу постраждалим від війни, а Група висловила солідарність з українцями та призупинила свою діяльність у Росії.

Як Credit Agricole підтримує суспільство з початку війни :

10 млн євро - фонд екстреної солідарності, який запустила Група. Ці кошти будуть спрямовані на допомогу українцям, де пріоритетом буде надання допомоги дітям, а також співробітникам Credit Agricole в Україні.

100 млн грн податків банк сплатив наперед.

7,5 млн грн банк перерахував на допомогу лікарням. На ці кошти було придбано життєво необхідне медичне обладнання для Запорізької обласної клінічної дитячої лікарні та мультивен для перевезення гуманітарної допомоги. Ініціатива реалізована за підтримки благодійного фонду «Твоя опора».

5 млн грн банк передав Червоному Хресту в Україні для надання життєво необхідної допомоги українцям.

600 тисяч злотих Credit Agricole у Польщі надала на допомогу українським лікарням.

270 тис. грн Credit Agricole пожертвувала на гуманітарну ініціативу нашого партнера Львівського IT Кластеру.

24 млн грн залучили клієнти банку на підтримку Збройних Сил України та благодійних фондів через мобільний додаток CA+.

Як Credit Agricole підтримувала клієнтів з початку війни:

Скасовано щомісячну оплату за всіма пакетами карток Credit Agricole: Smart, Voyage, Sommelier, La Force.

Скасовано комісію за переказ коштів на картки інших банків у мобільному додатку CA+.

З 24 лютого Credit Agricole запровадила спеціальні умови банківського

обслуговування та скасувала штрафні санкції за прострочення платежу.

Картки Credit Agricole, термін дії яких закінчився з лютого 2022 року, були автоматично оновлені для подальшого використання.

Credit Agricole у Польщі запровадила безкоштовне відкриття та ведення банківського рахунку для громадян України. Також безкоштовні грошові перекази в Україну.

Розрахунки карткою Credit Agricole доступні в торгових точках та Інтернеті без будь-яких обмежень.

Інтернет-банкінг та мобільний банк CA+ працюють цілодобово.Банк є лідером на ринку автокредитування, кожен третій автокредит виданий саме Креді Агріколь. Це єдиний в Україні банк, який має міжнародний сертифікат ISO 9001 за напрямом автомобільного кредитування, що гарантує клієнтам прозорі умови автокредитування та високу якість обслуговування. Як результат - п’ять років поспіль Креді Агріколь лідер у номінації «Автокредит» в рейтингу «Фінансового клубу».

Понад десять років Креді Агріколь є стратегічний партнером для агробізнесу. Банк обслуговує більше 3000 клієнтів агросектору, в тому числі міжнародні корпорації, великі агрохолдинги, середні агропромислові підприємства та невеликі фермерські господарства. Для цього у банку є широкий спектр продуктів фінансування: фінансування обігового капіталу, авалювання векселів, надання банківських гарантій та інвестиційне фінансування. У банку є також низка партнерських програм для агробізнесу, які передбачають привабливі умови фінансування.

Креді Агріколь - соціально орієнтована компанія, з 2016 року банк реалізовує програму КСВ «We Care!», яка включає три напрями: благодійність, зелені ініціативи і турботу про співробітників.

У 2020 році Креді Агріколь відзначений серед найкращих роботодавців України, найкращих програм КСВ, найкращих міжнародних компаній в Україні та отримав ще низку відзнак та нагород.

Credit Agricole Bank працює на фінансовому ринку України з 1993 року.

Власником цієї універсальної фінансової установи є Credit Agricole Group, французька компанія, що входить до числа найбільших фінансових груп світу. 500 000 приватних і 33 000 корпоративних клієнтів довірили свої фінансові операції та заощадження Креді Агріколь. Сьогодні її регіональна мережа налічує близько 160 відділень по всій Україні. Мережа Credit Agricole налічує майже 300 банкоматів. Банк також виступає партнером Атмосфери, об'єднаної мережі, що налічує понад 1000 банкоматів великих українських банків.

Історія.

Сучасний ПАТ «КРЕДІ АГРІКОЛЬ БАНК» був заснований 10 лютого 1993 року у Львові, як приватний банк під назвою Комерційний банк «Золотий Лев». У листопаді 1993 року банк було реорганізовано в Акціонерне товариство відкритого типу.

Протягом наступних трьох років економічні умови зазнали суттєвих змін. Банк значно розширився, здобув поважну репутацію в Західній Україні, перед ним постали нові цілі й завдання. Одним з важливих напрямів діяльності на той час був розвиток регіональної мережі в неохоплених регіонах.

28 жовтня 1996 року банк змінив свою назву на Акціонерне товариство «Індустріально-експортний банк» (АТ «Індекс-банк»).

Розвиток банку, зростання його фінансових показників і вихід на загальнонаціональний ринок зумовили прийняття рішення про переведення головного офісу АТ «Індекс-банк» зі Львова до Києва. Вже 10 лютого 2000 року Київ став новою адресою головного офісу Банку.

Після переведення головного офісу до Києва було прийнято нову стратегію розвитку банку та його регіональної мережі, спрямовану на максимальну універсалізацію, розширення спектру послуг, створення розгалуженої мережі відділень Банку для обслуговування клієнтів у всіх регіонах України.

У квітні 2000 року АТ «Індекс-банк» одним з перших банків в Україні розпочав реалізацію проекту «Автомобілі в кредит» і зараз є одним з найбільш активних і успішних банків, які здійснюють автокредитування. Разом з

різноманітними видами послуг для фізичних та юридичних осіб банк почав надавати кредити підприємствам аграрного сектора в рамках нових проектів «Трактор у кредит» та «Комбайн у кредит».

У 2001 році банк отримав право надавати послуги з виплати пенсій та грошової допомоги.

У лютому 2002 року Індекс-банк став асоційованим членом міжнародної платіжної системи Visa International.

У липні 2002 року Індекс-банк розпочав обслуговування мікропроцесорних карток російської платіжної системи «Золотая Корона».

30 жовтня 2003 року Банк став афілійованим членом міжнародної платіжної системи MasterCard, і отримав ліцензію на емісію карток MasterCard Gold, Standart, Maestro.

31 серпня 2006 року відповідно до Договіру купівлі-продажу Група «Crédit Agricole» (Франція) стала власником 99,967 % акцій АТ «Індекс-банк». У 2011 році змінив назву на «Crédit Agricole».

Після входження Індекс-банку до Групи «Credit Agricole» міжнародна рейтингова агенція Moody's повідомила про підвищення рейтингів ПАТ «КРЕДІ АГРІКОЛЬ БАНК» до «Baa1» від «B2» довгострокового і короткострокового «P-2» від «Not-Prime» рейтингів по депозитах у місцевій валюті.

Банк є:

- принциповим членом VISA International;
- принциповим членом MasterCard Int.;
- членом Української міжбанківської Асоціації членів платіжних систем «ЕМА»;
- членом ФБР і К (Форум з безпеки розрахунків і операцій з платіжними картами);
- членом Незалежної асоціації банків України (НАБУ);
- членом Американської торгової палати в Україні;
- членом Европейської бізнес-асоціації;

- членом Форуму Провідних Міжнародних Фінансових Установ;
- членом Асоціації «Французьке ділове співтовариство в Україні»;
- членом Британо-Української торгової палати;
- членом Харківського банківського Союзу.
- членом Фонду гарантування вкладів фізичних осіб.

Під час повномасштабного вторгнення військ РФ до України 2022 року компанія відмовилась зупиняти роботу на російському ринку та приєднатись до бойкоту фашистського режиму[3].

Діяльність в інших сферах.

- Банк організовує та підтримує ділові та культурні заходи, зокрема в 2020 році:
- Онлайн-захід «Макроекономічний огляд 2020». Організатор
- П’ять онлайн-зустрічей з преміум-клієнтами в рамках проекту Premium Talks. Організатор
- Міжнародна конференція «Black Sea Grain and Black Sea Oil Trade». Ексклюзивний фінансовий спонсор
- Щорічний Український форум агробізнесу. Генеральний партнер
- Міжнародна конференція «Large Farm Management». Ексклюзивний фінансовий партнер
- Проект «Автомобіль року». Фінансовий партнер
- Культурний захід «Плюс де Франс». Учасник

Рис 2.3 Цінності

Цінності – це основа нашого сталого розвитку, це втілення нашого бажання бути справжнім партнером для наших клієнтів, це відображення глибоких переконань і принципів нашого банку як частини групи Креді Агріколь:

ОРІЄНТАЦІЯ НА КЛІЄНТА (зовнішнього та внутрішнього).

Клієнт – центр нашого всесвіту. Наші колеги – наші внутрішні клієнти

КОМАНДНИЙ ДУХ.

Команда - це головне! Поєднання особистих талантів, досвіду та мотивації співробітників створює основу для загального успіху.

ПРОФЕСІЙНА КОМПЕТЕНТНІСТЬ.

Професійні знання та досвід кожного є загальною цінністю та реальним капіталом банку.

ДИНАМІЧНИЙ.

Готовність до змін та інновацій, гнучкість та проактивність.

ЕТИКА.

Етична та прозора поведінка є запорукою лояльних відносин, прикладом для наслідування в банку та за його межами.

ДОВГОСТРІКОВА СТАБІЛЬНА ВІДПОВІДАЛЬНІСТЬ.

Стабільна організація стабільна протягом тривалого часу. Стійкість як довгострокове зобов'язання.

КЛІМАТИЧНА СТРАТЕГІЯ

Прагнучи до сталого розвитку, Група Креді Агріколь реалізує Кліматичну стратегію, яка має високий пріоритет і є невід'ємною частиною Середньострокового плану розвитку Групи до 2022 року. Креді Агріколь Банк повністю підтримує Групу та планує скоротити свій прямий карбоновий слід на 10% протягом 2018-2022 рр. та бути у ТОП-3 зелених банків України.

Для цього банк впроваджує найкращі світові практики: має власні сонячні батареї, електромобілі та електросамокати для робочих поїздок, проводить енергоаудит та застосовує нові енергоефективні технології у будівництві. У рамках проекту «зелений» офіс співробітники банку сортують сміття, використовують папір зі 100% переробленої сировини та безсульфатні миючі засоби. Команда банку щорічно підтримує акцію «Година Землі» та висаджує дерева.

Ключова роль кліматичного фінансування.

Паризька угода, укладена в 2015 році наприкінці COP21, зобов'язує країни-підписанти дотримуватися траєкторії обмеження глобального потепління до рівня менше + 2 ° у 2100 році, що відповідає найамбітнішому сценарію IPCC. Він також має на меті «зробити фінансові потоки сумісними з профілем еволюції в напрямку розвитку з низькими викидами парникових газів і стійкістю до зміни клімату». Внесок фінансових установ справді важливий у цьому переході, і Crédit Agricole Group , провідний світовий кооперативний банк і провідний фінансист французької економіки, бере в ньому повну участь .

Довгий час відданий цій темі (підписант Принципів Екватора в 2003 році, Принципів клімату в 2008 році та член-співзасновник Принципів зелених облігацій), група Crédit Agricole з повною легітимністю та рішучістю прийняла в 2019 році кліматичну стратегію, який повністю є частиною стратегічного плану Групи «Амбіції 2022» . Відповідно до Паризької угоди, головною метою цієї стратегії Групи є поступовий перерозподіл портфелів фінансування та інвестицій, а також керованих активів на користь енергетичного переходу. Група також дотримується принципів відповідального банківського обслуговування та приєдналася до ООН Колективна відданість кліматичним діям, таким чином

підтверджуючи узгодженість своєї кліматичної стратегії з Цілями сталого розвитку ООН і Паризькою угодою 2015 року.

Світовий лідер у сфері «зелених» облігацій, піонер у кліматичному фінансуванні протягом майже 10 років (оцінка вуглецевого сліду нашого портфеля фінансування з 2011 року, реалізація галузевої політики, включаючи відмову від арктичних шельфів з 2012 року та в 2015 році, завершення проектного фінансування для шахт і вугільних електростанцій у 2015 і 2016 роках), група Credit Agricole як ніколи налаштована працювати на користь переходу на енергетику.

Рис 2.4 Загальна організаційна структура управління

Комплаєнс (контроль за дотриманням норм)

АТ «КРЕДІ АГРІКОЛЬ БАНК» (тут і далі – Банк), як частина міжнародної Групи Креді Агріколь, зобов'язується зробити комплаєнс основою свого розвитку. Надання послуг нашим клієнтам у повній відповідності до норм законодавства та високих етичних стандартів захисту клієнтів та прозорості діяльності є ключовими основами стратегії Банку. Банк впроваджує та регулярно оновлює свої внутрішні процедури відповідно до міжнародних та українських норм та стандартів.

Кодекс Етики Групи

Кодекс Етики Групи Креді Агріколь , опублікований у 2017 році, визначає, що основними цінностями будь-якої установи Групи у світі є орієнтація на клієнтів, відповідальність та солідарність. Цей документ визначає принципи подальших дій та поведінки, яких слід дотримуватися щодо наших клієнтів та всіх зацікавлених сторін, включаючи співробітників, постачальників товарів та послуг, державних представників, асоціацій та недержавних установ, акціонерів та інвесторів.

Кодекс Поводження Креді Агріколь Банку

Кодекс Етики визначає наші зобов'язання, ідентичність та цінності, а також принципи, що лежать в основі наших дій, а Кодекс Поводження Креді Агріколь Банку реалізує зазначені у Кодексі Етики зобов'язання Банку та його співробітників. Окрім застосування всіх юридичних, регулятивних та професійних правил, що регламентують різні види діяльності Банку, Кодекс Поводження, затверджений Наглядовою Радою Банку, відображає прагнення зробити ще більше для надання найкращого обслуговування клієнтам Банку та всім стейк-холдерам. Цей Кодекс Поводження – це інструмент та орієнтир, створений з метою кращого пояснення професійних обов'язків співробітників Банку та дотримання поведінки, що відповідає етиці та цінностям Банку та Групи.

Протидія відмиванню коштів та боротьба з фінансуванням тероризму

Банк впроваджує та реалізує процедури Групи, а також безумовно виконує чинне законодавство України, спрямоване на протидію відмиванню коштів та фінансуванню тероризму, зокрема Закон України «Про запобігання та протидію легалізації (відмиванню) доходів, отриманих злочинним шляхом, фінансування тероризму. знищення», Положення про здійснення банками фінансового моніторингу (затверджено постановою НБУ №417) та інші нормативно-правові акти у сфері фінансового моніторингу. Банк приділяє особливу увагу здійсненню коректної ідентифікації, верифікації та вивченню клієнтів (процедура KYC). Крім того, Банк не здійснює операцій, які мають ознаки ризикової та фіктивної діяльності, ознаки яких визначені Національним банком України (наприклад, відсутність економічного обґрунтування, невідповідність між операцією та

фінансовим станом клієнта, неповна ідентифікація кінцевого бенефіціарного власника, операції, що мають ознаки ухилення від сплати податків, відтоку капіталу чи легалізації кримінальних доходів тощо). Банк має право запросити у своїх клієнтів усю необхідну інформацію для підтвердження законності операцій. Спроба використання клієнтами послуг Банку для заборонених чи незвичайних операцій, або відмова клієнта надати необхідну інформацію, що підтверджує законність операцій, призведе до відмови у проведенні операцій, закритті рахунків та будь-яких інших наслідків відповідно до умов укладених договорів та вимог чинного законодавства України.

Українські та міжнародні санкції

Банк суворо виконує вимоги законодавства України про санкції та не здійснює операцій, які заборонені чи обмежені законодавством. Крім того, як частина міжнародної Групи Креді Агріколь, Банк не здійснює операцій, заборонених Організацією Об'єднаних Націй, США, Європейським Союзом та Політикою Банку.

FATCA

FATCA (The Foreign Account Tax Compliance Act) – це Закон США, спрямований на боротьбу з ухиленням від сплати податків громадянами США та резидентами США, які мають фінансові активи за межами США. Податкова служба США (Internal Revenue Service, IRS – податковий орган США) створила систему щорічного збору інформації від неамериканських фінансових установ про іноземні доходи та активи, які є у платників податків США за межами США. Цей закон вимагає від фінансових установ ввести процедури для ідентифікації платників податків серед своїх клієнтів, у тому числі шляхом надання такими клієнтами необхідної інформації у вигляді заповнених форм самосертифікації та передачі відповідної інформації до Податкової служби США (IRS).

Україна підписала міжурядову угоду зі США, яка набула чинності 18 листопада 2019 року, та погодилася на включення її до списку країн, опублікованого IRS на своєму веб-сайті. Відповідно до законодавства України, АТ „Креді Агріколь Банк“ застосовує вимоги щодо зобов'язань FATCA.

Для виконання вимог FATCA, АТ «Креді Агріколь Банк» зареєстровано на веб-сайті IRS (www.irs.gov) зі статусом іноземної фінансової установи, що відповідає вимогам FATCA, на яку поширюється Модель 1 IGA, та отримав глобальний ідентифікаційний номер посередника (GIIN) CEQ4EV. 00150.ME.804.

АТ "Креді Агріколь Банк" висловлює подяку своїм клієнтам за інформування Банку про свої податкові зобов'язання перед США, щоб забезпечити належне виконання зобов'язань FATCA.

Протидія корупції.

Банк застосовує політику нульової толерантності щодо будь-якого виду корупції чи хабарництва. Банк очікує, що його співробітники, клієнти, постачальники, партнери та будь-які треті сторони, які взаємодіють із Банком, боротимуться з корупцією.

Рис 2.4 EuroCompliance

Антикорупційна система Банку, як і Групи Креді Агріколь, за оцінкою незалежної зовнішньої компанії була відзначена сертифікатом ISO 37001. Це свідчить про рішучість Групи та Банку у становленні якісної програми запобігання корупції (виявленню та аналізу корупційних ризиків) заходів для зменшення таких ризиків. Банк запровадив Антикорупційну політику, затверджену Наглядовою Радою. Ця Політика визначає загальні заходи, що застосовуються всіма підрозділами Банку.

Зокрема, антикорупційна система базується на:

1. управлінні, що включає участь Правління та Наглядової Ради Креді

Агкріколь Банку;

2. картографування ризиків корупції;

3. Спеціальний розділ Кодексу Поведінки, що визначає очікувану поведінку співробітників Банку в антикорупційній сфері;

4. вимоги Кодексу Поводження, затвердженого Наглядовою Радою Банку та визначального норми поведінки працівників банку в різних аспектах, включаючи антикорупційний;

5. системі тренінгів та обізнаності для всіх співробітників;

6. систему конфіденційного інформування;

Система протидії корупції оновлюється на періодичній основі, щоб відповідати зовнішньому середовищу та забезпечувати покращення механізму виявлення та протидії.

Співробітники банку відіграють ключову роль у банківській політиці протидії корупції. Вони повинні діяти відповідно до принципів лояльності та відповідальності. Вони регулярно отримують інформацію шляхом проходження тренінгів щодо протидії корупції. Якщо співробітник став свідком спроби чи скоєння корупційної дії, він повинен проінформувати свого керівника чи компанію відповідно до принципу повної конфіденційності через систему конфіденційного інформування.

Запобігання конфлікту інтересів.

Політика запобігання конфлікту інтересів Банку, затверджена Наглядовою Радою Банку, визначає правила, які застосовуються до співробітників Банку, щоб уникнути конфлікту інтересів. Така Політика спрямована на забезпечення належного управління Банком, а також захист інтересів клієнта.

Інформування АТ «КРЕДІ АГРІКОЛЬ БАНК» про будь-які порушення законодавства, будь-які етичні провини чи події

Банк дуже серйозно ставиться до будь-якої події, яка шкодить її цінностям, етичним стандартам та репутації. Якщо вам відомі або ви стали свідками корупційних дій, шахрайства, порушень законодавства та/або внутрішніх процедур або будь-яких інших випадків неналежної професійної поведінки

співробітників Банку, які суперечать Кодексу Етики Групи Креді Агріколь та/або Кодексу Поведінки Банку, ви можете поінформувати про такі факти з безкорисливих спонукань із зазначенням суттєвих деталей, використовуючи платформу BKMS. Ця платформа використовується Групою Креді Агріколь з урахуванням гарантії захисту інформатора від впливу осіб та фактів, про які він інформує. Отримані факти будуть проаналізовані співробітниками підрозділу Комплаєнс з урахуванням принципів конфіденційності. Система BKMS доступна через інтернет у будь-якому місці та в будь-який час. Система інформування доступна 11 мовами, включаючи українську та англійську.

Зацікавлені особи та ресурси:

1. Замовник проекту:

- Креді Агріколь
- Управління бізнесу
- Відділ підримки

2 Ключові користувачі результатів проекту:

- Клієнти та співробітнки.

3 Куратор проекту:

- Немає куратора

4 Команда проекту

На період проектування та розробки:

- Керівник проекту
- Директори відділень
- ІТ департамент
- Управління бізнесом
- Аналітики
- Замісники
- Співробітники відділень

Вимоги до проектної команди:

- Орієнтація на замовника

- Досвід роботи за фахом не менше 5 розробених схожих проектів
- Командна робота
- Гнучкість у прийнятті рішень

2.4 Матриця відповідальності.

RACI - це скорочена назва системи, яка допомагає командам проясняти ролі в проекті та визначати, хто відповідає за виконання кожного конкретного завдання. Якщо ви ніколи не чули про RACI або хочете створити матрицю RACI для наступного проекту, тут ви знайдете всю необхідну інформацію про створення та використання цих матриць.

Що таке матриця RACI?

Матриця RACI (інша назва – матриця розподілу відповідальності) – це спосіб визначення ролей та обов'язків у групі за будь-яким завданням, віхою або очікуваним результатом проекту . Дотримуючись принципів RACI, можна чітко розподіляти обов’язки та усувати плутанину. RACI розшифровується так:

Відповідальна особа (Responsible). Це людина, на яку безпосередньо покладено роботу. Кожне завдання має мати лише одну відповідальну особу, щоб усі знали, до кого звертатися за інформацією чи з питаннями. Якщо завдання кілька відповідальних, втрачається чіткість і виникає плутанина. Натомість краще запрошувати додаткових учасників, які мають інші ролі по RACI, у яких може бути більше однієї людини.

Підзвітна особа (Accountable). Підзвітна особа відповідає за хід виконання завдань у цілому, хоча при цьому може не брати безпосередньої участі у роботі. Існують два способи визначення підзвітної особи. Іноді це менеджер проекту (або навіть відповідальна особа, хоча в такому разі вона виконуватиме дві різні ролі у процесі виконання завдання). У таких випадках підзвітна особа відповідає за виконання всієї роботи. В інших випадках це топ-менеджер або керівник, який відповідає за затвердження роботи, перш ніж вона вважатиметься виконаною. Як і у випадку з роллю відповідальної особи, роль підзвітної особи має виконувати

лише одна людина.

Консультуюча особа (Consulted). Ця людина чи люди перевіряють та візують роботу перед її здаванням. Кожне завдання, віха проекту або очікуваний результат може мати кілька таких фахівців.

Особа, що інформується (Informed). Це людина чи група фахівців, яких інформують про хід та завершення робіт. Вони здебільшого не залучені до інших аспектів досягнення результату.

Коли потрібно створювати матрицю RACI?

Матриці RACI – це корисний спосіб відстеження ролей усіх зацікавлених осіб у завданні, віху чи очікуваному результаті, особливо якщо ви керуєте складним проектом, у якому бере участь безліч фахівців, що приймають рішення, та експертів у різних галузях. За допомогою матриці RACI можна запобігати невдалим рішенням і уникати перешкод у процесах узгодження, які можуть вплинути на загальний успіх проекту.

Ці матриці (з урахуванням відмінностей від PERT-графіків) особливо корисні, якщо зацікавлені особи можуть виконувати різні ролі в ході проекту. Наприклад, хтось може бути відповідальною особою за одним очікуваним результатом, але інформованим по іншому. У матриці RACI можна чітко прописати цю інформацію, щоб усі знали, хто і за що відповідає.

Переваги та недоліки матриць RACI

Зрештою, все зводиться до наступного: чи потрібно вам створювати матрицю RACI? Вони є корисним інструментом для розподілу обов'язків у проекті, однак у міру його розвитку такі матриці можуть ставати громіздкими. Нижче наводяться переваги та недоліки створення матриць RACI для своєї команди:

Переваги матриць RACI.

Чіткий розподіл ролей та обов'язків у проекті допоможе вашій команді працювати швидше та усувати плутанину при визначенні того, хто що робить. За допомогою матриці RACI можна стежити за тим, щоб два співробітники не займалися однією роботою. В результаті вам буде простіше виконувати спільну

роботу у команді.

Матриці RACI особливо корисні, коли прийняття рішень розподілений між завданнями. Можливі ситуації, в яких інформована особа в одному завданні чи віху є відповідальною чи консультує в іншій. Щоб чітко це позначити, краще відстежувати таку роботу в матриці RACI.

Підводне каміння матриць RACI (і як їх уникати).

Моделі RACI зосереджені на деталях та не охоплюють всю роботу на рівні проекту. Ви можете знати, хто є консультантом у конкретній задачі (що корисно), але ця інформація не допоможе вам зрозуміти, як різні зацікавлені особи взаємодіють у масштабах роботи за проектом в цілому.

До того ж, якщо ви розписуватимете кожне завдання та роль, ваша матриця RACI стане величезною. Що ще гірше, якщо ваш проект якось зміниться, сформована матриця відразу втратить актуальність. В результаті вам буде важко отримувати в реальному часі чітке уявлення про місце кожного завдання у процесі проекту.

Матриці RACI обмежені у своїх можливостях, тому що не можуть адаптуватись до потреб вашого проекту у реальному часі. Щоб задати чіткі очікування та усунути плутанину на рівні проекту, вам потрібний засіб управління проектами.

	Члени правління	Директори відділень	Замісники	Співробітники відділень	Керівник бізнесу	Замісник кер. Бізнесу	Співробітники відділу бізнесу	Керівник ризиків	Замісник кер. ризиків	Керівник юрид. Від.	Зам. Кер. Юр. Відд.	Директор "WebVision"	UI/UX розробник	Програмісти	тестувальтники	Кер.відд.підтримки	Зам. Кер. Від. підтримки	співробітники
Пошук персоналу	К	Уз	П	В														
Співбесіда	К	Уз	П	В														
Підписання договору	К	Уз	П	В														
Проходження курсів	К	Уз	П	В														
Пошук підрядника	К				Уз	В												
Підписання контракту	К				Уз	В				Уз	П							
Закупівля матеріалу	К				Уз	В												
Проведення робіт та монтаж	К					Уз	В	Уз	П	П								
Введення в експлуатацію	К				Уз	П	В	Уз	П	Уз	П							
Проведення оцінювання необхідного обладнання	К	П			Уз	П	В	Уз	П	Уз	П							
Пошук постачальників	К	Уз	П		Уз	П	В	Уз	П									
Закупка	К				Уз	П	В	Уз	П									
Встановлення та введення в експлуатацію	К			В	Уз	П		Уз	П									

Рис 2.5 Матриця відповідальності(1)

	Члени правління	Директори відділень	Замісники	Співробітники відділень	Керівник бізнесу	Замісник кер. Бізнесу	Співробітники відділу бізнесу	Керівник ризиків	Замісник кер. ризиків	Керівник юрид. Від.	Зам. Кер. Юр. Відд.	Директор "WebVision"	UI/UX розробник	Програмісти	тестувальтники	Кер.відд.підтримки	Зам. Кер. Від. підтримки	співробітники
Логотип, гасло, сайт	К											Уз	В	В	В			
Просування у всіх ЗМІ	К				Уз	П	В											
Пошук UI/UX розробника	К	Уз	П	В				Уз	П	Уз	П							
Створення макету сайту	К											Уз	В	В	В			
Пошук розробників	К	Уз	П	В				Уз	П	Уз	П							
Створення	К											Уз	В	В	В			
Випуск в продуктив	К				Уз	П		Уз	П	Уз	П	Уз	В	В	В			
Пошук можливих партнерів	К				Уз	П	В											
Обговорення умов праці	К				Уз	П	В											
Підписання контракту	К				Уз	П	В	Уз	П	Уз	П							

Рис 2.6 Матриця відповідальності(2)

Ролі:

- К – консультування по даному процесу;
- Уз – узгоджує процес;
- П – повідомляє;

- В – відповідає за процес.

ВИСНОВОК ДО РОЗДІЛУ 2.

У другому розділі проведено аналіз ринку схожих програмних продуктів, розглянуто питання що таке клієнтська база, для чого вона, та чим важлива. Також які саме бувають клієнтські бази, це важливо, бо залежно від того, як компанія веде клієнтську базу, накопичує та аналізує інформацію, розрізняють кілька типів баз даних:

1. База «мертвих» клієнтів
2. Потенційна клієнтська база
3. Активна клієнтська база
4. Працююча клієнтська.
5. База постійних.
6. База даних перевірених часом.
7. VIP.
8. Ключові клієнти.

Розписано та представлено що таке SWOT, коли саме його необзідно

застосовувати та для визначення сильних та слабких сторін, а також ширші можливості та загрози проекту провнедено SWOT аналіз.

Проведено аналіз діяльності Креді Агріколь Банк.

Креді Агріколь Банк – найстаріший іноземний банк в Україні; працює на ринку з 1993 року та надає весь спектр банківських послуг, є стратегічним партнером агробізнесу та одним із лідерів ринку автокредитування. Банк входить до міжнародної Credit Agricole Group (Франція) , європейського лідера банківського страхування та управління активами та основного партнера французької економіки. Описана його історія, цінності, взаємодія та підтримка клієнтів де можна побачити, що представлення даних для клієнту буде необхідною та зручною функціональністю як для співробітників так і для самих кліжнтів. Зображена загальна організаційна структура управління, де зображено внутрішню структуру, та комплаенс, що засвідчує про турботу про клієнтів та персональну інформацію.

У розділі застосовано спосіб визначення ролей та обов’язків у групі за будь яким завданням, віхою або очікуваним результатом проекту . Дотримуючись принципів RACI, можна чітко розподіляти обов’язки та усувати плутанину, це зображено на матриці відповідальності(RACI).

РОЗДІЛ 3. РЕАЛІЗАЦІЯ УПРАВЛІННЯ ПРОЄКТОМ РОЗРОБКИ АРХІТЕКТУРИ МІКРОСЕРВІСУ ДЛЯ ОТРИМАННЯ ДАНИХ ПО КЛІЄНТУ.

«Архітектура — це організація системи, втілена в її компонентах, їх взаємозв'язку один з одним і з навколишнім середовищем.

Система — це набір компонентів, об'єднаних для виконання певної функції».

Таким чином, хороша архітектура - це, перш за все, модульна/блочна архітектура.

Критерії гарної архітектури.

Взагалі кажучи, не існує загальноприйнятого терміну «архітектура програмного забезпечення». Однак, коли справа доходить до практики, для більшості розробників вже зрозуміло, який код хороший, а який поганий. Хороша архітектура – це перш за все вигідна архітектура, яка робить процес розробки та підтримки програми простішим і ефективнішим. Добре архітектурну програму легше розширювати та змінювати, а також тестувати, налагоджувати та розуміти. Тобто фактично можна сформулювати перелік цілком розумних і універсальних критеріїв:

Ефективність системи. Перш за все, програма, звичайно, повинна вирішувати поставлені завдання і добре виконувати свої функції, причому в різних умовах. Сюди входять такі характеристики, як надійність, безпека, продуктивність, здатність справлятися з підвищеним навантаженням (масштабованість) тощо.

Гнучкість системи. Будь-яка програма має змінюватися з часом – змінюються вимоги, додаються нові. Чим швидше і зручніше вносити зміни в існуючий функціонал, чим менше проблем і помилок це викликає, тим гнучкіша і конкурентоспроможніша система. Тому під час процесу розробки спробуйте оцінити отримане з точки зору того, як вам, можливо, доведеться змінити це пізніше. Запитайте себе: «Що буде, якщо поточне архітектурне рішення виявиться

неправильним?», «Скільки коду буде змінено в цьому випадку?». Зміна одного фрагмента системи не повинна впливати на інші її фрагменти. Там, де це можливо, архітектурні рішення не повинні бути висіченими в камені, а наслідки архітектурних помилок мають бути розумно обмежені. «Хороша архітектура дозволяє ВІДСТАВИТИ ключові рішення» (Боб Мартін) і мінімізує «вартість» помилок.

Розширюваність системи. Можливість додавання нових сутностей і функцій до системи без порушення її базової структури. На початковому етапі має сенс включити в систему тільки базовий і найнеобхідніший функціонал (принцип ЯГНІ - не знадобиться, «Вам не знадобиться»), але в той же час архітектура має дозволити вам легко розширити додаткові функції за потреби. І так, щоб Вимога, щоб архітектура системи була гнучкою та розширюваною (тобто здатною до змін та еволюції) настільки важлива, що її навіть сформулювали як окремий принцип — Принцип відкритості-закритості — другий із п’яти принципів SOLID. : Сутності програми (класи, модулі, функції тощо) мають бути відкритими для розширення, але закритими для модифікації.

Іншими словами: повинна бути можливість розширити/змінити поведінку системи без зміни/переписування існуючих частин системи.

Це означає, що програма має бути розроблена таким чином, щоб зміна її поведінки та додавання нових функцій були досягнуті шляхом написання нового коду (розширень) без необхідності змінювати існуючий код. У цьому випадку поява нових вимог не спричинить за собою модифікації існуючої логіки, а може бути реалізована насамперед шляхом її розширення. Саме цей принцип лежить в основі «архітектури плагінів» (Plugin Architecture). Техніки, за допомогою яких цього можна досягти, будуть розглянуті пізніше.

Масштабованість процесу розробки. Можливість скоротити час розробки за рахунок додавання в проект нових людей. Архітектура повинна дозволяти розпаралелювати процес розробки, щоб багато людей могли працювати над програмою одночасно.

Перевіряємість. Код, який легше тестувати, міститиме менше помилок і

буде більш надійним. Але тести не тільки покращують якість коду. Багато розробників приходять до висновку, що вимога «хорошої тестованості» також є керівною силою, яка автоматично призводить до хорошого дизайну, і водночас одним із найважливіших критеріїв оцінки його якості: «Використовуйте принцип «класової тестованості». » як «лакмусовий папірець» класу хорошого дизайну. Навіть якщо ви не напишете жодного рядка тестового коду, відповідь на це питання в 90% випадків допоможе зрозуміти, наскільки все «добре» чи «погано» з її дизайн» (Ідеальна архітектура).

Існує ціла методологія розробки програм на основі тестів, яка називається Test-Driven Development (TDD).

Можливість повторного використання. Бажано проектувати систему так, щоб її фрагменти можна було повторно використовувати в інших системах.

Добре структурований, читабельний і зрозумілий код. Ремонтопридатність. Як правило, над програмою працює багато людей – одні йдуть, приходять нові. Після написання програми, як правило, також необхідно супроводжувати програму особам, які не брали участі в її розробці. Таким чином, хороша архітектура повинна полегшувати та швидко сприймати систему новими людьми. Проект повинен бути добре структурований, не містити дублікатів, мати добре сформований код і бажано документацію. І по можливості краще використовувати в системі стандартні, загальноприйняті рішення, знайомі програмістам. Чим екзотичніша система, тим важче її зрозуміти іншим (Принцип найменшого здивування — Принцип найменшого подиву. Він зазвичай використовується стосовно інтерфейсу користувача, але також застосовується до написання коду).

Ну і для повноти картини критерії поганого дизайну:

Це важко змінити, оскільки будь-яка зміна впливає на занадто багато інших частин системи. (Жорсткість, Ригідність).

Коли вносяться зміни, інші частини системи несподівано виходять з ладу. (Крихкість, Крихкість).

Код важко повторно використати в іншій програмі, оскільки надто важко

отримати його з поточної програми. (Нерухомість, Нерухомість). впровадження найімовірніших змін потребувало найменших зусиль.

3.1 Статут проєкту розробки та впровадження програмного забезпечення отримання даних по клієнту.

3.1.1 Опис продукту

Назва проєкту: Розробка архітектури мікросервісу для отримання даних по клієнту.

Суть проєкту: підвищити рівень підтримки, за допомогою сервісу клієнту зможуть отримувати швидко та водночас зручно необхідну для нього інформацію.

Цілі проєкту:

- Автоматизація роботи внутрішніх сервісів;
- Автоматизація роботи контакт центру;
- Створення сервісу для більш швидного та зручного отримання інформації клієнта;
- Впровадження сервісу у поточний функціональний процес;
- Ознайомлення співробітників відділу підтримки з функціоналом.

Продуктом є – сервіс для отримання інформації клієнта, що включає:

- Створення автоматизованої бази;
- Автоматизована взаємодія з іншими сервісами, що вже існують;
- Швидкий доступ клієнта до необхідної для нього інформації;
- Швидкий доступ співробітників до інформації.

Бізнес вимоги:

- Автоматизація роботи співробітників контакт-центру;
- Можливість зручного отримання інформації клієнта;
- Швидкість надання необхідної інформації клієнта повинна скоротитись до 5 хвилин на заявку;

Функціональні вимоги:

- Сумісність з вже існуючими сервісами;
- Стабільний сервіс, що працює цілодобово;
- Можливість редагувати інформацію клієнта;
- Можливість отримати загальну інформацію клієнта;
- Можливість вибору детальної інформації клієнта;
- Зберігати інформацію про авторизацію.

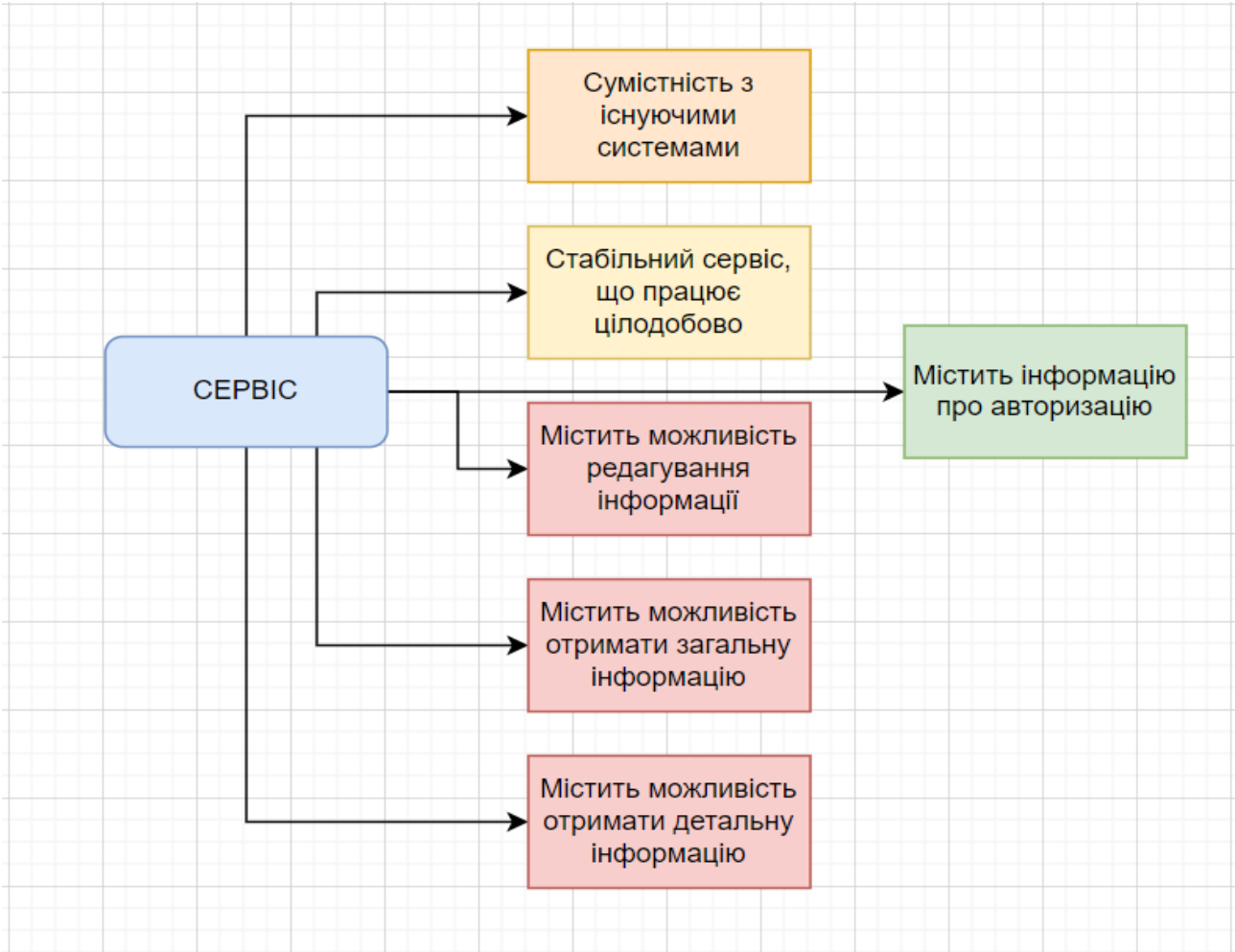


Рис 3.1 Функціональні вимоги до сервісу

Не функціональні вимоги:

- БД Oracle;
- MongoDB;
- Мова програмування JavaScript +React, Java + SpringBoot.
- Docker;

- Jenkins;
- FlayWay;
- Bitbacket.

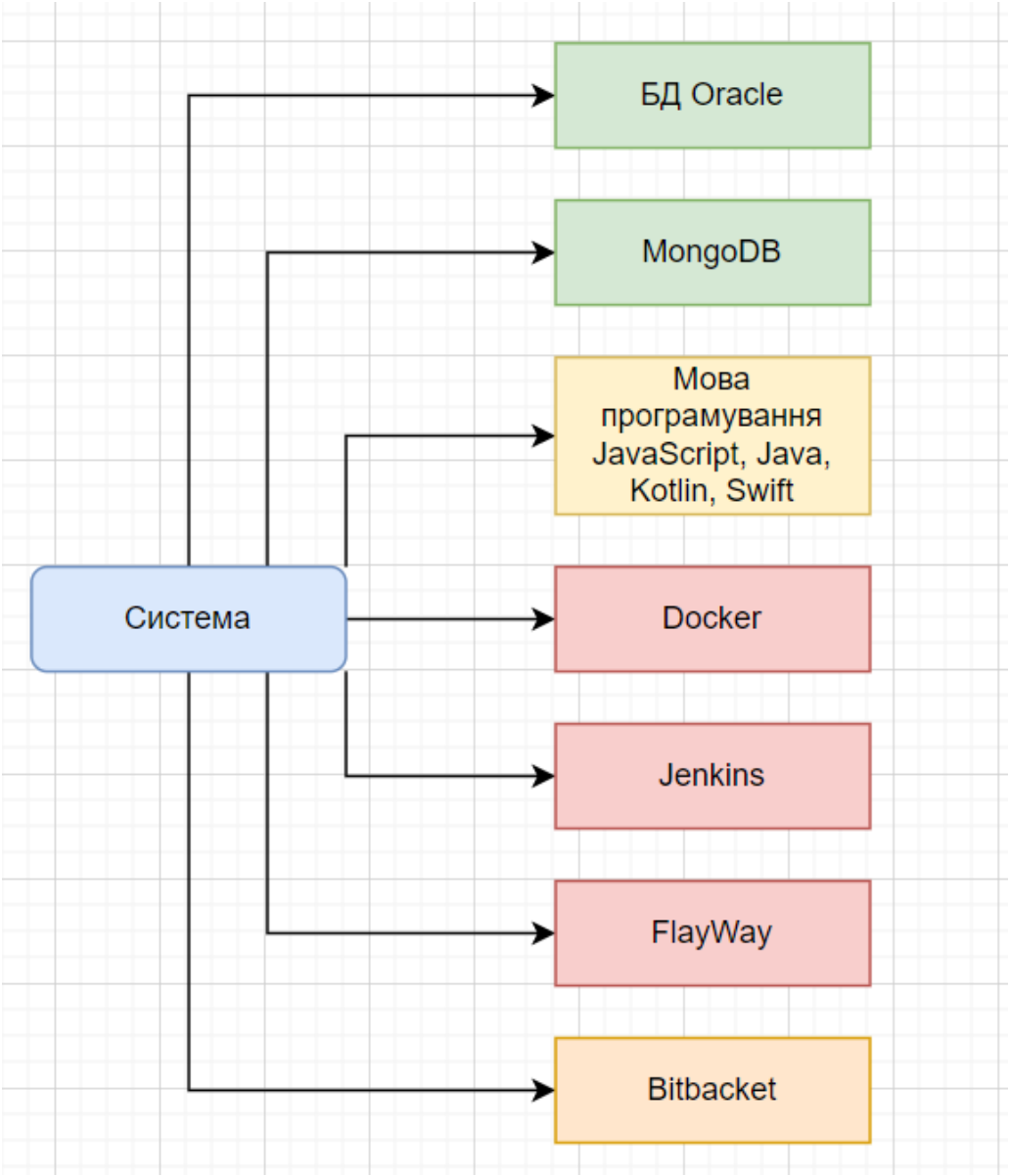


Рис 3.2 Не функціональні вимоги

Правила приймання:

- Приймати розробку буде керівник відділу підтримки із замісником, керівник відділу бізнесу та керівник IT департаменту з архітектором проєкту.
- Для приймання визначені наступні умови:
- Презентація з результатами роботи;

- Представлення роботи сервісу в онлайн форматі з трансляцією у MS Teams;
- Закриття договорів, надання документації - результатів роботи та функціональності, облікової, фінансової і т.д.;
- Проведення навчання робітників контакт-центру, а саме ознайомлення з функціоналом.
- Реалізація проєкту у встановлений термін – 5 місяців.

3.1.2 Структурна декомпозиція робіт

Щоб отримати хорошу архітектуру, потрібно знати, як правильно декомпозувати систему. Отже, варто розібратися - яке розкладання вважається «правильним» і як його краще проводити?

«Правильна» декомпозиція

1. Ієрархічний

Не обов’язково відразу різати додаток на сотні класів. Як уже було сказано, декомпозиція повинна проводитися ієрархічно - спочатку система розбивається на великі функціональні модулі / підсистеми, які описують її роботу в найбільш загальному вигляді. Потім отримані модулі аналізуються більш детально і, у свою чергу, поділяються на підмодулі або об’єкти.

Перед виділенням об'єктів хоча б подумки розділіть систему на основні смислові блоки. Для невеликих додатків часто достатньо двох рівнів ієрархії - спочатку система розділена на підсистеми / пакети, а пакети - на класи.

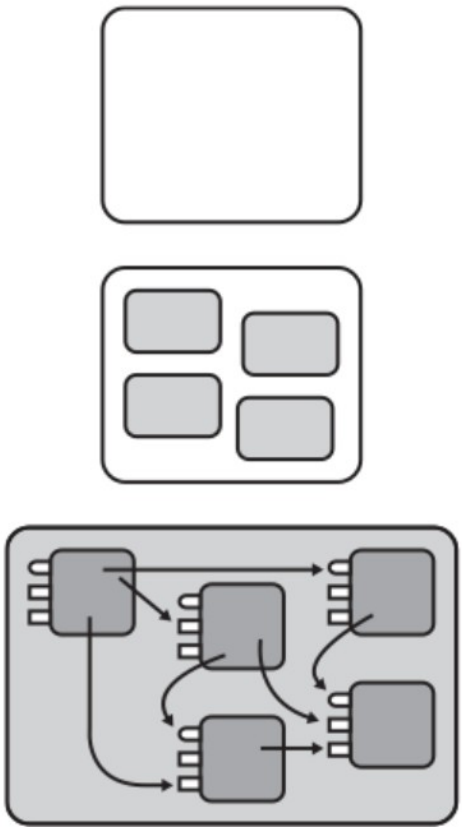


Рис 3.3 Декомпозиція архітектури

Ця ідея, при всій своїй очевидності, не така банальна, як здається. Наприклад, у чому полягає суть такого поширеного «архітектурного шаблону», як Model-View-Controller (MVC)? Йдеться лише про відокремлення презентації від бізнес-логіки, тобто будь-яка користувальницька програма спочатку ділиться на два модулі, один з яких відповідає за реалізацію самої бізнес-логіки (модель), а другий — за взаємодію з користувачем (користувач Інтерфейс або продуктивність). Потім, щоб ці модулі могли розроблятися незалежно, зв'язок між ними послаблюється за допомогою шаблону «Спостерігач» (деталі про те, як послабити зв'язки, буде розглянуто пізніше) і фактично ми отримуємо один із найпотужніших і затребуваних. «шаблони», які зараз використовуються. .

Типовими модулями першого рівня (отриманими в результаті першого поділу системи на найбільші компоненти) є саме «бізнес-логіка», «інтерфейс користувача», «доступ до бази даних», «зв'язок з конкретним обладнанням або ОС». ".

Для наочності на кожному ієрархічному рівні рекомендується виділяти від

2 до 7 модулів.

2. Функціональний

Поділ на модулі / підсистеми найкраще робити виходячи з завдань, які вирішує система. Основне завдання поділяється на складові підзадачі, які можна розв’язувати/виконувати незалежно одна від одної. Кожен модуль повинен відповідати за вирішення певної підзадачі і виконувати відповідну їй функцію. Крім функціонального призначення, модуль також характеризується набором даних, необхідних для виконання ним своїх функцій, а саме:

Модуль = функція + дані, необхідні для її виконання.

Крім того, бажано, щоб модуль міг виконувати свою функцію самостійно, без допомоги інших модулів, тільки на основі своїх вхідних даних.

Модуль - це не довільний фрагмент коду, а окрема функціонально значуща і завершена програмна одиниця (підпрограма), яка забезпечує вирішення певної задачі і, в ідеалі, може працювати самостійно або в іншому середовищі та використовуватися повторно. Модуль повинен бути свого роду «цілісністю, здатною до відносної незалежності в поведінці і розвитку» (Крістофер Александер).

Таким чином, грамотна декомпозиція базується, перш за все, на аналізі функцій системи та даних, необхідних для виконання цих функцій.

3. Висока когезія + низький зв'язок

Найважливішим критерієм якості декомпозиції є те, наскільки модулі орієнтовані на вирішення своїх завдань і є незалежними. Зазвичай це формулюють так: «Отримані в результаті декомпозиції модулі повинні бути максимально внутрішньо спряженими (висока внутрішня когезія) і мінімально взаємопов'язані один з одним (низький зовнішній зв'язок)».

Висока згуртованість, висока згуртованість або «згуртованість» у межах модуля вказує на те, що модуль зосереджений на вирішенні однієї вузької проблеми, а не займається виконанням різнорідних функцій чи непов’язаних обов’язків. (Cohesion - згуртованість, характеризує ступінь зв'язку між собою завдань, які виконує модуль)

Наслідком високої згуртованості є принцип єдиної відповідальності (перший з п'яти принципів SOLID), згідно з яким будь-який об'єкт/модуль повинен мати лише одну відповідальність і, відповідно, не повинно бути більше однієї причини для його зміни.

Low Coupling, слабкий зв'язок, означає, що модулі, на які розділена система, повинні бути, якщо можливо, незалежними або слабко пов'язаними один з одним. Вони повинні мати можливість взаємодіяти, але при цьому знати якомога менше один про одного (принцип мінімуму знань).

Це означає, що при правильному дизайні, змінюючи один модуль, вам не доведеться редагувати інші, або ці зміни будуть мінімальними. Чим слабший зв'язок, тим легше написати/зрозуміти/розширити/відремонтувати програму.

Вважається, що добре спроектовані модулі повинні мати такі властивості:

- ✓ функціональна цілісність і завершеність - кожен модуль реалізує одну функцію, але реалізує її якісно і повністю; модуль самостійно (без допомоги додаткових засобів) виконує повний набір операцій для реалізації своєї функції.
- ✓ один вхід і один вихід - на вході програмний модуль отримує певний набір вихідних даних, виконує змістовну обробку і повертає один набір результатних даних, тобто реалізується стандартний принцип ІПО - вхід-процес-вихід;
- ✓ логічна незалежність - результат роботи програмного модуля залежить тільки від вихідних даних, але не залежить від роботи інших модулів;
- ✓ слабкі інформаційні зв'язки з іншими модулями - обмін інформацією між модулями по можливості слід звести до мінімуму.

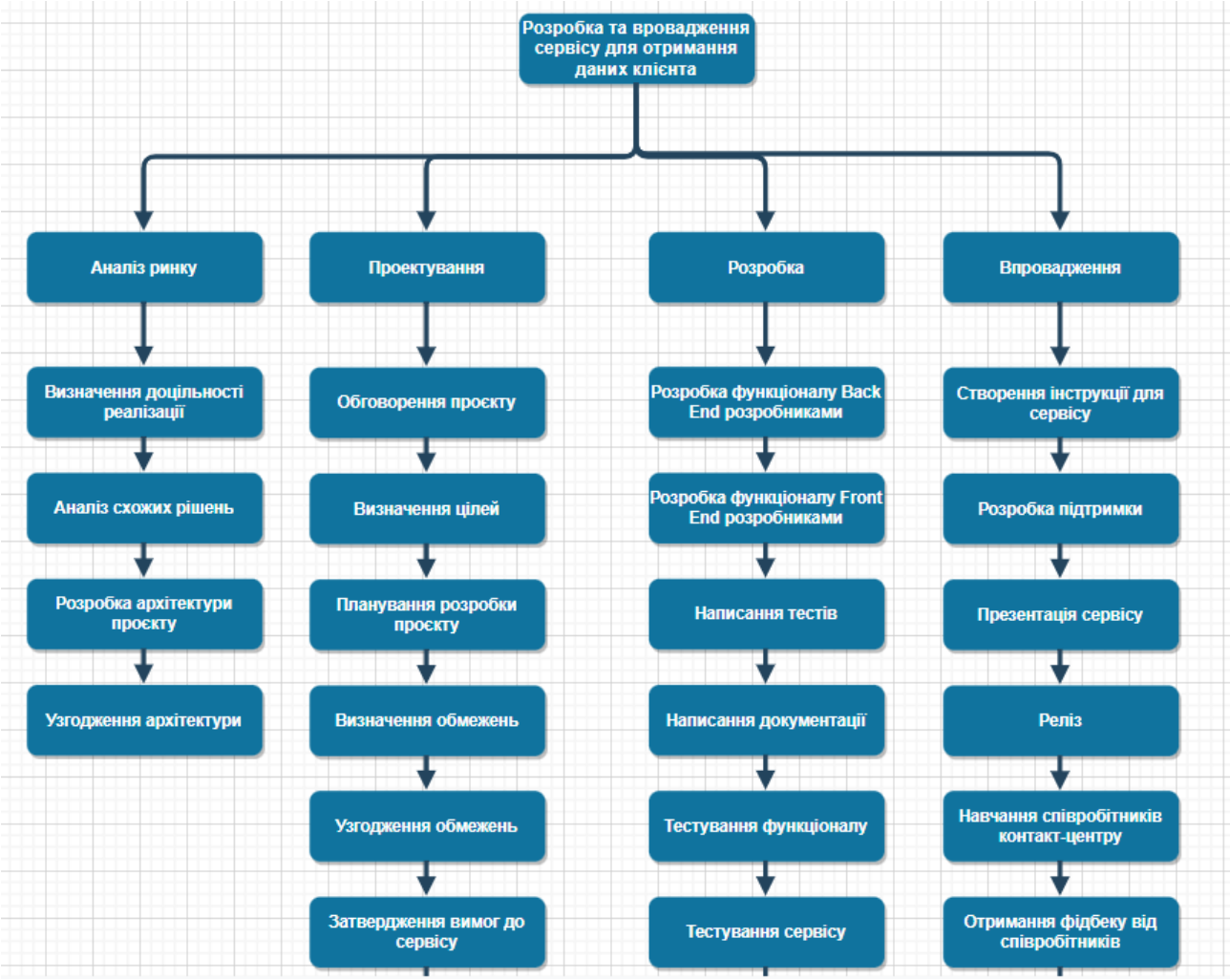


Рис 3.4 Структурна декомпозиція

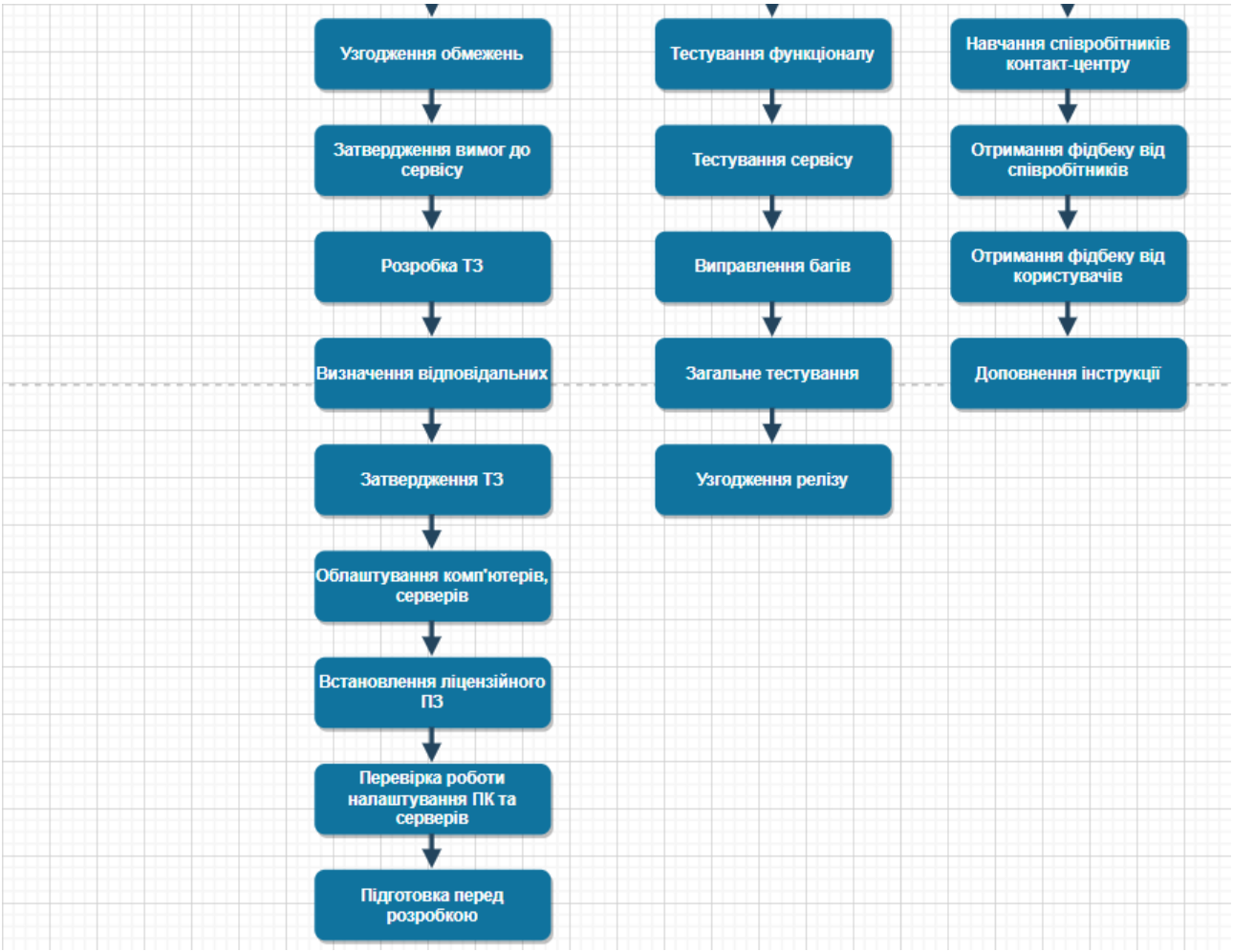


Рис 3.5 Структурна декомпозиція(продовження)

3.1.3 Обмеження проєкту

В ході реалізації проекту важливо, щоб співробітники виконавця враховували всі обмеження проекту.

Успіх будь-якого проекту залежить від продуманого та грамотного менеджменту, організації ефективної діяльності та її спрямованості на досягнення кінцевого результату. Але будь-яке управління здійснюється в певних обмеженнях, на які керівник проекту повинен звертати підвищену увагу.

Говорячи про обмеження проекту, вони часто наводять як приклад відомий закон Лермана: будь-яку технічну проблему можна подолати за достатньо часу та грошей, і вказують, що ні часу, ні грошей ніколи не вистачить.

У процесі реалізації проекту активно використовуються інструменти

управління обмеженнями. Так, для врахування часових параметрів використовуються методи побудови та контролю графіків роботи. Для управління фінансовими ресурсами - методи формування фінансового плану (бюджету) проекту; По ходу роботи бюджет контролюється, щоб переконатися, що витрати не виходять з-під контролю. З метою ресурсного забезпечення проекту використовуються спеціальні методи управління людськими і матеріальними ресурсами (наприклад, матриця відповідальності, діаграми завантаження ресурсів тощо). З усіх обмежень у проекті найважче відстежити обмеження щодо результатів. Основна проблема зводиться до постановки завдань і їх контролю. Для її вирішення, зокрема, використовуються методи управління якістю.

Усі гнучкі проекти розробки програмного забезпечення створюються з певними цілями: щось має бути виконано до певного терміну, у межах певного бюджету. Знайти баланс між цими трьома параметрами (вмістом, рядками та вартістю) може бути складно.

Класична форма тристоронньої позовної давності.

Форма потрійного обмеження описує обмеження управління проектом, які нерозривно взаємозалежні одне від одного: ви не можете змінити один параметр, не вплинувши на два інших.

ПАРАМЕТРИ ПОТРІЙНИХ ОБМЕЖЕНЬ

Обсяг роботи – робота, яку необхідно виконати, наприклад розробка функціональних можливостей для надання працездатного продукту.

Ресурси включають бюджет і членів команди, які відповідають за впровадження та виконання.

Час, потрібний командам для виведення своєї роботи на ринок, виражається в графіках випусків і етапах.

Далі зображено основні обмеження проєкту, де зображені віхи та дати проєкту зображено на таб. 3.1).

Таблиця 3.1 Обмеження проєкту

Код	Назва задачі	Тривалість	Початок	Завершенн

		задачі		я
1.	Початок проєкту	0 днів	05.09.22	05.09.22
2.	Аналіз ринку	12 днів	06.09.22	22.09.22
3.	Проєктування сервісу	39 днів	22.09.22	16.11.22
4.	Розробка сервісу	70 днів	16.11.22	22.02.23
5.	Впровадження сервісу	24 днів	22.02.23	27.03.23
6.	Завершення проєкту	0 день	27.03.23	27.03.23

1. Термін: планується виконати весь проект протягом 147 днів в термін з 05.09.2022 по 27.03.2023

2. Загальний бюджет проєкту 120 000 000 грн (без ПДВ)

Джерелом бюджету є кошти виділені із загального бюджету, який планується щорічно. Загальна вартість проєкту становить 1 мільйон 102 тисячі гривень 890 гривень, залишок – 97 тисяч 110 гривень, розподіляється на резерв та на непередбачувані витрати.

3.1.4 Команда

Сьогодні цифровізація торкнулася майже всіх сфер бізнесу: від фінансів і охорони здоров’я до транспорту та промисловості. Однак цифрова трансформація компанії неможлива без кваліфікованих IT-фахівців. Таким чином, IT-команди стають невід'ємною частиною структури дуже багатьох організацій.

Звичайно, всі компанії різні: у них різні бізнес-цілі, вони стикаються з різними проблемами і, відповідно, впроваджують різні технології. Усе це відображається на структурі та розмірі їхніх IT-відділів. Вони можуть сильно відрізнятися один від одного. Однак деякі закономірності простежуються. Як правило, команди формуються навколо певних функцій. У цій статті ми поговоримо про найпоширеніші типи IT-команд, їхні ролі та обов’язки.

В організаціях, які використовують сучасні підходи (Agile, Scrum), посадові обов’язки деяких спеціалістів, таких як розробники, адміністратори та

спеціалісти із забезпечення якості, можуть збігатися.

Команда проєкту складається зі співробітників IT департаменту. До команди входять:

- Менеджер проєкту;
- Архітектор проєкту;
- Системний аналітик;
- Back End розробник 1;
- Back End розробник 2;
- Fron End розробник 1;
- Full Stack розробник 1;
- Тестувальник 1;
- Тестувальник 2;
- DevOps інженер;

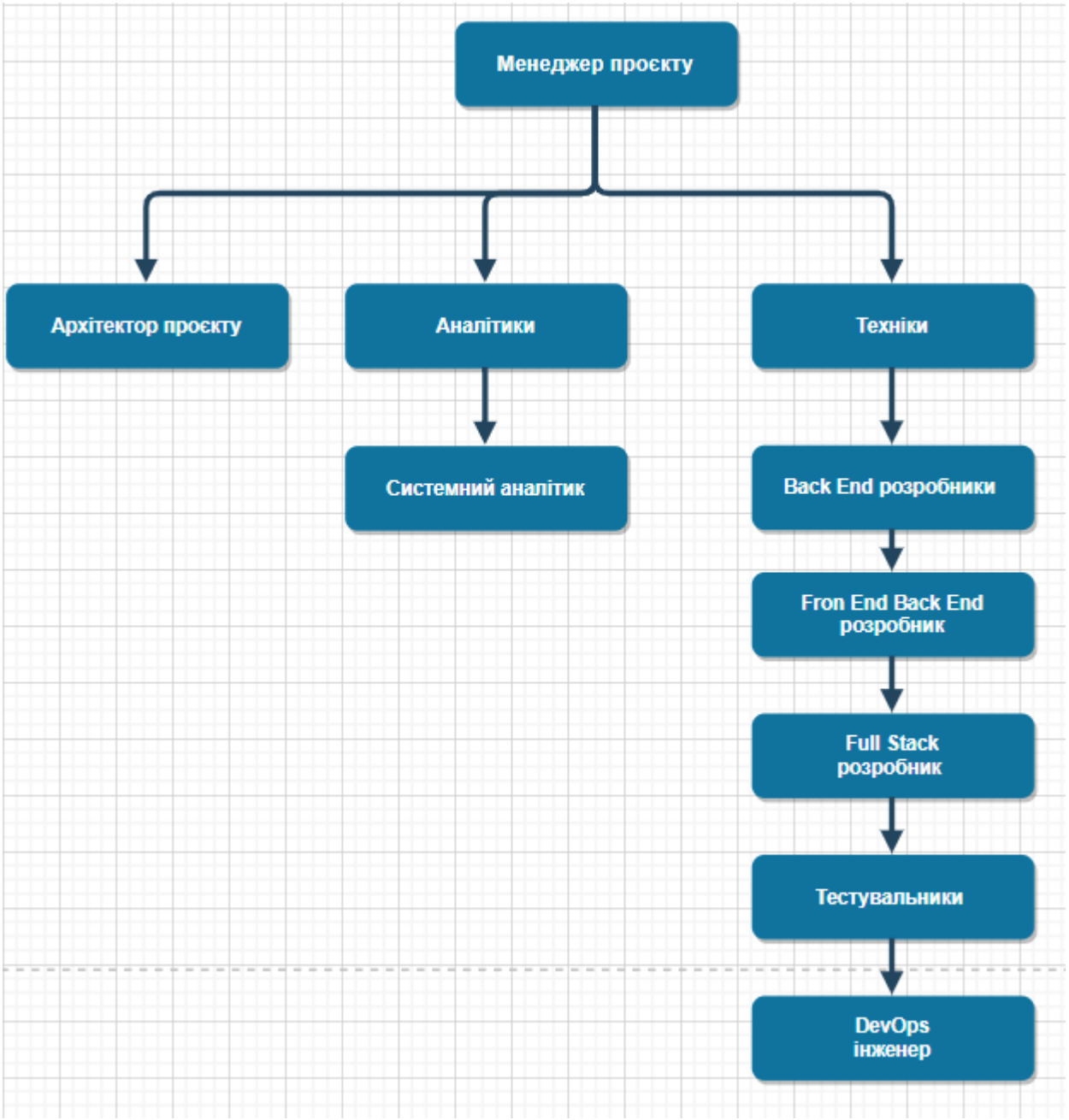


Рис 3.6 Ієрархічна модель організаційної структури

3.1.5 Зацікавлені особи і ресурси

Зацікавлені сторони проекту — це люди, які можуть впливати на проект, над яким ви працюєте, або на яких може вплинути цей проект. Вони можуть бути розташовані на різних рівнях організації, від окремих виконавців до вищого керівництва. Однак, якщо вони беруть участь у вашому проекті, усі вони важливі. Навіть якщо ці люди не беруть безпосередньої участі в повсякденній роботі

проекту, на них може вплинути результат проекту.

У широкому сенсі результат вашого проекту може вплинути на будь-кого. Проте в управлінні проектами учасниками проекту вважаються люди, які певною мірою беруть участь у процесі прийняття рішень щодо нього. До них можуть входити важливі люди, які перевіряють результати проекту, а також товариші по команді, які виконують роботу від пункту А до пункту Б. Члени цільової аудиторії також є зацікавленими сторонами проекту, оскільки рішення, які приймються, мають найбільший вплив на них.

По суті, визначити, чи є особа зацікавленою стороною проекту, можна, поставивши одне просте запитання: «Чи вплине на неї робота, яку я виконую?» Якщо відповідь ствердна, ця особа, швидше за все, є зацікавленою стороною проекту.

Зацікавлені сторони в проєкті:

1. Замовники проєкту – начальник бізнес відділу, начальник відділу підтримки.
2. Співробітники контакт-центру;
3. Директор ІТ департаменту;
4. Організаційні групи – внутрішні зацікавлені сторони, фокус група.
5. Менеджер проєкту.

3.2 План управління проєктом розробки сервісу

3.2.1 Управління термінами

Для відстежування індивідуальних завданнь кожного з учасників проєкту, менеджеру проєкту потрібен відповідний інструмент для планування та управління часом.

Пріоритетом управління часом є визначення взаємозв'язку проектних завдань і шляхів їх реалізації; це в першу чергу впливає на те, як розташовуються завдання в робочому графіку.

Опис плану проєкту з тривалістю робіт та віхами проєкту зображено на таб. 3.2:

Таблиця 3.2 Плану проєкту

Код	Назва задачі	Тривалість задачі
1.	Початок проєкту	0 днів
1.	**Аналіз ринку**	**12 днів**
1.1	Визначення доцільності реалізації	1 день
1.2	Аналіз схожих рішень	3 дні
1.3	Розробка архітектури проєкту	3 дні
1.4	Узгодження архітектури	2 дні
2.	**Проєктування сервісу**	**39 днів**
2.1	Обговорення проєкту	2 дні
2.2	Визначення цілей	2 дні
2.3	Планування розробки проєкту	3 дні
2.4	Визначення обмежень	3 дні
2.5	Узгодження обмежень	2 дні
2.6	Затвердження вимог до сервісу	2 дні
2.7	Розробка ТЗ	5 днів
2.8	Визначення відповідальних	1 день
2.9	Затвердження ТЗ	2 дні
2.10	Облаштування комп'ютерів, серверів	7 днів
2.11	Встановлення ліцензійного ПЗ	5 днів
2.12	Перевірка роботи налаштування ПК та серверів	3 дні
2.13	Підготовка перед розробкою	2 дні
3.	**Розробка**	**70 днів**
3.1	Розробка функціоналу back end розробниками	17 днів
3.2	Розробка функціоналу front end розробниками	14 днів
3.3	Написання тестів	7 днів
3.4	Написання документації	3 дні

3.5	Тестування функціоналу	5 днів
3.6	Тестування сервісу	6 днів
3.7	Виправлення багів	7 днів
	Продовження таблиці Таблиці 3.2	
3.8	Загальне тестування	9 днів
3.9	Узгодження релізу	2 дні
4.	**Впровадження**	**24 дні**
4.1	Створення інструкції для сервісу	4 дні
4.2	Розробка підтримки	4 дні
4.3	Презентація сервісу	1 дні
4.4	Реліз	2 дні
4.5	Навчання співробітників контакт-центру	5 днів
4.6	Отримання фідбеку від співробітників	2 дні
4.7	Отримання фідбеку від користувачів	2 дні
4.8	Доповнення інструкції	3 дні
	Завершення	**1 день**

3.2.2 Управління вартістю

Управління вартістю проекту на основі витрат пов’язане з одним із трьох основних обмежень у проектах – вимогами до вартості, часу та домену. Дотримання всіх цих обмежень дозволяє завершити проект у заплановані терміни та бюджет із повним задоволенням заздалегідь визначених очікувань замовника (тобто з повним досягненням усіх заздалегідь визначених результатів).

Важливою особливістю процесів управління вартістю проекту є їх дуже тісний зв’язок з іншими процесами планування. Зокрема, важко припустити, що без інформації про необхідні ресурси та календарного плану можна буде розробити правильний бюджет.

Інформація про ризики проекту також може істотно вплинути як на розмір,

так і на структуру проекту. Витрати на ресурси зображено на таб. 3.3:

Таблиця 3.3 Витрати на ресурси

Код	Назва ресурсу	Тип ресурсу (матеріальний, трудовий)	Вартість (за ос./год. чи штуку)
1.	Менеджер проєкту	Трудовий	160/год.
2.	Системний аналітик	Трудовий	100
3.	Back end розробник 1	Трудовий	130
4.	Back end розробник 2	Трудовий	125
5.	Тестувальник 1	Трудовий	100
6.	Тестувальник 2	Трудовий	110
7.	Нові ПК	Матеріальний	37000
8.	Монтажники	Трудовий	80
9.	Front end розробник	Трудовий	100
10.	Full stack розробник	Трудовий	110
11.	Dev ops інженер	Трудовий	120
12.	Архітектор проєкту	Трудовий	160
13.	Сервери	Матеріальний	150000

Опис плану проєкту з управління вартістю зображено на таб. 3.4:

Таблиця 3.4 Управління вартістю

Код	Назва задачі	Тривалість задачі	Вартість етапу
1.	Аналіз ринку	12 днів	21 010,00 гривень
2.	Проєктування сервісу	39 днів	903 200,00 гривень
3.	Розробка сервісу	70 днів	138 200,00 гривень
4.	Впровадження сервісу	24 днів	40 480,00 гривень

Загальний бюджет проєкту 120 000 000 грн. Загальна вартість проєкту становить 1 мільйон 102 тисячі гривень 890 гривень, залишок – 97 тисяч 110

гривень, який можна розподілити на резерв та на непередбачувані витрати.

3.2.3 Управління ресурсами

Розподіл ресурсів за завданнями полягає в чіткому розподілу по задачі та виділенню тривалості та самих ресурсів як матеріальних, трудових чи будь-яких інших, на кожному рівні декомпозиції процесів типів ресурсів, необхідних для виконання проєкту. Отже, можуть бути визначені види матеріально-технічних, людських і фінансових ресурсів. Кожен тип ресурсу деталізується на більш конкретні категорії. Виконання цих дій призводить до побудови ієрархічної структури ресурсу (Resource Breakdown Structure). Зрештою, базовий рівень ресурсів відображає детальний розподіл ресурсів між усіма типами робіт. План розподілу ресурсів на конкретних етапах проєкту зображено на таб. 3.5:

Таблиця 3.5 Розподіл ресурсів

Код	Назва задачі	Тривалість задачі	Ресурс
1.	**Аналіз ринку**	**12 днів**	**-**
1.1	Визначення доцільності реалізації	1 день	Системний аналітик
1.2	Аналіз схожих рішень	3 дні	Системний аналітик, менеджер проєкту
1.3	Розробка архітектури проєкту	3 дні	Системний аналітик, менеджер проєкту, архітектор
1.4	Узгодження архітектури	2 дні	Менеджер проєкту, архітектор
2.	**Проєктування сервісу**	**39 днів**	**-**
2.1	Обговорення проєкту	2 дні	Системний аналітик, менеджер проєкту,

			архітектор, dev ops інженер
2.2	Визначення цілей	2 дні	Системний аналітик, менеджер проєкту, архітектор, dev ops інженер
2.3	Планування розробки проєкту	3 дні	Архітектор, dev ops інженер
2.4	Визначення обмежень	3 дні	Менеджер проєкту, архітектор, системний аналітик
2.5	Узгодження обмежень	2 дні	Менеджер проєкту, архітектор, системний аналітик
2.6	Затвердження вимог до сервісу	2 дні	Менеджер проєкту, архітектор
2.7	Розробка ТЗ	5 днів	Архітектор, dev ops інженер, системний аналітик
2.8	Визначення відповідальних	1 день	Менеджер проєкту, архітектор
2.9	Затвердження ТЗ	2 дні	Менеджер проєкту, архітектор
2.10	Облаштування комп'ютерів, серверів	7 днів	Монтажники
2.11	Встановлення ліцензійного ПЗ	5 днів	Монтажники
2.12	Перевірка роботи налаштування ПК та серверів	3 дні	Монтажники, dev ops інженер
2.13	Підготовка перед розробкою	2 дні	Dev ops інженер
3.	**Розробка**	**70 днів**	**-**
3.1	Розробка функціоналу back end розробниками	17 днів	Back end розробник 1, Back end розробник 2
3.2	Розробка функціоналу front end	14 днів	Front end розробник,

	розробниками		Full stack розробник
3.3	Написання тестів	7 днів	Тестувальник 1
3.4	Написання документації	3 дні	Back end розробник 2, тестувальник 2
3.5	Тестування функціоналу	5 днів	Back end розробник 1, Full stack розробник, тестувальник 1, тестувальник 2
3.6	Тестування сервісу	6 днів	Тестувальник 1, тестувальник 2
3.7	Виправлення багів	7 днів	Back end розробник 1, Back end розробник 2, Full stack розробник
	Продовження таблиці Таблиці 3.2		
3.8	Загальне тестування	9 днів	Тестувальник 1, тестувальник 2
3.9	Узгодження релізу	2 дні	Менеджер проєкту, архітектор
4.	**Впровадження**	**24 дні**	**-**
4.1	Створення інструкції для сервісу	4 дні	Dev ops інженер, тестувальник 1
4.2	Розробка підтримки	4 дні	Back end розробник 1, Dev ops інженер, Full stack розробник
4.3	Презентація сервісу	1 дні	Менеджер проєкту
4.4	Реліз	2 дні	Dev ops інженер, архітектор
4.5	Навчання співробітників контакт-центру	5 днів	Системний аналітик, тестувальник 2
4.6	Отримання фідбеку від	2 дні	Менеджер проєкту

	співробітників		
4.7	Отримання фідбеку від користувачів	2 дні	Менеджер проєкту
4.8	Доповнення інструкції	3 дні	Тестувальник 2
	Завершення	**1 день**	

3.2.4 Управління якістю

Цілі в галузі якості. Головною ціллю є:

- збільшення розміру власних коштів, що забезпечує динаміку зростання обсягів бізнесу, щорічно залишати у розпорядженні не менше ніж 45% чистого прибутку;
- підтримання оптимального співвідношення ліквідності та прибутковості банківських операцій;
- забезпечення стабільності та стійкості по відношенню до існуючих та потенційних ризиків, приділяючи особливу увагу управлінню кредитним ризиком;
- забезпечення підвищення якості банківських послуг, удосконалення продуктового ряду з метою вибудовування комплексних, довготривалих відносин з клієнтами та завоювання лідируючих позицій з надання платіжних послуг населенню;
- забезпечення якісного розвитку бізнес-процесів на основі автоматизації та впровадження інформаційних технологій;

Довгостроковою ціллю є:

- збереження та розширення клієнтської бази до 200 тисяч клієнтів за наступні п’ять років;
- вдосконалення системи корпоративного управління та організаційної структури за наступні три роки;
- підтримка високої кваліфікації співробітників, удосконалення системи мотивації праці. Підтримка комфортних умов праці персоналу.

Політика в галузі якості

Банк розглядає якість як всебічну характеристику своєї діяльності (продукти/послуги, бізнес-процеси, персонал, інформаційні технології, технічні системи) та відповідність цієї діяльності найвищим вимогам Клієнтів, Партнерів, Акціонерів та Регулюючих органів.

Основні напрямки діяльності Банку – надання банківських продуктів/послуг для клієнтів – юридичних осіб та фізичних.

Основні принципи політики у сфері якості:

- Забезпечення високого рівня якості продуктів та послуг.
- Стабільне покращення якості продуктів та послуг та бізнес-процесів.
- Задоволення поточних та потенційних потреб клієнтів, орієнтація на споживача.
- Банк орієнтований на встановлення та підтримання взаємовигідних довгострокових відносин з клієнтами, партнерами та контрагентами.
- Гарантія збереження коштів клієнтів та виконання своїх зобов’язань.
- Застосування сучасних підходів, методик, технологій та інших засобів для покращення якості продуктів та послуг та діяльності Банку в цілому.
- Відповідальність кожного співробітника за якість та результати своєї роботи, а також за вдосконалення якості своєї роботи та діяльності Банку загалом.
- Зобов’язання керівництва щодо незмінного дотримання встановлених принципів Політики у сфері якості, створення та підтримання необхідного внутрішнього середовища.
- Менеджмент якості є невід’ємним та рівноправним завданням керівництва поряд з іншими завданнями управління.
- Однозначний розподіл повноважень, функціональних обов’язків та відповідальності серед усіх працівників Банку.
- Залучення у створення та розвиток системи менеджменту якості всього персоналу Банку, створення для співробітників мотивуючих умов, спрямованих на підвищення якості та ефективності роботи.

- Використання передових банківських та інформаційних технологій, що дозволяють здійснювати обслуговування та розрахунки на якісно високому рівні.

Основні цілі:

- Побудова та постійне покращення системи менеджменту якості у Банку відповідно до вимог міжнародного стандарту ISO 9001:2015 та успішних практик банківської галузі.
- Розробка та покращення Стандартів якості для продуктів / послуг та бізнес-процесів Банку.
- Розробка та модифікація продуктів та послуг відповідно до вимог клієнтів та кращих рішень чи пропозицій ринку.
- Систематичне та якісне навчання або атестація керівників та спеціалістів Банку.
- Проведення комплексних рекламних досліджень.
- Реалізація зворотного зв'язку з клієнтами на всіх встановлених каналах.
- Організація ефективної роботи із претензіями.
- Періодичний аудит діяльності Банку з позицій якості.

Ця політика доводиться до всіх співробітників і є основою для постановки цілей Банку в галузі якості, реалізується і знаходиться під особистим контролем директора з якості.

Керівництво Банку бере на себе зобов’язання щодо реалізації політики в галузі якості та її актуалізації у зв’язку із вимогами клієнтів і ринку, що змінюються.

IBAN – стандарт № 13616 Міжнародної організації зі стандартизації ISO та Європейського комітету з банківських стандартів ECBS – міжнародний номер банківського рахунку. Тобто це стандарт, за яким здійснюють кодування банківських рахунків за єдиними міжнародними правилами.

В Україні IBAN означає заміну комбінацій із коду МФО банку + номер

розрахункового рахунку на міжнародний формат рахунку, що складається з 29 символів (літери та цифри):

Це дозволить:

- вилучити із розрахункових документів реквізит «Код банку платника/отримувача/стягувача»;
- одночасно ідентифікувати країну та банк власника рахунку;
- зменшити витрати часу заповнення додаткових реквізитів.

Процедура внутрішніх перевірок проводиться згідно з вказівками, описаними в стандарті ISO 19011. Процедура, описана в Керівництві з управління якістю, розкриває основні етапи проведення внутрішнього аудиту.

3.2.5 Управління ризиками

Будь-який проєкт не є захищеним на сто відсотків від всіх ризиків які можуть виникнути. Деякі ризи виникають не через якісь дії людей чи не правильно спланованого проєкту, а є зовнішні ризики, на які люди не впливають, але потрібно вміти реагувати на них.

Управління ризиками проекту - це процес пошуку, оцінки та запобігання потенційним проблемам. Цей процес регулярний, превентивних дій на старті проекту недостатньо.

Управління ризиками не тільки зменшує вплив негативних ситуацій на проект. Це вивільняє ресурси: матеріальні, трудові.

Управління внутрішніми ризиками зображено на таб. 3.6:

Таблиця 3.6 Управління внутрішніми ризиками

Назва ризику	Ймовірність	Вплив	Спосіб уникнення (пом'якшення)
1. Ризики недотримання графіку виконання проекту	0,4	0,7	- Потрібно чітко розрахувати план дій та час, що буде затрачено; - Мати кваліфікованих співробітників для підтримки в команді на випадок форс-мажорів.
2. Перевищення бюджету	0,5	0,7	- Проаналізувати всі необхідні росходи на

проекту в процесі реалізації			матеріали. - Мати додаткові кошти на випадок форс-мажору.
3. Низька кваліфікація робітників	0,2	0,3	- Проводити чітко сплановані співбесіди. - Підвищувати кваліфікацію працівників курсами.
4. Ризик некоректно складеної документації, в результаті чого підрядник не виконає умови договору	0,1	0,8	- Перевірка юристами всієї складеної документації.
5. Ризик закупівлі неякісного матеріалу	0,2	0,8	- Чітко складений договір з переліком матеріалів та якістю; - Наявність сертифікації на матеріл; - Перевірка поставлених матеріалів.
6. Ризик виходу із строю техніки, сервері, обладнання.	0,3	0,7	- Гарантії на обладнання та техніку; - Сертифікація.
7. Ризик в затримці поставки необхідних матеріалів.	0,2	0,5	- Прописувати в договорі з постачальником штрафи за затримку матеріалів.
8. Витік конфіденційної інфомації.	0,1	0,8	- Перевірка службою безпеки кожного працівника; - Донесення до кожного співробітника стосовно відповідальності.
9. Ризик виявлення прихованих дефектів.	0,2	0,4	- Проведення детального обстеження та детальна перевірка за всіма заходами безпеки, а також монтаж та демонтаж.
10. Корупція.	0,2	0,6	- Високі зарплати, перевірені працівникі.

Управління зовнішніми ризиками зображено на таб. 3.7:

Таблиця 3.7 Управління зовнішніми ризиками

Назва ризику	Ймовір	Вплив	Спосіб уникнення (пом’якшення)

	ність		
1.Припинення фінансування	0,1	1	- Можливість отримання кредиту, заключення договорів з партнерами.
2. Заборгованості клієнтів.	0,1	0,6	- Перевірка кредитної історії та платіжоспроможності клієнта.
3. Ризик фінансової кризи в країні.	0,1	0,6	- Розробити систему антикризових заходів у випадку реалізації ризику; - Додаткове фінансування.
4. Поява конкурентів	0,2	0,5	- Стабільність та надійність; - Постійні та вигідні умови для клієнтів.
5. Ризик невиконання зобов'язань підрядниками	0,1	0,6	- Чітко визначені строки завершення робіт та прописати в договорі з підрядниками; - Систематичний контроль над виконанням.
6. Карантинні обмеження	0,8	0,2	- Вакцинувати працівників, дотримання карантинних норм.
7. Ускладнений порядок, тривалість проходження дозвільних (погоджувальних) процедур.	0,2	0,5	- Організація чіткого контролю за документами; - Розробка системи погоджень в середині проектної команди.
8. Політичний вплив.	0,3	0,5	- Не укладати договір з політичними інвесторами, компаніями.

ВИСНОВОК ДО РОЗДІЛУ 3.

У третьому розділі описано наступне:

Назва проєкту: Розробка архітектури мікросервісу для отримання даних по клієнту.

Суть проєкту: підвищити рівень підтримки, за допомогою сервісу клієнту зможуть отримувати швидко та водночас зручно необхідну для нього інформацію.

Цілі проєкту:

- Автоматизація роботи внутрішніх сервісів;
- Автоматизація роботи контакт центру;
- Створення сервісу для більш швидного та зручного отримання інформації клієнта;
- Впровадження сервісу у поточний функціональний процес;
- Ознайомлення співробітників відділу підтримки з функціоналом.

Продуктом є – сервіс для отримання інформації клієнта, що включає:

- Створення автоматизованої бази;
- Автоматизована взаємодія з іншими сервісами, що вже існують;
- Швидкий доступ клієнта до необхідної для нього інформації;
- Швидкий доступ співробітників до інформації.

Термін: планується виконати весь проект протягом 147 днів в термін з 05.09.2022 по 28.03.2023.

Загальний бюджет проєкту 120 000 000 грн (без ПДВ).

Джерелом бюджету є кошти виділені із загального бюджету, який планується щорічно. Загальна вартість проєкту становить 1 мільйон 102 тисячі гривень 890 гривень, залишок – 97 тисяч 110 гривень, розподіляється на резерв та

на непередбачувані витрати.

Команда проєкту складається зі співробітників IT департаменту. До команди входять:

- Менеджер проєкту;
- Архітектор проєкту;
- Системний аналітик;
- Back End розробник 1;
- Back End розробник 2;
- Fron End розробник 1;
- Full Stack розробник 1;
- Тестувальник 1;
- Тестувальник 2;
- DevOps інженер;

Було визначено зацікавлені сторони в проєкті:

1. Замовники проєкту – начальник бізнес відділу, начальник відділу підтримки.
2. 2Співробітники контакт-центру;
3. 3Директор IT департаменту;
4. Організаційні групи – внутрішні зацікавлені сторони, фокус група.
5. 5Менеджер проєкту.

Також, проаналізовані зовнішні та внутрішні ризи, їх йомвірність та вплив, та в результаті способи їх уникнення чи пом’якшення.

ВИСНОВОК

У роботі проводено аналіз та характеристику мікросервісної архитектури, плюси та мінуси цього підходу.

Проаналізовано найпопулярніші та зручні засоби для розробки мікросервісів, їх розгортання та розробки.

Проведено порівняння монолітної та мікросервісної архітектури, де були визначені наступні переваги другої, мікросервіси вирішують проблеми монолітної архітектури, використовуючи модульний підхід до розробки програмного забезпечення. Простіше кажучи, мікросервіси переосмислюють програми як комбінацію кількох окремих взаємопов'язаних служб. Кожна служба запускає спеціалізований процес і розгортається незалежно. За потреби служби можуть зберігати та обробляти дані за допомогою різних методів і можуть бути написані на інших мовах програмування.

Описана комунікація між мікросервісами за допомогою REST API.

Після проведення аналізу було визначено методи та засоби для розробки, розгортання, журнулювання та підтримки мікросервісної архітектури та в цілому - розробки проєкту.

Проведено аналіз ринку схожих програмних продуктів, розглянуто питання що таке клієнтська база, для чого вона, та чим важлива. Також які саме бувають клієнтські бази, це важливо, бо залежно від того, як компанія веде клієнтську базу, накопичує та аналізує інформацію.

Розписано та представлено що таке SWOT, коли саме його необзідно застосовувати та для визначення сильних та слабких сторін, а також ширші можливості та загрози проекту провнедено SWOT аналіз.

Проведено аналіз діяльності Креді Агріколь Банк.

Креді Агріколь Банк – найстаріший іноземний банк в Україні; працює на ринку з 1993 року та надає весь спектр банківських послуг, є стратегічним партнером агробізнесу та одним із лідерів ринку автокредитування. Банк входить

до міжнародної Credit Agricole Group (Франція) , європейського лідера банківського страхування та управління активами та основного партнера французької економіки. Описана його історія, цінності, взаємодія та підтримка клієнтів де можна побачити, що представлення даних для клієнту буде необхідною та зручною функціональністю як для співробітників так і для самих кліжнтів. Зображена загальна організаційна структура управління, де зображено внутрішню структуру, та комплаенс, що засвідчує про турботу про клієнтів та персональну інформацію.

Застосовано спосіб визначення ролей та обов'язків у групі за будь яким завданням, віхою або очікуваним результатом проекту . Дотримуючись принципів RACI, можна чітко розподіляти обов'язки та усувати плутанину, це зображено на матриці відповідальності(RACI).

Суть проєкту: підвищити рівень підтримки, за допомогою сервісу клієнту зможуть отримувати швидко та водночас зручно необхідну для нього інформацію.

Визначено головні цілі проєкту:

- Автоматизація роботи внутрішніх сервісів;
- Автоматизація роботи контакт центру;
- Створення сервісу для більш швидного та зручного отримання інформації клієнта;
- Впровадження сервісу у поточний функціональний процес;
- Ознайомлення співробітників відділу підтримки з функціоналом.

Продуктом є – сервіс для отримання інформації клієнта, що включає:

- Створення автоматизованої бази;
- Автоматизована взаємодія з іншими сервісами, що вже існують;
- Швидкий доступ клієнта до необхідної для нього інформації;
- Швидкий доступ співробітників до інформації.

Термін: планується виконати весь проект протягом 146 днів в термін з 05.09.2022 по 27.03.2023.

Загальний бюджет проєкту 120 000 000 грн (без ПДВ).

Джерелом бюджету є кошти виділені із загального бюджету, який планується щорічно. Загальна вартість проєкту становить 1 мільйон 102 тисячі гривень 890 гривень, залишок – 97 тисяч 110 гривень, розподіляється на резерв та на непередбачувані витрати.

Визначено команду проєкту складається зі співробітників IT департаменту. Визначені можливі обмеження проєкту, описані управління якістю, ризиками, ресурсами.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Бушуєв С. Д. Креативні технології управління проектами та програмами / С. Д. Бушуєв, Н. С. Бушуєва, І. А. Бабаєв [та ін.] – К. : «Саміт-Книга«, 2010. – 2-3 с.
2. Бушуєв С. Д. Методологія, методи і засоби проектного менеджменту. Практика проектного менеджменту «крок за кроком», методичні вказівки з питань занять: для студентів спеціальності «Проектний менеджмент». С.Д. Бушуєв.– КНУБА, 1999. – 34 с.
3. Кане М. М. Системи, методи та інструменти менеджменту якості / Кане М.М., Іванов Б.В., Корешков В.М., Стіртладзе А.Г. // – СПб.: Пітер. 2016. – 560 с.
4. Биковський, В. Управління інноваційними проектами та програмами: навч. посібник / В. Биковський, Є. Міщенко, В. Биковська [та ін.] – Тамбов: ГОУ ВПО ТДТУ, 2019. – 104 с.
5. Бушуєв С. Д. Словник-довідник з питань управління проектами / Бушуєв С. Д. – К.: Видавництво «Ділова Україна», 2001. – 640 с.
6. Богданов, В. Управління проектами. Корпоративна система – крок за кроком/В. Богданов. – М.: Манн, Іванов та Фербер, 2020. – 248 с.
7. Вайс. Дж. 5 стадій управління проектом: практичний посібник з планування та реалізації [Електронний ресурс] / Дж. Вайс, Р. Висоцькі. – Режим доступу: https://proglib.io/p/free-manager-books.
8. Матриці RACI [Електронний ресурс] - – Режим доступу: https://uk.economy-pedia.com/11035245-raci-matrix#
9. StudFiles Керування строками проєкту [Електронний ресурс] - – Режим доступу: https://studfile.net/preview/8893436/page:8/
10. ATLASSIAN Керування проєктами та agile [Електронний ресурс] - Режим доступу: https://www.atlassian.com/ru/agile/agile-at-scale/agile-iron-triangle
11. Studme Обмеження проєкту [Електронний ресурс] - Режим доступу: https://studme.org/184393/menedzhment/klyuchevye_ogranicheniya_proekta

12. Хабр Створення архітектури програми [Електронний ресурс] - Режим доступу: https://habr.com/ru/post/276593/

13. Офіційний сайт КРЕДИ АГРИКОЛЬ БАНКУ КОМПЛАЕНС [Електронний ресурс] - Режим доступу: https://credit-agricole.ua/ru/o-banke/komplaens

14. Офіційний сайт КРЕДИ АГРИКОЛЬ БАНКУ Акціонери [Електронний ресурс] - Режим доступу: https://credit-agricole.ua/ru/o-banke/akcioneri/akcioneri-ta-struktura-vlasnosti

15. Офіційний сайт КРЕДИ АГРИКОЛЬ БАНКУ Історія [Електронний ресурс] - Режим доступу: https://credit-agricole.ua/ru/o-banke/akcioneri/istoriya

16. Офіційний сайт КРЕДИ АГРИКОЛЬ БАНКУ Інформація про банк [Електронний ресурс] - Режим доступу: https://credit-agricole.ua/en/o-banke/pro-credit-agricole-ukraine/pro-credit-agricole-ukraine-2008

17. Діаграми та SWOT [Електронний ресурс] - Режим доступу: https://www.canva.com/design

18. Креді Агріколь Банк Вікіпедія [Електронний ресурс] - Режим доступу: https://uk.wikipedia.org/wiki/%D0%9A%D1%80%D0%B5%D0%B4%D1%96_%D0%90%D0%B3%D1%80%D1%96%D0%BA%D0%BE%D0%BB%D1%8C_%D0%91%D0%B0%D0%BD%D0%BA_(%D0%A3%D0%BA%D1%80%D0%B0%D1%97%D0%BD%D0%B0)

19. Medium The What, Why, and How of a Microservices Architecture [Електронний ресурс] - Режим доступу: https://medium.com/hashmapinc/the-what-why-and-how-of-a-microservices-architecture-4179579423a9

20. Хабр Що таке CI/CD [Електронний ресурс] - Режим доступу: https://habr.com/ru/company/otus/blog/515078/

21. InfoWorld What is CI/CD? Continuous integration and continuous delivery explained [Електронний ресурс] - Режим доступу: https://www.infoworld.com/article/3271126/what-is-cicd-continuous-integration-and-continuous-delivery-explained.html

22. Харрінгтон Дж. Досконалість управління змінами / Під наук. ред. В.В. Брагіна. – М.: РИА Стандарти та якість, 2018. – 192 с.

23. Metro Cash & Carry Ukraine – Ваш професійний партнер [Електронний ресурс] – Режим доступу: http://www.metro.ua/public/ua.

24. Оранжевий мармелад [Електронний ресурс] – Режим доступу: https://orangemarmelad.ru/kalkulyator-perescheta-ingredientov-pri-vypechke/.

25. Homebaked [Електронний ресурс] – Режим доступу: https://homebaked.ru/tortometr/.

26. Your-cake [Електронний ресурс] – Режим доступу: https://your-cake.ru/form-calculator.

27. Бушуєв С. Д. Методологія, методи і засоби проектного менеджменту. Практика проектного менеджменту «крок за кроком», методичні вказівки з питань занять: для студентів спеціальності «Проектний менеджмент». С.Д. Бушуєв.– КНУБА, 1999. – 34 с.

28. Переваги ієрархічної структури робіт (WBS) для менеджерів IT-проектів [Електронний ресурс] // habr.com. – Режим доступу: https://habr.com/ua/post/327872.

29. Кане М. М. Системи, методи та інструменти менеджменту якості / Кане М.М., Іванов Б.В., Корешков В.М., Стіртладзе А.Г. // – СПб.: Пітер. 2016. – 560 с.

30. Биковський, В. Управління інноваційними проектами та програмами: навч. посібник / В. Биковський, Є. Міщенко, В. Биковська [та ін.] – Тамбов: ГОУ ВПО ТДТУ, 2019. – 104 с.

ДОДАТКИ

Додаток 1

Проєкт в MS Project

	Режи задач	Название задачи	Длительнос	Начало	Окончание	Предц	Названия ресурсов	Затраты
1		**Проект**	**146 дней**	**Пн 05.09.22**	**Пн 27.03.23**			**1 102 890,00 ₴**
2		Початок	0 дней	Пн 05.09.22	Пн 05.09.22			0,00 ₴
3		**Аналіз ринку**	**12 дней**	**Вт 06.09.22**	**Чт 22.09.22**			**21 010,00 ₴**
4		Визначення доцільності реалізації	1 день	Вт 06.09.22	Ср 07.09.22	2	системний аналітик	800,00 ₴
5		Аналіз схожих рішень	3 дней	Ср 07.09.22	Пн 12.09.22	4	менеджер проєкт	6 240,00 ₴
6		Розробка архітектури проєкту	3 дней	Вт 13.09.22	Пт 16.09.22	5	системний аналітик;архітект	8 850,00 ₴
7		Узгодження архітектури	2 дней	Вт 20.09.22	Чт 22.09.22	6	архітектор проєкту;менедже	5 120,00 ₴
8		**Проєктування сервісу**	**39 дней**	**Чт 22.09.22**	**Ср 16.11.22**			**903 200,00 ₴**
9		Обговорення проєкту	2 дней	Чт 22.09.22	Пн 26.09.22	7	dev ops інженер ;	8 640,00 ₴
10		Визначення цілей	2 дней	Пн 26.09.22	Ср 28.09.22	9	dev ops інженер ;	8 640,00 ₴
11		Планування розробки проєкту	3 дней	Ср 28.09.22	Пн 03.10.22	10	dev ops інженер ;архітектор	6 720,00 ₴
12		Визначення обмежень	3 дней	Пн 03.10.22	Чт 06.10.22	11	менеджер проєкт	10 080,00 ₴
13		Узгодження обмежень	2 дней	Чт 06.10.22	Пн 10.10.22	12	архітектор проєкту;менедже	6 720,00 ₴
14		Затвердження вимог до сервісу	2 дней	Пн 10.10.22	Ср 12.10.22	13	архітектор проєкту;менедже	5 120,00 ₴
15		Розробка ТЗ	5 дней	Ср 12.10.22	Ср 19.10.22	14	dev ops інженер ;	15 200,00 ₴
16		Визначення відповідальних	1 день	Ср 19.10.22	Чт 20.10.22	15	архітектор проєкту;менедже	2 560,00 ₴
17		Затвердження ТЗ	2 дней	Чт 20.10.22	Пн 24.10.22	16	архітектор проєкт	5 120,00 ₴
18		Облаштування	7 дней	Пн 24.10.22	Ср 02.11.22	17	монтажники;нові	637 480,00 ₴

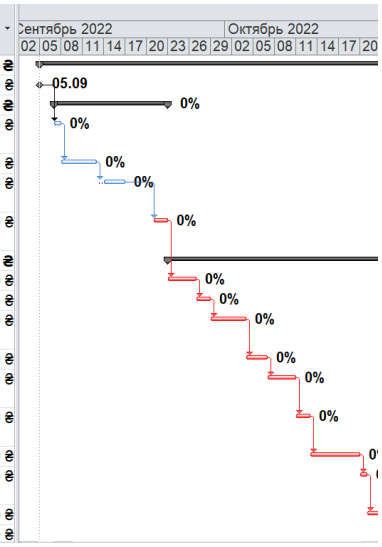


Рис 1 Додаток 1.1

	Режи задач	Название задачи	Длительнос	Начало	Окончание	Предц	Названия ресурсов	Затраты
17		Затвердження ТЗ	2 дней	Чт 20.10.22	Пн 24.10.22	16	архітектор проєкт	5 120,00 ₴
18		Облаштування комп'ютерів, серверів	7 дней	Пн 24.10.22	Ср 02.11.22	17	монтажники;нові ПК[9];сервери[2]	637 480,00 ₴
19		Встановлення ліцензійного ПЗ	5 дней	Ср 02.11.22	Ср 09.11.22	18	монтажники;нові ПК[1];сервери[1]	190 200,00 ₴
20		Перевірка роботи налаштування ПК та серверів	3 дней	Ср 09.11.22	Пн 14.11.22	19	монтажники;dev ops інженер	4 800,00 ₴
21		Підготовка перед розробкою	2 дней	Пн 14.11.22	Ср 16.11.22	20	dev ops інженер	1 920,00 ₴
22		**Розробка**	**70 дней**	**Ср 16.11.22**	**Ср 22.02.23**			**138 200,00 ₴**
23		Розробка функціоналу back end розробниками	17 дней	Ср 16.11.22	Пт 09.12.22	21	back end розробник 1;back end	34 680,00 ₴
24		Розробка функціоналу front end розробниками	14 дней	Пт 09.12.22	Чт 29.12.22	23	front end розробник;full stack розробник	23 520,00 ₴
25		Написання тестів	7 дней	Чт 29.12.22	Пн 09.01.23	24	тестувальник 1	5 600,00 ₴
26		Написання документації	3 дней	Пн 09.01.23	Чт 12.01.23	25	back end розробник	5 640,00 ₴
27		Тестування функціоналу	5 дней	Чт 12.01.23	Чт 19.01.23	26	back end розробник 1;full	18 000,00 ₴
28		Тестування сервісу	6 дней	Чт 19.01.23	Пт 27.01.23	27	тестувальник 1;т	10 080,00 ₴
29		Виправлення багів	7 дней	Пт 27.01.23	Вт 07.02.23	28	back end розробн	20 440,00 ₴
30		Загальне тестування	9 дней	Вт 07.02.23	Пн 20.02.23	29	тестувальник 1;т	15 120,00 ₴

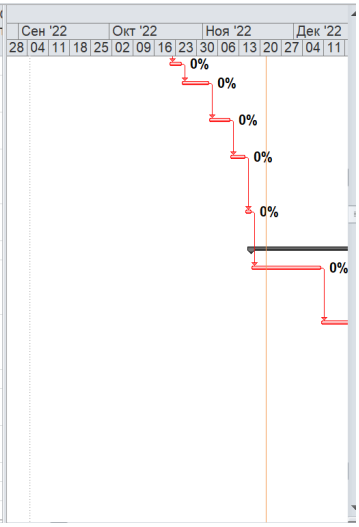
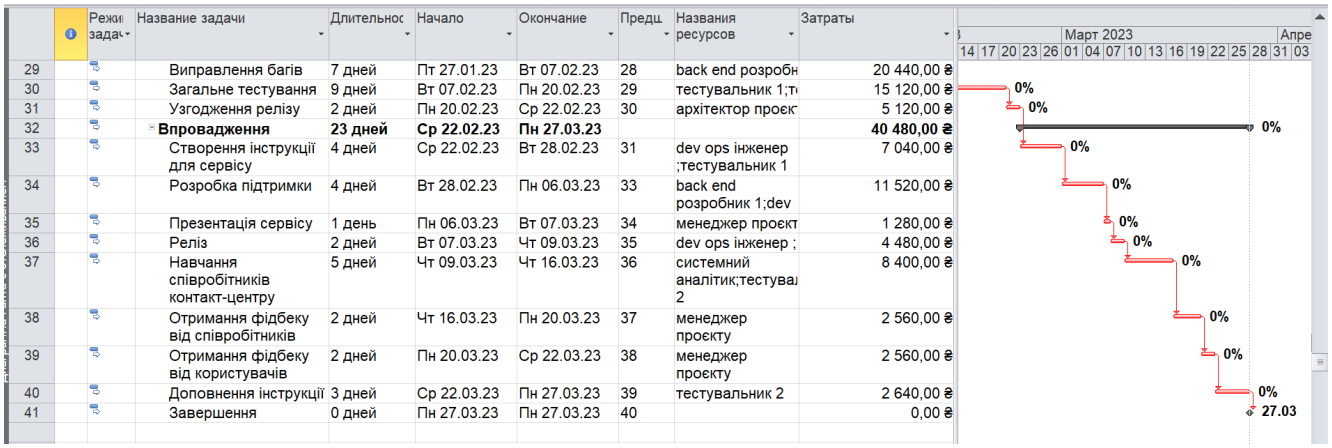


Рис 2 Додаток 1.2



	Режи задач	Название задачи	Длительнос	Начало	Окончание	Предш	Названия ресурсов	Затраты
29		Виправлення багів	7 дней	Пт 27.01.23	Вт 07.02.23	28	back end розробн	20 440,00 ₴
30		Загальне тестування	9 дней	Вт 07.02.23	Пн 20.02.23	29	тестувальник 1;т	15 120,00 ₴
31		Узгодження релізу	2 дней	Пн 20.02.23	Ср 22.02.23	30	архітектор проєкт	5 120,00 ₴
32		**Впровадження**	**23 дней**	**Ср 22.02.23**	**Пн 27.03.23**			**40 480,00 ₴**
33		Створення інструкції для сервісу	4 дней	Ср 22.02.23	Вт 28.02.23	31	dev ops інженер ;тестувальник 1	7 040,00 ₴
34		Розробка підтримки	4 дней	Вт 28.02.23	Пн 06.03.23	33	back end розробник 1;dev	11 520,00 ₴
35		Презентація сервісу	1 день	Пн 06.03.23	Вт 07.03.23	34	менеджер проєкт	1 280,00 ₴
36		Реліз	2 дней	Вт 07.03.23	Чт 09.03.23	35	dev ops інженер ;	4 480,00 ₴
37		Навчання співробітників контакт-центру	5 дней	Чт 09.03.23	Чт 16.03.23	36	системний аналітик;тестува 2	8 400,00 ₴
38		Отримання фідбеку від співробітників	2 дней	Чт 16.03.23	Пн 20.03.23	37	менеджер проєкту	2 560,00 ₴
39		Отримання фідбеку від користувачів	2 дней	Пн 20.03.23	Ср 22.03.23	38	менеджер проєкту	2 560,00 ₴
40		Доповнення інструкції	3 дней	Ср 22.03.23	Пн 27.03.23	39	тестувальник 2	2 640,00 ₴
41		Завершення	0 дней	Пн 27.03.23	Пн 27.03.23	40		0,00 ₴

Рис 2 Додаток 1.3

Додаток 2

Лист ресурсів в MS Project

	Название ресурса	Тип	Единицы измерения материалов	Краткое название	Группа	Макс. единиц	Стандартная ставка	Ставка сверхурочных
1	менеджер проєкту	Трудовой		м		100%	160,00 ₴/час	0,00 ₴/час
2	системний аналітик	Трудовой		с		100%	100,00 ₴/час	0,00 ₴/час
3	back end розробник 1	Трудовой		b		100%	130,00 ₴/час	0,00 ₴/час
4	back end розробник 2	Трудовой		b		100%	125,00 ₴/час	0,00 ₴/час
5	тестувальник 1	Трудовой		т		100%	100,00 ₴/час	0,00 ₴/час
6	тестувальник 2	Трудовой		т		100%	110,00 ₴/час	0,00 ₴/час
7	нові ПК	Материальный		н			37 000,00 ₴	
8	монтажники	Трудовой		м		100%	80,00 ₴/час	0,00 ₴/час
9	front end розробник	Трудовой		f		100%	100,00 ₴/час	0,00 ₴/час
10	full stack розробник	Трудовой		f		100%	110,00 ₴/час	0,00 ₴/час
11	dev ops інженер	Трудовой		d		100%	120,00 ₴/час	0,00 ₴/час
12	архітектор проєкту	Трудовой		а		100%	160,00 ₴/час	0,00 ₴/час
13	сервери	Материальный		с			150 000,00 ₴	

Рис 1 Додаток 2.1

	Название ресурса	Тип	Единицы измерения материалов	Краткое название	Группа	Макс. единиц	Стандартная ставка	Ставка сверхурочнь	Затраты на использ.	Начисление	Базовый календарь
1	Менеджер проекту	Трудовой		М		100%	160,00 ₴/час	0,00 ₴/час	0,00 ₴	Пропорциональн	Стандартный
2	Системний аналітик	Трудовой		С		100%	110,00 ₴/час	0,00 ₴/час	0,00 ₴	Пропорциональн	Стандартный
3	Спеціаліст з розвитку т	Трудовой		С		100%	100,00 ₴/час	0,00 ₴/час	0,00 ₴	Пропорциональн	Стандартный
4	Системний адміністрат	Трудовой		С		100%	100,00 ₴/час	0,00 ₴/час	0,00 ₴	Пропорциональн	Стандартный
5	Програміст 1	Трудовой		П		100%	130,00 ₴/час	0,00 ₴/час	0,00 ₴	Пропорциональн	Стандартный
6	Програміст 2	Трудовой		П		100%	130,00 ₴/час	0,00 ₴/час	0,00 ₴	Пропорциональн	Стандартный
7	UX/UI спеціаліст	Трудовой		U		100%	120,00 ₴/час	0,00 ₴/час	0,00 ₴	Пропорциональн	Стандартный
8	Тестувальник	Трудовой		Т		100%	100,00 ₴/час	0,00 ₴/час	0,00 ₴	Пропорциональн	Стандартный
9	Контент-менеджер	Трудовой		К		100%	90,00 ₴/час	0,00 ₴/час	0,00 ₴	Пропорциональн	Стандартный
10	Технічний письменник	Трудовой		Т		100%	80,00 ₴/час	0,00 ₴/час	0,00 ₴	Пропорциональн	Стандартный
11	Ноутбук HP	Материальны	шт.	Н			23 000,00 ₴		0,00 ₴	Пропорциональн	
12	Ліцензійне ПЗ	Материальны	од.	Л			5 560,00 ₴		0,00 ₴	Пропорциональн	
13	Хостинг	Материальны	од.	Х			2 400,00 ₴		0,00 ₴	Пропорциональн	
14	Домен	Материальны	од.	Д			1 500,00 ₴		0,00 ₴	Пропорциональн	
15	Інтернет-з'єднання	Материальны	од.	І			10 000,00 ₴		0,00 ₴	Пропорциональн	
16	Вище керівництво	Трудовой		В		100%	0,00 ₴/час	0,00 ₴/час	0,00 ₴	Пропорциональн	Стандартный
17	Бухгалтер	Трудовой		Б		100%	0,00 ₴/час	0,00 ₴/час	0,00 ₴	Пропорциональн	Стандартный

Рис. Д2.1

Додаток 3

Презентація роботи в MS PowerPoint

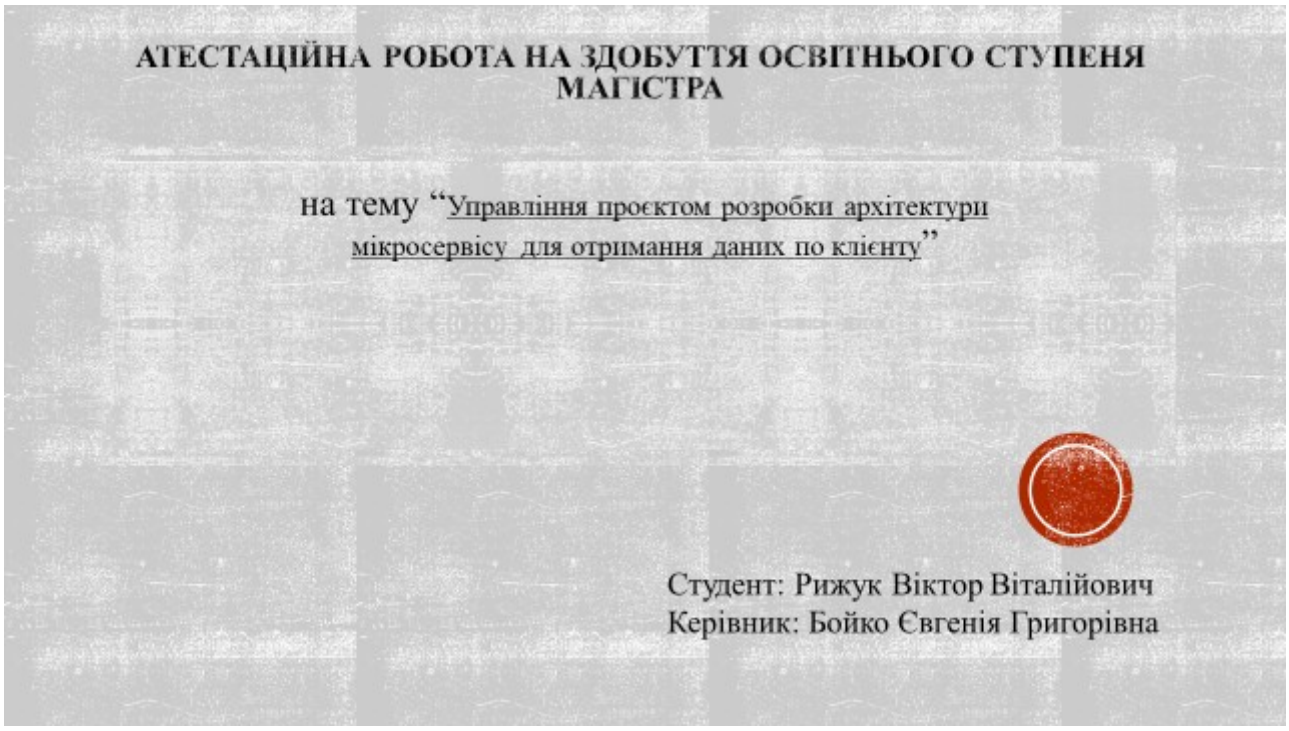


Рис. Д3.1

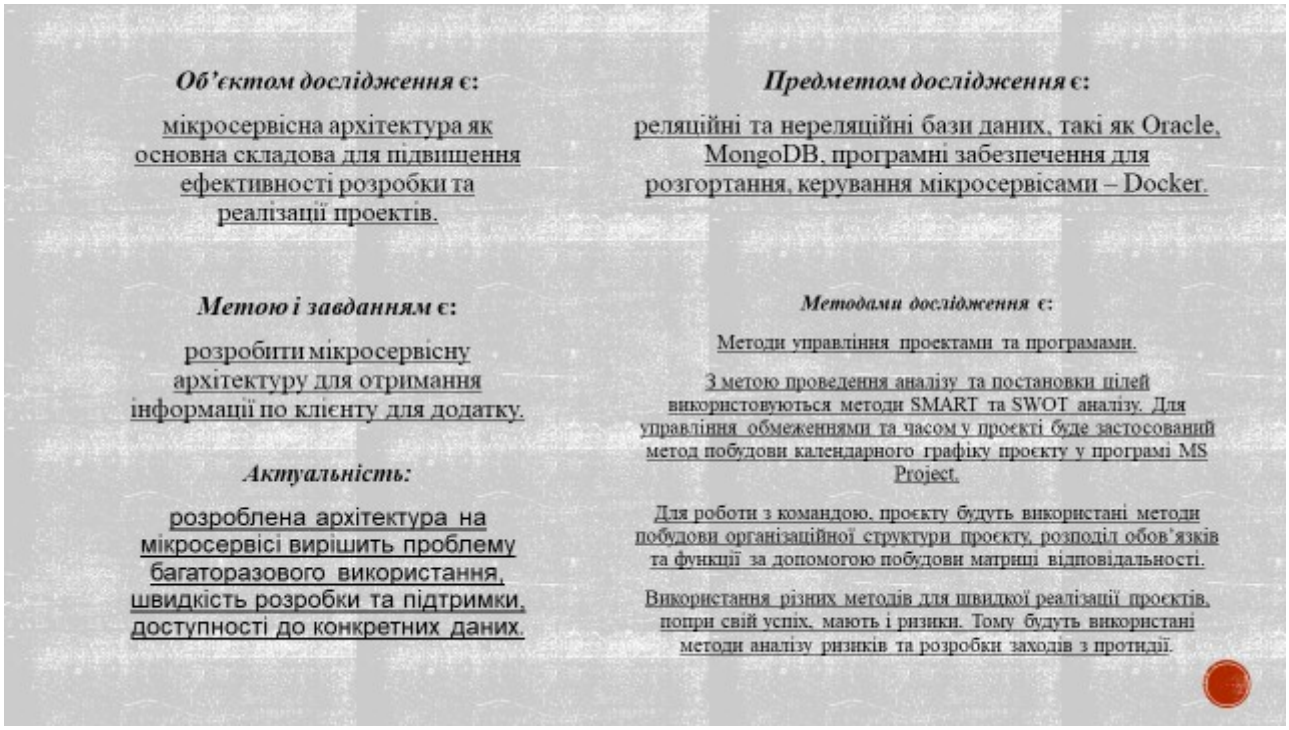


Рис. Д3.2

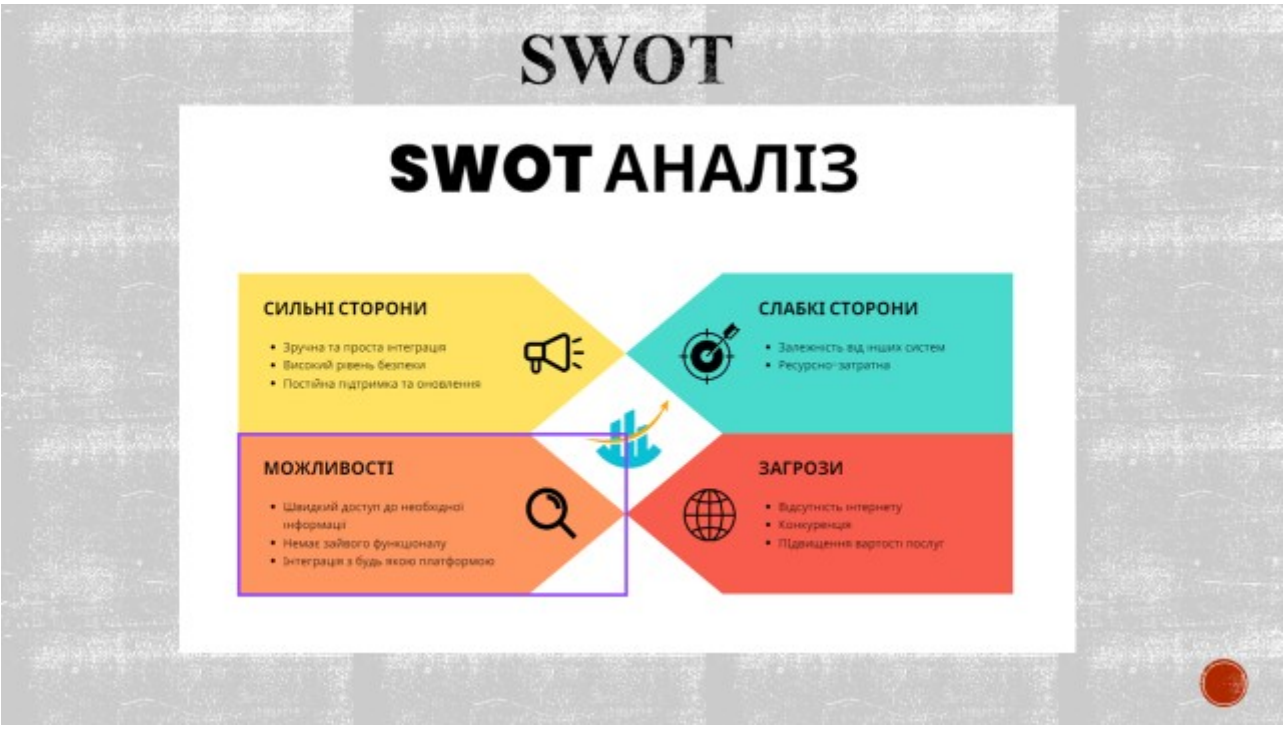


Рис. Д3.3

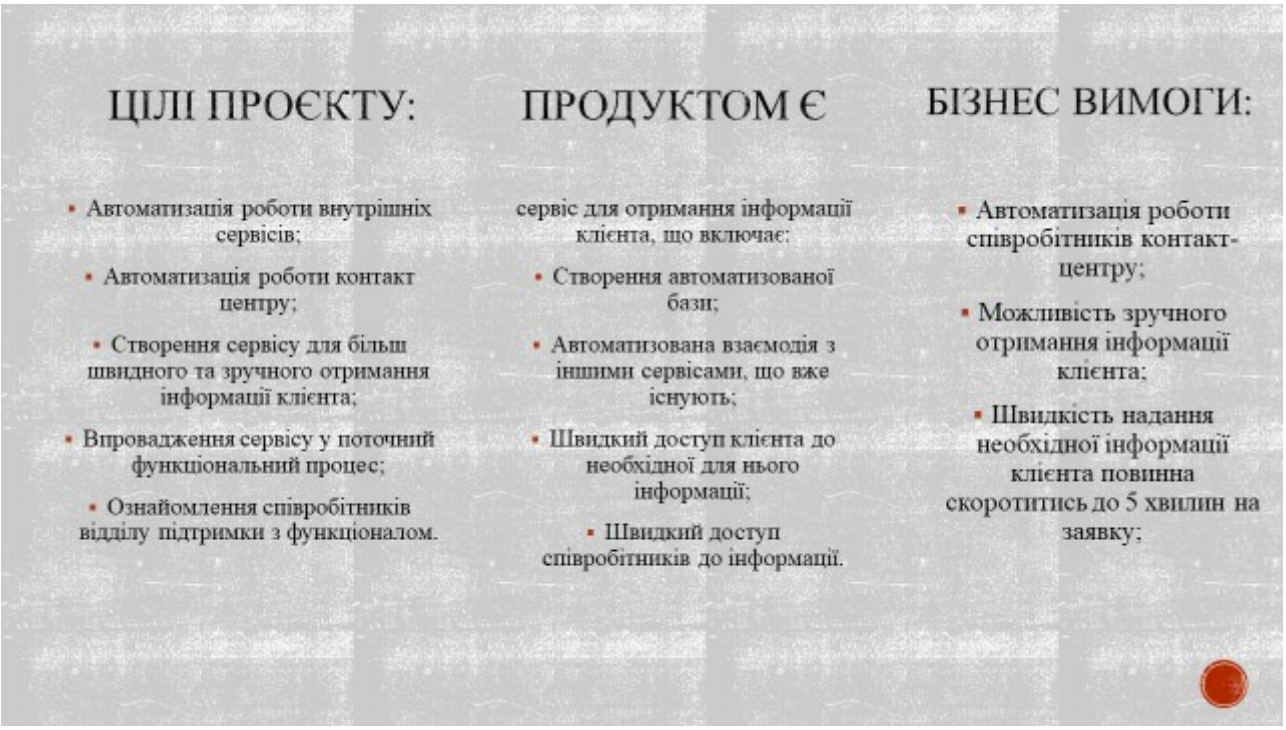


Рис. Д3.4

Рис. Д3.5

Рис. Д3.6

Рис. Д3.7

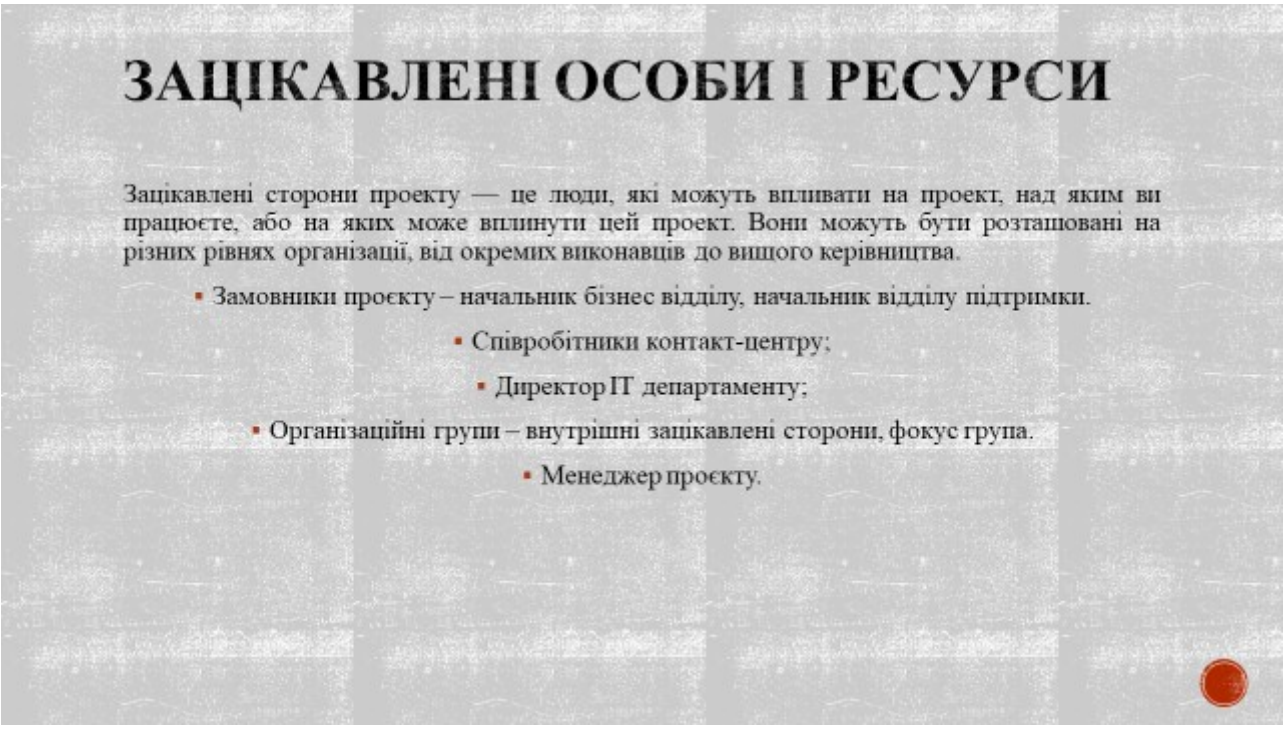


Рис. Д3.8

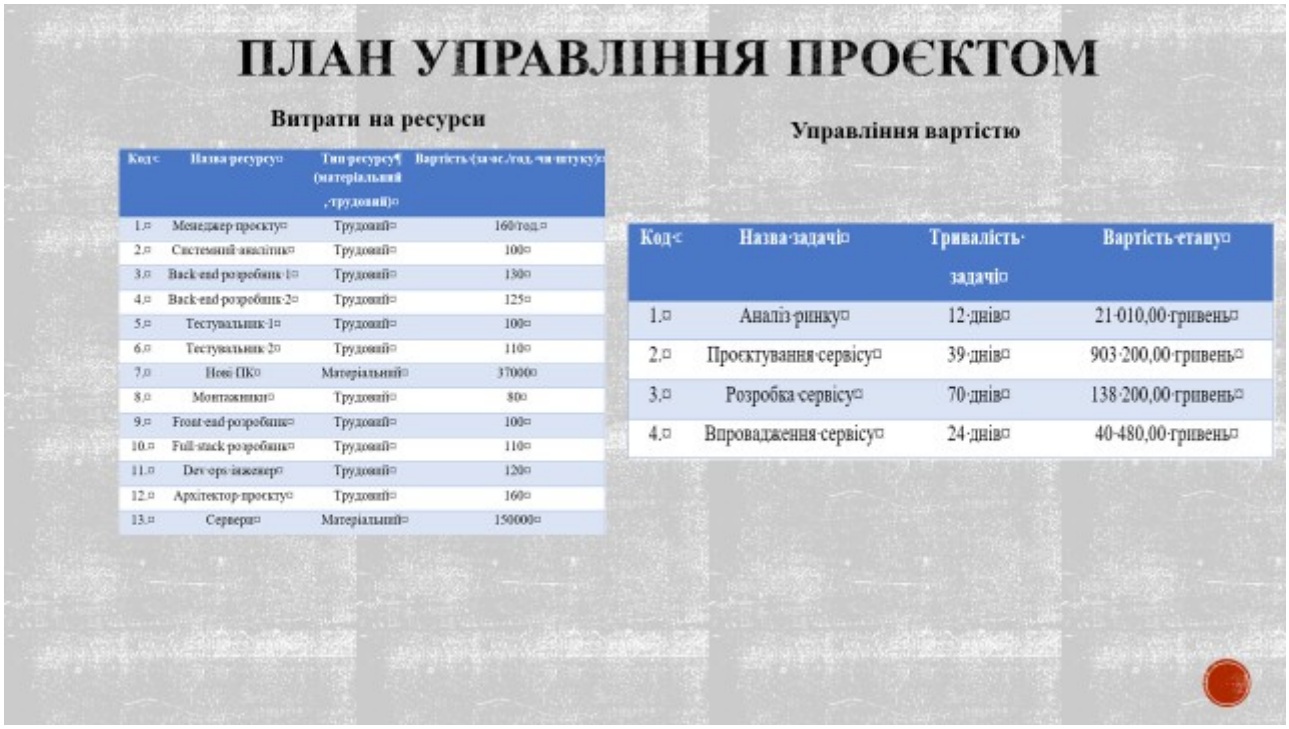


Рис. Д3.9

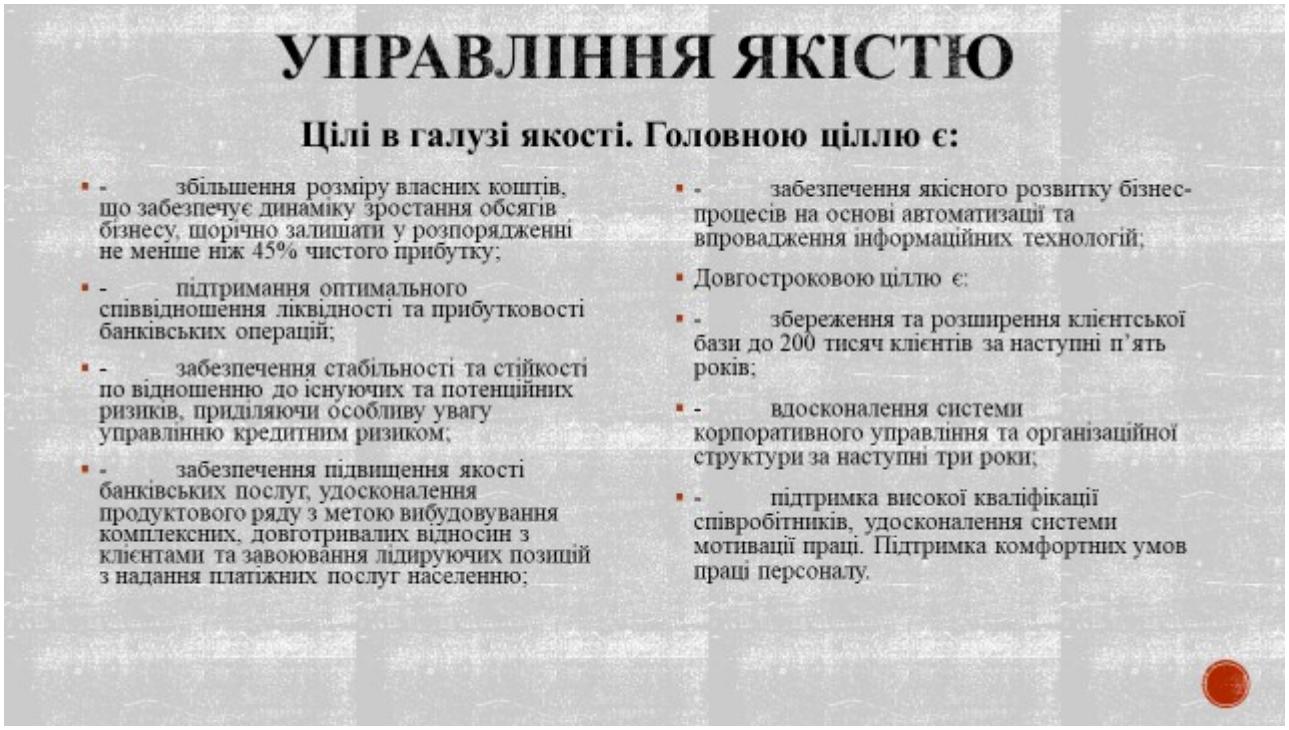


Рис. Д3.10

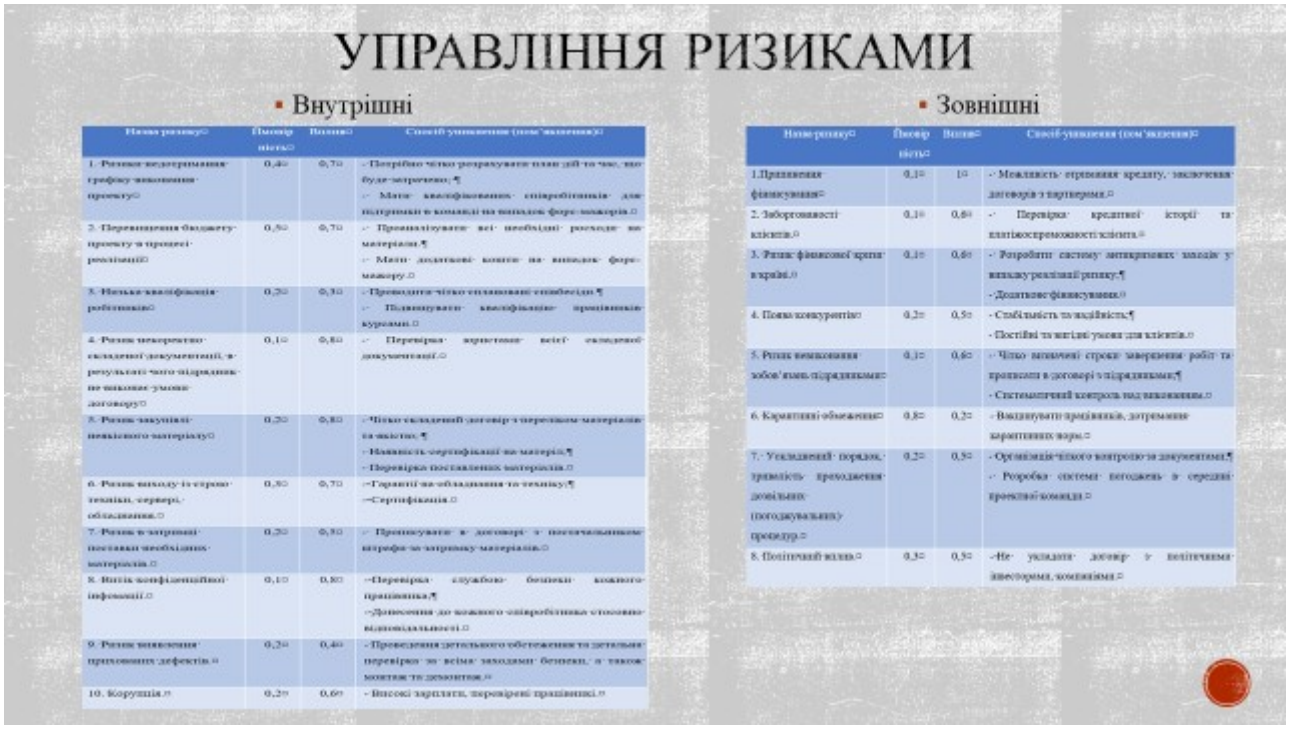


Рис. Д3.11

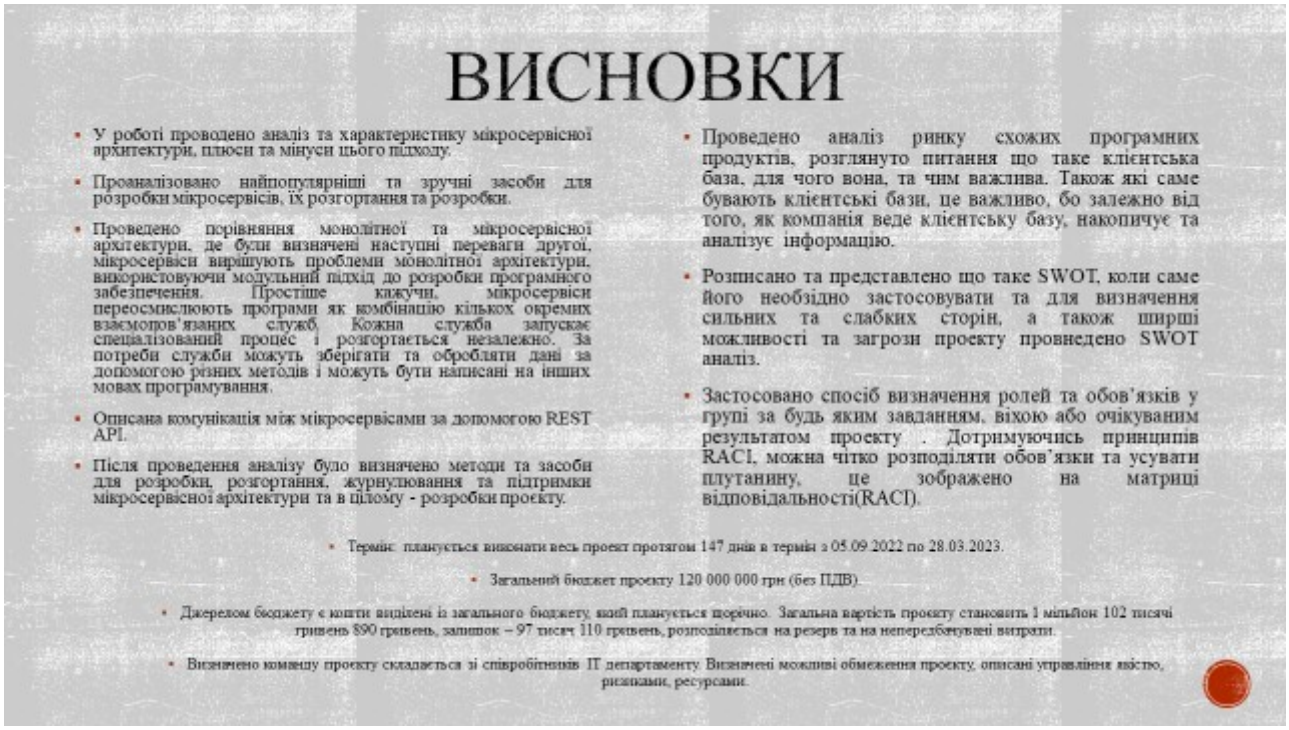


Рис. Д3.12

Рис. Д3.13