Вступ

Сьогодні навколо нас розвивається новий інформаційний світ - світ, який дозволяє нам переміщатися, взаємодіяти і вести бізнес набагато швидше, ніж будь-коли раніше. В цьому світі, в якому до Інтернету може підключитися практично будь-який людина і безліч пристроїв, ми спостерігаємо вибухове зростання кількості технічних і інформаційних ресурсів і підключених до мережі об'єктів, що має суттєвий вплив на нашу повсякденне життя.

Хмарні провайдери надають можливість користувачам зберігати, обробляти та отримувати доступ до власних даних і програм з будь-якого місця та пристрою за наявності доступу до інтернету, а бізнесу сильний інструмент для вирішення залежності від потужності локальних обчислювальних ресурсів, вирішуючи разом з цим проблему масштабованості та економічних втрат на придбання та обслуговування власних залів з серверами, підтримку інфраструктури, при цьому отримуючи високу продуктивність, гнучкість і можливість надійно зберігати дані у різних куточках світу.

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. ТЕОРЕТИЧНИЙ АНАЛІЗ СУЧАСНИХ МЕТОДОЛОГІЙ УПРАВЛІННЯ ПРОЄКТАМИ ТА ХМАРИНИХ ТЕХНОЛОГІЙ

1.1 Проєкт, визначення поняття та основні особливості

Проєкт — це унікальний тимчасовий набір дій, спрямований на досягнення запланованих цілей, які можна визначити в термінах результатах, або вигодах. Будь-який проєкт є тимчасовим, оскільки він має визначені терміни початку і кінця, і відповідно, визначений обсяг і ресурси. Також проєкт унікальний тим, що це не рутинна операція, а певний набір дій, призначених для досягнення окремої мети.

Управління проєктом — це застосування процесів, методів, навичок, знань і досвіду для досягнення конкретних цілей відповідно до критеріїв прийнятності в межах узгодженого терміну та бюджету, Управління проєктом має кінцеві результати, які обмежені кінцевим часом і бюджетом. Планування та організація ресурсів, організації з метою виконання конкретного завдання чи події. Управління проєктом зазвичай передбачає одноразовий проєкт, а не поточну діяльність, а керовані ресурси включають як людський, так і фінансовий капітал.[1]

Багато людей не розуміють що таке проєкт і часто плутають його з операціями. Це пов'язано з тим, що вони не володіють перерахованими вище якостями.

Як показано в наведеній нижче таблиці (див. табл. 1.1), операції, як правило, є поточними підприємствами без обмежених часових рамок і не є унікальними через їх рутинний характер.

Таблиця 1.1

Відмінності між проєктами та операційною діяльністю

Проєкти	**Операційна діяльність**
1	**2**
Тимчасові (обмежені у часі)	Постійні
Забезпечує унікальний результат	Має постійний результат
Має інноваційний характер	Має повторювальний характер
Існує перед продуктом	Існує після продукту
Трансформаційний	Покращує ефективність звичайної практики

Управління проєктами часто тісно пов'язане з інженерними проєктами, які, як правило, мають повний набір компонентів, які потрібно завершити та зібрати певним чином, щоб створити функціонуючий продукт. Керівники проєктів використовують візуальні представлення робочого процесу, наприклад різні діаграми, щоб визначити, які завдання мають бути виконані якими відділами.

Ключ до успішного управління полягає в зосередженні на:

- *Людях.* Найважливіший елемент успішного проєкту;
- *Продукті*. Проєктний продукт, який буде створено;
- *Процесі.* Набір вимог і завдань для людей для виконання роботи;
- *Проєкті*. Всі роботи, необхідні для втілення продукту в реальність.

Синергія всіх чотирьох аспектів разом дає змогу успішно керувати проєктами. [2]

1.1.1 Життєвий цикл проєкту

Існує багато фазових моделей, які можна використовувати для поділу проєкту на різні частини. Оскільки різноманітність проєктів є дуже великою, від простих до дуже складних, зазвичай виділяють чотири основні та одну додаткову фазу. У таблиці (див. табл. 1.2) подано загальний теоретичний огляд фаз, а також завдань, для яких слід модифікувати та адаптувати кожен проєкт.

Таблиця 1.2

Фази виконання проєкту

Етап	**Завдання етапу**
1	**2**
1.Ініціація	1. Ситуативний аналіз; 2. Визначення зацікавлених сторін; 3. Формування «візії» проєкту; 4. Визначення вимог до проєкту; 5. Визначення ризиків та створення ризикової стратегії; 6. Формування SMART цілей;
2.Планування	1.Фінальне формування списку зацікавлених сторін; 2. Формування головних та другорядних цілей; 3.Створення WBS структури ; 4. Створення календарного плану ; 5. Підрахунок необхідних ресурсів та витрат; 6. Створення OBS структури; 7.Візуалізація плану проєкту.
3. Виконання	1. Створення команди проєкту;

	2.Фінальне погодження з зацікавленими сторонами; 3. Комунікативний менеджмент; 4. Вирішення конфліктів

Продовження табл.1.2

1	2
4.Моніторинг	1.Контроль за виконанням робіт по проєкту;
5. Завершення	1. Перевірка відповідності результату до вимог; 2. Підсумковий звіт про результати; 3. Закриття проєкту 4. Оцінка того, які результати можна використати в наступних проєктах

Фаза ініціації проєкту

Фаза ініціації проєкту знаменує собою початок проєкту. Як правило, проєкт буде розпочато у відповідь на можливість, яку потрібно дослідити, або проблему, яку потрібно вирішити. На самому початку потрібно провести аналіз витрат і вигод.

Частина аналізу витрат і вигод включає проведення техніко-економічного обґрунтування, визначення обсягу проєкту, визначення кінцевих результатів проєкту та зацікавлених сторін, залучених до створення бізнес-обґрунтування.

На цьому етапі потрібно розробити статут проєкту стає найважливішим документом, оскільки в ньому зазначена головна інформаціє, що відноситься до проєкту. Після того, як усі ці деталі перевірено та проєкт затверджено, проєкт офіційно стартує, проєктні групи збираються, і починається планування.

Фаза планування проєкту

Під час планування проєкту керівнику проєкту важливо розуміти вимоги та цілі проєкту. Етап планування є найважливішим для будь-якого проєкту, оскільки планування впливає на ризики та результати проєкту. На етапі планування розробляється план проєкту, щоб надати всім зацікавленим сторонам дорожню карту проєкту. У ньому описано всі дії, завдання, часові рамки, ролі, витрати, етапи, результати та інші залежності, необхідні для успішного виконання проєкту.

План проєкту має вирішальне значення на етапах виконання, моніторингу та закриття проєкту, оскільки в ньому деталізуються не лише цілі та завдання проєкту, але й «як» і «хто що робить» під час реалізації.

Етап виконання проєкту

Планування та виконання проєкту є двома основними етапами досягнення цілей проєкту. Етап виконання, як правило, найдовший і потребує найбільшого розподілу ресурсів у міру фактичного впровадження проєкту. На цьому етапі контроль над ресурсами проєкту, моніторинг прогресу проєкту та підтримка чіткої комунікації між усіма зацікавленими сторонами стають вирішальними.

Команда проєкту використовує WBS і календарний графік проєкту для виконання завдань, викладених у плані проєкту. Крім того, проводяться часті зустрічі команди, щоб звітувати про хід проєкту, оцінювати відхилення в проєкті, а також розглядати запити на зміни та оновлювати план проєкту, якщо такі є. Керівник проєкту гарантує, що він тримає всіх зацікавлених сторін в курсі прогресу проєкту за допомогою звітів про стан. Спілкування має відповідати плану спілкування. Після того, як готові результати, кінцевий продукт доставлений і прийнятий замовником відповідно до критеріїв прийнятності, проєкт готовий до закриття.

Етап моніторингу та контролю

Хоча моніторинг і контроль призначені для перевірки всього процесу управління проєктом, це зручніше на етапі виконання. Моніторинг і контроль здійснюються, щоб переконатися, що проєкт рухається в правильному напрямку

та в межах визначеного обсягу. Коли проєкт просувається за планом, ризик мінімізується.

В ідеалі моніторинг фактичної продуктивності проєкту порівнюється із запланованою продуктивністю та відповідним планом дій, вжитих у разі виникнення розбіжностей.

Фаза закриття

Проєкт закривається після того, як він досяг поставлених цілей і продукт готовий до випуску та доставки клієнту. Ця остання фаза також відома як наступна фаза, коли керівник проєкту та команди збираються разом, щоб обговорити події та ідеї проєкту на заключній зустрічі. Вони підведуть підсумки всього життєвого циклу, щоб витягти з нього уроки та висновки, визначити сильні сторони та можливості для вдосконалення та задокументувати їх разом з іншими проєктними даними для подальшого використання.

Іноді проєкт закривається до завершення, в основному через невдачу.

1.1.2 Типи проєктів в проєктному менеджменті

Хоча багато проєктів мають однакові характеристики, вони ніколи не бувають однаковими. Таким чином, вони поділяються на різні категорії на основі різних факторів. Важливо класифікувати проєкти, оскільки це допомагає організації висвітлити їх особливості та знайти найбільш прийнятний підхід до їх виконання.

Типи проєктів за джерелом фінансування:

- *Громадські проєкти*, джерелом фінансування яких є уряд або державні установи;
- *Приватні проєкти*, які приватно фінансуються бізнесом або венчурним капіталом;

- *Гібридні проєкти*, джерело фінансування яких є як приватним, так і державним.

Типи проєктів за змістом проєкту:

- *Будівельні проєкти* – це ті, результатом яких є артефакт, наприклад, проєкт розробки IT-системи.

- *Бізнес-проєкти впровадження*, спрямовані на запровадження нової/покращеної функції або зміни в бізнес-системах чи процесах.

- *Дослідницькі проєкти* виконуються з метою пошуку знань або розуміння для прийняття рішень.

- *Проєкти закупівель*, які встановлюють відносини B2B для пошуку продуктів та/або послуг.

Види проєктів за часом їх реалізації :

- *Звичайні проєкти* – це проєкти, для виконання яких відведено достатній час; як такі, вони пройдуть через життєвий цикл проєкту, щоб отримати очікувану якість результату.

- *Проєкти стихійних лих* — це імпровізовані проєкти, які передбачають дуже великі капіталовкладення з мінімальним часом виконання.

- *Аварійні проєкти* – це проєкти, виконання яких потребує додаткових витрат протягом короткого періоду часу, а фази проєкту, як правило, збігаються.

1.1.3 Важливість проєктного менеджменту в діяльності організацій

Управління певним проєктом дозволяє компанії стати більш конкурентоспроможною на своєму ринку. Для управління кожним окремим проєктом може використовуватись спеціальне програмне забезпечення яке може ніде більше не застосовуватись , при цьому кожен проєкт має унікальні вимоги.

Проєктні менеджери керують усіма аспектами проєкту, від планування до введення корегувальних змін, а також роблять все, щоб все було зроблено вчасно відповідно до календарного плану. Вони знають, як зробити так, щоб зробити все якісно, при цьому заощаджуючи кошти компанії. Проєктні менеджери розробляють стратегію, яка дозволяє їм організовувати та керувати співробітниками, дотримуватись термінів і доводити проєкт до завершення. Від них очікують вирішення будь-яких проблем, що виникають під час виконання проєкту, дозволяючи керівникам організацій займатися іншими аспектами бізнесу.

Загальне завдання або мета управління проєктами полягає в тому, щоб планувати та реалізовувати в життя інновації, які допомагають вирішувати виклики – незалежно від того, що це за проєкт. Проєктний менеджмент щороку стає дедалі необхіднішим у нашому складному світі. Отже, можемо зробити висновки, що управління проєктами буде невід'ємною частиною процесів розвитку бізнесу в майбутньому.

1.1.4 Ролі та зобов'язання в проєктах

Успішні менеджери проєктів повинні вміти адаптовуватись то тих чи інших обставин, які можуть виникнути в ході реалізації. Зацікавленими сторонами проєкту можуть бути люди різних культур, віку, та з певними професійними особливостями. Гарному управлінцю потрібно завжди знаходити взаєморозуміння з такими людьми та уникати конфліктних ситуацій, які можуть серйозно вплинути на якість.

Для успішного проєкту також дуже важлива готовність до роботи і відкритість усіх зацікавлених сторін. В іншому випадку це призводить до нових непорозумінь і конфліктів. На фазі ініціації потрібно провести розподіл ролей та зобов'язань, ці зобов'язання повинні бути донесені та прийняті усіма зацікавленими сторонами.

1.2 Методології з управління проєктами

Теорія та практика управління проєктами надали нам велику кількість практично застосовних методологій для ефективного управління проєктом. Методологія управління проєктом являє собою набір методів, технік, процедур, найкращих практик тощо, які використовуються в проєкті. Зазвичай вони базується на конкретному підході до управління проєктом, який визначає набір принципів і вказівок, які визначають спосіб управління проєктом. Найвідомішими методологіями управління проєктами є: PMI, IPMA, PRINCE2, Agile тощо. Це в основному методології процесів і вони включають певні під процеси або фази в управлінні проєктом. Основна проблема цих методологій полягає в тому, що вони зазвичай пропонуються незалежно від типу проєкту. Проте очевидно, що відмінності між структурами окремих груп проєктів досить великі; отже, методи управління цими проєктами також мають відрізнятися. Залежно від особливостей і типу проєкту, філософії управління та культури організації, яка керує проєктом, слід обрати певну методологію. Це означає необхідність визначення конкретних методологій для деяких груп подібних проєктів.

Теорія і практика управління проєктами пропонують і застосовують ряд методологій з більшим чи меншим успіхом. Ці методології в основному пропонуються національними або міжнародними асоціаціями управління проєктами, а також певними організаціями та установами.

1.2.1 Методологія PMI

Методологія PMI була розроблена великою та потужною асоціацією керівників проєктів – Project Management Institute (PMI) із США та представлена в довіднику Project Management Body of Knowledge – PMBoK. Це методологія процесу, яка пропонує керувати проєктом за областями знань:

- *управління інтеграцією проєкту*, яке включає процеси та дії для ідентифікації, об’єднання, уніфікації та координації різних процесів і дій з управління проєктом у групах процесів управління проєктом;

- *управління обсягом проєкту* – включає процеси, необхідні для забезпечення того, щоб проєкт включав усю роботу, необхідну для успішного завершення проєкту;

- *управління часом* – включає процеси, пов’язані з визначенням часових аспектів проєктів і процесів, необхідних для управління своєчасним завершенням проєкту;

- *управління витратами* – процеси, пов’язані з плануванням вартості проєкту, бюджетуванням, фінансуванням, контролінгом для завершення проєкту в межах затвердженого бюджет;

- *управління якістю* – включає процеси, які дозволяють завершити проєкт відповідно до необхідної якості та задовольнити потреби, для яких він був розпочатий;

- *управління людськими ресурсами* – процеси, які організовують, керують і керують командою проєкту;

- *управління комунікаціями* – займається збиранням та використанням всієї інформації, пов’язаної з виконанням проєкту;

- *управління закупівлями* – включають процеси, пов’язані із закупівлею матеріалів і продуктів, необхідних для виконання проєкту;

- *управління ризиками* – включає процеси проведення планування управління ризиками, ідентифікації, аналізу, планування реагування та контроль ризиків у проєкті;

- *управління зацікавленими сторонами проєкту* – процеси, необхідні для ідентифікації та аналізу зацікавлених сторін, їхніх очікувань, а також для розробки відповідних стратегій управління для ефективного залучення

зацікавлених сторін до прийняття рішень та виконання проєкту. [4]

1.2.2 Методологія PRINCE2

Методологія PRINCE2 була розроблена у Великій Британії з метою управління інформаційними/комп'ютерними проєктами. Пізніше її було змінено та розширено, щоб її можна було використовувати й для інших проєктів. З огляду на те, що вона в основному орієнтована на комп'ютерні проєкти, ця методологія наполягала на гарній комунікації та залученні клієнтів до процесу управління проєктом, на розподілі проєктів на фази та на орієнтації на очікувані результати проєкту. Методологія PRINCE 2 складається з дев'яти основних елементів: Організація; Планування; Контроль; Фази; Управління ризиками; Якість у проєктному середовищі; Управління конфігурацією; Контроль змін. Процес управління проєктом, що здійснюється за допомогою методології PRINCE2, найчастіше спостерігається через вісім основних підпроцесів: запуск проєкту; Ініціація проєкту; Фазовий контроль; Менеджмент на основі результатів; Управління фазовими обмеженнями; Планування; Керівництво проєктом і закриття проєкту. На додаток до цієї широкої версії пропонується вужча версія, яка містить чотири основні підпроцеси: запуск проєкту; Ініціація проєкту; Реалізація проєкту та закриття проєкту. Враховуючи складність цієї методології, її реалізація вимагає всебічного знання управління проєктом, щоб включити всі основні елементи та основні підпроцеси.

1.2.3 Стандарти IPMA

Бачення IPMA полягає в тому, щоб «Сприяти розвитку компетентності в суспільстві, щоб створити світ, у якому всі проєкти будуть успішними». Тому IPMA визначила міжнародний стандарт компетенцій у сферах управління проєктами, програмами та портфоліо. Для фізичних осіб було визначено IPMA Individual Competence Baseline. На основі ICB4 було також визначено базовий

рівень компетентності IPMA для тренерів і консультантів у сфері проєктів, програм і портфоліо, ICB4CCT (див рис 1.1).

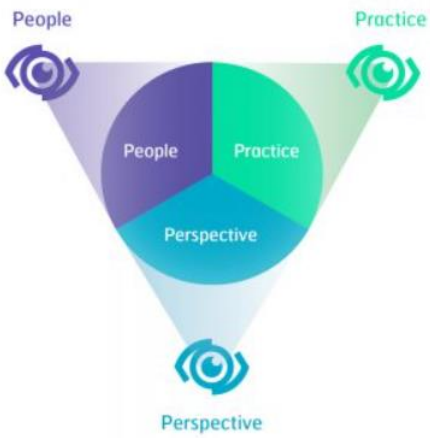

Рис.1.1 Сфери компетенції IPMA

ICB4 — це міжнародний стандарт компетентності для менеджерів проєктів, програм і портфоліо. Компетенція, необхідна для кожної з цих областей, визначається в таких сферах компетенції:

- «Люди»: як ви взаємодієте з оточуючими людьми та самим собою;

- «Практика» приклади роботи: потрібна для проєктів, програм і портфоліо.

- «Перспектива» ініціатив, які ви запускаєте: контекст, у якому здійснюється ініціатива, і зв’язок із тим, що потрібно досягти.

1.2.4 Методології Agile

На відміну від традиційного підходу, філософія гнучкого управління проєктами була представлена як спроба зробити розробку програмного забезпечення більш гнучкою та ефективною. Він швидко став галузевим стандартом управління проєктами. За оцінками, близько 95 відсотків організацій у тій чи іншій формі прийняли Agile. У той же час, залишилося багато роботи, щоб зробити практику зрілою.

Гнучкі методології використовують ітеративний підхід до розробки програмного забезпечення. На відміну від простої лінійної моделі водоспаду, гнучкі проєкти складаються з кількох менших циклів – спринтів. Кожен із них — це проєкт у мініатюрі: він має відставання та складається з етапів проєктування, впровадження, тестування та розгортання в рамках заздалегідь визначеного обсягу робіт.

Наприкінці кожного спринту доставляється потенційно придатний для доставки продукт. Таким чином, з кожною ітерацією до продукту додаються нові функції, що призводить до поступового зростання проєкту. Оскільки функції перевіряються на ранніх стадіях розробки, шанси отримати потенційно невдалий продукт значно нижчі. Давайте підсумуємо основні аспекти (див. табл. 1.3) Agile:

Таблиця 1.3

Відмінність традиційних та гнучких методологій

Традиційні методології:	**Agile методології:**
1	**2**
Постійний процес	Послідовний процес
Тверда структура проєкту	Гнучка структура проєкту
Проєктна документація є точною та детальною	Проєктна документація є попередньою
Технологія виконання проєкту чітко визначена та не змінюється	Технологія виконання проєкту може змінюватися
Можливі зміни	Постійні зміни
Рідкісні зустрічі з клієнтом	Регулярні зустрічі з клієнтом
Багатофункціональна команда	Самоорганізована команда
Високий авторитет керівника проєкту	Не дуже високий авторитет керівника

Agile — це загальний термін для широкого розмаїття методологій і технік, які поділяють описані вище принципи та цінності. Кожен з них має свої сфери

застосування і відмінні риси. Найпопулярнішими фреймворками та практиками є Scrum, Kanban, Lean, Перш ніж обговорювати їх більш детально, давайте розглянемо їх основні характеристики (див. табл. 1.4).

Таблиця 1.4

Основні особливості різних гнучких методологій

Фраємворк	**Основні особливості**
1	**2**
SCRUM	– Весь обсяг робіт розбитий на короткі відрізки – спринти; – Тривалість одного спринту 1-4 тижні; – Люди які приймають участь в розробці, мають чітко визначе ролі.
Kanban	– Розробка побудована на візуалізації робочого процесу; – Поточна робота (WIP) має пріоритет; – Існує n обмежених у часі циклів розробки; – Команда може змінити план розробки в будь-який момент.
Lean	– Сприяє швидкій розробці ПЗ з меншими зусиллями; – Цикл розробки максимально короткий; – Продукт постійно вдосконалюється; – Команда є незалежною та має широке коло обов'язків; – Розробники залучені до формування концепції продукту.
Гібридна	– Поєднання Waterfall та Agile; – Гнучка розробка ПЗ проводиться в умовах водоспаду.

Scrum: найпоширеніша гнучка структура для регулярної доставки випусків

Scrum — це домінуюча гнучка методологія, яка визначає хід процесу. Його використовують виключно 58 відсотків організацій, а ще 18 відсотків компаній поєднують його з іншими методами.

Scrum спрямований на підтримку тісної співпраці між людьми, які працюють над складними продуктами, і деталі змінюються або додаються. Він заснований на систематичній взаємодії між трьома основними ролями з визначеною відповідальністю: Scrum Master, Product Owner і команди..

Scrum Master є центральною фігурою в проєкті. Його основний обов'язок — усунути всі перешкоди, які можуть завадити команді ефективно працювати.

Product Owner, зазвичай клієнт або інша зацікавлена сторона, бере активну участь у проєкті, передаючи глобальне бачення продукту та надаючи своєчасний відгук про виконану роботу після кожного спринту.

Команда Scrum — це між функціональна та самоорганізована група людей, яка відповідає за впровадження продукту. Вона має складатися з 3 до 7 членів команди, щоб залишатися гнучкою та продуктивною. Останнім часом акцент було зміщено з самоорганізованої scrum-команди на самокеровану, тобто команда сама вирішує, хто, що, коли та як робитиме всередині.

Основна одиниця роботи в scrum – спринт – це короткий цикл розробки, який необхідний для створення продукту, який можна відправляти. Зазвичай спринт триває від 1 до 4 тижнів: більш тривалим ітераціям бракує передбачуваності та гнучкості, які є основними перевагами Scrum. Не маючи стандартної тривалості (якщо вона становить менше 4 тижнів), усі спринти в рамках проєкту повинні матифіксовану тривалість. Це полегшує планування та відстеження прогресу (див. рис 1.2).

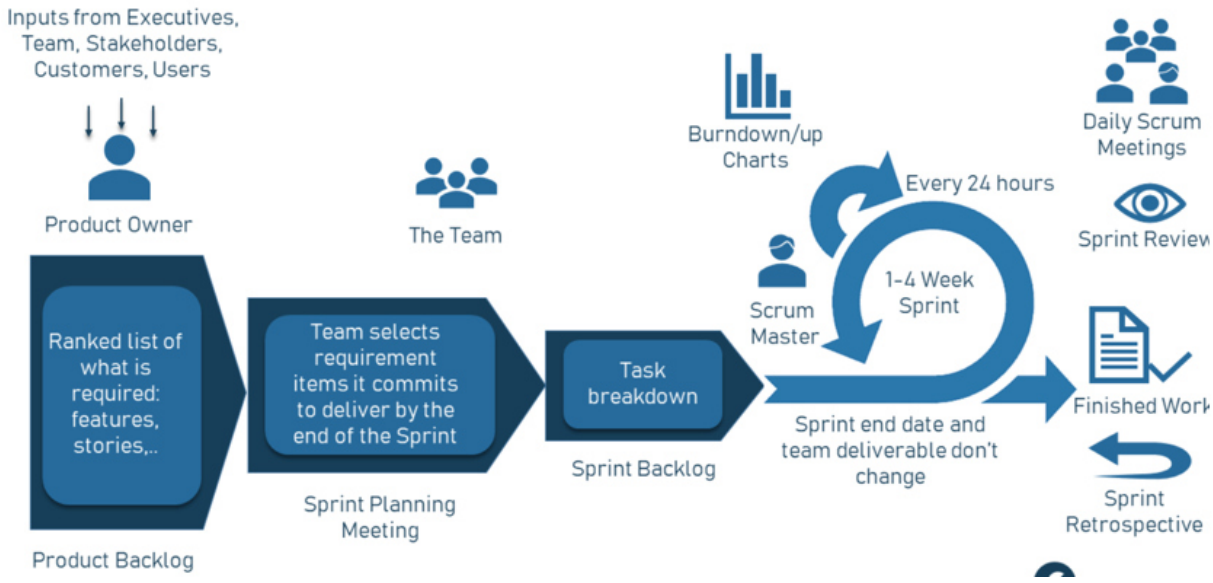

Рис 1.2 Життєвий цикл по методології SCRUM

Беклог продукту — це впорядкований список функцій, які можуть знадобитися в кінцевому продукті проєкту. Це єдине джерело вимог. Беклог продукту оновлюється, коли змінюються або додаються нові вимоги, виправлення, функції та деталі.

Беклог Спринту – це список завдань, які команда має виконати, щоб забезпечити приріст функціонального програмного забезпечення наприкінці кожного Спринту. Іншими словами, члени команди домовляються про те, які товари постачати, і визначають план, як це зробити.

Kanban: комплексне рішення для обробки незавершених робіт

Kanban — це ще одна поширена структура, яка використовується для впровадження гнучкого управління проєктами. Тим не менш, 43 відсотки компаній заявили, що вони використовують Kanban як один із підходів Agile. Kanban — це простий, але потужний підхід до розробки програмних продуктів. Його витоки сходять до візуальної системи карток, які використовувалися на виробництві Toyota як метод контролю виробництва.

У перекладі з японської як «візуальний сигнал», Kanban зосереджується на візуалізації робочого процесу та визначає пріоритетність незавершеної роботи

(WIP) (див. рис 1.3), обмежуючи її обсяг, щоб ефективно відповідати можливостям команди.

Рис 1.3 Дошка Kanban

Як тільки завдання буде виконано, команда може взяти наступний елемент із конвеєра. Таким чином, процес розробки пропонує більшу гнучкість у плануванні, швидшу реалізацію, чіткі цілі та прозорість.

У Kanban, на відміну від Scrum, не потрібні стандартні процедури в рамках процесу, а також фіксовані ітерації. Розробка проєкту базується на візуалізації робочого процесу за допомогою дошки Kanban, яка зазвичай представлена липкими нотатками та дошками або онлайн-інструментами, такими як Trello.

Використовуючи Kanban, команди можуть робити невеликі випуски та адаптуватися до мінливих пріоритетів. На відміну від Scrum, тут немає спринтів із заздалегідь визначеними цілями. Канбан зосереджений на виконанні невеликих фрагментів роботи в міру їх появи. Наприклад, якщо тестувальники знаходять помилки в продукті, розробники намагаються їх негайно виправити. Kanban, наприклад, добре працює після основного випуску продукту та підходить для цілей оновлення та обслуговування.

1.3 Історія та термінологія хмарних технологій.

Історія хмарних технологій почалася досить давно. До 70-х – 80-х років відносять перші ідеї, які побічно торкалися того, що згодом і стало хмарними обчисленнями. Але все таки датою відліку сучасної історії cloud computing став 2006 рік, коли компанія Amazon, яка вже на той момент була однією з найбільших, презентувала свою інфраструктуру веб-сервісів, яка була здатна забезпечити користувачеві не лише хостинг, а й надати віддалені обчислювальні потужності клієнтові. Новинку сприйняли і схвалили такі гіганти, як Google, Sun і IBM, а в 2008 році про свій інтерес у цій галузі заявила корпорація Microsoft.

Хмарні технології пропонують масштабовану інфраструктуру і програмні засоби без прямої прив'язки до фізичних машин, при цьому економлячи витрати, серверні потужності і енергоспоживання під час простоювання. Хмарні технології – це можливість безлічі фізичних серверів бути єдиним обчислювальним середовищем. В цілому, сервіси хмарних обчислень є додатками, доступ до яких забезпечується через Інтернет за допомогою браузера або інших мережевих застосувань, наприклад, FTP-клієнта. Головна відмінність від звичного методу роботи з ПЗ полягає в тому, що користувач використовує не ресурси свого комп'ютера, або сервера своєї локальної мережі, а потужності, які надаються йому як Інтернет-послуга. При цьому користувач має повний доступ до власних даних і можливість роботи з ними з будь-якої точки світу і з будь-якого пристрою, але не обтяжений управлінням операційною системою, програмною базою, обчислювальними потужностями, за допомогою яких ця робота відбувається. Зберігання в хмарі не лише даних, а й додатків змінює обчислювальну парадигму в бік традиційної клієнт-серверної моделі, при якій на стороні користувача зберігається мінімально необхідна функціональність. Таким чином, необхідність встановлювати необхідні оновлення програмного забезпечення, проводити перевірку на віруси й інше обслуговування покладається на провайдера хмарного сервісу. Це також означає, що загальний доступ, управління версіями, спільне

редагування стають набагато простішими, ніж коли додатки і дані розміщені на призначених для користувача комп’ютерах.

Розглянемо основні визначення:

Хмарні технології – це технології обробки даних, в яких комп’ютерні ресурси надаються Інтернет користувачеві як онлайн сервіс, одна велика концепція, що включає в себе багато різних понять, що надають послуги.

Хмарний сервіс – послуга надання хмарних ресурсів за допомогою технологій «хмарних обчислень».

Хмарні обчислення (англ. cloud computing) – це програмно-апаратне забезпечення, доступне користувачеві через Інтернет або локальну мережу у вигляді сервісу, що дозволяє використовувати зручний інтерфейс для віддаленого доступу до виділених ресурсів (обчислювальних ресурсів, програм і даних). Комп’ютер користувача виступає при цьому рядовим терміналом, підключеним до Мережі. Комп’ютери, які здійснюють cloud computing, називаються «обчислювальною хмарою». При цьому навантаження між комп’ютерами, що входять в «обчислювальну хмару», розподіляється автоматично.

Хмарні обчислення – це модель надання зручного мережевого доступу в режимі «на вимогу» до колективно використовуваного набору налаштовуваних параметрів обчислювальних ресурсів (наприклад, мереж, серверів, сховищ даних, додатків і / або сервісів), які користувач може оперативно задіяти під свої задачі і вивільняти при зведенні до мінімуму числа взаємодій з постачальником послуги або власних управлінських зусиль. Ця модель спрямована на підвищення доступності обчислювальних ресурсів і поєднує в собі п’ять головних характеристик, три моделі обслуговування і чотири моделі розгортання.

1.3.1 Загальні відомості та характеристики хмарних технологій

Суть хмарних технологій, таким чином, полягає в перенесенні обробки даних з персональних комп'ютерів і робочих станцій на сервери всесвітньої мережі. В області комп'ютерного моделювання це означає розгортання програмних комплексів на ресурсах Інтернет. Користувач стає не покупцем обчислювальних програм і комплексів, а їх орендарем, якому надаються різноманітні послуги. Форма купівлі–продажу товару з відчуженням прав власності від продавця до покупця змінюється на форму оренди, в даному випадку – продажу не продукту, а послуг з його використання клієнтом без зміни власника продукту. При цьому забезпечена повна відповідність виробничих потужностей інфраструктури фактичним потребам користувача.

Хоча термін "хмарні технології" є сталим, в українській мові він має інше значення, ніж оригінал. "Cloud" окрім хмари має й інше значення – розсіяний; власне значення "розсіяний" і мається на увазі в англомовній термінології.

Хмарна обробка даних як концепція включає поняття:

- інфраструктура як послуга;
- платформа як послуга;
- програмне забезпечення як послуга;
- дані як послуга;
- робоче місце як послуга;
- інші технологічні тенденції, загальною рисою яких є впевненість, що мережа Інтернет у змозі задовільнити потреби користувачів в обробці даних.

Характеристики хмарних обчислень:

Самообслуговування на вимогу

Споживач, коли це йому необхідно, може самостійно задіяти обчислювальні можливості, такі як серверний час або мережеве сховище даних, в автоматичному режимі, без взаємодій з персоналом постачальника послуг.

Широка доступність через мережу (Інтернет)

Можливості доступні через мережу; доступ до них здійснюється на основі стандартних механізмів, що забезпечує використання різнорідних тонких і

товстих клієнтських платформ (наприклад, мобільних телефонів, ноутбуків, КПК).

Об’єднання ресурсів в пул

Постачальник об’єднує свої обчислювальні ресурси в пул для обслуговування великого числа споживачів, використовуючи принцип множинної оренди (Multi-tenancy). Різні фізичні і віртуальні ресурси динамічно розподіляються і перерозподіляються відповідно до потреб користувачів. Виникає відчуття незалежності від місця розташування, коли замовник не знає і не контролює, де конкретно знаходяться обчислювальні ресурси, якими він користується, але, можливо, може визначити їх розташування на більш абстрактному рівні (наприклад, країна, регіон або дата-центр). Прикладом ресурсів можуть бути сховище даних, обчислювальна потужність, оперативна пам’ять, пропускна здатність, віртуальні машини.

Здатність до швидкої адаптації

Обчислювальні можливості можуть швидко і гнучко резервуватися (часто автоматично) для оперативного масштабування під завдання замовника, і також швидко звільнятися. З точки зору споживача доступні можливості часто виглядають нічим не обмеженими і можуть бути придбані в будь-якій кількості в будь-який час.

Вимірюваність послуги

Хмарні системи автоматично контролюють і оптимізують використання ресурсів через вимір деяких абстрактних параметрів. Параметри варіюються в залежності від типу послуги. Наприклад, це можуть бути: розмір сховища даних, обчислювальна потужність, пропускна здатність і / або число активних записів користувача. Використання ресурсів відстежується, контролюється; формуються звіти. Таким чином і постачальник, і споживач отримують прозору інформацію про обсяг наданих (спожитих) послуг.

З точки зору постачальника, завдяки об’єднанню ресурсів і непостійному характеру споживання з боку споживачів, хмарні обчислення дозволяють економити на масштабах, використовуючи менші апаратні ресурси, ніж вимагалися б у виділених апаратних потужностях для кожного споживача, а за рахунок автоматизації процедур модифікації виділення ресурсів істотно знижуються витрати на абонентське обслуговування.

З точки зору споживача, ці характеристики дозволяють отримати послуги з високим рівнем доступності (англ. high availability) і низькими показниками непрацездатності, забезпечити швидке масштабування обчислювальної системи

завдяки еластичності без необхідності створення, обслуговування і модернізації власної апаратної інфраструктури.

Зручність і універсальність доступу забезпечується широкою доступністю послуг і підтримкою різного класу термінальних пристроїв (персональних комп'ютерів, мобільних телефонів, інтернет-планшетів).

Для демонстрації різниці між звичайними і хмарними сервісами можна розглянути послуги, що надаються хостинг-провайдером. При традиційному (звичайному) підході провайдер на місячній основі отримує фіксовану винагороду за використання його обчислювальних ресурсів (CPU, RAM, HDD і ін.). При цьому не має значення, чи використовував клієнт виділені йому ресурси в повному обсязі впродовж усього місяця чи тільки декілька днів, а решта часу обчислювальні ресурси простоювали.

У випадку надання хмарного сервісу використовується тип оплати "плата за використання". Зазвичай за одиницю виміру часу роботи береться хвилина або година користування ресурсами. Під час оцінювання обсягів даних за одиницю виміру береться Мегабайт інформації, що зберігається. В цьому випадку користувач оплачує тільки той об'єм ресурсів, який ним в реальності використовувався протягом певного часу. Крім того, хмарна інфраструктура надає користувачеві можливість за необхідності збільшувати чи зменшувати максимальні ліміти виділених ресурсів, користуючись тим самим еластичністю сервісу, що надається. Користувачеві хмарних сервісів не потрібно піклуватися про інфраструктуру, яка забезпечує працездатність сервісів, що надаються йому. Усі завдання по налаштуванню, усуненню несправностей, розширенню інфраструктури і ін. бере на себе сервіс-провайдер.

1.4. Моделі хмарного розміщення

У сучасному світі існує декілька типів хмар. Завдання полягає в тому, щоб зрозуміти, яка модель краще всього підходить конкретній організації в конкретних умовах, а потім вибрати оптимальний спосіб підключення до інших хмарних ресурсів, щоб реалізувати увесь потенціал цієї технології.

Класифікувати хмарні обчислення можна таким чином: загальна "хмара", публічна "хмара" (public cloud), приватна "хмара" (private cloud) і гібридна "хмара".

Розглянемо основні відмінності типів хмар.

Приватна хмара (англ. private cloud) – інфраструктура, призначена для використання однією організацією, що включає декілька споживачів (наприклад,

підрозділів однієї організації), можливо також клієнтами і підрядчиками цієї організації. Приватна хмара може знаходитися у власності, управлінні і експлуатації як самої організації, так і третьої сторони (чи яких-небудь їх комбінацій), і вона може фізично існувати як усередині, так і поза юрисдикцією власника.

Публічна хмара (англ. public cloud) – інфраструктура, призначена для вільного використання широкою публікою. Публічна хмара може знаходитися у власності, управлінні і експлуатації комерційних, наукових і урядових організацій (чи яких-небудь їх комбінацій). Публічна хмара фізично існує в юрисдикції власника – постачальника послуг. Загальнодоступна хмара – модель, коли незалежний провайдер надає в оренду ПЗ, інфраструктуру або платформи хмарних обчислень за принципом "ПЗ як послуга" (SaaS), "інфраструктура як послуга" (IaaS) або "платформа як послуга" (PaaS).

Гібридна хмара (англ. hybrid cloud) – це комбінація з двох або більше різних хмарних інфраструктур (приватних, публічних або громадських), що залишаються унікальними об'єктами, але пов'язані між собою стандартизованими або приватними технологіями передачі даних і додатків (наприклад, короткочасне використання ресурсів публічних хмар для балансування навантаження між хмарами). Гібридна хмара – архітектура, що поєднує в собі риси приватних і громадських моделей хмарних обчислень. В цьому випадку критично важливі додатки або конфіденційні дані зберігаються в приватній хмарі, що належить самій компанії. У загальнодоступній же частині хмари розміщуються усі інші додатки, особливо складні, які нерегулярно використовуються або вимагають частого оновлення.

Громадська хмара (англ. community cloud) – вид інфраструктури, призначений для використання конкретним співтовариством споживачів з організацій, що мають загальні завдання (наприклад, місії вимог безпеки, політики, і відповідності різним вимогам). Громадська хмара може знаходитися в кооперативній (спільній) власності, управлінні і експлуатації однієї або більше організацій, співтовариств або третьої сторони (чи яких-небудь їх комбінацій), і вона може фізично існувати як усередині, так і поза юрисдикцією власника.

Приватні системи хмарних обчислень відрізняються високим рівнем безпеки і керованості, недосяжним для загальнодоступних хмар, і при цьому обходяться набагато дешевше. З іншого боку, в них може бути відсутньою можливість швидкого масштабування, характерна для загальнодоступних хмар. Зрозуміло, свої недоліки є і у останніх: стандартизовані застосування або платформи не дозволяють встановити індивідуальні налаштування. Крім того, користувачам доведеться змиритися з додатковими ризиками, оскільки конфіденційна інформація буде довірена сторонній компанії. Оптимальним

рішенням для більшості організацій служать гібридні системи. Їх застосування зводить до мінімуму можливі ризики, оскільки найбільш важливі застосування залишаються під контролем власника, а менш важливі програми з нерівномірним коефіцієнтом використання зберігаються на серверних фермах.

Розглянемо основні відмінності хмар від традиційних хостинг-рішень:

• на відміну від dedicated-серверів, установка і налаштування яких займає багато часу, хмарні сервіси повинні бути доступні для використання відразу після покупки, те ж саме стосується більшості класичних послуг дата-центрів;

• на відміну від shared-хостингу, в хмарах є можливість нарощувати обсяг закуплених потужностей миттєво, без звернення до служби технічної підтримки оператора;

• на відміну від того, що було на ринку хостингу до хмар, хмарні продукти надають схему оплати за фактом, тобто оплату тільки тієї потужності, яку користувач дійсно використовує з досить коротким проміжком тарифікації.

Це основні принципи, за якими можна чітко відрізняти хмарні продукти. Однак, вони досить широкі, хоч і строгі. У поняття хмарного продукту вкладається три види поставки продукту користувачеві за ступенем гнучкості. Це надання інфраструктури як сервісу (в оренду), платформи як сервісу або програми як сервісу.

Хмарні обчислення нині включають наступні технології:

- **SaaS:** Software as a Service, “Програмне забезпечення як послуга”;
- **PaaS:** Platform as a Service, “Платформа як послуга”;
- **IaaS:** Infrastructure as a Service, “Інфраструктура як послуга”;
- **DaaS:** Data as a Service, “Дані як послуга”;
- **WaaS:** Workplace as a Service, “Робоче місце як послуга”;
- **AaaS:** All as a Service, “Усе як послуга”.

В даний час прийнято виділяти три основні моделі обслуговування хмарних технологій, які іноді називають шарами хмари. Можна сказати, що ці три шари - послуги інфраструктури, послуги платформи і послуги застосунків - відображають будову не тільки хмарних технологій, а й інформаційних технологій в цілому. Зупинимося докладніше на кожному з них.

SaaS – це модель використання бізнес-додатків в форматі інтернет-сервісів.

SaaS додатки працюють на сервері SaaS-провайдера, а користувачі отримують до них доступ через інтернет-браузер. Користувач не купує SaaS-додаток, а орендує його – платить за його використання деяку суму в місяць. Таким чином досягається економічний ефект, який вважається одним з головних переваг SaaS.

SaaS провайдер піклується про працездатність додатків, здійснює технічну підтримку користувачів, самостійно встановлює оновлення. Таким чином, користувач менше думає про технічну сторону питання, а зосереджується на своїх бізнес-цілях.

PaaS – модель надання хмарних обчислень, при якій споживач отримує доступ до використання інформаційно-технологічних платформ: операційних систем, систем управління базами даних, зв'язного програмного забезпечення, засобів розробки і тестування розміщених у хмарних провайдерах. У цій моделі вся інформаційно-технологічна інфраструктура, включаючи обчислювальні мережі, сервери, системи зберігання, цілком керується провайдером, ним же визначається набір доступних для споживачів видів платформ та набір керованих параметрів платформ, а споживачеві надається можливість використовувати платформи, створювати їх віртуальні екземпляри, встановлювати, розробляти, тестувати, експлуатувати на них прикладне програмне забезпечення, при цьому динамічно змінюючи кількість споживаних обчислювальних ресурсів.

Провайдер хмарної платформи може стягувати плату зі споживачів залежно від рівня споживання, тарифікація можлива за часом роботи додатків споживача, за обсягом оброблювальних даних і кількості транзакцій над ними, по мережному трафіку. Провайдери хмарних платформ досягають економічного ефекту за рахунок використання віртуалізації та економії на масштабах, коли з безлічі споживачів в один і той же час лише частина з них активно використовує обчислювальні ресурси, споживачі – за рахунок відмови від капітальних вкладень в інфраструктуру і платформи, розрахованих під пікову потужність і непрофільних витрат на безпосереднє обслуговування всього комплексу.

IaaS – це модель обслуговування, в межах якої споживачу надається можливість керувати засобами обробки та збереження, комунікаційними мережами, та іншими фундаментальними обчислювальними ресурсами, на базі яких споживач може розгортати та виконувати довільне програмне забезпечення, до складу якого можуть входити операційні системи та прикладні програми. Споживач не керує фізичною та віртуальною інфраструктурою, що лежить в основі хмари, проте він контролює операційні системи, системи збереження, встановлені програми та, можливо, має обмежений контроль над деякими мережевими компонентами (наприклад, мережевими екранами вузлів).

IaaS складається з трьох основних компонентів:

Апаратні засоби (сервери, системи зберігання даних, клієнтські системи, мережеве обладнання);

Операційні системи та системне ПЗ (засоби віртуалізації, автоматизації, основні засоби управління ресурсами);

Зв’язуюче ПЗ (наприклад, для управління системами).

1.5. 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.

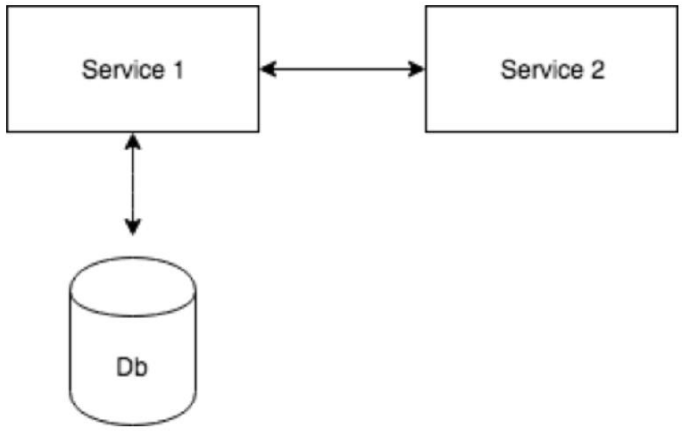

Рис 1.2 Схема взаємодії мікросервісу

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 систем

ВИСНОВКИ ДО РОЗДІЛУ 1

Метою даного розділу роботи було проаналізувати та обговорити що таке проєкт, його види, відмінність від звичайних операцій, можливості впровадження вже існуючих методологій управління проєктами для різних типів проєктів.

Було проведено короткий аналіз методологій, запропонованих та впроваджених у практику управління проєктами (PMI, IPMA, PRINCE 2 та інші). Проаналізовано особливості цих методологій з урахуванням можливостей їх реалізації.

Також в розділі було проведено аналіз сучасних хмарних технологій, різних моделей хмарних сервісів, дано визначення хмарним обчисленням та розглянута архітектура, що дозволяє хмарним сервісам бути такими якими вони є.

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

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

Результати проведеного аналізу показали, що загальні характеристики традиційних методологій роблять їх більш придатними для впровадження у більші та складніші проєкти. На відміну від них, загальні характеристики гнучких методологій дозволяють зробити висновок, що ці методології більше підходять для використання в IT-проєктах, а також для деяких менших і менш складних проєктах, таких як проведення різноманітних досліджень, звітів про проєкти тощо.

2. ОБГРУНТУВАННЯ ДОЦІЛЬНОСТІ РЕАЛІЗАЦІЇ ПРОЄКТУ ТА АНАЛІЗ РИНКУ ПОДІБНИХ РІШЕНЬ.

2.1 Опис організації – середовища реалізації проєкту

Компанія UCloud — це один із лідерів ринку хмарних рішень, що пропонує клієнтам IaaS (інфраструктуру як сервіс) для вирішення завдань будь-якої складності й масштабу: від підключення віртуальних сервісів до розгортання власної IT-інфраструктури.

UCloud - найбільший український хмарний провайдер, який має дата-центри як в України, так і за кордоном у Польщі та Франкфурті. Вже 11 років ми допомагаємо бізнесам зберігати, резервувати та користуватися своїми даними, які розміщені на серверах у наших ЦОД.

Бренд UCLOUD був заснований в 2011 році командою сертифікованих фахівців. UCLOUD сьогодні - це сучасний хмарний провайдер, обладнання якого розміщено як в Україні (ЦОД BeMobile, Київ, Україна), так і за кордоном (ЦОД Atman, Варшава, Польща).

Місія компанії - допомога бізнесу в захисті і збереження даних, оптимізації витрат на IT, масштабування процесів.

Для забезпечення безперебійного доступу до даних всім клієнтам надається доступ до хмари через Інтернет та прямий канал передачі даних L2. UCLOUD співпрацює з Microsoft Veeam, Citrix, Cisco, 1C, Бітрікс і багатьма іншими компаніями для надання якісного сервісу клієнтам.

UCLOUD надає послуги:

- Приватної або гібридної хмари,
- Віртуального ЦОД,
- Резервного ЦОД,
- 1С в хмарі,
- Office 365,
- Оренда ПО Microsoft,
- ExpressRoute в хмару Microsoft Azure,
- Прямі канали зв’язку.

Організаційна структура (див. рис. 2.2)

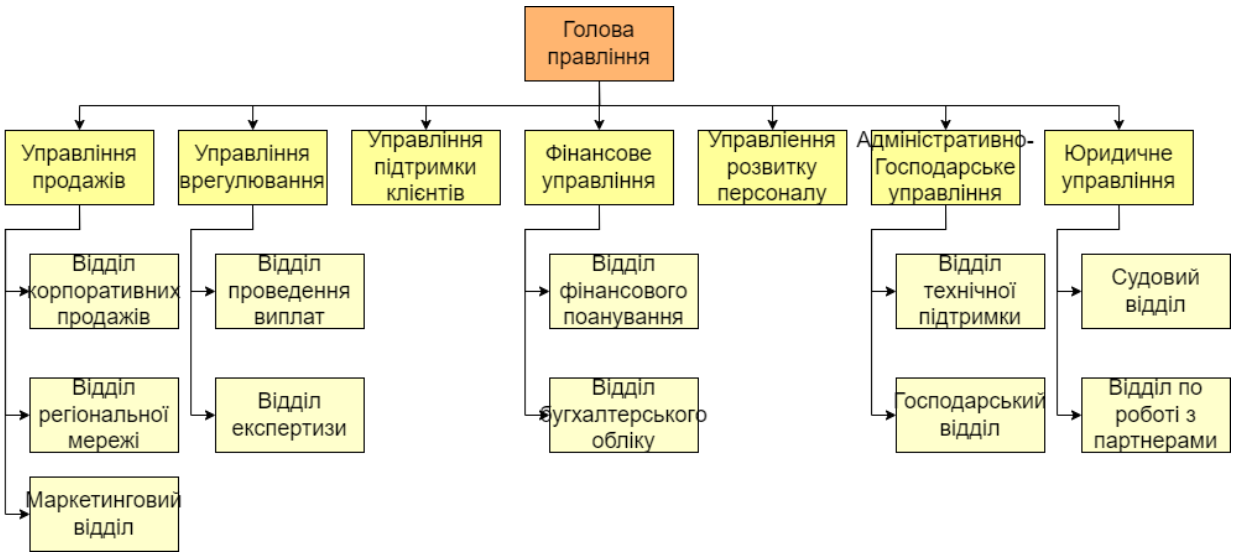

Рис.2.2 Організаційна структура компанії «Ucloud»

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.

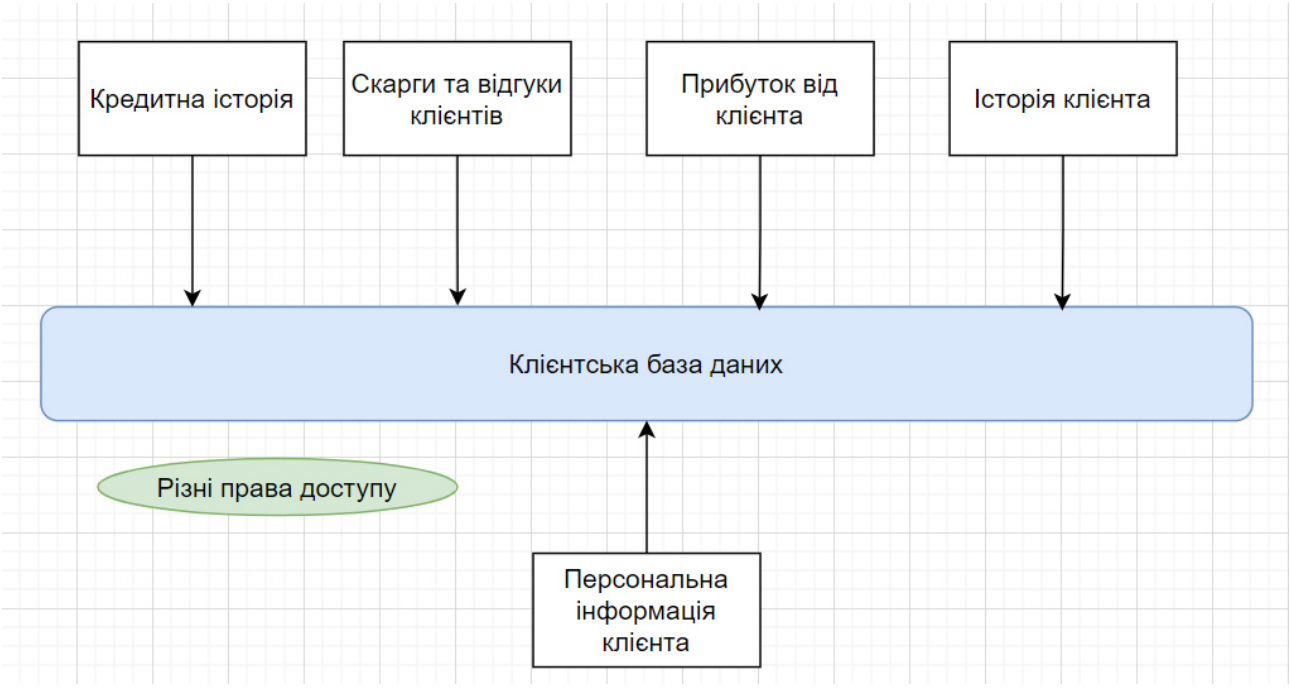

Рис 2.1 Клієнтська база даних

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:

Дослідити можливості для нових зусиль або вирішення проблем.

Приймайте рішення про найкращий шлях для вашої ініціативи. Визначення ваших можливостей для успіху в контексті загроз успіху може прояснити напрямки та вибір.

Визначте, де можливі зміни. Якщо ви перебуваєте на переломному або поворотному етапі, аналіз ваших сильних і слабких сторін може виявити пріоритети, а також можливості.

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 (О, Т).

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. табл. 2.2):

SWOT-аналіз проєкту

Сильні сторони	*Слабкі сторони*
- Зручна та проста інтеграція; - Цілодобова підтримка клієнтів та оновлень; - Покращення умов праці для співробітників; - Високий рівень безпеки.	- Подібні рішення є на ринку, тобто ідея не нова; - Для реалізації проєкту потрібно навчати персонал; - Залежність від інших систем.

Можливості	*Загрози*
- Збільшити кількість клієнтів, та їх задоволення якістю послуг; - Швидкий доступ до необхідної інформації; - Інтеграція з будь якою платформою. - Немає зайвого функціоналу.	- Відсутність інтернету; - Конкуренти створюватимуть подібні рішення; - Підвищення вартості послуг.

Таблиця 2.2 SWOT-аналіз

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. Від.	Директор "WebVision"	UI/UX розробник	Програмісти	тестувальники	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	Співробітник відділу бізнесу	Керівник ризиків	Замісник кер. ризиків	Керівник юрид. Від.	Зам. Кер. Юр. Відд.	Директор "WebVision"	UI/UX розробник	Програмісти	тестувалтники	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:

- База «мертвих» клієнтів
- Потенційна клієнтська база

- Активна клієнтська база
- Працююча клієнтська.
- База постійних.
- База даних перевірених часом.
- VIP.
- Ключові клієнти.
-

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

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.

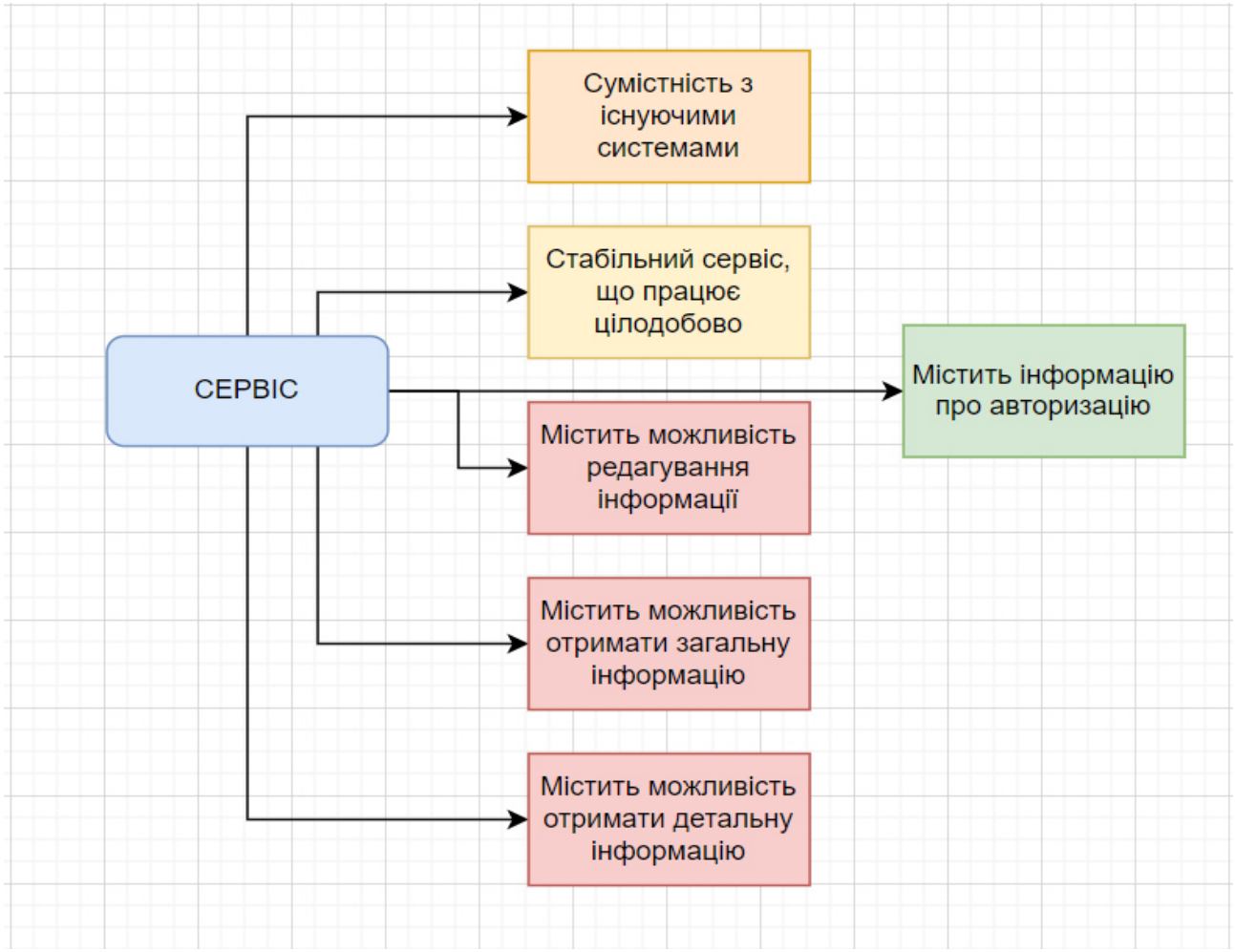

Рис 3.1 Функціональні вимоги до сервісу

Не функціональні вимоги:

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

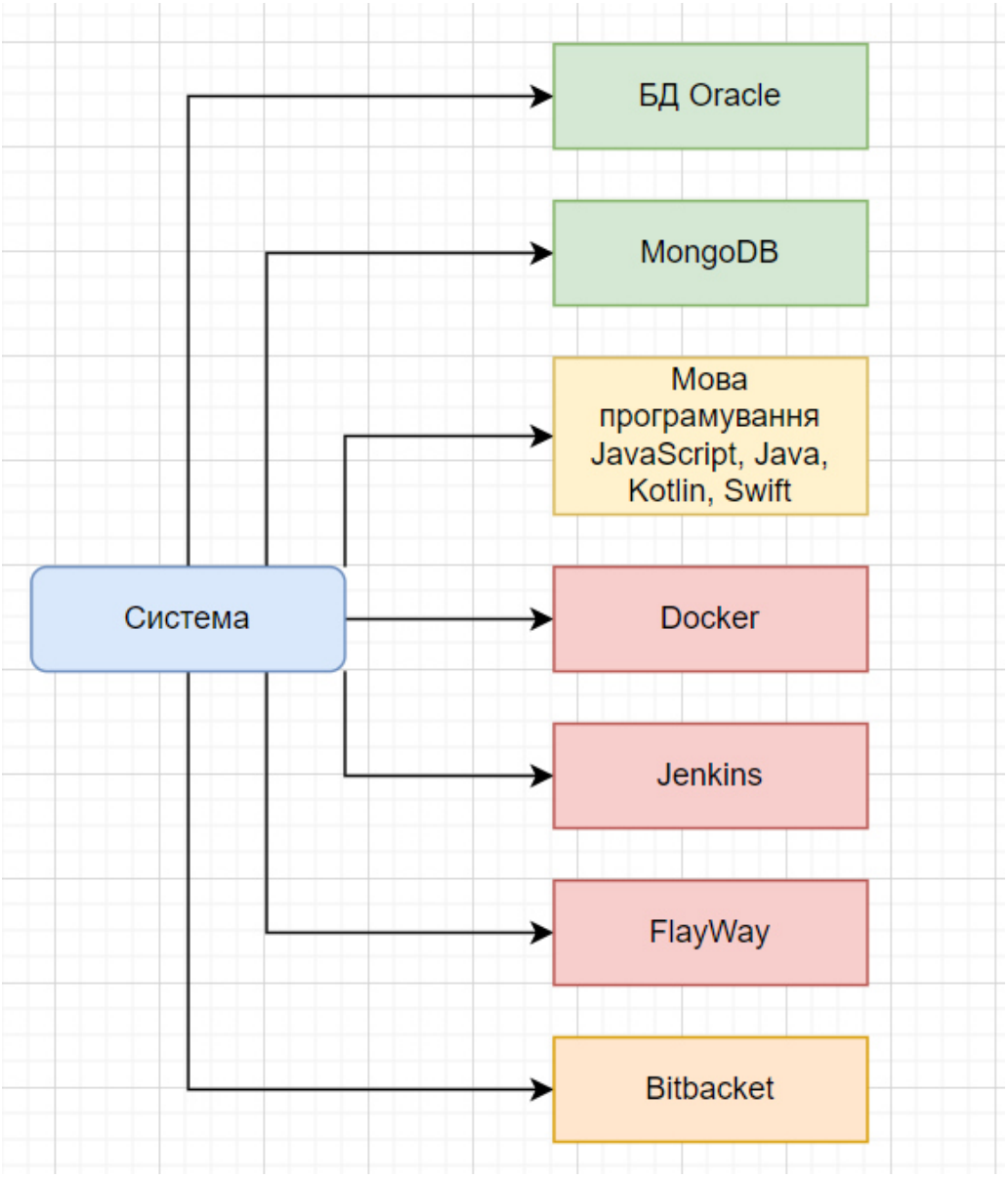

Рис 3.2 Не функціональні вимоги

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.

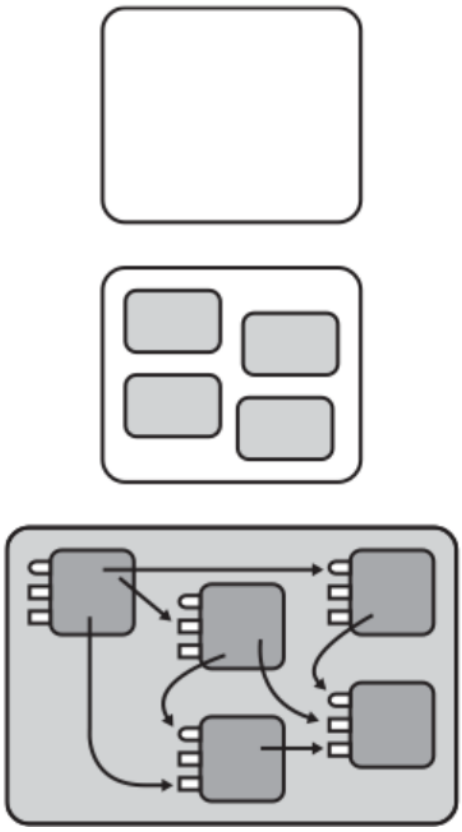

Рис 3.3 Декомпозиція архітектури

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.

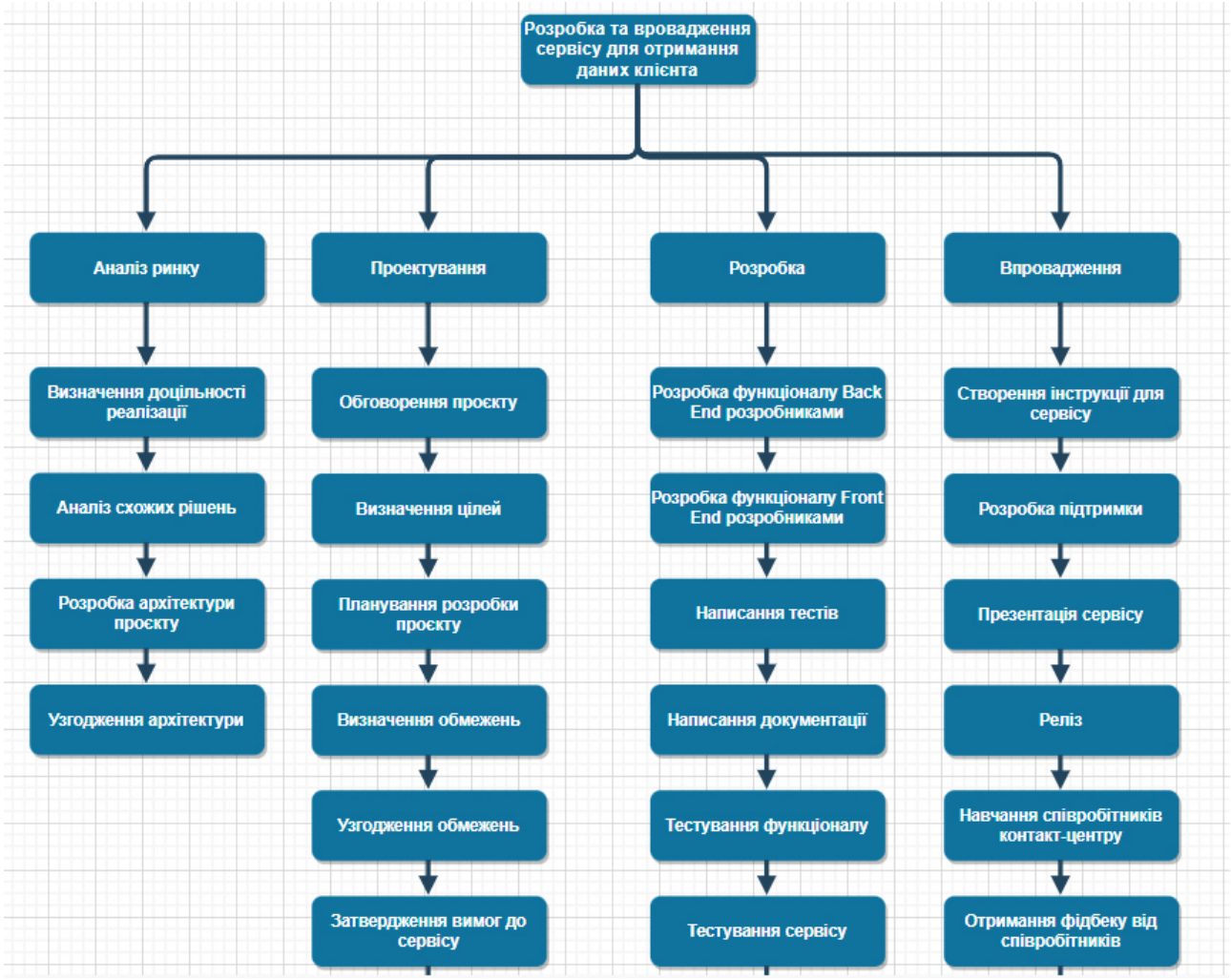

Рис 3.4 Структурна декомпозиція

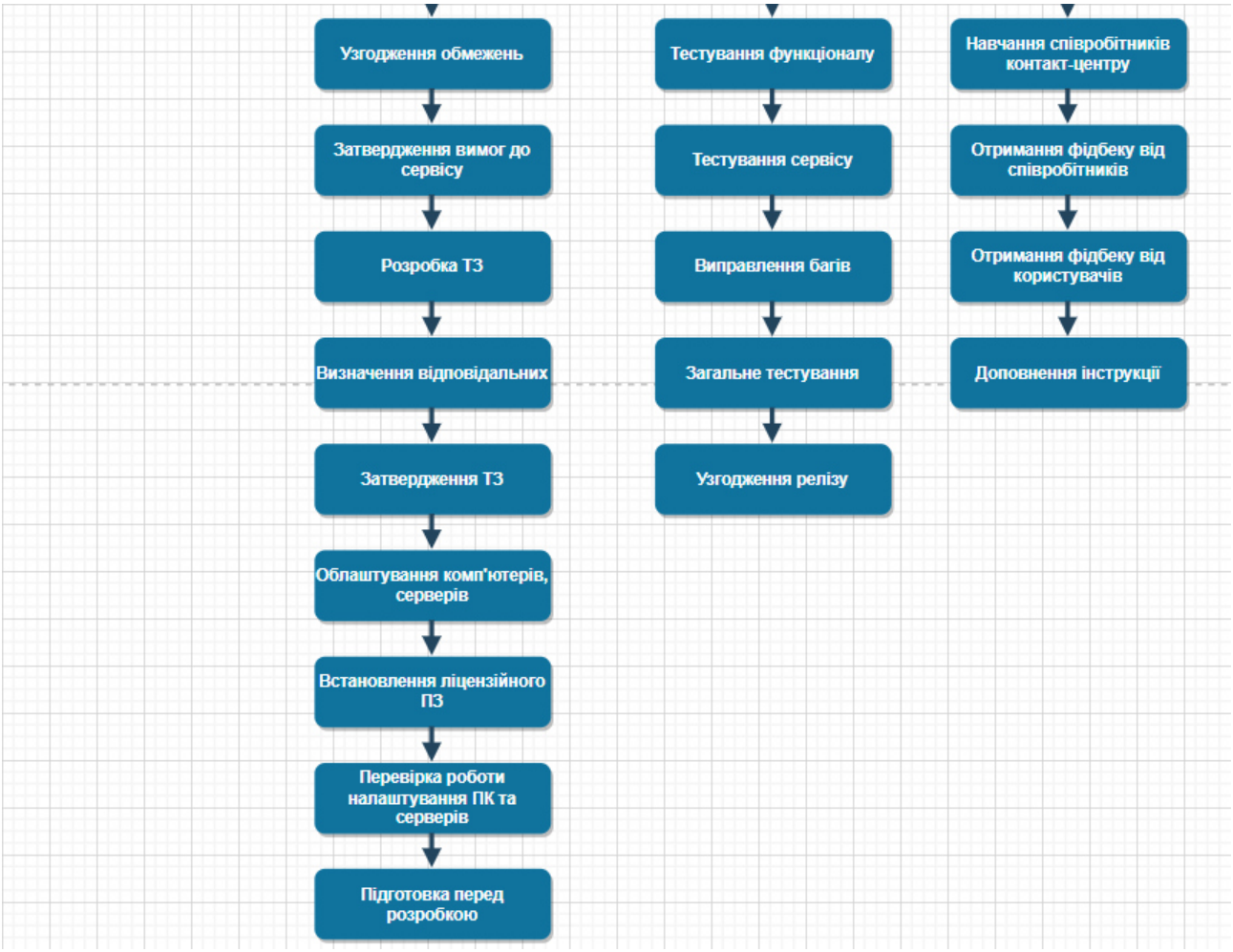

Рис 3.5 Структурна декомпозиція(продовження)

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

Код	Назва задачі	Тривалість задачі	Початок	Завершенн

				Я
1.	Початок проєкту	0 днів	05.09.23	05.09.23
2.	Аналіз ринку	12 днів	06.09.23	22.09.23
3.	Проєктування сервісу	39 днів	22.09.23	16.11.23
4.	Розробка сервісу	70 днів	16.11.23	22.02.24
5.	Впровадження сервісу	24 днів	22.02.24	27.03.24
6.	Завершення проєкту	1 день	27.03.24	28.03.24

1. Термін: планується виконати весь проект протягом 147 днів в термін з 05.09.2023 по 28.03.2024

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;

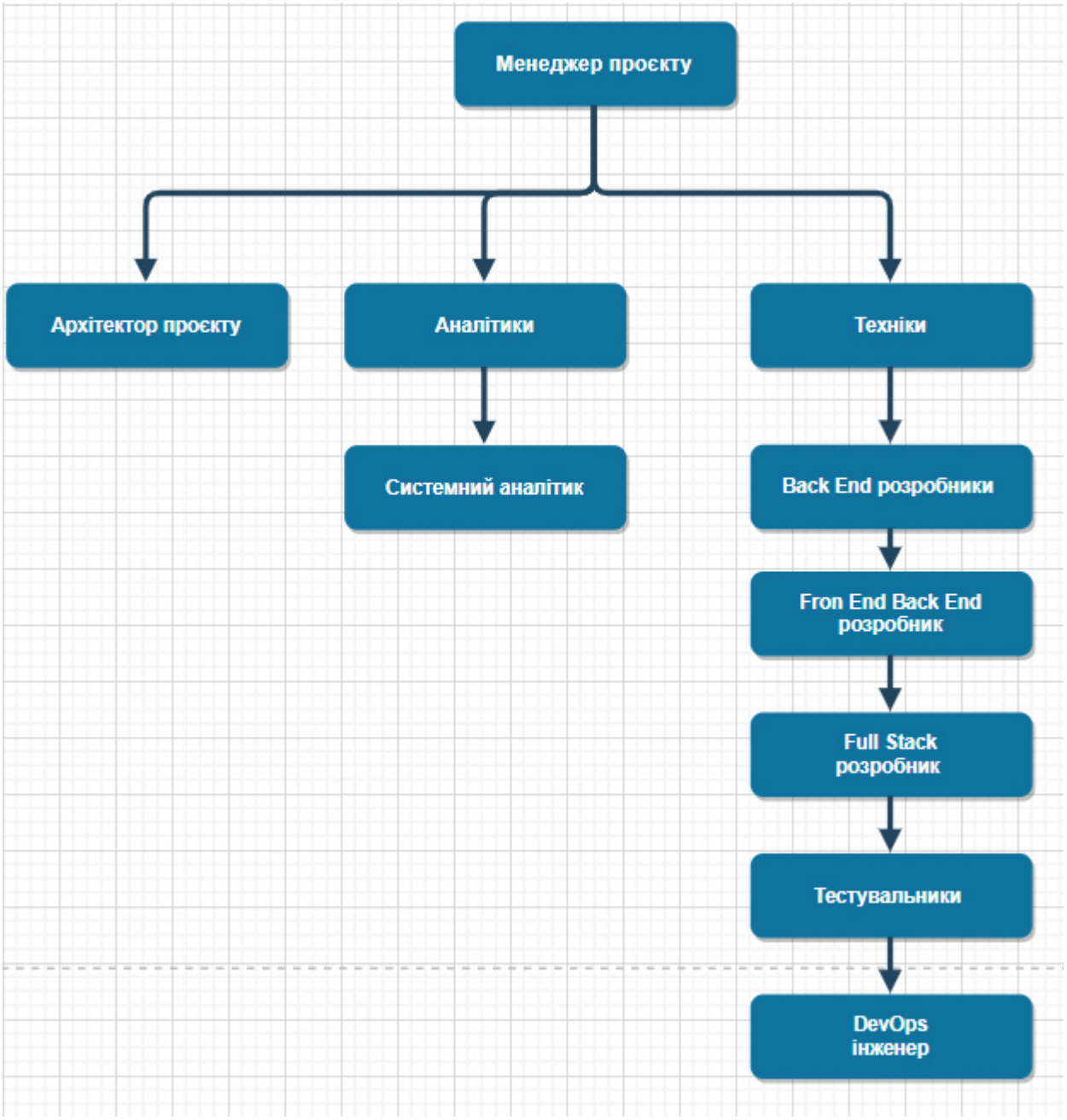

Рис 3.6 Ієрархічна модель організаційної структури

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:

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

3.2 План управління проєктом розробки сервіс

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

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

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

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

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 розробниками	14 днів	Front end розробник, 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 співробітників	2 дні	Менеджер проєкту
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;

- забезпечення стабільності та стійкості по відношенню до існуючих та потенційних ризиків, приділяючи особливу увагу управлінню кредитним ризиком;

- забезпечення підвищення якості 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

Компанія 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 компанії – надання 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.
- Компанія 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 компанії в цілому.
- Відповідальність кожного співробітника за якість та результати своєї роботи, а також за вдосконалення якості своєї роботи та діяльності компанії 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 компанії.
- Залучення у створення та розвиток системи менеджменту якості всього персоналу компанії, створення для співробітників мотивуючих умов, спрямованих на підвищення якості та ефективності роботи.
- Використання передових та інформаційних технологій, що дозволяють здійснювати обслуговування та розрахунки на якісно високому рівні.

Основні цілі:

- Побудова та постійне покращення системи менеджменту якості у компанії відповідно до вимог міжнародного стандарту ISO 9001:2008 та успішних практик.
- Розробка та покращення Стандартів якості для продуктів / послуг та бізнес-процесів компанії.
- 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 компанії.
- Проведення комплексних рекламних досліджень.
- Реалізація зворотного зв'язку з клієнтами на всіх встановлених каналах.
- Організація ефективної роботи із претензіями.
- Періодичний аудит діяльності компанії з позицій якості.

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

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

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 ресурсами зображено на таб. 3.6:

Таблиця 3.6

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 проекту в процесі реалізації	0,5	0,7	- Проаналізувати всі необхідні росходи на 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	0,1	0,8	- Перевірка юристами всієї складеної документації.

договору			
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 ресурсами зображено на таб. 3.7:

Таблиця 3.7

Назва ризику	Ймовірність	Вплив	Спосіб уникнення (пом'якшення)
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 05.09.2023 по 28.03.2024.

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 ІТ 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.