Міністерство освіти і науки України
Криворізький національний університет
Факультет інформаційних технологій
Кафедра комп’ютерних систем та мереж

Пояснювальна записка
до кваліфікаційної роботи магістра
за спеціальністю 123 «Комп’ютерна інженерія»

на тему: МЕТОДИ РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ НА БАЗІ ОПТИМАЛЬНОЇ АРХІТЕКТУРИ

Проектував _____ Б. С. Кікачеішвілі

Керівник роботи _____ І. О. Музика

Нормоконтроль _____ Д. І. Кузнєцов

Завідувач кафедри _____ А. І. Купін

Кривий Ріг
2023

Криворізький національний університет
Факультет інформаційних технологій
Кафедра комп'ютерних систем та мереж

Ступінь вищої освіти магістр
Спеціальність 123 «Комп'ютерна інженерія»

ЗАТВЕРДЖУЮ
Завідувач кафедри, голова циклової комісії

_____ А. І. Купін

"____" _____ 20__ року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

_____
(прізвище, ім'я, по батькові)

1. Тема роботи _____
_____
_____

керівник роботи _____,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу від "___"_____ 20__року №__

2. Строк подання студентом роботи _____

3. Вихідні дані до роботи _____
_____
_____
_____
_____
_____

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)_____
_____
_____
_____
_____
_____

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)_____
_____
_____
_____
_____

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання_____

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Строк виконання етапів роботи	Примітка

Студент _____ _____
(підпис) (прізвище та ініціали)

Керівник роботи _____ _____
(підпис) (прізвище та ініціали)

РЕФЕРАТ

Пояснювальна записка: 112 сторінок, 49 рисунки, 29 використаних джерел.

Мета даної кваліфікаційної роботи – методи розробки програмного забезпечення на базі оптимальної архітектури.

Даний проект складається з 3 розділів.

У першому розділі виконано огляд компанії IDAP та її програмних продуктів.

Другий розділ виконана розробка моделі та структури розділів IDAP.

У третьому розділі проведно проектування програмного забезпечення.

ПРОЕКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ, ОГЛЯД КОМПАНІЇ, IDAP, РОЗДІЛИ КОМПАНІЇ IDAP, ПРОГРАМНІ ПРОДУКТИ IDAP, МЕТОДИ РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ.

Explanatory note: 112 pages, 49 figures, 29 sources used.

The purpose of this qualification work is to explore methods for developing software based on optimal architecture.

The project consists of 3 sections.

The first section provides an overview of the company IDAP and its software products.

The second section involves the development of the model and structure of IDAP sections.

The third section covers the design of software.

SOFTWARE DESIGN, COMPANY OVERVIEW, IDAP SECTIONS, IDAP SOFTWARE PRODUCTS, SOFTWARE DEVELOPMENT METHODS.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

ПЗ – програмне забезпечення;
ПК – персональний комп’ютер.

Backend – внутрішня, прихована від користувача начинка сайту або веб-програми;
DevOps – development & operations (IT-фахівець);
Frontend – це інтерфейс для взаємодії між користувачем і backend;
QA – quality assurance (тестувальник);
UI – user interface (призначений для користувача інтерфейс);
UX – user experience (враження від роботи з інтерфейсом).

ВСТУП

Розробка програмного забезпечення визначається комплексом рішень, які визначають його майбутню ефективність, надійність та зручність підтримки. Одним із визначальних аспектів є вибір оптимальної архітектури, що стає фундаментальним каменем успіху проекту. У цьому контексті, оптимальна архітектура – це така, що гармонійно вплітається в специфіку проекту, сприяє зручності розробки та підтримки, а також забезпечує готовність до майбутніх змін.

В сучасному цифровому віці, коли програмне забезпечення проникає в усі аспекти життя, від економіки і науки до повсякденних рутинних завдань, важливість оптимальної архітектури програм стає ключовою. Розробка програмного забезпечення є складним завданням, що вимагає глибокого розуміння як технічних, так і стратегічних аспектів, адже вона має значний вплив на ефективність та успішність бізнес-процесів та проектів.

У цьому обсязі ми подамо обширний огляд методів розробки програмного забезпечення, зосереджуючись на їхньому зв'язку з оптимальною архітектурою. З перших кроків проектування до фінальної реалізації, ми вивчимо різноманітні стратегії, методології та інструменти, які впливають на формування архітектури програм.

Розглядаючи традиційні та інноваційні підходи до розробки, ми проведемо докладний аналіз важливих концепцій, таких як модульність, масштабованість, безпека та ефективність. Вивчимо принципи та практики, що дозволяють досягти оптимальної архітектури, і дослідимо вплив цих аспектів на якість та продуктивність програмного забезпечення.

Також буде проводитись огляд методів розробки програмного забезпечення на базі оптимальної архітектури, зокрема, на прикладі компанії IDAP та її програмних продуктів. Для досягнення цієї мети розглянемо загальний опис компанії IDAP, її різновиди та модель спіральної динаміки для бізнесу. Проведемо детальний огляд структури компанії та програмного забезпечення, яке вона використовує для реалізації своїх продуктів.

Ця робота присвячена вивченню найсучасніших технологій та підходів, спрямованих на оптимізацію розробки програмного забезпечення. Надаючи читачеві глибоке розуміння інструментів та стратегій, які формують оптимальну архітектуру програм, ми маємо на меті надати комплексний погляд на сучасний ландшафт розробки програмного забезпечення.

Актуальність роботи «Методи розробки програмного забезпечення на базі оптимальної архітектури» визначається необхідністю адаптації розробницьких підходів до зростаючих вимог сучасного інформаційного середовища. Розвиток технологій, високий темп змін у галузі програмного забезпечення та збільшення обсягів данних ставлять перед розробниками нові виклики та завдання.

Оптимальна архітектура в програмному забезпеченні стає ключовим фактором для досягнення цих цілей. Забезпечення ефективної архітектури стає завданням важливим не лише для розробників програмного забезпечення, але й для бізнесу та кінцевих користувачів. Раціонально побудована архітектура дозволяє створювати продукти, які відповідають найвищим стандартам продуктивності, надійності та швидкості розгортання.

Мета і завдання дослідження. Метою даного дослідження є вивчення та аналіз методів розробки програмного забезпечення на базі оптимальної архітектури з метою вдосконалення процесів створення програмних продуктів.

Основні завдання:

1. Виявлення сучасних тенденцій у розробці програмного забезпечення.
2. Аналіз та класифікація поточних тенденцій у галузі розробки програмних продуктів.
3. Систематизація методів розробки ПЗ на базі оптимальної архітектури.
4. Визначення основних принципів та підходів до вибору та реалізації оптимальних архітектурних рішень.
5. Аналіз впливу архітектурних рішень на характеристики програмного забезпечення.
6. Встановлення взаємозв'язку між обраними архітектурними підходами та властивостями програмних продуктів.
7. Оцінка ефективності розроблених програмних рішень.
8. Розгляд можливостей та інструментів для вимірювання ефективності та якості програмного забезпечення.
9. Розробка рекомендацій для вдосконалення процесу розробки ПЗ на підставі оптимальних архітектурних рішень.

Формулювання практичних порад та рекомендацій для впровадження оптимальних архітектурних підходів у процес розробки програмного забезпечення.

Ці завдання є ключовими для досягнення загальної мети дослідження та визначення практичних кроків для оптимізації процесів розробки програмного забезпечення на основі оптимальної архітектури.

Об’єкт дослідження у рамках теми «Методи розробки програмного забезпечення на базі оптимальної архітектури» включає сам процес розробки програмного забезпечення та його етапи, а також конкретні аспекти оптимізації архітектури програм для досягнення кращої продуктивності та ефективності.

Об'єкт дослідження в даному випадку охоплює всі важливі складові, пов'язані з розробкою програмного забезпечення, включаючи вибір оптимальної архітектури, використання сучасних методик та технологій, а також аспекти, пов'язані із забезпеченням високої якості та конкурентоспроможності розроблених продуктів.

Предмет дослідження у контексті теми «Методи розробки програмного забезпечення на базі оптимальної архітектури» визначає область, на яку спрямована увага магістерської роботи. В даному випадку предметом дослідження є сукупність властивостей і відношень об’єкта, який включає процес розробки програмного забезпечення з використанням оптимальних архітектурних рішень.

Основні аспекти предмета дослідження включають:

1. Процес розробки ПЗ: вивчення етапів, методологій та підходів, що використовуються в розробці програмного забезпечення.
2. Оптимальна архітектура: аналіз та порівняння різних архітектурних рішень, їх переваги та недоліки, вибір оптимальних підходів для конкретних завдань.
3. Властивості програмного забезпечення: дослідження характеристик програм, таких як продуктивність, масштабованість, надійність та інші.
4. Технології та інструменти: використання сучасних технологій та інструментів у розробці для досягнення оптимальної архітектури.
5. Управління якістю: аналіз методів та підходів до забезпечення високої якості програмного забезпечення з точки зору архітектурних рішень.

Методи досліджень.

Для досягнення мети магістерської роботи, спрямованої на вивчення методів розробки програмного забезпечення на базі оптимальної архітектури, використовувалися різноманітні методи дослідження. Нижче перераховані основні методи, що використовувалися в рамках даного дослідження:

1. Літературний аналіз.

Проведено аналіз наукових публікацій, статей, книг, тез та інших джерел, що стосуються оптимальних архітектурних підходів у розробці програмного забезпечення.

2. Кейс-стаді.

Вивчено практичні приклади успішної реалізації оптимальних архітектур у великих та складних програмних проектах. Розглянуто випадки відомих компаній та проектів, які використовують оптимальні архітектурні підходи.

3. Експериментальне дослідження.

Проведено серію експериментів для визначення впливу різних архітектурних рішень на продуктивність, масштабованість та інші характеристики розроблених програмних продуктів.

4. Опитування та інтерв'ю.

Здійснено опитування та інтерв'ю з досвідченими розробниками програмного забезпечення, архітекторами систем та іншими фахівцями для отримання експертної думки та практичних порад.

5. Моделювання.

Використано інструменти моделювання архітектури для аналізу та порівняння різних архітектурних варіантів під час розробки програмного забезпечення.

6. Аналіз статистичних даних.

Застосовано методи статистичного аналізу для оцінки результатів експериментів та визначення статистично значущих відмінностей між різними архітектурними рішеннями.

Ці методи дозволили отримати комплексне розуміння та детальний аналіз оптимальних архітектурних підходів у розробці програмного забезпечення.

Наукова новизна одержаних результатів.

Наукова новизна дослідження полягає в наступних аспектах:

1. Поглиблення Теоретичного Розуміння:

Розширено та уточнено теоретичне розуміння оптимальних архітектурних підходів у розробці програмного забезпечення. Вивчено та систематизовано нові тенденції, що виникають у цій галузі.

2. Інтеграція Емпіричних Даних:

Подано результати широкого спектру емпіричних досліджень, що включають в себе аналіз кейс-стадів, експериментів та опитувань. Це надає новий вигляд на практичне застосування оптимальних архітектурних рішень у реальних проектах.

3. Методологія Оцінки Продуктивності:

Розроблено новий підхід до оцінки продуктивності програмного забезпечення з урахуванням його архітектурних особливостей. Застосовано статистичні методи для об'єктивного порівняння різних архітектурних рішень.

Практичне значення отриманих результатів.

Практичне значення отриманих результатів цього дослідження виявляється в кількох аспектах, що можуть сприяти вдосконаленню практик розробки програмного забезпечення:

Оптимізація Процесу Розробки.

Результати дослідження надають нові підходи та рекомендації для оптимізації процесу розробки програмного забезпечення, спрощуючи вибір оптимальних архітектурних рішень та зменшуючи час на їх прийняття.

Підвищення ефективності робочих груп:

Висновки дослідження можуть слугувати як основа для вдосконалення взаємодії у робочих групах розробників, надаючи їм інструменти та критерії для спільного вибору та реалізації оптимальних архітектурних рішень.

Підтримка Прийняття Рішень.

Рекомендації дослідження можуть слугувати як керівництво для прийняття обґрунтованих рішень з питань архітектури під час розробки проектів програмного забезпечення.

Створення високоякісних продуктів.

Застосування вивчених підходів та рекомендацій може призвести до створення високоякісних програмних продуктів, які відповідають сучасним стандартам та вимогам користувачів.

Апробація роботи.

1. Кікачеішвілі Б.С., Музика І.О Комплексні методи захисту локальної комп’ютерної мережі підприємства. XVI Всеукраїнська науково-практична WEB конференція аспірантів, студентів та молодих вчених «Комп’ютерні інтелектуальні системи та мережі» : 21–23 березня 2023 р. : матер. Кривий Ріг, 2023. С. 54–56.

1 ОГЛЯД КОМПАНІЇ IDAP ТА ЇЇ ПРОГРАМНИХ ПРОДУКТІВ

1.1 Загальний опис та різновиди IT-компанії

IT компанії – це підприємства, які спеціалізуються на розробці програмного забезпечення, розробці веб-сайтів, створенні мобільних додатків та інших інструментів. Вони можуть мати власну команду розробників та тестувальників, або працювати зі збалансованою командою фрілансерів.

Аутсорсинг – це стратегія бізнесу, при якій компанія використовує послуги зовнішнього партнера (аутсорсера) для виконання певних функцій або завдань, які раніше виконувалися внутрішньо. Замість того, щоб мати внутрішні відділи або команди, які виконують певні завдання, компанія передає ці функції або послуги під контроль зовнішнього партнера (рис.1.1).

Основні причини використання аутсорсингу включають економію коштів, спеціалізовану експертизу, фокус на основних функціях бізнесу та можливість зосередитися на стратегічних завданнях. Аутсорсінг може охоплювати різні аспекти бізнесу, такі як IT-послуги, бухгалтерія, кадрове управління, маркетинг, логістика та інше [1].

Цей підхід дозволяє компаніям покращити ефективність, знизити витрати, зосередитися на стратегічних завданнях та використовувати зовнішній досвід та ресурси для вирішення конкретних завдань. Аутстаф - це процес найму фахівців для роботи в компанії на повний робочий день, але ці співробітники працюють від імені та на замовлення замовника. Таким чином, аутстаф-компанії надають підприємствам можливість наймати фахівців з різних галузей на повний робочий день, без необхідності забезпечувати їх заробітну плату та інші соціальні вигоди.

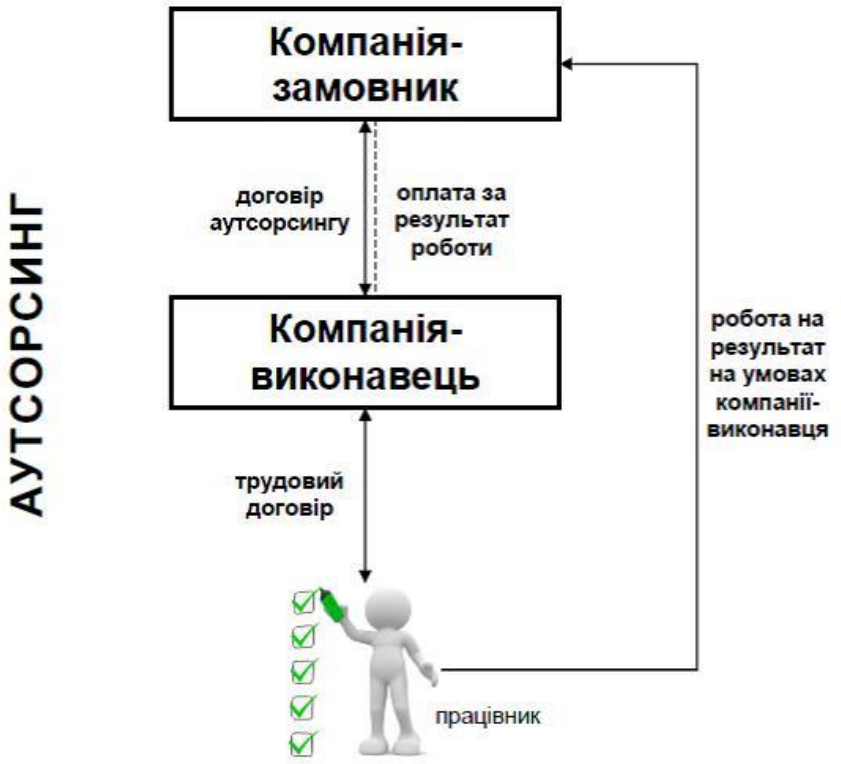

Рисунок 1.1 – Схема роботи аутсорсингу

Переваги аутсорсингу:

1. Економія витрат: однією з головних переваг є можливість значного зменшення витрат. Зовнішні постачальники послуг часто можуть пропонувати свої послуги за більш доступними цінами, особливо якщо вони базуються у країнах з меншими витратами на працю.
2. Спеціалізована експертиза: аутсорсинг дозволяє компанії отримати доступ до висококваліфікованих фахівців із спеціалізованою експертизою, яку може бути важко знайти внутрішньо [1].
3. Фокус на основних функціях: компанія може зосередитися на своїх стратегічно важливих функціях, переклавши рутинні або спеціалізовані завдання на зовнішніх партнерів.
4. Гнучкість і масштабованість: аутсорсинг дозволяє легко масштабувати або зменшувати обсяги робіт відповідно до потреб компанії.

Недоліки аутсорсингу:

1. Втрата контролю: передача функцій під контроль зовнішнього партнера може призвести до втрати контролю над процесами та якістю.

2. Ризик конфіденційності: при передачі конфіденційної інформації стороннім партнерам існує ризик її розголошення або неналежного використання.
3. Труднощі в управлінні відстанню: управління відстанню може бути викликом, особливо якщо зовнішні постачальники базуються в інших країнах.
4. Залежність від зовнішніх факторів: зміни в політиці, економіці або інших зовнішніх чинниках можуть вплинути на діяльність зовнішніх партнерів і, відповідно, на компанію.
5. Можливість втрати корпоративної культури: розподіл функцій на зовнішні партнери може призвести до втрати корпоративної культури та єдності в команді.

Аутстафінг є формою управління персоналом, де компанія укладає угоду із сторонньою фірмою (аутстафінговою агенцією), щоб передати своїх співробітників на роботу в цю агенцію [2]. Цей підхід реалізується через визначення спеціальних умов та обов'язків у контракті між компанією-клієнтом і зовнішньою агенцією, яка бере на себе управління цим персоналом (рис.1.2).

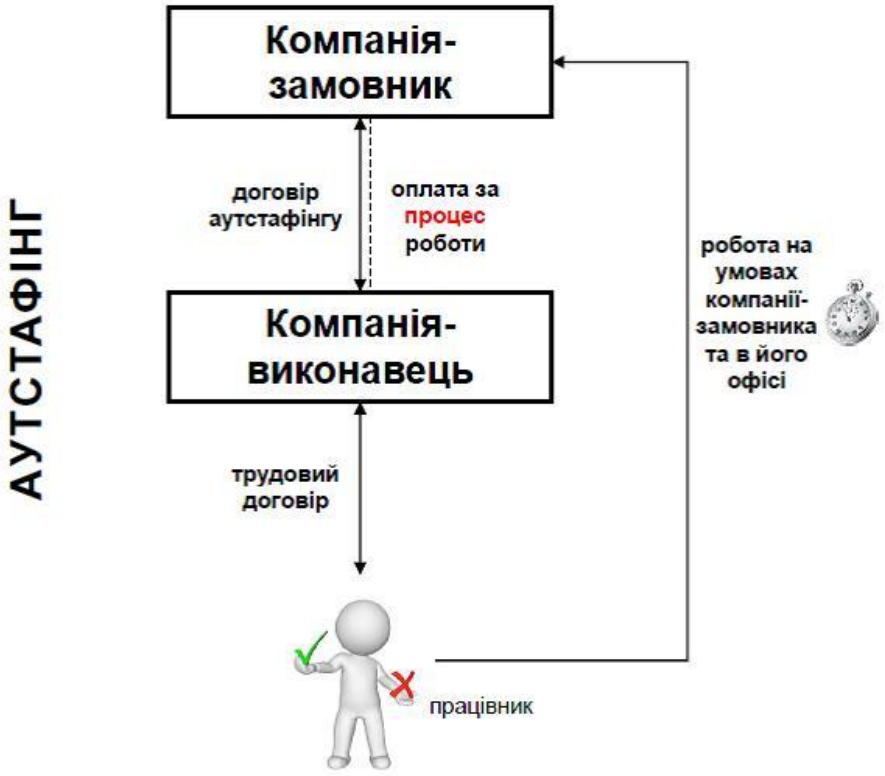

Рисунок 1.2 – Схема роботи аутстафінг

Переваги Аутстафінгу:

1. Ефективність витрат: компанії можуть заощаджувати на оплаті заробітної плати та інших витратах на утримання персоналу, так як оплата за аутстафінгові послуги може бути менше, ніж витрати на внутрішній персонал.
2. Гнучкість у використанні робочої сили: компанії можуть легко змінювати кількість працівників відповідно до своїх потреб, не залучаючи додатковий внутрішній персонал або не звільняючи зайвих співробітників.
3. Доступ до талантів: аутстафінгові агенції можуть мати доступ до широкого спектру фахівців, що дозволяє підприємствам залучати висококваліфікованих працівників із різних галузей.
4. Спрощення управління персоналом: управління аспектами, такими як оплата заробітної плати, податки та інші адміністративні питання, може бути делеговане аутстафінговій компанії [2].

Недоліки Аутстафінгу:

1. Втрата контролю: компанія може втратити частину контролю над своїм персоналом, оскільки вони працюють на іншу компанію.
2. Ризик конфіденційності: велика кількість персоналу, який переходить до аутстафінгової агенції, може збільшити ризик витоку конфіденційної інформації.
3. Нестабільність: угоди про аутстафінг можуть бути менш стабільними, оскільки підприємство залежить від умов контрактів із зовнішніми агентствами.
4. Специфічні вимоги до управління: управління відносин

Продакт компанії, або продуктова лінійка, представляє собою асортимент товарів або послуг, які компанія пропонує на ринку. Це може включати в себе основні продукти чи послуги, а також їхні модифікації, варіації та додаткові послуги.

Ключові аспекти продуктової компанії:

1. Асортимент продукції.

Основні продукти: це основні товари чи послуги, які компанія пропонує на ринку.

Додаткові опції: різноманітні варіанти, які дозволяють клієнтам вибирати серед різних характеристик, функцій чи властивостей продукції.

Апгрейди: можливості покращення продуктів, щоб задовольнити змінюючіся потреби споживачів [3].

2. Політика ціноутворення.

Стратегії ціноутворення: визначення того, як встановлюються ціни на продукцію, включаючи конкурентоспроможність, преміальність чи стратегії диференціації.

Знижки та акції: можливості залучення клієнтів через різні форми знижок, акцій та бонусних програм.

3. Маркетингові стратегії.

Реклама: розробка ефективних рекламних кампаній для залучення уваги та підвищення усвідомленості бренду.

Брендування: створення та підтримка сильного бренду, який відрізняє компанію від конкурентів.

Продажі: використання різних каналів збуту та стратегій продажу для досягнення максимального обсягу продажів.

4. Дослідження та Розробка.

Створення нових продуктів: інвестування в дослідження для створення нових інноваційних продуктів.

Вдосконалення існуючих: постійне удосконалення та оновлення існуючих продуктів для відповіді на змінюючіся потреби ринку.

Впровадження технологій: застосування нових технологій для поліпшення продукції та забезпечення конкурентоспроможності [3].

5. Споживчі Властивості.

Задовольнення потреб та очікувань: гарантування того, що продукція відповідає потребам та очікуванням клієнтів.

Якість: забезпечення високої якості продуктів для збереження довгострокових відносин з клієнтами.

Інновації: впровадження новаторських елементів та унікальних функцій для залучення споживачів.

Переваги продуктової компанії.

1. Широкий асортимент: можливість задовольнити різні потреби різних сегментів ринку.
2. Брендування: змога побудувати сильний бренд за допомогою вдосконалених та відомих продуктів.
3. Гнучкість у ціноутворенні: можливість експериментувати з цінами та створювати пакетні пропозиції для різних клієнтів.
4. Стійкість на ринку: здатність адаптуватися до змін у смаках та вимогах споживачів.

Недоліки продуктової компанії.

1. Високі витрати на дослідження та розробку: необхідність інвестування у створення та покращення продуктів.

2. Ризик неуспіху нового продукту: нові продукти можуть не завжди приносити очікувані прибутки.
3. Конкуренція на ринку: змагання з іншими компаніями за увагу та лояльність клієнтів.
4. Обмежена гнучкість в обслуговуванні клієнтів: важко задовольнити індивідуальні потреби кожного клієнта [3].

Основна Різниця.

Сфера діяльності.

Аутсорсинг: сфера діяльності охоплює передачу бізнес-процесів чи завдань зовнішнім підрядникам.

Аутстафінг: основний аспект - це виведення внутрішнього персоналу на роботу в агенцію.

Продуктова компанія: зосереджена на створенні та продажу власних продуктів.

Використання персоналу.

Аутсорсинг: персонал використовується для виконання конкретних завдань або функцій.

Аутстафінг: компанія використовує персонал агенції на своїй території під керівництвом агентства.

Продуктова компанія: зосереджена на розробці та продажу власних продуктів.

Фокус бізнесу.

Аутсорсинг: фокус на оптимізації бізнес-процесів та зниженні витрат.

Аутстафінг: спрощення управління персоналом та зменшення адміністративних завдань.

Продуктова компанія: фокус на розробці та просуванні власних продуктів.

Кожен з цих підходів має свої конкретні цілі та переваги, залежно від потреб бізнесу.

1.2 Модель спіральної динаміки для бізнесу

У моделей IT-компаній існують різноманітні відмінності, такі як розмір, здатність обслуговування клієнтів, маркетингові стратегії та інші фактори. Для більшого розуміння їх потреб часто корисно мати чітке уявлення про саму компанію. Однією з широко відомих моделей є «Спіральна динаміка». Ця концепція дозволяє класифікувати організації за рівнями розвитку та взаємодії з їхніми співробітниками та клієнтами.

Спіральна динаміка – це теорія розвитку культур та організацій, яка описує рівні свідомості та значення, що присвоюються різними людьми та організаціями у суспільстві [5].

У рамках компанії Спіральна динаміка описує, які значення та підходи присутні серед працівників та управління, та як вони можуть впливати на ефективність та успіх організації в цілому. Згідно з теорією Спіральної динаміки, кожен рівень розвитку має свої унікальні характеристики та цінності. Рівні можна поділити на дві групи – «масові» рівні, які відповідають за масову культуру та свідомість, та «індивідуальні« рівні, які більш виражені у суспільстві в якості окремих індивідів та організацій. Наприклад, на перших рівнях працівники можуть бути більш орієнтовані на виконання своїх обов'язків та бути менш самостійними. На рівні системного мислення (сьомий рівень) працівники можуть відчувати більшу відповідальність за результат своєї роботи та бути більш творчими та інноваційними у своїх підходах (рис.1.3).

Рисунок 1.3 – Модель спіральної динаміки

Перша ступінь теорії «Спіральна динаміка» відома як або «рівень виживання». Це рівень, який відображає початкові етапи еволюції людини, коли люди зосереджувались на задоволенні своїх базових фізіологічних потреб і виживанні в своєму середовищі.

Характерні риси «виживання» включають простий спосіб мислення, інстинктивне сприйняття світу та схильність до поведінки, орієнтованої на виживання [5].

Колективи на рівні «виживання» можуть зосереджуватись на масовому виробництві та створенні масового ринку, або де пріоритетним є

максимізація прибутку та ефективності. Ці колективи можуть мати низький рівень інноваційності та консервативний підхід до розвитку. Також треба зауважити що це саме колективи а не компанії, тому що в більшості випадків це є маленька група людей, або навіть мікробізнес де співробітником є тільки власник.

Другий рівень Спіральної динаміки відомий як «пурпуровий» або «анімістичний». Цей рівень характеризується з'явленням племінних суспільств та розвитком магічного мислення та анімістичних переконань.

На пурпуровому рівні люди мають тенденцію жити в згуртованих спільнотах та надають велике значення традиції та ритуалу. Вони сприймають світ через призму магії та містики, вірять у духів, привидів та надприродні сили, що впливають на їхнє життя.

Організації на пурпуровому рівні Спіральної динаміки можуть мати структуру, подібну до кланової, з сильними зв'язками з традицією та звичаєм. Управління на цьому рівні може базуватись на племінних лідерах чи старшинах, а процес прийняття рішень часто керується інтуїцією та містичними переконаннями.

Компанії на пурпуровому рівні Спіральної динаміки можуть зосереджуватись на наданні товарів та послуг, що відповідають переконанням та цінностям їхніх спільнот. Вони можуть також пріоритезувати добробут своїх працівників та їхніх сімей, а також збереження свого культурного спадку. Іноді може здатися, що пурпуровий рівень мислення та поведінки є дещо застарілим або обмеженим, однак він може бути корисним у відповідних контекстах, наприклад, у збереженні традицій та культурної спадщини, або у створенні спільнот, які мають сильні зв'язки між собою та піклуються про добробут кожного її члена.

Третій рівень Спіральної динаміки називається «червоний». Цей рівень пов'язаний із зародженням першого індивідуалізму та сильним наголосом на владу та контроль над оточуючим середовищем [5].

На рівні червоного взаємодії людей стають конкурентами, які стежать тільки за своїми інтересами та прагнуть отримати владу над іншими. Люди на цьому рівні мають сильний індивідуальний егоїзм та вважають, що їх особисті інтереси та потреби мають бути задоволені в будь-який спосіб.

Організації на рівні червоного зазвичай мають централізовану структуру та виражену ієрархію. Управління на цьому рівні зосереджується в руках лідера, який контролює процеси та приймає рішення від імені всієї організації. Робітники на цьому рівні зазвичай працюють на зарплату та не мають значного впливу на процеси виробництва.

Компанії на рівні червоного зазвичай фокусуються на збільшенні своєї влади та впливу на ринку, а також на максимізації прибутку. Їхній бізнес-підхід може бути спрямований на підкорення ринку, конкуренцію та перевагу над конкурентами.

Хоча червоний рівень Спіральної динаміки має дещо егоїстичну та конкурентну природу, він є важливим кроком у розвитку людської свідомості та культури. Це перехід від племінних спільнот до індивідуалізму та

самостійності, що в подальшому розвитку відкриває нові можливості для інновацій та розвитку.

Важливо зауважити, що червоний рівень Спіральної динаміки може мати як позитивні, так і негативні аспекти. З одного боку, він може сприяти створенню ефективних лідерів та підприємств, які забезпечують економічний розвиток та зростання. З іншого боку, червоний рівень може також призвести до зловживання владою, занадто високого рівня конкуренції та індивідуалізму, які можуть завдати шкоди як окремим індивідам, так і суспільству в цілому.

Загалом, третій рівень Спіральної динаміки є важливим етапом у розвитку людської свідомості та культури, який відкриває нові можливості для розвитку та інновацій. Проте, як і на кожному рівні, важливо розуміти, що кожна людина та організація мають свій унікальний шлях розвитку, тому важливо розглядати Спіральну динаміку як інструмент для розуміння та пояснення, а не для оцінювання та порівняння [5].

Четвертий рівень Спіральної динаміки називається «синій». На цьому рівні значення приділяються порядку, дисципліні та дотриманню правил, а головна мета – забезпечити стабільність та безпеку.

Люди на рівні синього мислення вірять у справедливість, закони та порядок. Їм важливо дотримуватися правил та процедур, тому що це забезпечує відчуття стабільності та безпеки. На цьому рівні значення приділяються державі, релігії, традиціям та іншим формам організації.

Організації на рівні синього зазвичай мають сильну структуру та виражену ієрархію. Вони дотримуються встановлених процедур та правил та вірять у стабільність та безпеку, що забезпечується за рахунок дисципліни та контролю.

Компанії на рівні синього зазвичай фокусуються на створенні довгострокових планів та стратегій розвитку, забезпеченні дисципліни та порядку в роботі, а також на створенні стабільної платформи для забезпечення розвитку.

Важливо зауважити, що синій рівень Спіральної динаміки може мати як позитивні, так і негативні аспекти. З одного боку, синій рівень може допомогти забезпечити стабільність та безпеку в організації, а також в суспільстві в цілому. З іншого боку, занадто великий фокус на дотриманні правил та процедур може призвести до догматичного мислення та відчуття відокремленості від інших.

Компанії на рівні синього Спіральної динаміки можуть бути здатні до довгострокового планування та стратегічного розвитку. Вони можуть створювати високоякісні продукти та послуги, які дотримуються встановлених стандартів та вимог. Організації на рівні синього також можуть бути ефективними у вирішенні проблем та конфліктів, оскільки вони мають ясні процедури та правила для управління ситуаціями.

Проте, важливо зазначити, що рівень синього мислення може бути обмежувальним, особливо в умовах швидкої зміни та нестабільності. Організації, які занадто дотримуються процедур та правил, можуть не мати

достатньої гнучкості та адаптивності до змін на ринку та у суспільстві в цілому.

Загалом, синій рівень Спіральної динаміки відображає значення стабільності та безпеки в організації та суспільстві. Він є важливим етапом у розвитку, що допомагає створити фундамент для подальшого розвитку та інновацій. Однак, як і на кожному рівні, важливо розуміти, що кожна людина та організація мають свій унікальний шлях розвитку.

П'ятий рівень Спіральної динаміки називається «помаранчевий». Це рівень, на якому значення приділяється досягненням, інноваціям та особистій свободі [5].

Люди на рівні оранжевого мислення прагнуть до високих досягнень, індивідуальної свободи та особистої вигоди. Вони розуміють світ через призму логіки та раціональності, вірять у самовираження та відкритість до нових ідей. Люди на цьому рівні можуть бути пристрасними підприємцями, лідерами або інноваторами.

Організації на рівні оранжевого мислення можуть мати децентралізовану структуру та бути фокусовані на досягненні цілей та результатів. Управління на цьому рівні може бути досить свобідним та гнучким, а рішення можуть прийматися на основі логіки та аналітики.

Компанії на рівні оранжевого мислення зазвичай фокусуються на інноваціях та створенні високоякісних продуктів, що задовольняють вимоги ринку та забезпечують високий рівень ефективності. Ці компанії можуть бути більш гнучкими та адаптивними до змін, ніж компанії на попередніх рівнях Спіральної динаміки.

Оранжевий рівень Спіральної динаміки є важливим етапом у розвитку, оскільки він сприяє інноваціям та досягненню нових рівнів ефективності. Проте, важливо зазначити, що оранжевий рівень може мати свої негативні наслідки, такі як більш високий ризик стереотипного мислення та загострення індивідуалістичного підходу до управління. Крім того, помаранчевий рівень може привести до зростання соціальної нерівності та несправедливості, оскільки фокус на досягненнях та особистій вигоді може приводити до знехтування потребами менш привілейованих груп.

Загалом, оранжевий рівень Спіральної динаміки відображає зростаючу роль інновацій, ефективності та самовираження в організації та суспільстві. Це важливий етап у розвитку, що допомагає забезпечити ефективність та здатність організації адаптуватися до змін.

Шостий рівень Спіральної динаміки, також відомий як «зелений» рівень, більше стосується соціальної справедливості та духовності. Цей рівень характеризується розумінням того, що існує взаємозв'язок між людьми.

Люди на рівні зеленого мислення прагнуть до рівності та справедливості, розвивають відкритість до інших культур та стиль життя, що більше спрямований на «екологічні»та сталий розвиток. Вони вірять у співпрацю та колективну дію та дбайливе ставлення до один одного.

Організації на рівні зеленого мислення можуть мати децентралізовану структуру, де важливі рішення приймаються через консенсус та демократичний процес. Управління на цьому рівні може бути більш співпрацювальним та орієнтованим на потреби співробітників та громадськості.

Компанії на рівні зеленого мислення можуть бути спрямовані на створення продуктів та послуг, що допомагають зберегти навколишнє середовище та зменшити екологічний вплив. Також, компанії можуть працювати з місцевими спільнотами та здійснювати благодійні проекти з метою підтримки соціально вразливих групп [5].

Зелений рівень Спіральної динаміки є важливим етапом у розвитку, оскільки він сприяє розвитку соціальної справедливості та свідомості про сталість розвитку. Цей рівень також позитивно впливає на розвиток нових форм управління та співпраці, які можуть стати важливим інструментом для створення більш сталого та справедливого світу.

Однак, важливо зазначити, що зелений рівень Спіральної динаміки також може мати свої негативні наслідки. Наприклад, недостатнє увага до ефективності та досягнення результатів може призвести до зменшення конкурентоспроможності компаній на цьому рівні. Також, іноді розгляд зеленого рівня може призвести до відкидання технічних рішень та інновацій у підтримку більш екологічних та соціальних рішень.

Загалом, зелений рівень Спіральної динаміки є важливим етапом у розвитку людської свідомості та культури, оскільки він привертає увагу до соціальних та екологічних проблем та стимулює розвиток нових форм управління та співпраці.

Сьомий рівень Спіральної динаміки називається «жовтий» рівень. Цей рівень характеризується зростаючою складністю мислення та розумінням взаємозв'язку між різними системами та процесами.

Люди на рівні жовтого мислення починають бачити світ як складну систему, в якій все взаємопов'язане та залежне одне від одного. Ці люди віддають перевагу гнучким та адаптивним рішенням, а також відкриті до нових ідей та підходів.

Організації на рівні жовтого мислення можуть мати децентралізовану структуру, де прийняття рішень базується на гнучких та інноваційних підходах. Управління на цьому рівні орієнтоване на досягнення стратегічних цілей та збереження конкурентоспроможності компанії.

Компанії на рівні жовтого мислення можуть бути спрямовані на створення продуктів та послуг, що відповідають потребам ринку та споживачів, а також зменшення «екологічного» впливу своєї діяльності. Такі компанії можуть використовувати нові технології та інновації для досягнення своїх цілей.

Однак, з жовтим рівнем Спіральної динаміки пов'язані й деякі виклики та ризики. Люди на цьому рівні можуть бути спокушені складністю та досконалістю, що може призвести до затримки прийняття рішень та розробки продуктів.

Також, компанії можуть стикатися з труднощами в управлінні та координації діяльності через складну структуру та різноманітність проектів.

Восьмий рівень Спіральної динаміки називається «білий» рівень. Цей рівень характеризується зростаючою свідомістю та розумінням всесвіту та космосу.

Люди на рівні білого мислення починають розуміти, що все у світі взаємопов'язане та залежне одне від одного, і розглядають світ як одну велику систему. Ці люди мають глибоке розуміння принципів науки та техніки, а також мають сильну духовну складову та багато різноманітних інтересів.

Організації на рівні білого мислення можуть мати децентралізовану структуру, що дає можливість керівництву використовувати інноваційні технології та методи, щоб досягнути стратегічних цілей. Управління на цьому рівні орієнтоване на досягнення більш високих рівнів ефективності та сталості діяльності.

Компанії на рівні білого мислення можуть бути спрямовані на розробку продуктів та послуг, які сприяють розвитку суспільства та екологічній сталості. Такі компанії можуть бути лідерами у використанні нових технологій та методів роботи, а також мати сильну соціальну складову.

Важливо зазначити, що білий рівень Спіральної динаміки є рідкісним, і його представники можуть зустрічатися тільки в обмеженій кількості в суспільстві. Цей рівень є дуже важливим для розвитку людської культури та свідомості, оскільки він сприяє розвитку науки, технологій, інновацій та соціальної відповідальності. Компанії, які працюють на рівні білого мислення, можуть бути відкритими до нових ідей та принципів роботи, що дає їм можливість досягати більш високих рівнів ефективності та досягати успіхів в різних сферах [5].

Однак, з восьмим рівнем Спіральної динаміки пов'язані й деякі виклики та ризики. Люди на цьому рівні можуть бути вкрай вимогливі та претензійні, що може викликати труднощі в управлінні та координації діяльності. Також, компанії на цьому рівні можуть стикатися з великими витратами на дослідження та розробку нових продуктів та послуг.

Важливо пам'ятати, що Спіральна динаміка є теорією, яка може допомогти розуміти та пояснити еволюцію людських цінностей та світогляду, але не є жорстким правилом або шаблоном для кожної людини чи організації. Кожна людина та організація мають свій унікальний шлях розвитку, тому важливо розглядати Спіральну динаміку як інструмент для розуміння та пояснення, а не для оцінювання та порівняння.

1.3 Огляд структури компанії IDAP

Одна з цілей цієї роботи є створити модель IT компанії по розробці додатків на базі оптимальної архітектури. Один з методів створення моделі є дослідження існуючого об‘єкта, та документація процесів. Тож за існуючий об‘єкт буде взята компанія IDAP. Спираючись на вищесказане опишемо існуючу компанію.

IDAP – українська компанія повного циклу розробки програмного забезпечення, яка заснована у 2012 році. Компанія спеціалізується на розробці нативних додатків для iOS, Android та веб-платформ, займається фронтендом та бекендом, а також забезпечує повне тестування продукту перед його випуском [6].

За більше ніж десять років роботи IDAP успішно здійснила більше 200 проектів в різних сферах: від роздрібної торгівлі до охорони здоров'я та транспортування. Компанія працює зі стартапами, відомими бізнесами та відомими брендами, надаючи їм якісні та інноваційні рішення, які дозволяють їм бути на крок попереду у своїх галузях.

У компанії більше 100 технічних фахівців, які створюють технологічні продукти з використанням сучасних технологій та підходів.

IDAP знаходиться на четвертому рівні спіральної динаміки, бізнес-процеси всередені відповідають цінностям «захисту» та «стабільності». Компанія моє досить ієрархічну структуру управління, а керівництво визначає стратегії та приймати рішення на основі стабільності та підтримки статус-кво. Компанія робить акцент на створенні якісних та інноваційних продуктів з використанням сучасних технологій, але з підхідом, спрямованим на підтримку стабільності та захисту від невизначеності. IDAP зосереджується на ретельному плануванні та контролі за процесами розробки продуктів, щоб гарантувати якість та дотримання строків. Розроблені продукти компанії спрямовані на різні ринки та галузі, але з метою забезпечення стабільності та захисту від ризиків, мають певний рівень консерватизму у своїх рішеннях. Подалі буде розглянуто технічні рішення та процеси котрі використовуються всередині компанії. Для того щоб визначити якими продуктами користується компанія IDAP, спочатку верхорівнемо опишемо процеси всередині компанії.

Почнемо з процесу створення технічного завдання, з ідеї котра пропонується замовником рис. 1.4.

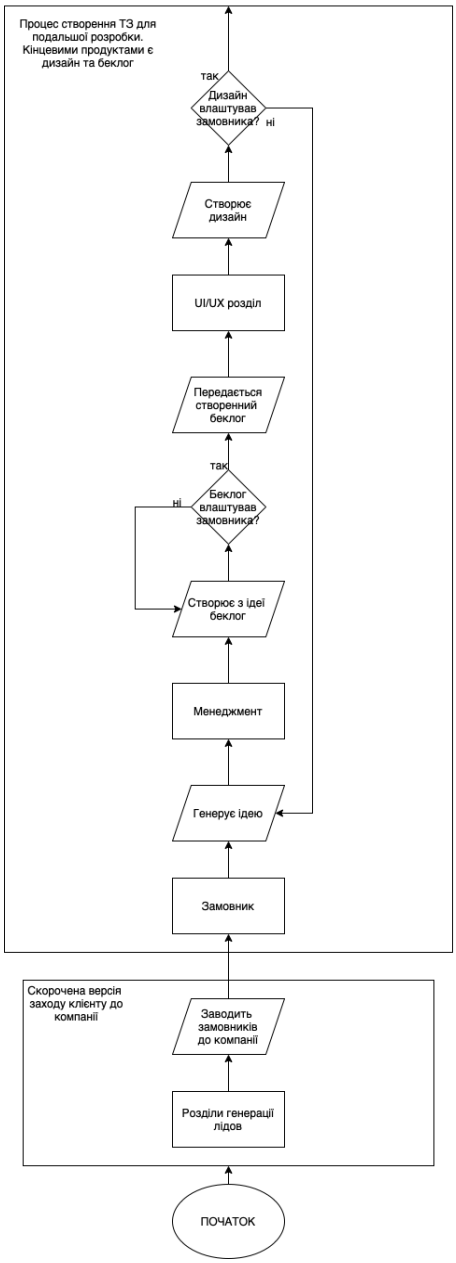

Рисунок 1.4 – Діаграма створення ТЗ

Розглянемо діаграму рис. 1.2. Та внесемо роз‘яснення стосовно використаних термінів. А саме: «генерація лідів», «беклог», «UI/UX», «дизайн».

Генерація лідів – це процес залучення потенційних клієнтів або лідів, які проявляють інтерес до продукту чи послуги компанії. Головна мета генерації лідів полягає в тому, щоб залучити нових клієнтів та збільшити продажі. Це можна зробити через різні канали, такі як веб-сайти, соціальні мережі, рекламні кампанії, електронні листи та інші методи. Компанії використовують різні стратегії генерації лідів, залежно від їхньої галузі та цільової аудиторії. Важливо пам'ятати, що генерація лідів - це лише перший крок в продажу і потребує подальшої роботи з лідами, щоб перетворити їх на реальних клієнтів [6]. Отже розділ з генерації лідів зазвичай займається розробкою та впровадженням стратегій та тактик для залучення потенційних клієнтів. Основна мета такого розділу – це залучити якомога більше лідів, які

проявляють інтерес до продукту або послуги компанії, та підготувати їх для подальшого продажу.

Беклог рис 1.5 – це список завдань або робіт, які ще не були виконані в процесі розробки продукту або проекту. Це може бути список функцій, які потрібно додати до продукту, проблеми, які потрібно вирішити, або будь-які інші завдання, які потрібно виконати, щоб успішно завершити проект. Беклог часто використовується в методології Agile, де він є основним інструментом для планування і контролю процесу розробки продукту. В Agile беклог представляється в вигляді списка завдань, які повинні бути виконані в розробці продукту, і який постійно оновлюється залежно від потреб проекту та змін у вимогах. Беклог є важливим інструментом для комунікації між усіма учасниками проекту і забезпечує більшу прозорість та зрозумілість процесу розробки продукту.

Home screen	Home screen with meeting lists	- Home screen according to design - List of current & nearest meetings (from groups & users that I subscribed to)	12	12	Design: https://www.figma.com/file/mMqV9E7wayfbkwBFq2MLha/CB-one?node-id=2%3A5315
	Recommendations feed - groups (+list with details)	- List of groups based on shared interests - New & active group prioritization	10	8	Based on interests system
	Recommendations feed - contacts (+list with details)	- List of users based on shared interests - Active users prioritization	8	8	
	Recommendations feed - meetings (+list with details)	- Public meetings & meetings from similar groups recommendations	8	8	

Рисунок 1.5 – Приклад беклога

UI/UX – це скорочення від двох понять: User Interface (UI) та User Experience (UX). UI описує те, як інтерфейс продукту виглядає, як користувачі взаємодіють з різними елементами інтерфейсу, які кольори та шрифти використовуються і т.д. UI-дизайнер має зрозуміти, як користувачі будуть використовувати продукт та як його елементи мають бути організовані для досягнення максимальної ефективності та зручності для користувачів. UX зосереджується на тому, як користувачі взаємодіють з продуктом в цілому, на їхньому враженні та задоволенні від використання продукту. UX-дизайнер має створювати продукти, які будуть легкими та зрозумілими для користувачів, мають інтуїтивно зрозумілий та логічний інтерфейс, який дасть користувачам приємний досвід використання продукту. У сучасному світі UI/UX дизайн має велике значення, оскільки забезпечує зручність та ефективність взаємодії користувачів з продуктом, що в свою чергу сприяє підвищенню конкурентоспроможності продукту та його популярності серед користувачів [6].

У контексті UI/UX, «дизайн» рис 1.6 означає процес створення взаємодії користувача з продуктом або сервісом через розробку інтерфейсу та взаємодії з ним. «Дизайн» включає в себе розробку естетичної та функціональної складових інтерфейсу, таких як шрифти, кольори, кнопки, піктограми, макети, та інші елементи, які допомагають користувачам легко зрозуміти та взаємодіяти з продуктом.

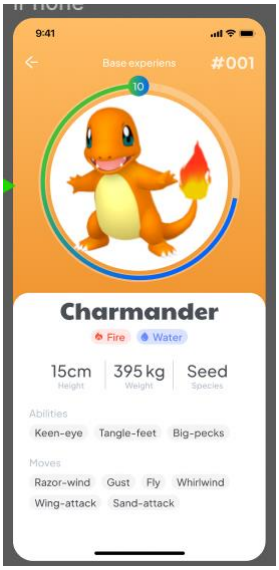

Рисунок 1.6 – Готовий дизайн екрана

«Дизайн» також включає в себе розробку структури та організації інформації, щоб забезпечити легкість навігації та використання продукту. Крім того, дизайн враховує контекст використання продукту, включаючи різні пристрої та ситуації використання, які можуть впливати на “дизайн”. У загальному розумінні, «дизайн» в UI/UX може бути розглянутий як процес створення продукту, який буде зручним та приємним у використанні, та забезпечити користувачам позитивний досвід взаємодії з ним.

Розглянемо подальший рух беклогу та дизайну рис. 1.7. Після подготовки ці об‘єкти надходять до розробки [6].

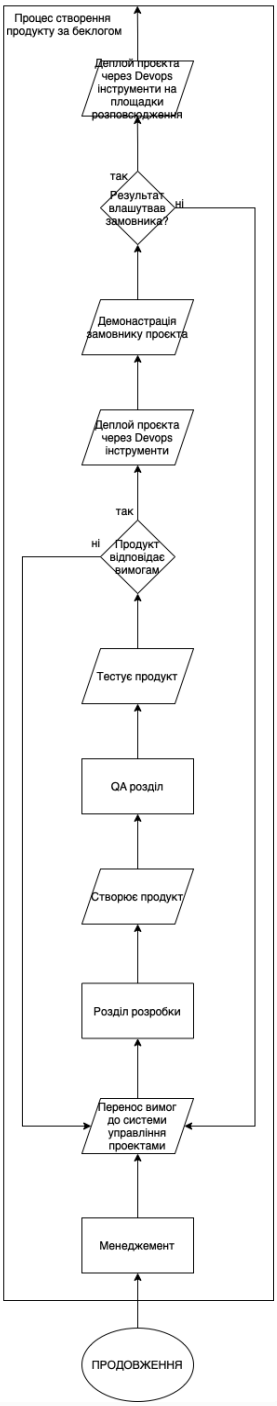

Рисунок 1.7 – Процес створення продукту за беклогом

Розглянемо діаграму рис. 1.7. Та внесемо роз‘яснення стосовно використаних термінів. А саме: «менеджмент», «система управління проектами», «розділ розробки», «продукт», «QA», «тестування», «деплой», «devops», «площадка розповсюдження».

У попередніх діаграмах часто є згадка «менеджмент», туди входять два розділи це: «Business analytics», «Project manager».

Business analytics - це процес збору, обробки та аналізу даних з метою зрозуміти, які фактори впливають на діяльність підприємства, і які дії можуть покращити його результативність. Аналітики бізнесу використовують різноманітні методи, такі як статистичний аналіз, машинне навчання, бізнес-моделювання, дашборди, звіти та інші технології, щоб

зрозуміти, які фактори впливають на бізнес та як їх можна вплинути на підприємство.

У сфері IT-компаній бізнес-аналітики займаються розумінням потреб користувачів, аналізують поведінку клієнтів, розробляють стратегії маркетингу, проводять аналіз ринку та конкурентів, підтримують прийняття рішень з питань розробки продуктів та стратегії компанії.

Крім того, бізнес-аналітики допомагають впроваджувати нові технології та підходи в компанії, оцінюють ефективність різних проектів, допомагають визначити пріоритети та стратегії, щоб компанія могла досягнути максимальних результатів та розширити свій бізнес.

Узагальнюючи, бізнес-аналітики в айті компаніях займаються використанням даних для прийняття рішень, оптимізації діяльності компанії, а також вивченням ринку та конкурентів для забезпечення успішності бізнесу [6].

Project manager – це фахівець, який відповідає за керування проектами в компанії. Їхні обов'язки включають в себе: Планування проектів: проектний менеджер розробляє плани проектів, визначає мету проекту, визначає обсяг робіт, призначає терміни виконання та розробляє розклад робіт. Організація робочих процесів: проектний менеджер розподіляє роботу між учасниками команди, організовує збори та звіти, визначає потрібні ресурси та вирішує питання, що виникають під час роботи. Керування бюджетом: проектний менеджер забезпечує контроль над витратами на проект, визначає бюджет та управляє його розподілом. Керування ризиками: проектний менеджер визначає ризики, які можуть виникнути під час роботи, та розробляє стратегії для їх управління та мінімізації. Комунікація з учасниками проекту: проектний менеджер забезпечує ефективну комунікацію між учасниками команди та замовниками, вирішує конфлікти та веде звіти про прогрес проекту. Контроль та оцінка результатів: проектний менеджер контролює виконання робіт, оцінює результати та забезпечує, щоб вони відповідали поставленим цілям та вимогам замовника. Узагальнюючи, проектний менеджер відповідає за керування проектами в компанії та забезпечення їх успішного виконання, враховуючи бюджет, терміни виконання, ризики та комунікацію з учасниками проекту.

Система управління проектами (Project Management System) – це набір інструментів, методів та процесів, які використовуються для керування проектами в організації. Система управління проектами допомагає організувати та контролювати роботу над проектом від початку до закінчення, забезпечуючи досягнення мети проекту з мінімальними витратами ресурсів та в строк.

Система управління проектами включає в себе такі елементи: Інструменти планування: ці інструменти допомагають розробляти розклад робіт, призначати терміни виконання, призначати ресурси та забезпечувати контроль за їх використанням.

Інструменти контролю: ці інструменти допомагають відстежувати виконання робіт, контролювати витрати, управляти змінами та відповідати на виклики, що виникають під час роботи над проектом. Інструменти комунікації: ці інструменти допомагають забезпечити ефективну комунікацію між учасниками проекту, обмінюватися даними та документами, вирішувати проблеми та конфлікти. Інструменти оцінки: ці інструменти допомагають оцінювати прогрес проекту, визначати успішність проекту відповідно до його мети та вимог замовника, а також забезпечувати звітність та документацію. Застосування системи управління проектами дозволяє ефективно керувати ресурсами, планувати та контролювати роботу над проектом, зменшувати ризики та забезпечувати досягнення мети проекту в строк та з мінімальними витратами ресурсів.

«Розділ розробки» – це структурний підрозділ ІТ-компанії, який відповідає за розробку програмного забезпечення або рішень на основі технологій. Цей відділ складається з команд розробників, які працюють над різними аспектами програмного забезпечення, наприклад, фронтенд та бекенд-розробкою, тестуванням, дизайном і т.д. Зазвичай, "розділ розробки" включає в себе ряд професійних спеціалістів, таких як програмісти, архітектори, тестувальники, дизайнери, проектні менеджери та інші фахівці зі спеціальностей, пов'язаних з розробкою ПЗ. Основні завдання «розділу розробки» включають в себе: Розробка програмного забезпечення: «розділ розробки» відповідає за розробку програмного забезпечення згідно з вимогами клієнтів або потребами компанії. Тестування: "розділ розробки" проводить тестування програмного забезпечення, щоб забезпечити його якість, відповідність стандартам та вимогам замовника. Підтримка: «розділ розробки» забезпечує підтримку та оновлення вже існуючого програмного забезпечення [6]. Вдосконалення: «розділ розробки» досліджує та впроваджує нові технології та методи розробки, щоб забезпечити ефективність та якість роботи. Керування проектами: "розділ розробки" відповідає за керування проектами розробки програмного забезпечення, включаючи планування, організацію та контроль за виконанням робіт. Узагальнюючи, «розділ розробки» є важливим елементом в будь-якій ІТ-компанії, що займається розробкою програмного забезпечення. Цей відділ відповідає за процес розробки, тестування та підтримки програмного забезпечення. Крім того, він забезпечує вдосконалення технологій та методів розробки, щоб забезпечити ефективність та якість роботи. Також «розділ розробки» відповідає за керування проектами розробки програмного забезпечення, включаючи планування, організацію та контроль за виконанням робіт. У склад «розділу розробки» зазвичай входять професійні спеціалісти з різних областей розробки програмного забезпечення, що забезпечує різноманітність компетенцій і можливість забезпечення високої якості продукту.

«Продукт» – це програмне забезпечення або технічний продукт, який розробляється на замовлення клієнта. Продуктом може бути будь-яке програмне забезпечення, що відповідає певним вимогам та специфікаціям, від сайту або мобільного додатку до складної системи електронного документообігу або іншої програмної продукції. Розробка продукту в аутсорс IT компанії зазвичай проводиться на основі процесу, що передбачає взаємодію між замовником та розробником, включаючи детальний аналіз вимог клієнта, розробку технічної специфікації, програмування та тестування продукту, а також підтримку та оновлення продукту в майбутньому.

QA розділ - це структурний підрозділ в IT компанії, який відповідає за забезпечення якості програмного забезпечення, розробленого в компанії. QA розділ забезпечує контроль якості продукту на всіх етапах його життєвого циклу, починаючи від розробки та тестування і закінчуючи підтримкою та оновленням вже існуючого програмного забезпечення. Основні завдання QA розділу включають в себе: Розробка та впровадження стратегій тестування: QA розділ відповідає за розробку та впровадження стратегій тестування, які забезпечують високу якість та надійність програмного забезпечення. Виконання тестування: QA розділ забезпечує проведення тестування програмного забезпечення на всіх етапах його життєвого циклу, включаючи ручне та автоматизоване тестування. Оцінка якості продукту: QA розділ проводить оцінку якості продукту з метою виявлення проблем та дефектів, а також визначення рівня задоволення клієнтів. Підтримка та оновлення програмного забезпечення: QA розділ забезпечує підтримку та оновлення вже існуючого програмного забезпечення, зокрема виявлення та виправлення дефектів та помилок. Розробка та впровадження процесів управління якістю: QA розділ розробляє та впроваджує процеси управління якістю, які забезпечують високу якість та ефективність розробки програмного забезпечення. У склад QA розділу можуть входити різні спеціалісти, такі як тестувальники, інженери з якості програмного забезпечення, автомізатори тестів, спеціалісти з автоматизованого тестування, аналітики, менеджери якості та інші фахівці зі спеціальностей, пов'язаних з контролем та забезпеченням якості програмного забезпечення [6].

QA розділ є дуже важливим для IT компанії, оскільки він допомагає забезпечити високу якість та надійність програмного забезпечення, що є критично важливим для задоволення потреб клієнтів та забезпечення успішного функціонування бізнесу компанії. До того ж, розробка програмного забезпечення з використанням методологій Agile та DevOps вимагає постійного контролю якості продукту на всіх етапах його розробки та тестування, що підкреслює важливість QA розділу в IT компанії.

«Тестування» – це процес перевірки програмного забезпечення з метою виявлення помилок, дефектів та проблем, що можуть знизити якість та надійність продукту. Цей процес може включати в себе ручне та автоматизоване тестування, а також забезпечення контролю якості продукту на всіх етапах його життєвого циклу – від розробки та тестування до підтримки та оновлення вже існуючого програмного забезпечення.

«Деплой» – це процес розгортання або встановлення програмного забезпечення на сервері або іншому пристрої з метою його запуску та функціонування. Цей термін також використовується для опису процесу випуску та виливки мобільних додатків в магазини додатків, наприклад, App Store та Google Play. Після того, як програмне забезпечення було розроблено, протестовано та відлагоджено, воно готове до розгортання на сервері або на іншому пристрої. В разі мобільних додатків, виливка додатка в магазин також вимагає проведення попередніх перевірок та відповідності вимогам платформи. Деплой може бути здійснений на різних середовищах, наприклад, на тестовому сервері, стейджинговому сервері або виробничому сервері. Перш ніж встановлювати програмне забезпечення на виробничому сервері, воно повинно бути відповідно перевірене та протестоване на інших середовищах. Деплой може бути здійснений різними способами, включаючи вручну або за допомогою автоматичних засобів. За допомогою автоматичних засобів деплой може бути здійснений швидше та з меншим ризиком помилок.

У рамках процесу деплою можуть використовуватися різні інструменти, наприклад, системи керування конфігурацією, контейнерні рішення, інструменти автоматичного розгортання та інші. Деплой є важливим етапом в життєвому циклі програмного забезпечення та мобільних додатків, оскільки від нього залежить успішність запуску програмного забезпечення та його функціонування в реальному середовищі, а також доступність та якість мобільного додатку для користувачів магазинів додатків.

Після успішного деплою, програмне забезпечення або мобільний додаток стає доступним для користувачів та може бути використане відповідно до своєї призначеності. Під час функціонування програмного забезпечення або мобільного додатку можуть з'являтися помилки та несправності, що вимагають виправлення через процес розгортання патчів або нових версій [6].

Усі етапи деплою повинні бути добре організовані та підтримуватися документацією та »журналами», щоб забезпечити належну якість та контроль над процесом.

DevOps – це методологія розробки програмного забезпечення, яка поєднує в собі розробку (Dev - Development) та експлуатацію (Ops - Operations) програмного забезпечення, з метою забезпечення швидкого та надійного випуску програмного забезпечення на ринок. DevOps охоплює в собі автоматизацію процесів, спільну відповідальність розробників та операторів за результат, інструменти та практики для швидкого розгортання та відладки програмного забезпечення, а також практики неперервної інтеграції та неперервної доставки.

DevOps включає в себе колаборацію між командами розробників та операторів, щоб забезпечити ефективність та прозорість усіх процесів, а також знизити час випуску нових функцій та продуктів.

DevOps-інженери допомагають встановлювати та налаштовувати інфраструктуру для розгортання та тестування програмного забезпечення, автоматизують процеси збірки та тестування коду, забезпечують моніторинг та аналіз результатів функціонування системи. Також орієнтований на практики, що сприяють швидкому та надійному випуску програмного забезпечення, такі як неперервна інтеграція (Continuous Integration), неперервна доставка (Continuous Delivery), неперервне вирішення проблем (Continuous Problem Resolution), моніторинг процесів та показників продуктивності, а також автоматизація рутинних завдань та операцій.

«Площадка розповсюдження» (distribution platform) – це онлайн-магазин або маркетплейс, де можна знайти та завантажити мобільні додатки на свій пристрій. Такі платформи зазвичай забезпечують розповсюдження мобільних додатків на основі певних критеріїв, таких як категорія додатків, країна розташування користувачів, мова та інші параметри. Найбільш відомі площадки розповсюдження мобільних додатків - це App Store від Apple для iOS-пристроїв та Google Play для Android-пристроїв. На площадках розповсюдження, мобільні додатки пройшли процес модерації та перевірки, щоб забезпечити відповідність вимогам платформи та забезпечити безпеку користувачів. Розробники мобільних додатків повинні дотримуватися правил та політик платформи, щоб забезпечити успішне публікування своїх додатків на платформі розповсюдження.

Крім того, на платформах розповсюдження мобільних додатків можуть бути надані інструменти для просування та монетизації мобільних додатків, такі як реклама та платіжні сервіси. Площадки розповсюдження є важливим елементом в життєвому циклі мобільного додатка, оскільки вони забезпечують доступність та розповсюдження додатка для користувачів, а також можуть впливати на успіх та прибутковість додатка.

Також має сенс згадати, що крім розділів відповідальних безпосередньо за розробку, є наступні: бухгалтерія, HR розділ а також навчальні курси, ці розділи будуть розглянуті подалі [6].

IT компанії можуть бути корисними для наукових досліджень, оскільки ці компанії займаються розробкою технологічних продуктів, які мають значний вплив на сучасне суспільство. У цій дисертації ми проаналізували типи IT компаній, їх ролі в економіці, процеси розробки ПЗ та UI/UX дизайну, а також спіральну динаміку в контексті розвитку компаній. Було визначено, що IT компанії можуть бути класифіковані за кількома параметрами, включаючи тип продукту, розмір компанії, спеціалізацію та модель бізнесу. Кожен тип компанії має свої особливості та проблеми, які можуть виникати в процесі роботи. Для розробки продуктів компанії використовують різні методики та підходи. Розробка UI/UX дизайну також є важливою складовою процесу розробки ПЗ, оскільки цей елемент може значно вплинути на користувацький досвід. Спіральна динаміка, може бути застосована для аналізу розвитку IT компаній та їхніх процесів.

Кожен рівень спіралі відображає унікальну систему цінностей та поглядів на світ, що може відображатися в культурі компанії, методиках розробки та моделях бізнесу. Оскільки сфера IT є однією з найшвидше зростаючих в світі, компанії, які діють у цьому секторі, повинні бути готові до неперервних змін та інновацій. У спробі відповідати на потреби ринку та залучити нових клієнтів, компанії можуть використовувати різні стратегії, такі як аутсорсинг, аутстафінг та власний продукт. Крім того, для ефективної роботи у сфері IT, компанії повинні мати глибокі знання про різноманітні технології та програмні засоби. У цьому контексті надзвичайно важливо мати висококваліфікований технічний персонал та забезпечувати їм належну підтримку та розвиток [6]. Під час аналізу було виявлено, що успішне впровадження процесів розробки вимагає ефективної комунікації та співпраці між різними структурними підрозділами IT-компанії, такими як розробка, тестування та експлуатація. Дослідження показали, що управління проектами та контроль якості є ключовими аспектами розробки програмного забезпечення, а практики DevOps можуть допомогти забезпечити швидкий та надійний випуск програмного забезпечення на ринок. Отже, дослідження засвідчує, що успішна розробка програмного забезпечення в IT-компаніях вимагає співпраці різних структурних підрозділів, використання ефективних практик розробки та контролю якості, а також забезпечення безпечного та ефективного розповсюдження додатків на ринку.

1.3 Програмне забезпечення IDAP для реалізації продукта

IDAP вкладає особливу увагу у розробку своїх продуктів, використовуючи передові інструменти та технології у галузі дизайну UI/UX. Наша команда вдається до найефективніших рішень завдяки використанню таких інструментів, як Figma, Adobe Photoshop, та інші. Це дозволяє створювати продукти високої якості, що відповідають найвищим стандартам в індустрії дизайну (рис.1.8).

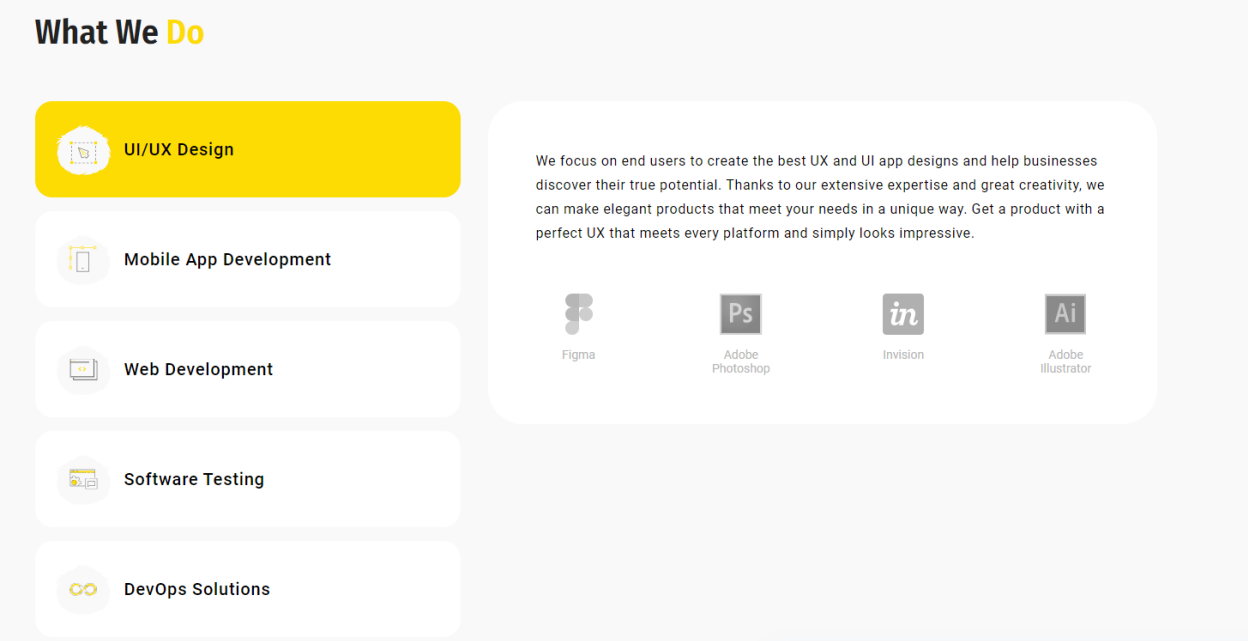

Рисунок 1.8 – Інструменти UI/UX

Figma – це інтерактивний веб-сервіс та програмне забезпечення для дизайну, що дозволяє командам створювати, спільно працювати та редагувати графічні проекти в реальному часі. В основному використовується для розробки дизайну інтерфейсів користувачів (UI) та дизайну користувацького досвіду (UX). Однією з головних переваг Figma є можливість спільної роботи кількох користувачів над одним проектом в режимі реального часу, що полегшує процес колективної роботи та комунікації між учасниками проекту.

Adobe Photoshop - це програмне забезпечення для редагування та обробки графіки, що входить до складу Creative Cloud, продуктової лінійки компанії Adobe. Воно використовується для ретушування фотографій, створення графічних дизайнів, малювання, а також для роботи з різними видами медіа-контенту. Adobe Photoshop має широкий спектр інструментів і можливостей, які дозволяють дизайнерам, фотографам та іншим творчим професіоналам створювати і редагувати зображення з високою якістю.

InVision - це інструмент для дизайну та прототипування, призначений для дизайнерів та розробників програмного забезпечення. За допомогою InVision користувачі можуть створювати інтерактивні прототипи веб-сайтів, мобільних додатків та інших цифрових продуктів безпосередньо в браузері.

Основні функції InVision включають:

1. Створення прототипів: користувачі можуть створювати інтерактивні прототипи, які відображають поведінку елементів інтерфейсу на різних етапах взаємодії.
2. Спільна робота: кілька членів команди може спільно працювати над проектом, обмінюючись відгуками та вносячи зміни.
3. Збір відгуків: дизайнери можуть збирати відгуки та коментарі від команди та клієнтів безпосередньо в інтерфейсі.

4. Інтеграція з іншими інструментами: InVision часто використовується разом з іншими програмами для дизайну, такими як Sketch або Adobe XD.
5. Анімація: є можливість додавати базову анімацію до прототипів для кращого розуміння взаємодії.

InVision дозволяє дизайнерам та розробникам ефективно співпрацювати, вирішувати дизайнерські питання та вдосконалювати користувацький досвід.

Adobe Illustrator – це векторний графічний редактор, розроблений компанією Adobe Inc. Це програмне забезпечення призначене для створення ілюстрацій, малюнків, логотипів, емблем та інших векторних зображень.

IDAP впроваджує передові технології для створення своїх мобільних додатків, надаючи користувачам високоякісний інтерфейс та функціонал. Для розробки мобільних застосунків компанія IDAP використовує такі потужні технології, як Flutter та React Native (рис.1.9). Ці інструменти дозволяють швидко та ефективно створювати крос-платформені додатки, які працюють як на iOS, так і на Android, забезпечуючи зручність та доступність для широкого кола користувачів. За допомогою Flutter і React Native IDAP забезпечує інноваційні та високоякісні мобільні рішення, які відповідають найвищим стандартам в галузі розробки програмного забезпечення [6].

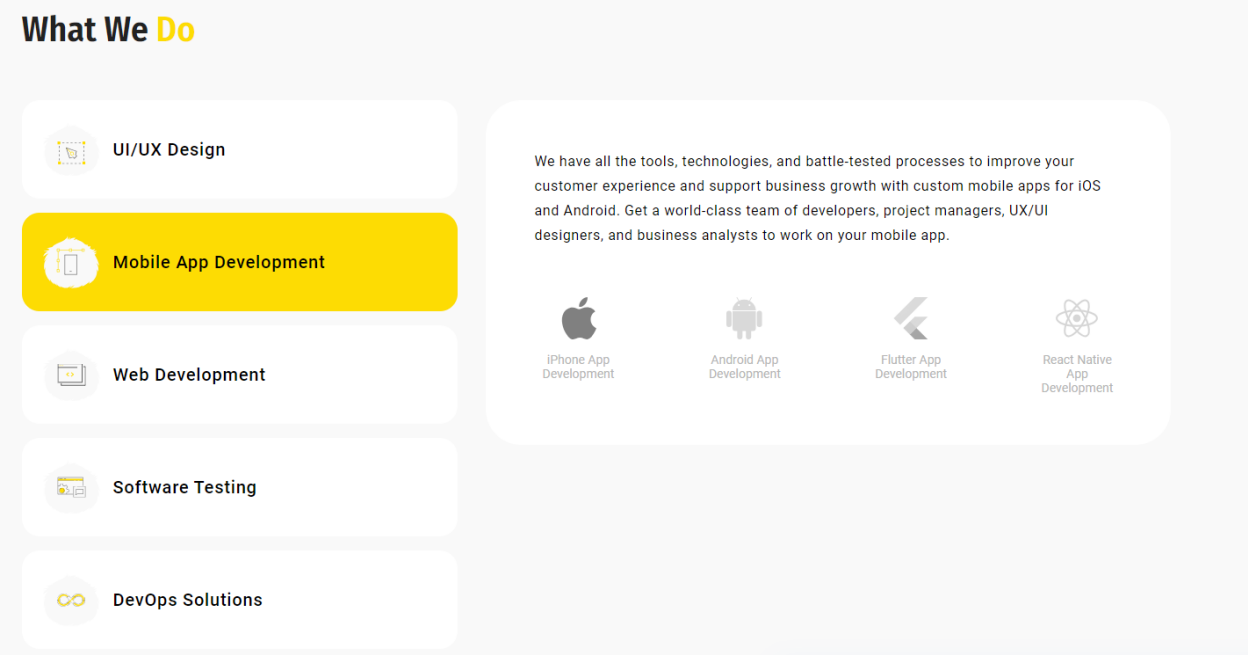

Рисунок 1.9 – Інструменти UI/UX

Flutter – це відкрите програмне забезпечення для створення користувальницьких інтерфейсів (UI). Розроблений компанією Google, цей інструмент використовується для швидкої та ефективної розробки мобільних, веб- та настільних додатків з крос-платформовими можливостями. Flutter використовує власний високопродуктивний двигун Dart

для відтворення інтерфейсів та дозволяє розробникам створювати красиві та плавні UI,
що працюють однаково добре на різних платформах, таких як Android та iOS. За допомогою Flutter розробники можуть ефективно взаємодіяти з різними аспектами інтерфейсу та забезпечувати стабільні та інноваційні додатки для різних пристроїв.

React Native – це відкрите програмне забезпечення для розробки мобільних додатків, яке дозволяє використовувати мову програмування JavaScript та бібліотеку React для створення крос-платформових додатків. Розроблений компанією Facebook, React Native дозволяє розробникам використовувати один і той самий код для створення додатків для платформ Android та iOS, що робить процес розробки більш ефективним та швидким.

Однією з ключових особливостей React Native є можливість використання компонентів React для побудови інтерфейсу користувача, що спрощує розробку та забезпечує високий рівень переносимості між платформами. Розробники можуть швидко оновлювати та впроваджувати зміни у своїх додатках, а також використовувати розширені можливості React Native для доступу до пристройових API та інших функцій мобільних платформ.

В розробці веб-продуктів, IDAP використовує передові технології та інструменти для створення сучасних та інноваційних веб-додатків. HTML5, CSS3, jQuery та інші технології стають основними будівельними блоками у розробці відмінних користувацьких інтерфейсів та функціональності, що відповідають високим стандартам та очікуванням користувачів [6]. Давайте зглибимося в світ інновацій та технологічних досягнень, які використовує IDAP для створення потужних та ефективних веб-продуктів (рис.1.10).

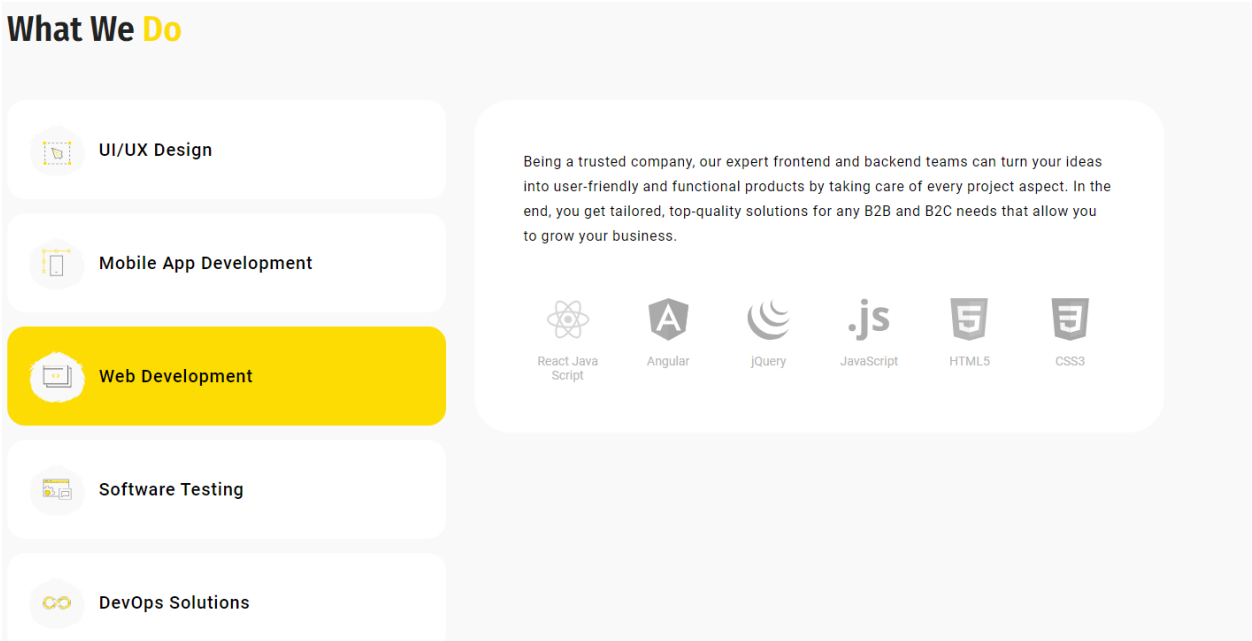

Рисунок 1.10 – Інструменти web development

React – це бібліотека JavaScript для створення інтерфейсів користувача, розроблена компанією Facebook. React дозволяє розбити інтерфейс на компоненти та ефективно оновлювати їх при зміні даних. Використання React спрощує створення складних інтерфейсів, робить код більш читабельним і підтримуваним.

JavaScript – це мова програмування, яка використовується для створення динамічних веб-сайтів та веб-додатків. Вона використовується як мова для взаємодії з користувачем на веб-сторінках та для реалізації різноманітних функціональностей в браузерах. JavaScript є однією з найпоширеніших мов програмування в інтернет-розробці.

Angular – це фреймворк для створення веб-додатків та односторінкових застосунків (SPA), розроблений компанією Google. Він використовується для створення динамічних та масштабованих інтерфейсів користувача. Angular пропонує різні інструменти та можливості для ефективної розробки великих проектів.

Основні риси Angular:

1. Двостороннє зв'язування (Two-way binding): це механізм, що автоматично синхронізує дані між моделлю та представленням.
2. Модульність: angular дозволяє розбити додаток на невеликі, самодостатні модулі, що спрощує розробку та підтримку.
3. Dependency Injection: механізм, що дозволяє внедрювати залежності в компоненти, що полегшує тестування та управління кодом.
4. Directives: angular має вбудовані директиви, що дозволяють розширювати HTML новими атрибутами та тегами.

Розширена система шаблонів: angular використовує шаблони для створення користувацького інтерфейсу, що робить їх потужним та гнучким інструментом.

DAP вдало використовує ряд ефективних інструментів у сфері тестування програмного забезпечення для забезпечення високої якості своїх продуктів. Серед цих інструментів використовуються такі популярні рішення, як Jira та Postman. Jira відображається як надійний інструмент для управління завданнями та відстеження помилок, що дозволяє ефективно взаємодіяти з розробниками та вирішувати поточні завдання [6]. З іншого боку, Postman використовується для автоматизації тестування API, надаючи швидкі та надійні засоби для взаємодії з веб-сервісами. Цей набір інструментів сприяє забезпеченню високої якості програмних продуктів IDAP та ефективному процесу тестування (рис.1.11).

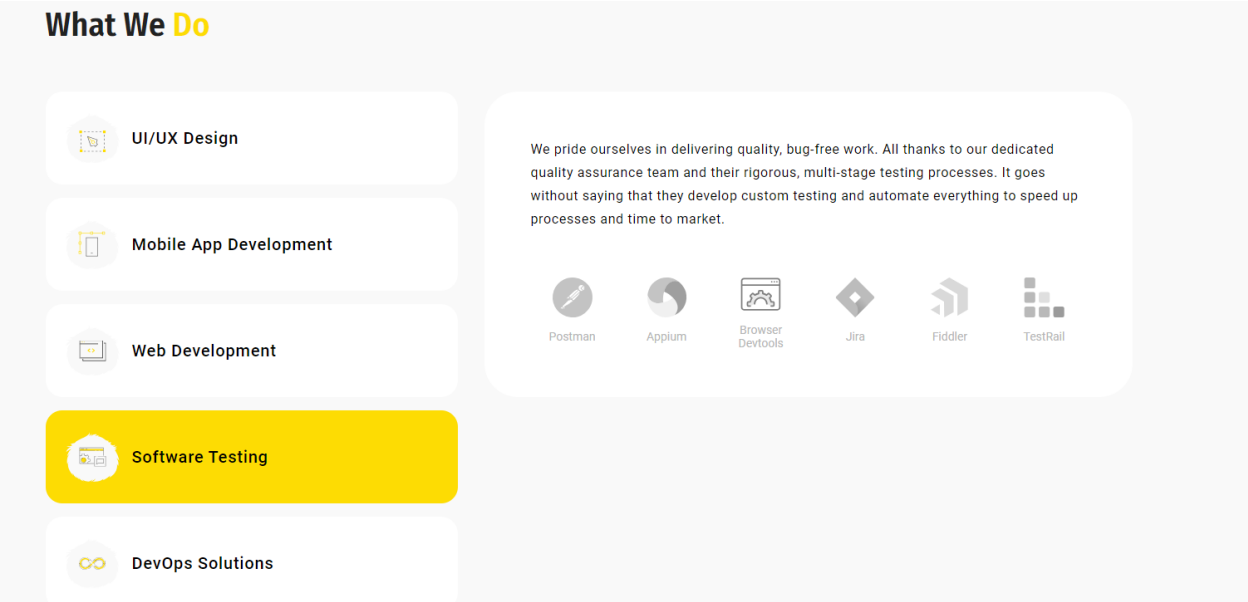

Рисунок 1.11 – Інструменти software testing

Jira – це інструмент для управління проектами та відстеження завдань, розроблений компанією Atlassian. Він широко використовується розробниками програмного забезпечення, командами розробки та іншими професійними групами для ефективного планування проектів, відстеження завдань, спільної роботи та звітування про результати.

Вона дозволяє створювати задачі, розподіляти їх між учасниками команди, визначати терміни виконання та відстежувати прогрес. Також він надає можливості для створення дошок Kanban та Scrum, щоб візуалізувати та контролювати хід роботи. Завдяки своїм різнобічним можливостям, Jira допомагає командам більш ефективно керувати своїми проектами та завданнями.

Appium – це відкрите програмне забезпечення для автоматизації тестування мобільних додатків на різних платформах, включаючи iOS, Android та Windows. Appium дозволяє тестувати мобільні додатки, які написані на різних мовах програмування, таких як Java, Python, Ruby, PHP, C#, і використовувати різні технології, такі як Native, Hybrid та Mobile Web.

Однією з ключових переваг Appium є те, що він надає єдиний інтерфейс для автоматизації тестів на різних платформах без необхідності зміни коду тестування. Він використовує стандартні протоколи, такі як WebDriver, що дозволяє взаємодіяти з мобільними додатками так само, як і з веб-додатками [6].

Appium підтримує автоматизацію для різних мобільних платформ і пропонує багатофункціональність для ефективного тестування мобільних додатків.

Postman – це інструмент для розробки та тестування API (інтерфейсів програмування застосунків). Він надає зручний інтерфейс для взаємодії з API, відправлення HTTP-запитів, тестування різних методів та перевірки відповідей сервера.

Основні функції Postman включають:

1. Створення запитів: postman дозволяє створювати різні типи HTTP-запитів, такі як GET, POST, PUT, DELETE, і багато інших. Ви можете вказати параметри, заголовки, тіло запиту та інші налаштування.
2. Організація та збереження запитів: ви можете організовувати ваші запити у папки, надавати їм зрозумілі назви та зберігати для майбутнього використання.
3. Тестування API: postman дозволяє вам писати автоматизовані тести для перевірки відповідей від сервера. Ви можете визначати умови успіху та перевіряти, чи відповіді відповідають вашим очікуванням.
4. Моніторинг та документація API: postman дозволяє створювати документацію для вашого API та вести моніторинг його працездатності.

Postman є корисним інструментом для розробників, QA-інженерів та інших фахівців, які працюють з веб-сервісами і мають потребу у зручному інструменті для тестування та взаємодії з API.

IDAP для своєї реалізації продуктів в галузі DevOps використовує ряд передових технологій та платформ, серед яких важливе місце займають такі провідні хмарні сервіси, як DigitalOcean та Microsoft Azure [6]. Ці платформи надають надійні та потужні інструменти для автоматизації процесів розробки, тестування та впровадження програмного забезпечення (рис.1.12).

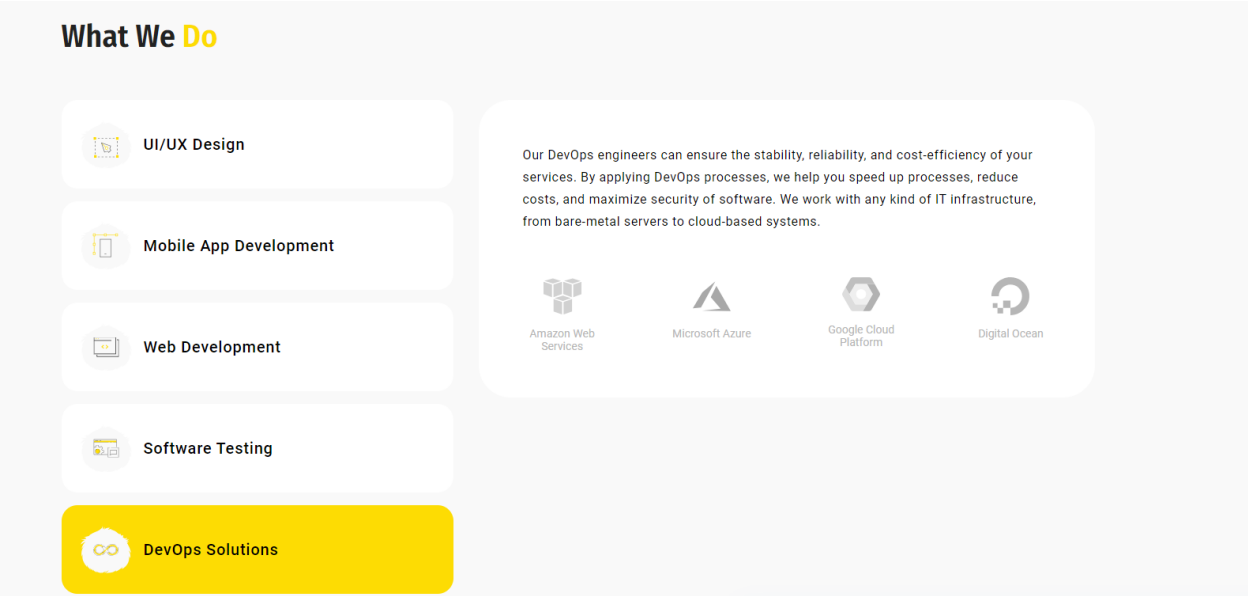

Рисунок 1.12 – Інструменти devops solution

DigitalOcean: цей хмарний сервіс надає розгортання віртуальних серверів (droplets) з широким спектром конфігурацій, що відповідають вимогам будь-якого проекту. DigitalOcean дозволяє автоматизувати інфраструктурні задачі та забезпечує швидке розгортання, що робить його ідеальним вибором для DevOps-процесів.

Microsoft Azure: як одна з провідних хмарних платформ від Microsoft, Azure пропонує широкий спектр сервісів для розробки, тестування та впровадження програмного забезпечення. Від інфраструктури до рішень штучного інтелекту, Azure дозволяє ефективно впроваджувати DevOps-практики та автоматизовувати процеси розробки.

Ці інноваційні інструменти в поєднанні з експертною екіпажем IDAP дозволяють створювати та управляти високопродуктивними DevOps-рішеннями для клієнтів, відповідаючи найвищим стандартам ефективності та надійності.

1.4 Огляд альтернативних продуктів для реалізації проектів

В умовах постійного розвитку сучасних технологій та росту вимог до управління проектами, важливо звертати увагу на різноманіття альтернативних продуктів, призначених для ефективної реалізації проектів. В даному огляді ми детально розглянемо кілька інструментів та рішень, які можуть служити альтернативою у сфері управління проектами. Кожен з цих продуктів має свої особливості та переваги, спрямовані на задоволення конкретних потреб команд та організацій. Отже, розглянемо цей список альтернативних інструментів та проведемо огляд їхніх ключових характеристик.

Тепер розглянемо альтернативні програми, які можна використовувати для реалізації проектів, орієнтовані на досягнення високих стандартів та ефективних рішень в галузі дизайну UI/UX [6].

Альтернативами для Adobe Photoshop та Figma можуть бути такі інструменти:

1. Sketch: це популярний інструмент для дизайну інтерфейсів, особливо популярний серед користувачів macOS.
2. Adobe XD: інший продукт від Adobe, який спеціалізується на дизайні інтерфейсів та прототипуванні.
3. Canva: це онлайн-інструмент, спрямований на швидке та просте створення графіки для соціальних мереж, презентацій та іншого контенту.
4. Affinity Designer: відмінна альтернатива для роботи з векторною графікою та дизайном.

Кілька альтернатив для розробки крос-платформених мобільних додатків, окрім Flutter і React Native:

1. Xamarin: це фреймворк від Microsoft, який дозволяє розробляти кросплатформенні додатки, використовуючи мову програмування C#.
2. PhoneGap / Apache Cordova: використовується для розробки мобільних додатків з використанням веб-технологій, таких як HTML, CSS і JavaScript.
3. Ionic: це фреймворк для розробки мобільних додатків, який використовує HTML, CSS та JavaScript.
4. NativeScript: дозволяє розробляти нативні мобільні додатки за допомогою JavaScript або TypeScript.
5. Corona SDK: фреймворк для швидкої розробки ігор та додатків, використовуючи мову Lua.
6. Appcelerator Titanium: використовує JavaScript для розробки кросплатформених додатків, які можуть використовувати нативний код.
7. SwiftUI / UIKit (для iOS) та Kotlin / Android SDK (для Android): якщо необхідно розробляти нативні додатки для конкретних платформ.

Існує безліч альтернатив для React, JavaScript та Angular у світі розробки веб-додатків [6]. Ось кілька популярних альтернатив:

1. Vue.js: легкий JavaScript фреймворк для створення інтерфейсів користувача. Він легко інтегрується з існуючими проектами та має зручну документацію.
2. Svelte: інноваційний фреймворк, який перетворює код під час компіляції, що дозволяє зменшити обсяг вихідного коду та поліпшити продуктивність.
3. Ember.js: зручний фреймворк для розробки амбітних веб-додатків, який надає структуру та конвенції для швидкої розробки.
4. Backbone.js: легкий фреймворк, який дозволяє організувати код та створювати односторінкові додатки.
5. Aurelia: модульний фреймворк, який пропонує чистий та розширюваний код для створення динамічних інтерфейсів.
6. Mithril: легкий фреймворк з акцентом на простоту та продуктивність.
7. Preact: мінімалістичний фреймворк, який надає швидку та ефективну роботу з віртуальним DOM.
8. Alpine.js: JavaScript-бібліотека для декларативного програмування, яка працює без необхідності компіляції.

У сфері тестування програмного забезпечення.

Існує кілька альтернатив для Jira, які також відомі своєю ефективністю в управлінні проектами та розробкою програмного забезпечення. Деякі з них включають (рис.1.13):

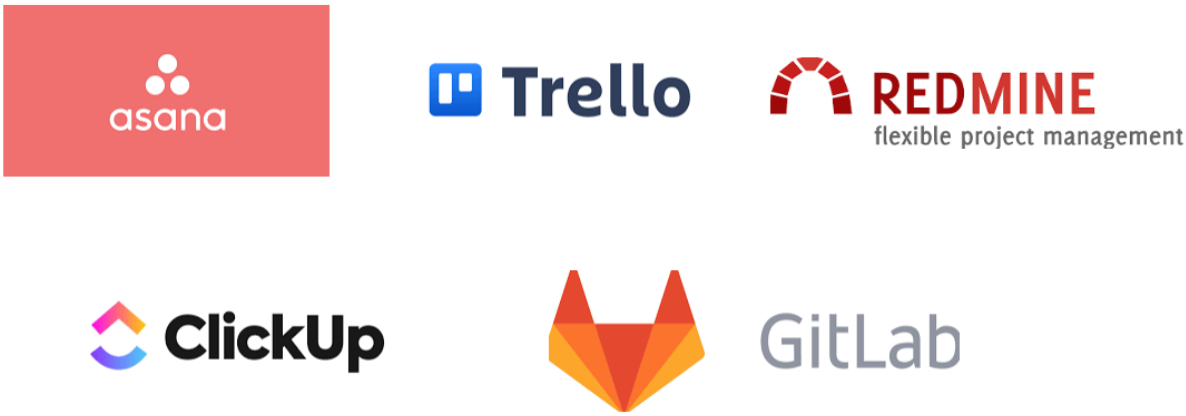

Рисунок 1.13 – Приклда програм

1. Asana – це популярний інструмент для управління проектами, який дозволяє створювати завдання, призначати їх відповідальним особам, встановлювати терміни та ведення комунікації в межах команди.
2. Trello використовує карти і дошки для візуального представлення завдань та їхнього прогресу. Це інтуїтивно зрозумілий інструмент, особливо для тих, хто віддає перевагу методології Kanban.
3. GitLab – це не тільки система керування версіями Git, але і інтегрована платформа DevOps, яка включає в себе управління проектами, CI/CD, відстеження проблем та інші інструменти.
4. Redmine – це відкрите програмне забезпечення для управління проектами, яке включає в себе функції відстеження проблем, календарі, графіки Gantt та інші.
5. ClickUp – це універсальний інструмент для управління проектами та завданнями, який має велику кількість функцій, включаючи засоби для спільної роботи, документації та збереження нотаток.

Кожен з цих інструментів має свої унікальні особливості та підходить для різних потреб команд. Вибір конкретної альтернативи залежить від конкретних вимог та уподобань вашої команди.

Перелік продуктів в галузі DevOps [6].

Існує кілька альтернатив DigitalOcean та Microsoft Azure для хмарних послуг та хостингу:

1. Amazon Web Services (AWS): найбільший і найбільш розповсюджений хмарний провайдер, який пропонує широкий спектр послуг для розробки, хостингу та обслуговування веб-додатків.
2. Google Cloud Platform (GCP): інша велика платформа хмарних послуг, яка пропонує різноманітні інструменти та сервіси для розробки та управління додатками.

3. IBM Cloud: платформа хмарних послуг від IBM, яка надає рішення для розробки, розгортання та управління додатками.
4. Heroku: платформа, що дозволяє розробникам легко розгортати, масштабувати та управляти додатками без необхідності у глибоких знаннях про інфраструктуру.
5. Vultr: постачальник хмарних та віртуальних серверів, який пропонує швидке та ефективне хостингове рішення.
6. Linode: ще один постачальник віртуальних серверів і хостингових послуг, який дозволяє розгортати веб-додатки та інші проекти.
7. Oracle Cloud Infrastructure (OCI): платформа хмарних послуг від Oracle, яка надає інфраструктурні та платформенні послуги для розробки.

Висновки за розділом

У результаті аналізу загального опису компанії IDAP можна застосувати, що вона є динамічною та інноваційною IT-компанією, спеціалізована яка охоплює різні технології та послуги. Дійсність широкого спектру проектів та розмаїття продукції свідчать про компетентність та гнучкість компанії.

Модель спіральної динаміки з використанням компанії IDAP для стратегічного управління, що підвищує про високий рівень професіоналізму в управлінні та гнучкість у виборі стратегій у відповідності зі змінами в індустріальних та ринкових умовах.

Ретельний огляд структури компанії дозволяє зрозуміти внутрішню організацію та взаємодію різних підрозділів. Ієрархічна структура та розподіл обов'язків виконує ефективну функцію та реагувати на зміни.

Висновок із розділу дозволяє використовувати, що компанія IDAP обробляє та обирає оптимальні альтернативи для реалізації проектів. Аналіз конкурентних продуктів компанії залишаються конкурентоспроможною та інноваційною на ринку.

2 МОДЕЛЬ ТА СТРУКТУРА РОЗДІЛІВ IDAP

2.1 Загальний опис розділів IDAP

Сучасний IT-світ насичений різноманітними компаніями, які впроваджують інноваційні рішення та технології для досягнення своїх бізнес-цілей. Однією з таких компаній, що займає лідерські позиції в галузі розробки програмного забезпечення та інформаційних технологій, є IDAP.

Цей розділ виділено детальним аналізом різних аспектів та розділів компанії IDAP, щоб краще зрозуміти її структуру, стратегії та технічні особливості. Мета вивчення цього розділу - розкрити загальний опис розділів компанії та вийти, як їхні функції та взаємодія впливають на загальний успіх та розвиток IDAP.

Розглянемо ключові аспекти структури та функціонування IDAP, висвітлимо різні розділи компанії та їхні основні завдання. Аналізуючи ці аспекти, ми приймаємо глибше розуміння внутрішньої роботи компанії, її підходів до управління та розробки продуктів, що стане основою для подальших досліджень та висновків (рис.2.1).

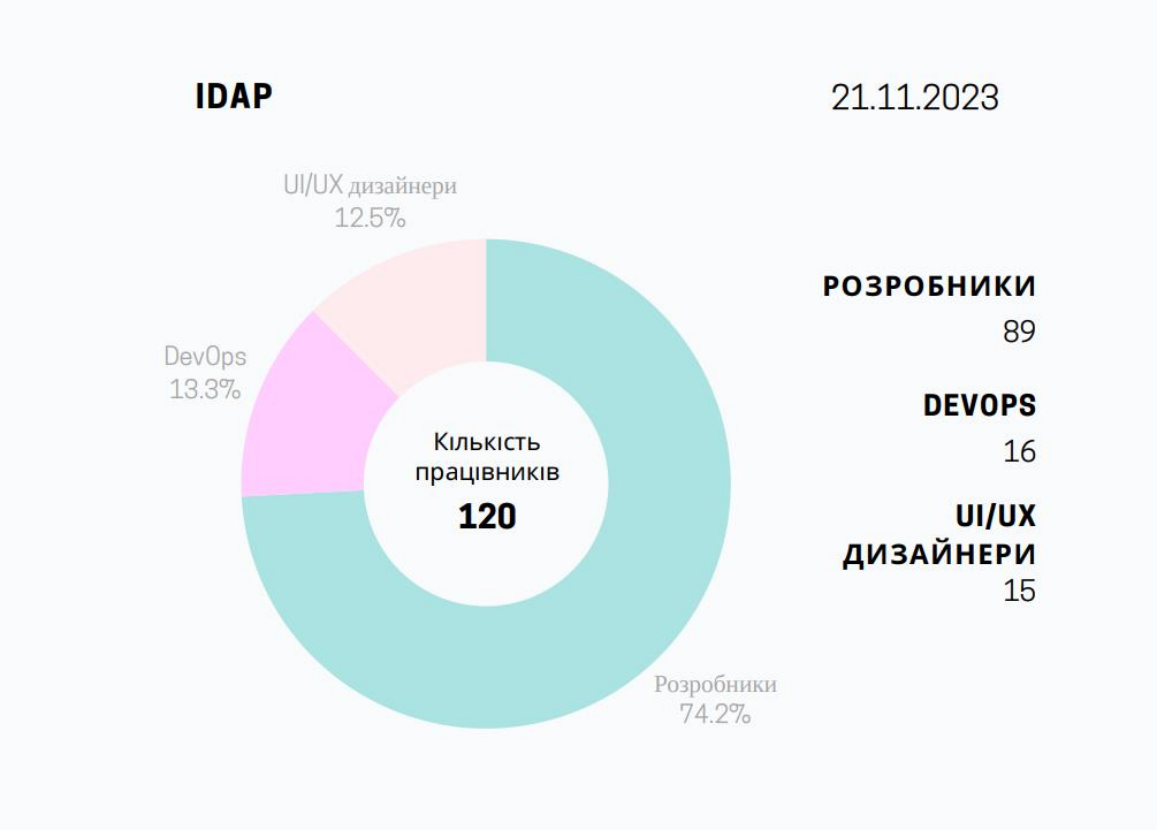

Рисунок 2.1 – Діаграма робітників

					КНУ.РМ.123.23.6.МТСРІ		
Змн.	Арк.	№ документа	Підпис	Дата			
Розробив		Кікачеішвілі			МОДЕЛЬ ТА СТРУКТУРА РОЗДІЛІВ IDAP	Літера	Аркуш / Аркушів
Перевірив		Музика					
Н.контроль		Кузнєцов				КІ-22м	
Затвердив		Купін					

Компанія IDAP являє собою колектив талановитих та висококваліфікованих спеціалістів, які становлять основний ресурс для досягнення успіхів та впровадження інновацій. На даний момент в об'єднанні працює понад 120 співробітників, кожен з яких володіє глибокими знаннями та великим досвідом у своїй галузі.

Ключовою складовою колективу є 89 розробників із різними спеціалізаціями, включаючи експертів для платформ Android та iOS, а також фахівців у сферах фронтенду та бекенда. Їхні унікальні навички та розширений досвід у програмуванні не дозволяють нам ефективно створювати продукти високої якості, що відповідають вимогам наших клієнтів [6].

Команда включає також 16 девопсів, відповідних для оптимізації та автоматизації процесів розробки, що гарантує ефективну та безперебійну роботу наших продуктів. При цьому 15 UI/UX дизайнерів активно співпрацюють над створенням привабливого та інтуїтивно зрозумілого інтерфейсу наших продуктів, підвищуючи їхню привабливість для користувачів. Злагоджена взаємодія всіх груп фахівців закладає основу для успішної реалізації інноваційних проектів та забезпечення високого стандарту якості в області розробки програмного забезпечення.

2.2 Опис frontend розділу

Фронтенд веб-розробка – це один із етапів веб-розробки, який перетворює дані в графічний інтерфейс для перегляду та взаємодії з даними за допомогою цифрової взаємодії, використовуючи HTML, CSS та JavaScript. Простими словами, фронтенд розробка - це той процес, яким дизайн веб-сайту фактично реалізується в Інтернеті.

Фронтенд-розробник – це програміст, який пише код для розробки фронтенду веб-сайту. Іншими словами, фронтенд-розробники перетворюють файли дизайну веб-сайту в код HTML, JavaScript (JS) і/або CSS, що включає основний дизайн/структуру веб-сайту, зображення, вміст, кнопки, навігацію та внутрішні посилання. Результатом є код, який служить структурою фронтенду веб-сайту, який використовується бекенд-розробником для додавання бізнес-логіки та підключення баз даних і процесів (рис.2.2).

Фронтенд-розробник відповідає тому, щоб візуальний фронтенд веб-сайту був вільним від помилок і виглядав точно так, як задумано. Він також забезпечує однаковий вигляд веб-сайту на різних комп'ютерних і мобільних веб-переглядачах [7].

Також у веб-додатках фронтенд-розробник створює графічний інтерфейс користувача (GUI), який надає доступ до функцій та можливостей бекенда програмного забезпечення.

Існують різні платформи та інструменти для фронтенд розробки, і кожна з них використовує власну мову програмування та технології. Ось декілька різновидів фронтенд розробок та їхні основні мови:

1. WEB (Веб):

Мови програмування: HTML, CSS, JavaScript

Опис: веб-фронтенд розробка орієнтована на створення користувацьких інтерфейсів для веб-сайтів. HTML використовується для структурізації контенту, CSS – для стилізації та оформлення, а JavaScript – для надання інтерактивності та динамічності.

2. Android:

Мови програмування: Kotlin, Java

Опис: розробка фронтенду для Android орієнтована на створення робочих інтерфейсів для мобільних додатків, які працюють на платформі Android. Kotlin та Java є основними мовами програмного забезпечення для розробки Android-додатків.

3. IOS:

Мови програмування: Swift, Objective-C

Опис: розробка фронтенду для iOS орієнтована на створення користувацьких інтерфейсів для мобільних додатків, які працюють на платформі iOS. Swift та Objective-C є основними мовами програмування для розробки iOS-додатків.

4. Flutter:

Мова програмування: Dart

Опис: Flutter є фреймворком, який дозволяє розробникам створювати крос-платформені мобільні додатки з єдиною кодовою базою. Dart використовується як мова програмування для розробки інтерфейсу та логіки додатків у Flutter.

Кожна з цієї платформи має свої особливості та інструменти, але загальна мета – забезпечує високоякісний та ефективний користувацький інтерфейс для відповідної платформи чи пристрою [8].

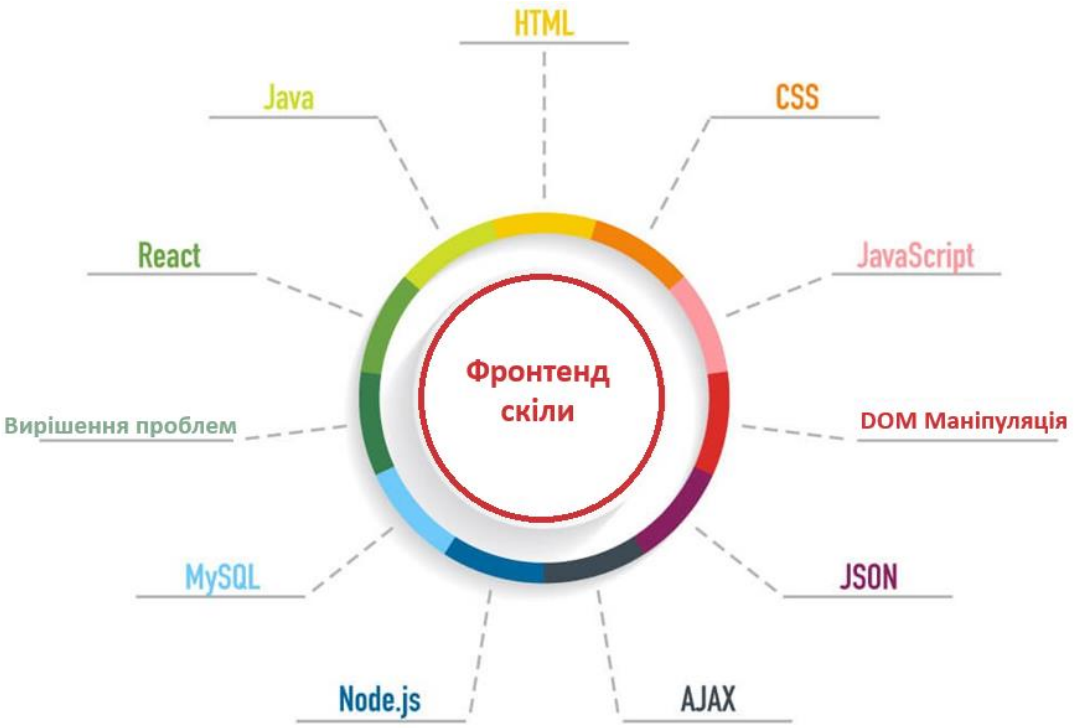

Рисунок 2.2 – Схема знань для розробника

Основні обов'язки фронтенд-розробника включають:

Розробка користувацького інтерфейсу (UI): створення структури та вигляду веб-сторінок або додатків, щоб забезпечити приємний та легкий для розуміння дизайн.

Реалізація інтерактивності: додавання функціональності, яка дозволяє користувачам взаємодіяти з вмістом сторінки або додатків, наприклад, форми, кнопки, анімація тощо.

Оптимізація для різних пристроїв: забезпечення того, що веб-сайт або додаток працює коректно на різних пристроях і розмірах екранів (від комп'ютерів до смартфонів) [8].

Взаємодія з бекендом: співпраця з бекенд-розробниками для обміну даними між клієнтською та серверною частиною програмного забезпечення.

Щодо мовного програмування, яке вимагає фронтенд-розробників, основними є HTML, CSS та JavaScript. HTML використовується для створення структури сторінок, CSS - для оформлення та стилізації, а JavaScript – для додавання динамічної функціональності та взаємодії з користувачем. Крім того, багато інших фреймворків та бібліотек для JavaScript, таких як React, Angular, чи Vue.js, які полегшують розробку фронтенду та роблять його більш організованим та ефективним.

HTML означає мову розмітки гіпертексту. Це стандартна мова розмітки для створення веб-сторінок. Вона дозволяє створювати та структурувати розділи, абзаци та посилання за допомогою HTML-елементів (основних будівельних блоків веб-сторінок), таких як теги та атрибути.

HTML використовує багато сфер, а саме:

Веб-розробка. Розробники вибирають HTML-код для визначення того, як веб-браузер відображає елементи веб-сторінок, такі як текст, гіперпосилання та мультимедійні файли.

Інтернет-навігація. Користувачі можуть легко навігувати та встановлювати посилання між пов'язаними сторінками та веб-сайтами, після широкого застосування HTML для введення гіперпосилання.

Веб-документація. HTML дозволяє організовувати та форматувати документи аналогічно Microsoft Word.

Важливо відзначити, що HTML не підтримує програмування, після чого вона не може створити динамічну функціональність, хоча тепер вона визнана офіційним веб-стандартом. Консорціум всесвітньої павутини (W3C) забезпечує підтримку та розвиток специфікацій HTML, а також регулярно надає оновлення.

Середня веб-сторінка включає кілька різних HTML-сторінок. Наприклад, головна сторінка, сторінка «Про нас» та сторінка контактів ваших окремих файлів HTML [9].

HTML-документи – це файли, які закінчуються розширенням .html або .htm. Веб-браузер читає файл HTML і відображає його вміст так, щоб користувачі могли його переглядати.

Усі HTML-сторінки складаються з набору HTML-елементів, які включають у себе набір тегів та атрибутів. HTML-елементи - це будівельні блоки веб-сторінок. Тег приєднується до веб-браузера, де починається і закінчується елемент, тоді як атрибут описує характеристики елемента.

Три основні частини елементів:

Відкриваючий тег – використовується для вказівки, де починається елемент. Тег обгортається відкриваючими та закриваючими кутовими дужками. Наприклад, використовуйте початковий тег , щоб створити абзац.

Вміст – це вид, який бачать інші користувачі.

Закриваючий тег - той же самий, що і відкриваючий тег, але зі слешем перед ім'ям елемента. Наприклад, , щоб закрити абзац.

Комбінація цих трьох частин створити HTML-елемент:

Ось як ви додаєте абзац у HTML.

Ще одна важлива частина HTML-елемента – це його атрибут, який має дві частини – ім'я та значення атрибута. Ім'я ідентифікує додаткову інформацію, яку користувач хоче додати, тоді як значення атрибута дає подальшу специфікацію.

Наприклад, елемент стилю, який додає колір фіолетового кольору та шрифт verdana, буде виглядати наступним чином:

Ось як ви додаєте абзац у HTML.

Ще одним атрибутом, який є висновком для розробки та програмування, є атрибут class HTML. Атрибут класу надає інформацію про стиль, який можна замінити до різних елементів із цим самим значенням класу.

CSS(Cascading Style Sheets). CSS є стандартною технологією веб-розробки, яка використовується для стилізації та форматування веб-сторінок. Основним завданням CSS є визначення зовнішнього вигляду елементів HTML на веб-сторінці.

Основні можливості CSS включають в себе зміну кольорів, шрифтів, розмірів тексту, вирівнювання, відступи, рамки, фони та інші візуальні аспекти веб-сторінок. CSS дозволяє розробникам відокремити структуру HTML від її представлення, що полегшує зміну стилів і вигляду без внесення змін у сам HTML-код.

Наприклад, застосування CSS можна виглядати так:

```
/* Стилі для елемента <p> */
p {
  color: blue;
  font-size: 16px;
  margin-bottom: 10px;
}
/* Стилі для елемента з ідентифікатором "header" */
#header {
  background-color: gray;
  padding: 20px;
  text-align: center;
}
/* Стилі для класу "button" */
.button {
  background-color: green;
  color: white;
  padding: 8px 16px;
  border-radius: 5px;
}
```

У цьому прикладі застосовуються стилі до елементів , елемента з ідентифікатором «header» та елемента з класу «button». CSS дозволяє вам докладно керувати виглядом веб-сторінок і забезпечує багатий набір можливостей для стилізації [10].

JavaScript – це високорівнева, інтерпретована мова програмування, яка використовується для забезпечення динамічності та інтерактивності веб-

сторінок. Вона є однією з ключових технологій для розробки фронтенду (клієнтської частини) веб-додатків [10].

Основні характеристики JavaScript включають:

1. Динамічність: JavaScript дозволяє змінювати вміст і структуру веб-сторінок під час її виконання. Це робить взаємодію з користувачем більш динамічною та захоплюючою.
2. Інтерактивність: за допомогою JavaScript можна створювати інтерактивні елементи, такі як форми для введення даних, форми перевірки, анімації, слайд-шоу тощо.
3. Робота з DOM: JavaScript взаємодіє з Document Object Model (DOM), що представляє структуру HTML-документа. Завдяки цьому ви можете змінювати та маніпулювати вмістом сторінок.
4. Асинхронність: JavaScript підтримує асинхронну обробку подій, що дозволяє виконувати операції без блокування виконання інших операцій.
5. Кросбраузерність: мова є стандартом для реалізації скриптів у браузерах, і вона підтримується практично всіма сучасними браузерами.
6. Розширюваність: є багато бібліотек та фреймворків на основі JavaScript, таких як React, Angular або Vue.js, які полегшують розробку великих та складних веб-додатків.

Приклад простого коду на JavaScript:

```
// Виведення повідомлення в консоль браузера
console.log("Привіт, світ!");
// Змінні та умовний оператор
var number = 5;
if (number > 0) {
  console.log("Число додатнє");
} else {
  console.log("Число від'ємне або рівне нулю");
}
// Функція для додавання двох чисел
function add(a, b) {
  return a + b;
}
// Виклик функції
var result = add(3, 4);
console.log("Результат додавання: " + result);
```

JavaScript використовується як на стороні клієнта (в браузері), так і на стороні сервера (за допомогою платформи, таких як Node.js). Вона є невід'ємною частиною веб-розробки та дозволяє створювати динамічні та ефективні веб-додатки.

Android – операційна система, розроблена компанією Google, яка використовується в основному для мобільних пристроїв, таких як смартфони та планшети. Розробники Android створюють додатки, використовуючи Java або Kotlin. Android дозволяє створювати різноманітні додатки, від ігор до продуктивних інструментів, і використовувати на мільйонах пристроїв по всьому світу. Android також включає в себе власний фреймворк для розробки інтерфейсів, відомий як Android SDK (Software Development Kit).

Kotlin – це статично типізована мова програмування, яка працює на віртуальній машині Java (JVM) і може бути використана для розробки різноманітних додатків, включаючи Android-додатки. Вона була розроблена компанією JetBrains і представлена в 2011 році [11].

Основні особливості Kotlin включають:

1. Сумісність з Java: Kotlin повністю сумісний з Java, що означає, що код Kotlin може використовувати бібліотеки та інструменти Java і навпаки. Це полегшує поступовий перехід від Java до Kotlin або їх спільне використання в одному проекті.
2. Безпека та Чистота: Kotlin дає деякі безпечні та чисті концепції програмування, що робить код менш помилковим і більш читабельним.
3. Сучасні Можливості: мова має численні сучасні можливості, такі як лямбда-вирази, розширення функцій, інтерфейси зі змінним числом методів та інші, що полегшують написання коду.
4. Підтримка Android: Kotlin є офіційною мовою програмування для розробки Android-додатків, і вона широко використовується серед розробників для створення мобільних застосунків.

```
// Приклад Kotlin-коду

// Визначення класу
class Person(val name: String, var age: Int) {
    // Функція для виведення інформації про особу
    fun introduce() {
        println("Мене звуть $name і мені $age років.")
    }
}
fun main() {
    // Створення об'єкта класу Person
    val person = Person("Іван", 25)
```

```
    // Виклик функції для виведення інформації про особу
    person.introduce()

    // Зміна віку та виведення оновленої інформації
    person.age = 26
    person.introduce()
}
```

Пояснення:

1. Клас Особа: описує особу з властвостями ім'я та вік. Клас має конструктор, який замінює значення для цих властей при створенні об'єкта.
2. Функція введення(): виводить інформацію про особу, використовуючи значення її властивостей.
3. Функція main(): точка входу в програму. Створюється об'єкт класу Person з певними характеристиками, виводиться інформація про особу, змінюється вікно і знову виводиться оновлена інформація.

Java – це високорівнева, об'єктно-орієнтована мова програмування, яка використовується для розробки різноманітних програм, від мобільних додатків до великих корпоративних систем [11].

Основні особливості Java:

1. Об'єктно-орієнтована: Java базується на принципах об'єктно-орієнтованого програмування, що дозволяє створювати код, який легко розширюється та обслуговується.
2. Переносимість: код, написаний на Java, може використовуватися на будь-якому пристрої чи операційній системі, яка підтримує віртуальну машину Java (JVM). Це забезпечує переносимість програм між різними платформами.
3. Безпека: Java використовує вбудовану систему безпеки, що дозволяє запускати код в "пісочниці" (sandbox), обмежуючи його доступ до системних ресурсів.
4. Багатопотоковість: мова має вбудовану підтримку багатопотоковості, що дозволяє одночасно виконувати кілька завдань.

Велика стандартна бібліотека: Java має обширну бібліотеку класів і методів, що спрощує розробку програм і забезпечує готові рішення для багатьох завдань.

Популярні області застосування Java включають розробку веб-додатків, мобільних додатків (зокрема для Android-платформи), великих корпоративних систем, інтеграцію програмних рішень та багато іншого.

Простий приклад програми на Java, яка виводить «Hello, World!» на екран:

```
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello, World!");
    }
}
```

Основні елементи цього коду:

1. public class HelloWorld: визначає клас з іменем HelloWorld. У Java кожен код повинен бути розміщений у класах.
2. public static void main(String[] args): це метод main, який є точкою входу в програму. Викликається при запуску програми.
3. System.out.println("Hello, World!");: команда виведення тексту «Hello, World! » на консоль.

Щоб виконати цю програму, вам потрібно скомпілювати її в байт-код (завдяки чому вона стає переносимою між платформами) та запустити на віртуальній машині Java (JVM).

Порівняльний малюнок двох мов Kotlin та Java (рис.2.3).

Рисунок 2.3 – Порівняння двох мов програмування

IOS – це мобільна операційна система, розроблена компанією Apple Inc. для їхніх пристроїв, таких як iPhone та iPad. iOS включає в себе середовище для виконання додатків, а також основні додатки, такі як Safari, Камера, Карти та інші [11].

Swift та Objective-C - це мови програмування, які використовуються для розробки додатків під iOS.

Objective-C є об'єктно-орієнтованою мовою програмування, яка використовується для розробки програмного забезпечення, особливо на платформі Apple, такій як iOS та macOS.

Основною особливістю Objective-C є те, що вона є розширенням мови програмування C з додатковими можливостями, пов'язаними з об'єктно-орієнтованою розробкою [12].

Основні риси Objective-C:

1. Об'єктно-орієнтований підхід: Objective-C використовує об'єктно-орієнтований підхід до програмування, що дозволяє розробникам створювати програми, орієнтовані на об'єкти (класи та об'єкти), що сприяє полегшенню розробки та підтримці коду.
2. Спадкування і поліморфізм: Objective-C підтримує концепції спадкування та поліморфізму, що є ключовими складовими об'єктно-орієнтованого програмування. Це дозволяє створювати загальні класи, які можна успадковувати та розширювати.
3. Синтаксис: синтаксис Objective-C включає як елементи мови C, так і додаткові конструкції для роботи з об'єктами. Наприклад, для виклику методу може використовуватися синтаксис [об'єкт methodName].
4. Автоматичне управління пам'яттю: Objective-C використовує систему автоматичного управління пам'яттю (Automatic Reference Counting - ARC), що дозволяє автоматично визначати моменти, коли об'єкт може бути видалений з пам'яті.
5. Інтеграція зі звичайним C: Objective-C може викликати функції зі звичайного коду мови C, що робить його гнучким для використання у вже існуючих проектах.

Хоча Objective-C залишається історичним в контексті розробки для Apple, введення мови Swift в 2014 році принесло більше сучасний інструментарій для розробників та поступово витіснило Objective-C.

```
#import <Foundation/Foundation.h>
// Оголошення класу Person
@interface Person : NSObject
// Властивості класу
@property NSString *name;
@property NSInteger age;
// Методи класу
- (void)introduce;  // Метод для представлення особи
@end
// Реалізація класу Person
@implementation Person

// Реалізація методу introduce
- (void)introduce {
    NSLog(@"Привіт, мене звати %@ і мені %ld років.", self.name, (long)self.age);
```

```
}
@end
int main(int argc, const char * argv[]) {
   @autoreleasepool {

      // Створення об'єкта класу Person
      Person *person = [[Person alloc] init];
            // Встановлення значень властивостей
      person.name = @"Іван";
      person.age = 25;
            // Виклик методу introduce
      [person introduce];
   }
   return 0;
}
```

У цьому прикладі створюється клас Person з властивостями name і age, а також методом introduce, який виводить інформацію про особу в консоль. В main функції створюється об'єкт класу Person, встановлюються значення його властивостей і викликається метод introduce.

Swift – це мова програмування, розроблена компанією Apple для створення програм для iOS, macOS, watchOS та tvOS. Вона була представлена у 2014 році та стала заміною мови Objective-C для розробки на платформах Apple. Ось деякі ключові особливості Swift:

Сучасність та Безпечність: Swift була розроблена з урахуванням новітніх підходів до програмування.

Вона включає в себе безпечні елементи, які допомагають уникнути багатьох типових помилок, що можуть виникнути при розробці [13].

Інтерактивна платформа Playgrounds:

1. Розробники можуть використовувати Swift у відкритому середовищі Playgrounds, де вони можуть тут же бачити результат виконання свого коду. Це дозволяє ефективніше експериментувати та вдосконалювати код.
2. Спрощений синтаксис: у порівнянні з Objective-C, Swift має більш зрозумілий та читабельний синтаксис. Він більше схожий на інші сучасні мови програмування, що полегшує вивчення та використання.
3. Висока продуктивність: Swift призначена для максимальної продуктивності. Вона пропонує високу швидкість виконання коду, що робить її ідеальним вибором для розробки додатків для платформ Apple.
4. Активна спільнота та підтримка Apple: Swift користується підтримкою від компанії Apple, і вона постійно розвивається.

Крім того, у ній активно бере участь велика спільнота розробників.

Приклад простого коду на Swift для створення класу «Person» та виведення інформації про нього:

```
class Person {
   var name: String
   var age: Int
      init(name: String, age: Int) {
      self.name = name
      self.age = age
   }
      func introduce() {
      print("Привіт, мене звати \(name) і мені \(age) років.")
   }
}
// Створення об'єкта класу Person
let person = Person(name: "Іван", age: 25)
// Виклик методу introduce
person.introduce()
```

У цьому прикладі створюється клас Person з властивостями name і age, конструктором init для ініціалізації об'єкта та методом introduce для виведення інформації про особу.

Flutter – це відкритий фреймворк для розробки мобільних додатків, веб-додатків та настільних застосунків з єдиною кодовою базою. Він розроблений компанією Google і використовується для створення красивих та високопродуктивних інтерфейсів користувача. Основною мовою програмування для Flutter є Dart [14].

Dart – це мова програмування, яка використовується для розробки з багатьма платформами, але особливо підходить для роботи з Flutter. Ось кілька ключових рис Dart [15]:

1. Єдина кодова база для Flutter: Dart дозволяє розробникам використовувати єдиний код для створення мобільних, веб- та настільних додатків з використанням Flutter.
2. Швидкодія: Dart пропонує високу швидкодію виконання, що робить його ефективним для мобільних додатків та інших проектів.
3. Об'єктно-орієнтована та функціональна мова: Dart поєднує в собі принципи об'єктно-орієнтованого та функціонального програмування, що дозволяє розробникам вибирати підходи, які найкраще відповідають їхнім потребам.

4. Сучасний синтаксис: Dart має читабельний та сучасний синтаксис, що полегшує вивчення та використання мови.

Приклад простого віджету в Flutter на мові Dart:

```
import 'package:flutter/material.dart';

void main() {
  runApp(MyApp());
}

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        appBar: AppBar(
          title: Text('Мій перший додаток Flutter'),
        ),
        body: Center(
          child: Text('Привіт, Flutter!'),
        ),
      ),
    );
  }
}
```

У цьому прикладі створюється додаток Flutter з однією сторінкою, на якій розташований текст «Привіт, Flutter!». Dart використовується для опису структури додатка та вигляду інтерфейсу користувача за допомогою віджетів Flutter.

2.3 Опис backend розділу

Backend (бекенд) веб-розробки представляє собою серверну частину веб-додатка, яка відповідає за обробку бізнес-логіки та взаємодію з базою даних [16]. Це та компонента, яка працює в кулісах, взаємодіючи з інфраструктурою бази даних, обробляючи запити, які надходять від клієнтського інтерфейсу (фронтенд) та виконуючи різноманітні операції, необхідні для правильного функціонування веб-додатка (рис.2.4).

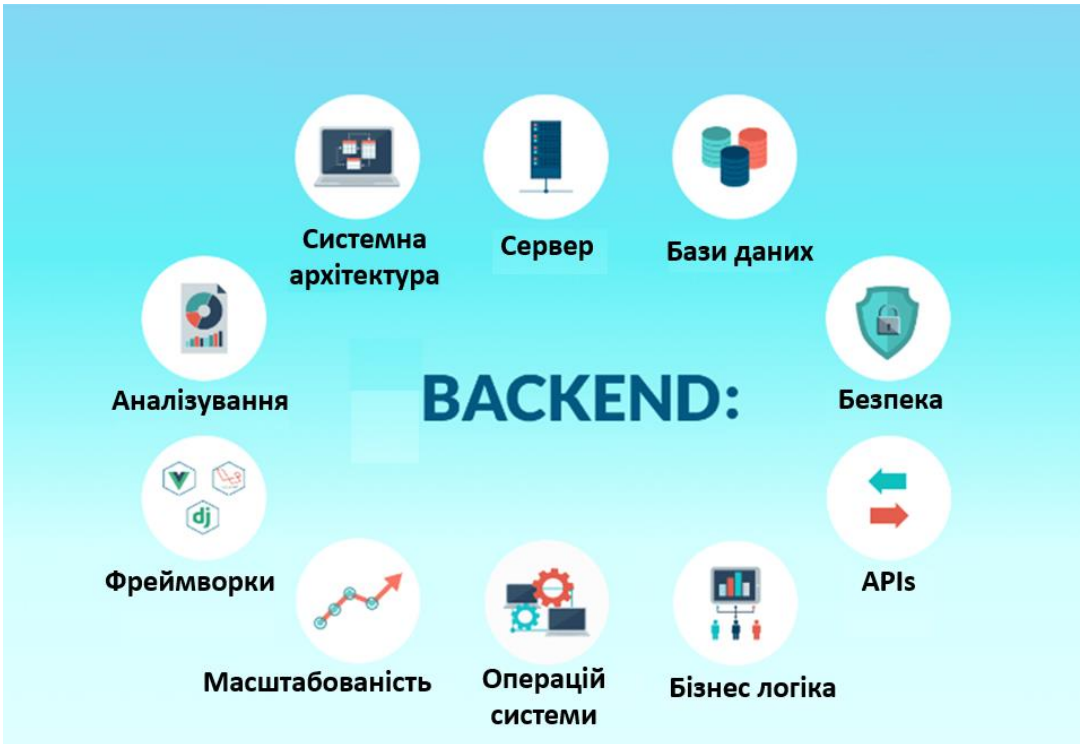

Рисунок 2.4 – Перелік знань для бекенд

Основні завдання бекенду включають обробку запитів, взаємодію з базою даних для зберігання та отримання інформації, забезпечення безпеки даних, обчислення та виконання бізнес-логіки. Backend також відповідає за забезпечення високої продуктивності та швидкодії веб-додатка.

У контексті розділення на фронтенд та бекенд, бекенд займається всією «задньою» частиною системи, яка не є відображеною на сторінці користувача. Це включає в себе обробку серверних запитів, роботу з базою даних, автентифікацію, авторизацію та інші аспекти, які потрібні для коректного функціонування веб-додатка.

Основні елементи бекенду включають:

1. Сервер: це програмне забезпечення, яке слухає та обробляє запити від клієнтського інтерфейсу, виконуючи певні дії та надсилаючи відповіді (рис.2.5).
   Опис: сервер є програмним забезпеченням або комп'ютерною системою, яка призначена для обробки та відповіді на запити, надсилані від клієнтського інтерфейсу. Він грає ключову роль у взаємодії між фронтендом та бекендом, виконуючи обчислення, обробку даних та відповідей на клієнтські запити [16].

   Приклад: нехай у веб-додатку користувач взаємодіє з електронною формою, вводячи дані та натискуючи кнопку «Відправити». Коли це відбувається, клієнтський інтерфейс (фронтенд) відправляє запит на сервер. Сервер, у свою чергу, обробляє цей запит, перевіряє дані, можливо, взаємодіє з базою даних для збереження інформації, і повертає відповідь, яка може

бути повідомленням про успішне відправлення форми чи повідомленням про помилку. У цьому контексті сервер відповідає за обробку та керування логікою додатку, тоді як фронтенд відображає інформацію та забезпечує користувачський інтерфейс

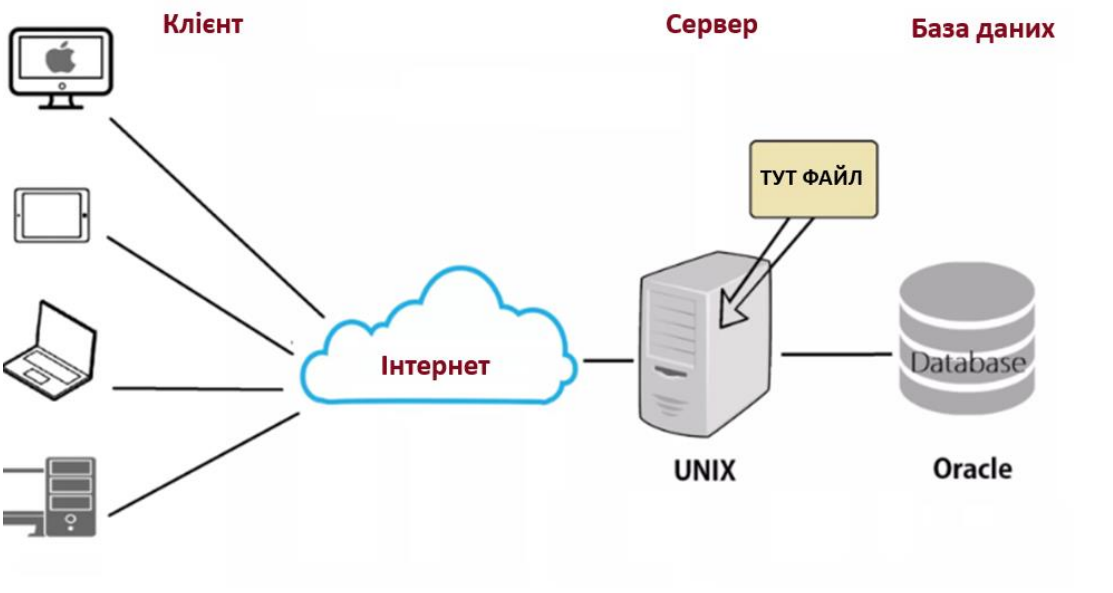

Рисунок 2.5 – Приклад роботи сервера

2. База даних: Місце, де зберігається інформація, необхідна для додатка. Взаємодія з базою даних дозволяє зберігати, оновлювати та видаляти дані (рис.2.6).
   Опис: база даних – це організована колекція даних, яка зберігається та управляється таким чином, щоб її можна було легко оновлювати, отримувати та управляти. В контексті бекенд-розробки, база даних використовується для зберігання інформації, яка необхідна для функціонування додатка [17].

   Приклад: розглянемо веб-магазин. У базі даних можуть бути таблиці для зберігання інформації про продукти (назва, ціна, категорія), користувачів (ім'я, електронна пошта, пароль) та замовлення (продукти, кількість, адреса доставки). Коли користувач робить замовлення через фронтенд, сервер взаємодіє з базою даних, щоб зберегти цю інформацію. Пізніше, коли користувач перевіряє стан свого замовлення, сервер знову взаємодіє з базою даних, щоб отримати відповідні дані. Така організація дозволяє зберігати та взаємодіяти з інформацією, необхідною для роботи веб-додатка.

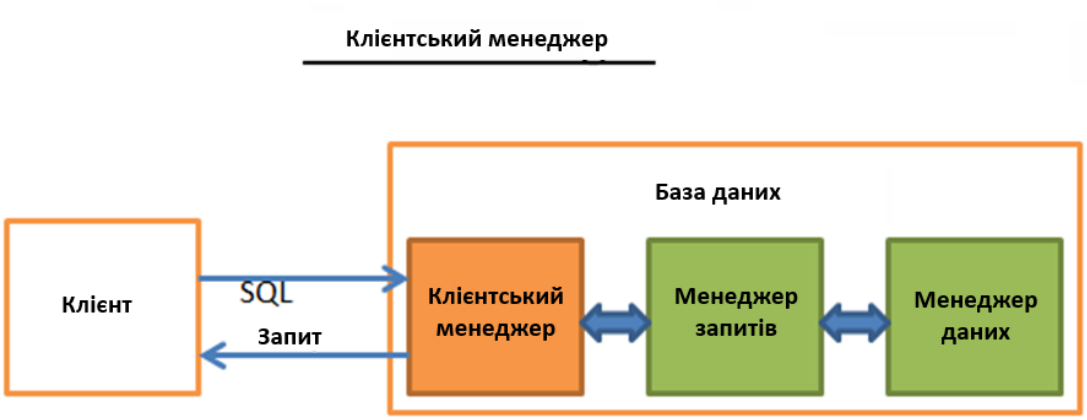

Рисунок 2.6 – Приклад роботи баз даних

Приклади Програм:

1. MySQL: MySQL – це популярна система управління базами даних (СУБД), яка використовує мову SQL для взаємодії з даними. Вона часто використовується для зберігання структурованих даних.

Основні риси та особливості MySQL включають:

1. Відкритість та безкоштовність: MySQL є відкритою програмою з відкритим кодом, що означає, що ви можете отримати доступ до її вихідного коду, модифікувати його та використовувати безкоштовно.
2. Мова SQL: вона використовує мову SQL для роботи з базою даних. SQL дозволяє виконувати різноманітні операції, такі як вибірка (SELECT), вставка (INSERT), оновлення (UPDATE) та видалення (DELETE) даних.
3. Структуровані дані: MySQL відмінно справляється зі зберіганням структурованих даних, таких як таблиці, які мають жорстко визначені схеми з фіксованими колонками та типами даних.
4. Транзакції та ACID: MySQL підтримує транзакції, що забезпечує консистентність бази даних в умовах збоїв. Вона також відповідає вимогам ACID (Atomicity, Consistency, Isolation, Durability).
5. Підтримка індексів та зовнішніх ключів: MySQL дозволяє створювати індекси для поліпшення швидкодії операцій пошуку та підтримує використання зовнішніх ключів для забезпечення цілісності даних.
6. Масштабованість: вона може бути успішно використана як для невеликих веб-сайтів, так і для великих корпоративних систем [17].

Приклад SQL-запиту для створення таблиці користувачів:

```
CREATE TABLE users (
  id INT AUTO_INCREMENT PRIMARY KEY,
```

```
  username VARCHAR(50) NOT NULL,

  email VARCHAR(100) NOT NULL UNIQUE,
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

2. MongoDB: MongoDB - це NoSQL база даних, яка використовує документи у форматі JSON для зберігання даних. Вона часто використовується для роботи з неструктурованими або півструктурованими даними.

Основні характеристики та особливості MongoDB включають:

1. NoSQL (Not Only SQL): MongoDB належить до категорії NoSQL баз даних, що означає, що вона не використовує традиційний мову SQL для взаємодії з даними та не має фіксованої схеми.
2. Документоорієнтована: вона працює з документами у форматі JSON, де кожен документ може містити різні поля та дані. Це дозволяє зручно зберігати неструктуровані або півструктуровані дані.
3. Гнучкість схеми: MongoDB дозволяє вам додавати нові поля до документів без необхідності зміни схеми всіх існуючих записів.
4. Індексація та запити: підтримує можливість індексації для покращення швидкодії запитів. Дозволяє виконувати різноманітні запити, використовуючи мову запитів MongoDB.
5. Масштабованість: забезпечує горизонтальну масштабованість, що дозволяє легко розширювати базу даних на великі обсяги даних та високий обсяг трафіку.
6. Реплікація та шардування: можливість налаштовувати реплікацію для забезпечення високої доступності та шардування для розподілу даних [18].

Приклад структури документа в MongoDB:

```
{
  "_id": ObjectId("5f5e7a2f9238182b8a35fd56"),
  "name": "John Doe",
  "age": 30,
  "email": "john.doe@example.com",
  "address": {
    "city": "New York",
    "zipcode": "10001"
  }
}
```

У прикладі кожен документ має унікальний ідентифікатор (_id) та різні поля, включаючи вкладений об'єкт адреси

У цих прикладах SQL та MongoDB-запитів показано, як вибрати дані з бази даних за певними умовами. SQL-запити використовуються для реляційних баз даних, тоді як MongoDB-запити застосовуються до NoSQL баз даних, таких як MongoDB [19].

3. Бізнес-логіка: це логіка або правила, які визначають, які операції потрібно виконувати з даними та як обробляти різні сценарії.
   Опис: бізнес-логіка представляє собою сукупність правил, які визначають, як додаток має взаємодіяти з даними та як він повинен обробляти різні сценарії відповідно до потреб бізнесу.

   Функції:
   1. Обробка даних: бізнес-логіка визначає правила для збору, обробки та збереження даних. Наприклад, у системі електронної комерції вона визначатиме, які дані про товари варто зберігати, як їх структурувати та як виводити користувачам.
   2. Прийняття рішень: бізнес-логіка включає в себе логіку прийняття рішень на основі вхідних даних. Наприклад, система управління запасами може вирішувати, коли замовляти нові товари на склад, базуючись на різних факторах, таких як кількість наявних товарів, швидкість їх реалізації та прогнозована попит.
   3. Автоматизація процесів: бізнес-логіка визначає автоматизовані процеси в додатку. Наприклад, у системі управління клієнтами вона може визначати, що при отриманні нового замовлення потрібно автоматично створювати профіль клієнта та відсилати підтвердження.

Приклад.

Розглянемо бізнес-логіку в системі онлайн-банкінгу. Коли клієнт робить переказ коштів, бізнес-логіка визначає, як зменшити суму на одному рахунку і збільшити на іншому. Якщо виникає помилка або переказ неможливий, бізнес-логіка визначає, як обробити цю ситуацію, можливо, надсилаючи повідомлення користувачу чи блокуючи операцію.

4. API (Application Programming Interface): інтерфейс, який дозволяє взаємодіяти з бекендом через HTTP-запити. API може повертати дані у форматі JSON або XML.
   Опис: API (інтерфейс програмування додатків) визначає набір правил та інструментів, які дозволяють різним програмам взаємодіяти одна з одною. У веб-розробці, API використовується для забезпечення комунікації між фронтендом та бекендом, а також для обміну даними з іншими сервісами чи додатками [20].

Функції:

1. Комунікація бекенд-фронтенд: API дозволяє фронтенду (клієнтській стороні) звертатися до бекенду (серверної сторони) для отримання або відправлення даних. Наприклад, фронтенд може використовувати API для запиту інформації про користувача [20].
2. Взаємодія з іншими сервісами: зовнішні сервіси та додатки можуть використовувати API для обміну даними з основним додатком. Наприклад, соціальні мережі можуть використовувати API для інтеграції з іншими веб-сайтами.
3. Стандартизація запитів та відповідей: API визначає структуру та формат запитів та відповідей. Це сприяє стандартизації взаємодії між різними компонентами системи.

Приклад:
Розглянемо API для взаємодії з сервісом погоди. Запит до API може мати вигляд:
GET /weather?city=Kyiv
Відповідь може бути у форматі JSON і виглядати так:

```
{
  "city": "Kyiv",
  "temperature": 25,
  "conditions": "Sunny"
}
```

Фронтенд може використовувати це API для отримання актуальної погоди в Києві та відображення цих даних на веб-сторінці.

Різні мови програмування використовуються для розробки бекенду, і ось деякі з них:

1. Node.js (JavaScript): використовується для розробки бекенду з використанням JavaScript. Зазвичай використовується з фреймворками Express або Koa.
2. Django (Python): Python часто використовується для розробки бекенду за допомогою фреймворка Django, який полегшує і прискорює процес.
3. Ruby on Rails (Ruby): використовується для швидкого розвитку веб-додатків, надає структурований підхід до розробки.
4. Java (Spring): Java використовується для розробки бекенду, часто з використанням фреймворка Spring.
5. PHP (Laravel): PHP використовується для розробки бекенду, а Laravel є одним з популярних фреймворків для PHP.

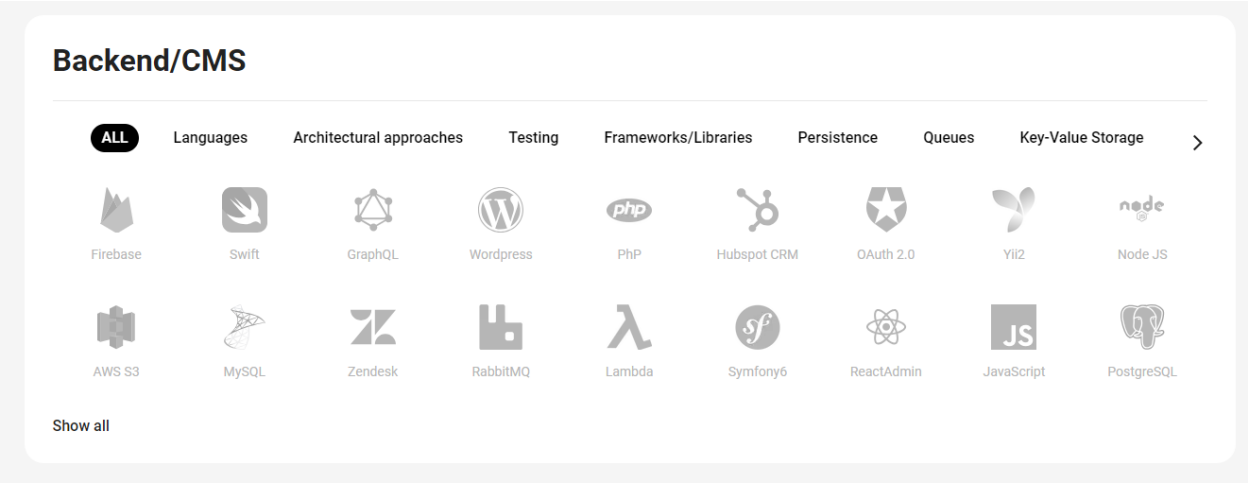

Рисунок 2.7 – Перелік інструментів бекенд

На рисунку 2.7 продемонстровано добірку мов програмування та технічних засобів які використовує IDAP у своїй розробці проектів.

Для прикладу розглянемо 2 популярні мови, які використовує IDAP у реалізації своїх проектів.

Node.js – це середовище виконання для JavaScript, яке дозволяє використовувати цю мову програмування для розробки серверної частини додатків. Використання JavaScript на обох боках (фронтенді та бекенді) дозволяє розробникам використовувати ту ж мову для побудови всього додатка, що полегшує розробку та обслуговування [21].

Функції та Особливості:

1. Серверна розробка: використовуючи Node.js, розробники можуть створювати сервери та обробляти запити від клієнтських додатків. Це робить Node.js популярним для створення бекенду веб-додатків.
2. Асинхронність: Node.js використовує асинхронний підхід, що дозволяє обробляти багатозадачні операції без блокування інших подій. Це особливо важливо в сучасних веб-додатках з великою кількістю одночасних запитів.
3. Модульність: Node.js побудований на модульній архітектурі, що дозволяє розробникам використовувати різні модулі та бібліотеки для полегшення розробки.

Приклад:

Розглянемо простий веб-сервер, створений з використанням Node.js та фреймворку Express:

```
// Підключення модуля Express
const express = require('express');
const app = express();
```

```
// Визначення маршруту
app.get('/', (req, res) => {
  res.send('Привіт, це веб-сервер на Node.js!');
});

// Прослуховування порту 3000
app.listen(3000, () => {
  console.log('Сервер слухає на порту 3000');
});
```

У цьому прикладі Node.js використовується з Express для створення простого веб-сервера, який відповідає на запити та відправляє повідомлення «Привіт, це веб-сервер на Node.js! » при відкритті головної сторінки.

PHP – це скриптова мова програмування, спеціально призначена для розробки веб-застосунків та взаємодії з веб-серверами. Вона вбудована безпосередньо в HTML-код, що полегшує створення динамічних веб-сторінок.

Основні особливості Laravel включають:

1. Елегантний синтаксис: laravel пропонує чистий та елегантний синтаксис коду, що полегшує розробку та підтримку.
2. ORM (Object-Relational Mapping): вбудований ORM - Eloquent дозволяє взаємодіяти з базою даних, використовуючи звичний для розробників об'єктно-орієнтований підхід.
3. Маршрутизація: легко визначати маршрути для обробки HTTP-запитів.
4. Шаблонізація Blade: Blade – простий та потужний движок шаблонів, який дозволяє вставляти PHP-код безпосередньо в шаблони.
5. Міграції та засіб захисту: забезпечує можливість контролю версій бази даних та забезпечує базовий захист від атак[22].

Приклад PHP-коду:

```
<?php
  $name = "John Doe";
  echo "Hello, $name!";
?>
```

2.4 Опис QA розділу

QA (Quality Assurance) або тестувальник – це фахівець, чий основний завданням є перевірка програмного продукту на відповідність визначеним вимогам та виявлення помилок чи недоліків. Робота QA спрямована на забезпечення якості програмного забезпечення перед його випуском або впровадженням (рис.2.8).

Основні обов'язки тестувальника включають у себе:

1. Функціональне тестування: перевірка того, чи виконує програма свої основні функції вірно.
2. Тестування взаємодії: визначення того, як програма взаємодіє з користувачем та іншими компонентами системи.
3. Тестування витривалості: визначення та перевірка витривалості та стабільності програмного продукту.
4. Автоматизація тестування: використання інструментів автоматизації для виконання тестів, що дозволяє швидше та ефективніше проводити тестування.
5. Тестування безпеки: виявлення та усунення потенційних загроз безпеці програмного забезпечення [23].

Рисунок 2.8 – Приклад роботи тестувальника

Для виконання своєї роботи QA використовує різноманітні інструменти та підходи:

1. Інструменти автоматизації тестування: наприклад, Selenium для веб-тестування, JUnit або TestNG для автоматизації тестів у Java, PyTest для Python тощо.

Інструменти автоматизації тестування відіграють важливу роль у розробці програмного забезпечення, забезпечуючи ефективне виконання тестів та виявлення помилок. Декілька прикладів таких інструментів:

1. Selenium.

Призначення: використовується для автоматизації тестів веб-додатків.

Приклад коду (Java):

```
WebDriver driver = new ChromeDriver();
driver.get("https://www.example.com");
WebElement element = driver.findElement(By.name("username"));
element.sendKeys("john_doe");
element.submit();
```

2. JUnit та TestNG:

Призначення: фреймворки для написання та виконання тестів у мові програмування Java.

Приклад коду (JUnit):

```
import static org.junit.jupiter.api.Assertions.assertEquals;
import org.junit.jupiter.api.Test;

class MyTest {
  @Test
  void addition() {
    assertEquals(2, 1 + 1);
  }
}
```

3. PyTest:

Призначення: інструмент для автоматизації тестів у мові програмування Python.

Приклад коду (Python):

```
def test_addition():
 result = add(1, 2)
 assert result == 3
```

4. Cypress:

Призначення: інструмент для тестування веб-додатків, який працює безпосередньо в браузері.

Приклад коду (JavaScript):

```
describe('My First Test', () => {
  it('Does not do much!', () => {
    cy.visit('https://www.example.com');
    cy.contains('Hello').should('be.visible');
  });
});
```

Ці інструменти допомагають розробникам та тестувальникам автоматизувати тестові сценарії, щоб забезпечити швидке виявлення та виправлення помилок у програмному забезпеченні [24].

2. Системи керування тестовими випробуваннями (Test Management Tools): забезпечують організацію та виконання тестів, ведення звітності. Приклади включають TestRail, Zephyr, TestLink.

Системи керування тестовими випробуваннями (Test Management Tools):

Системи керування тестовими випробуваннями відіграють ключову роль у забезпеченні ефективного управління тестовим процесом, відслідковуванні виконання тестів та генерації звітності. Нижче подано огляд кількох таких інструментів:

TestRail.

Призначення: забезпечує централізоване управління тестами, ведення та організацію тестової документації, а також генерацію звітів.

Основні функції:

1. Організація тест-кейсів та тест-сьютів.
2. Запуск, відстеження виконання та автоматичне оновлення статусу тестів.
3. Аналіз результатів випробувань та генерація звітів.

Zephyr.

Призначення: інтегрована система керування тестовими випробуваннями для великих та середніх підприємств.

Основні функції:

1. Створення, редагування та виконання тест-кейсів.
2. Співпраця з іншими інструментами для автоматизації тестування.
3. Звітність про прогрес випробувань та виконані тестові цикли.

TestLink.

Призначення: відкрите програмне забезпечення для керування тестовими випробуваннями та тестовою документацією.

Основні функції:

1. Зберігання та організація тестових сценаріїв.
2. Відстеження виконання тестів та результатів.
3. Інтеграція з іншими інструментами для автоматизації та управління випробуваннями[24].

Інструменти для тестування безпеки: наприклад, OWASP ZAP для тестування вразливостей веб-додатків.

Інструменти для тестування продуктивності: Apache JMeter для тестування швидкодії та витривалості систем.

Одним із найпопулярніших інструментів в цій категорії є Apache JMeter. Давайте розглянемо його більш детально:

Apache JMeter.

Призначення: Apache JMeter є потужним інструментом для виконання функціонального, навантажувального, стрес-тестування та тестування продуктивності веб-додатків.

Основні функції та можливості:

Стрес-тестування: вимірювання швидкодії та стійкості системи при великому навантаженні.

Навантажувальне тестування: симуляція реального навантаження на систему для визначення її працездатності.

Функціональне тестування: перевірка різних аспектів функціональності веб-додатків.

Тестування API: можливість тестування API за допомогою HTTP-запитів.

Графічне інтерфейсне та консольне використання: Надає інтерфейс для графічного редагування та конфігурації тестових планів.

Розширюваність: підтримує плагіни для розширення можливостей[25].

Приклад використання.

Допустимо, ви розробляєте великий веб-портал, і вам потрібно визначити, як він веде себе при одночасному доступі тисяч користувачів. За допомогою Apache JMeter ви можете створити тестові плани, які будуть генерувати різні види навантаження на ваш веб-портал, а потім вимірювати його продуктивність та стабільність під час роботи під високим навантаженням.

3. Інструменти для тестування візуального інтерфейсу: наприклад, Applitools для автоматизованого візуального тестування [26].

Один із таких інструментів – Applitools, є потужним рішенням для автоматизованого візуального тестування. Розглянемо його основні характеристики та застосування:

Призначення: applitools використовується для автоматизованого тестування візуального інтерфейсу веб-додатків та мобільних додатків.

Основні функції та можливості:

Візуальне тестування масштабів: перевірка, як коректно відображається вміст на різних роздільних здатностях та пристроях.

Виявлення візуальних помилок: виявлення різниці між очікуваним та фактичним візуальним представленням елементів сторінки або екрану.

Автоматизація скріншотів: збір та порівняння скріншотів сторінок для виявлення будь-яких змін.

Підтримка різних платформ: можливість тестування як веб-додатків, так і мобільних додатків.

Інтеграція з іншими інструментами: можливість легко інтегрувати Applitools з іншими інструментами для автоматизації.

Приклад використання.

При розробці веб-сайту або мобільного додатку, важливо переконатися, що елементи і сторінки правильно відображаються на різних пристроях та браузерах. За допомогою Applitools, ви можете автоматизувати процес перевірки візуальної ідентичності та виявлення найменших змін у вигляді елементів інтерфейсу під час розробки та після внесення змін. Це допомагає забезпечити консистентність та високу якість користувацького інтерфейсу.

Робота QA важлива для забезпечення якості програмного продукту та задоволення потреб користувачів.

2.4 Опис devops розділу

DevOps (Development and Operations) – це підхід до розробки програмного забезпечення, який об'єднує розробку і операції з метою покращення ефективності і швидкості постачання продуктів. DevOps включає в себе автоматизацію процесів розробки, тестування та впровадження програм, а також спільну відповідальність розробників і операторів за весь життєвий цикл продукту (рис. 2.9).

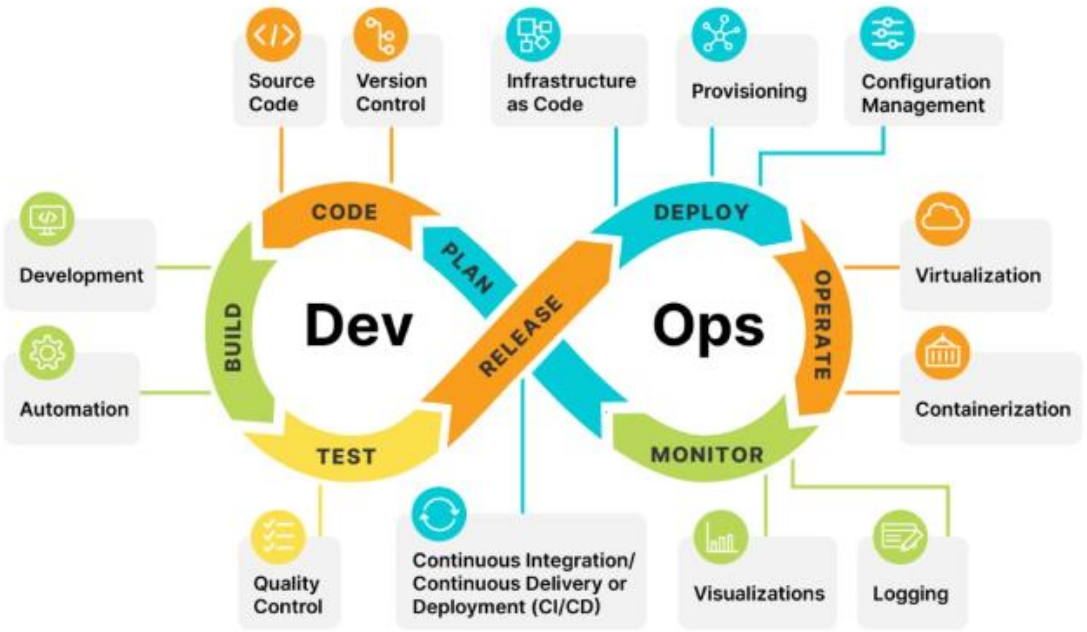

Рисунок 2.9 – Схема роботи девопс

Основні аспекти роботи DevOps включають:

Автоматизація інфраструктури: використання інструментів, таких як Ansible, Puppet, чи Chef для автоматизації створення та управління інфраструктурою [27].

Автоматизація інфраструктури є важливою складовою сучасного підходу до розробки програмного забезпечення і управління інфраструктурою. Інструменти, такі як Ansible, Puppet та Chef, надають зручний та ефективний спосіб автоматизації процесів створення, налаштування та управління інфраструктурою.

Ansible.

Призначення: Ansible використовується для автоматизації рутинних задач, таких як установка програм, налаштування серверів, розгортання програмного забезпечення та інші операції управління конфігурацією.

Основні функції та можливості:

Декларативний код: опис інфраструктури здійснюється у вигляді YAML-файлів, що полегшує зрозуміння та редагування конфігурації.

Агент-less архітектура: Ansible не вимагає встановлення агентів на керованих вузлах, що спрощує розгортання та управління.

Масштабованість: здатність керувати багатьма серверами одночасно, що дозволяє легко масштабувати інфраструктуру.

Інтеграція з іншими інструментами: Ansible легко інтегрується з іншими інструментами розробки та конфігураційного управління.

Приклад використання.

Нехай у вас є велика ферма серверів, і ви хочете забезпечити, щоб всі вони мали однаковий стек програм та конфігурацію. Ansible може бути використаний для написання декларативного коду, який описує необхідний стан системи. При застосуванні цього коду Ansible автоматично налаштує кожен сервер згідно з описаною конфігурацією. Це полегшує рутинні завдання адміністрування та забезпечує консистентність інфраструктури.

Continuous Integration (CI): процес автоматизованої перевірки коду розробників в репозиторії, наприклад, з використанням інструментів, таких як Jenkins, GitLab CI, або Travis CI.

Continuous Integration – це практика розробки програмного забезпечення, яка передбачає автоматизоване об'єднання коду розробників у спільний репозиторій та автоматичну перевірку цього коду з метою виявлення можливих конфліктів та помилок. Основна ідея CI - інтегрувати зміни в коді регулярно та швидко, щоб уникнути проблем, які можуть виникнути при об'єднанні великої кількості коду пізніше.

Інструменти Continuous Integration:

Jenkins.

Опис: jenkins є одним з найпоширеніших інструментів CI/CD. Він надає можливість автоматизації будь-яких завдань, пов'язаних з процесом розробки та тестування.

Приклад використання: після кожного коміту у репозиторій, Jenkins може автоматично запускати набір тестів, перевіряти стиль коду, та, в разі успішного проходження тестів, автоматично розгортати оновлення на тестовий сервер.

GitLab CI.

Опис: інтегрована система Continuous Integration в GitLab. Вона дозволяє організувати та автоматизувати процеси CI/CD безпосередньо в інтерфейсі GitLab.

Приклад використання: коли розробник робить коміт у гілку, GitLab CI автоматично запускає процес компіляції, виконання тестів та, в разі успішного завершення, розгортання на визначений сервер.

Travis CI.

Опис: travis CI – це інструмент для автоматизації тестування та розгортання, який інтегрується з репозиторіями на GitHub.

Приклад використання: після пушу коду в GitHub, Travis CI може автоматично стягувати оновлення, запускати тести та вести розгортання на живий сервер у випадку успішного тестування [27].

Переваги Continuous Integration:

1. Забезпечує раннє виявлення помилок у коді.
2. Зменшує час між написанням коду та його впровадженням в продакшн.
3. Забезпечує стабільність роботи програмного забезпечення завдяки автоматизованим тестам.
4. Покращує співпрацю команди розробників через регулярні об'єднання коду [25].

Continuous Delivery (CD): автоматизоване впровадження коду в середовище виробництва з мінімальними ручними втручаннями.

Continuous Delivery – це практика, що передбачає автоматизоване впровадження коду в середовище виробництва з мінімальними ручними втручаннями. Основна ідея полягає в тому, щоб забезпечити готовність до випуску продукту на виробництво в будь-який момент.

Це забезпечується за допомогою автоматизованих процесів, які включають в себе тестування, збірку, розгортання та інші етапи впровадження коду.

Елементи Continuous Delivery.

Автоматизована збірка (Automated Build):

Опис: автоматизована процедура компіляції та збірки програмного коду для створення виконуваних файлів або інших артефактів.

Приклад використання: Використання інструментів, таких як Maven, Gradle або npm, для автоматичної збірки програмного коду.

Автоматизоване тестування (Automated Testing).

Опис: використання автоматизованих тестів для перевірки функціональності, продуктивності та стабільності програмного продукту.

Приклад використання: використання Selenium для веб-тестування або JUnit для автоматизованого тестування в Java.

Автоматизоване розгортання (Automated Deployment).

Опис: автоматична передача виконуваних файлів чи артефактів на цільовий сервер або середовище.

Приклад використання: використання інструментів розгортання, таких як Ansible, Docker або Kubernetes, для автоматизованого розгортання.

Контейнеризація.

Опис: використання технологій контейнеризації (наприклад, Docker) для упакування програмного забезпечення та всіх його залежностей у відокремлені контейнери.

Приклад використання: створення Docker-контейнера для програмного продукту та використання його для автоматизованого розгортання.

Переваги Continuous Delivery:

1. Зниження ризику людських помилок завдяки автоматизованому процесу впровадження.
2. Збільшення ефективності розробки завдяки швидкому виявленню та виправленню помилок.
3. Забезпечення готовності до випуску продукту в будь-який момент.
4. Зниження часу, необхідного для впровадження нового функціоналу чи виправлення помилок.

Моніторинг і логування: використання інструментів, таких як Prometheus, ELK Stack, Grafana для збору та аналізу даних про продуктивність та надійність.

Моніторинг і логування є важливою частиною розробки та управління інфраструктурою. Ці практики дозволяють вам відстежувати та аналізувати продуктивність, надійність та стан системи в реальному часі [27].

Елементи моніторингу і логування.

Prometheus.

Опис: Prometheus – це система моніторингу та оповіщення, яка призначена для збору та аналізу метрик системи в реальному часі.

Приклад використання: Збір та відображення метрик HTTP-сервера, вимірювання завантаження системи.

ELK Stack (Elasticsearch, Logstash, Kibana).

Опис: ELK Stack – це комбінація інструментів, які включають Elasticsearch для зберігання та пошуку логів, Logstash для їх обробки та Kibana для візуалізації та аналізу.

Приклад використання: Збір та аналіз логів додатка для виявлення проблем та відслідковування подій.

Grafana.

Опис: grafana – це інструмент для візуалізації та аналізу даних з різних джерел, включаючи метрики, логи та бази даних.

Приклад використання: створення інтерактивних та графічних панелей для моніторингу системи.

Переваги моніторингу і логування:

1. Виявлення проблем: моніторинг дозволяє оперативно виявляти проблеми та реагувати на них до того, як вони вплинуть на користувачів.
2. Оптимізація продуктивності: аналіз метрик дозволяє вдосконалювати продуктивність та реагувати на зміни завантаження.
3. Безпека: логи можуть бути використані для виявлення потенційних загроз та аналізу безпекових подій.
4. Планування ресурсів: моніторинг допомагає в плануванні ресурсів та розподілі їх відповідно до потреб системи. Моніторинг і логування є важливою частиною розробки та управління інфраструктурою. Ці практики дозволяють вам відстежувати та аналізувати продуктивність, надійність та стан системи в реальному часі.

Контейнеризація: використання технологій контейнерів, таких як Docker, для упаковки та розгортання додатків та їх залежностей.

Контейнеризація – це підхід до розробки та розгортання програмного забезпечення, який дозволяє упаковувати додатки та їх залежності разом в легкі та переносні контейнери. Один з найпопулярніших інструментів для контейнеризації - Docker.

Основні аспекти контейнеризації:

Docker.

Опис: Docker – це платформа для розробки, доставки та запуску додатків в контейнерах.

Приклад використання: Упакування веб-додатка, його залежностей та конфігурації в контейнер для легкого розгортання в будь-якому середовищі.

Користь від контейнеризації:

1. Портативність: контейнери дозволяють переносити додатки між різними середовищами, забезпечуючи консистентність робочого середовища.
2. Легкість розгортання: завдяки контейнерам, розгортання додатків стає простим та швидким процесом.
3. Ізоляція: контейнери ізолюють додаток та його залежності, запобігаючи конфліктам та забезпечуючи стабільність [27].

Оркестрація контейнерів.

Опис: для керування та оркестрації багатьох контейнерів використовуються інструменти, такі як Kubernetes або Docker Swarm.

Приклад використання: оркестрація роботи сотень або тисяч контейнерів, управління їхнім життєвим циклом.

Переваги контейнеризації:

1. Стандартизація: усі залежності та конфігурації включені в контейнер, що забезпечує стандартизацію середовища.
2. Швидке розгортання: додатки можуть бути швидко розгорнуті в будь-якому середовищі, яке підтримує Docker.
3. Масштабованість: легкість копіювання та розгортання контейнерів дозволяє масштабувати додатки злегкістю.

Оркестрація контейнерів: використання інструментів, таких як Kubernetes, Docker Swarm, для автоматизованого управління та оркестрації контейнерами в середовищі виробництва.

Оркестрація контейнерів – це процес автоматизованого управління та координації роботи контейнеризованих додатків в розподіленому середовищі виробництва. Інструменти, такі як Kubernetes та Docker Swarm, відповідають за цей процес, забезпечуючи ефективне впровадження та масштабування контейнерів.

Основні аспекти оркестрації контейнерів.

Kubernetes.

Опис: Kubernetes – це відкрите програмне забезпечення для автоматизованого розгортання, масштабування та управління контейнеризованими додатками.

Приклад використання: управління та масштабування сотень або тисяч контейнерів, оркестрація додатків в реальному часі.

Docker Swarm.

Опис: Docker Swarm – це офіційний інструмент для оркестрації контейнерів від Docker, призначений для простоти використання.

Приклад використання: розгортання та керування контейнерами за допомогою вбудованих засобів Docker Swarm.

Групування та масштабування.

Опис: оркестраційні інструменти дозволяють групувати контейнери в служби та забезпечують автоматичне масштабування в залежності від навантаження.

Приклад використання: збільшення або зменшення кількості контейнерів у групі для забезпечення стабільної роботи додатка [27].

Балансування навантаження.

Опис: автоматичне розподілення запитів між контейнерами для збалансування навантаження та оптимізації продуктивності.

Приклад використання: рівномірне розподілення трафіку між різними екземплярами додатка для підтримки стабільності та ефективності.

Переваги оркестрації контейнерів.

Автоматизація: забезпечення автоматичного розгортання та управління контейнерами.

Масштабованість: можливість масштабувати додатки за потреби.

Відмовостійкість: забезпечення неперервності роботи додатків під час відмови окремих контейнерів чи вузлів.

Collaboration and Communication: забезпечення ефективного спілкування та співпраці між розробниками і операторами [27].

DevOps дозволяє прискорити час розробки та впровадження продукту, знизити ймовірність помилок, покращити якість програмного забезпечення та забезпечити більш ефективне використання ресурсів.

Висновки за розділом

В даному розділі був проведений загальний огляд структури IDAP, що надало зрозуміння загального ландшафту системи.

Виділені основні складові та їх взаємодія, що створює основу для подальшого розгляду часткових аспектів.

Опис frontend розділу ретельно розглянутий фронтенд IDAP, описана його структура та функціональність.

Виділені ключові елементи інтерфейсу та їх роль у забезпеченні зручного користування системою.

Опис backend розділу був детально проаналізований, включаючи огляд архітектури та технологій, що використовуються в цьому компоненті.

Особливий акцент зроблено на обробці бізнес-логіки та взаємодії з базою даних для забезпечення ефективного функціонування.

Опис QA розділу визначено стратегії та методи QA, використовувані в IDAP з метою забезпечення високої якості продукту.

Здійснено аналіз процесів тестування та їх вплив на вдосконалення якості розробки.

Опис devOps розділу детально розглядає процеси інтеграції розробки та операцій з метою автоматизації та прискорення процесу впровадження.

Зазначені інструменти та практики, які використовуються для досягнення мети забезпечення стабільності та надійності розробки.

Комплексний аналіз цих розділів дозволяє отримати повний обзор структури та взаємодії різних компонентів IDAP, визначити ключові фактори, які впливають на якість та ефективність системи.

3 ПРОЕКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Генерація, розробка та реалізація проекту

Для розробки завдань та ефективного управління проектами, використовують різноманітні інструменти. Один із найбільш популярних та потужних засобів для цього – Jira. Jira є високорівневою платформою для спільної роботи, що надає широкий спектр можливостей для планування, відстеження та керування розробкою програмного забезпечення.

Завдяки Jira можна легко створювати та призначати завдання, визначати пріоритети, встановлювати строки виконання та відстежувати прогрес робіт. Кожне завдання в Jira може бути докладно описане, призначене конкретним виконавцем, і мати відповідність до певного проекту чи задачі (рис.3.1).

Jira також підтримує роботу з різними типами завдань, такими як задачі, історії користувачів, помилки, завдання на вдосконалення та інші. Інтеграція з іншими інструментами розробки, такими як Bitbucket чи Confluence, робить Jira ідеальним інструментом для повного циклу розробки програмного забезпечення.

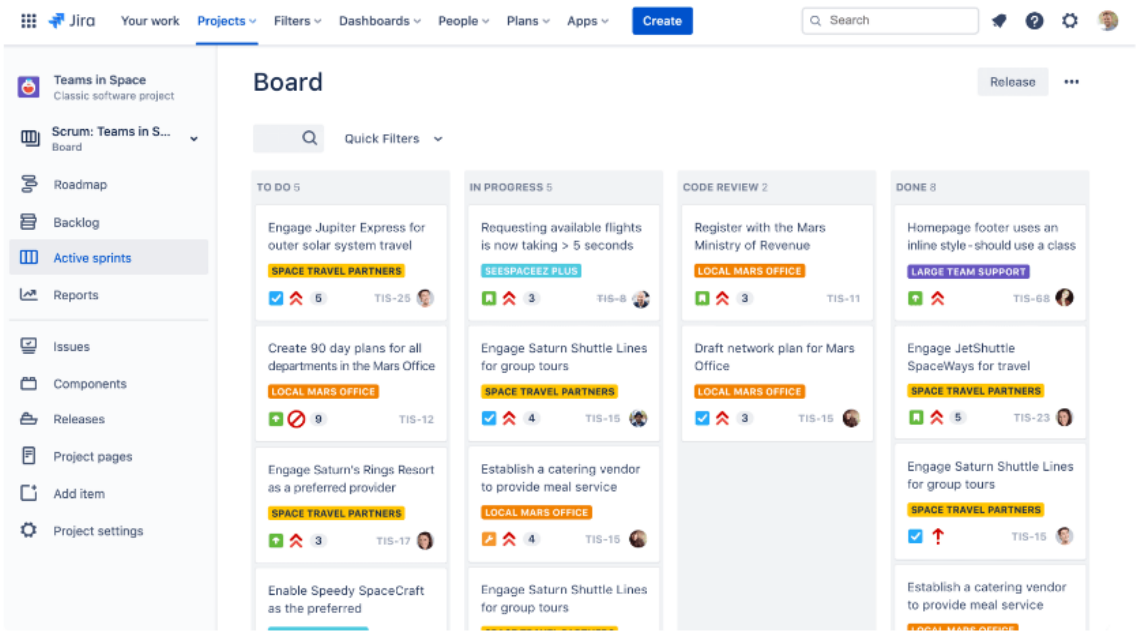

Рисунок 3.1 – Робоче вікно Jira

Розробка проекту – це складний і багатоетапний процес, який включає в себе ряд послідовних етапів, на кожному з яких задіяні різні команди та фахівці з різноманітних областей (рис.3.2). Розглянемо детальніше кожен із цих етапів:

Початкова постановка задачі.

Замовник: ініціює процес, визначає основні вимоги та постановку задачі.

Бізнес-аналітик: проводить аналіз вимог замовника для подальшого розроблення технічного завдання.

Аналіз і проектування.

Бізнес-аналітик: визначає більш детальні вимоги, розробляє технічне завдання.

Дизайнер: розробляє концепції та визначає вигляд та інтерфейс продукту.

Планування.

Проект-менеджер: розробляє план проекту, визначає бюджет, ресурси та строки.

Девопс: організовує інфраструктуру для подальшого розгортання та тестування.

Рисунок 3.2 – Flow проекту

Реалізація.

Програмісти та розробники: виконують написання коду відповідно до встановлених вимог та технічного завдання.

Тестувальники: проводять тестування для виявлення помилок та недоліків.

Впровадження та підтримка.

Девопс: розгортає розроблений продукт в робочому середовищі.

Автоматизація та оптимізація: вирішує завдання автоматизації та оптимізації розробки.

Технічна підтримка: веде моніторинг роботи системи та вирішує проблеми.

Цей процес є циклічним, і на кожному етапі може виникнути необхідність внесення змін чи коригувань. Співпраця різних команд та взаємодія між фахівцями гарантують успішну і високоякісну реалізацію проекту.

Після загального огляду процесу розробки проекту, розглянемо конкретний приклад – розробку мобільного додатка. У цьому випадку, етапи проекту будуть адаптовані до особливостей роботи над мобільними застосунками.

У процесі розробки програмного забезпечення проекти створюються та об'єднуються в системі контролю версій Git. Усі зміни та новий функціонал додаються у відокремлені гілки, проте для подальшої інтеграції вони об'єднуються в спільну вітку «develop». При цьому кожен проект отримує унікальне ім'я, що формується за схемою «назва проекту + номер проекту», який вказаний в системі управління проектами, такій як Jira. Цей підхід сприяє структурованій та організованій роботі команди розробників та ефективному об'єднанню різних аспектів розробки (рис.3.3).

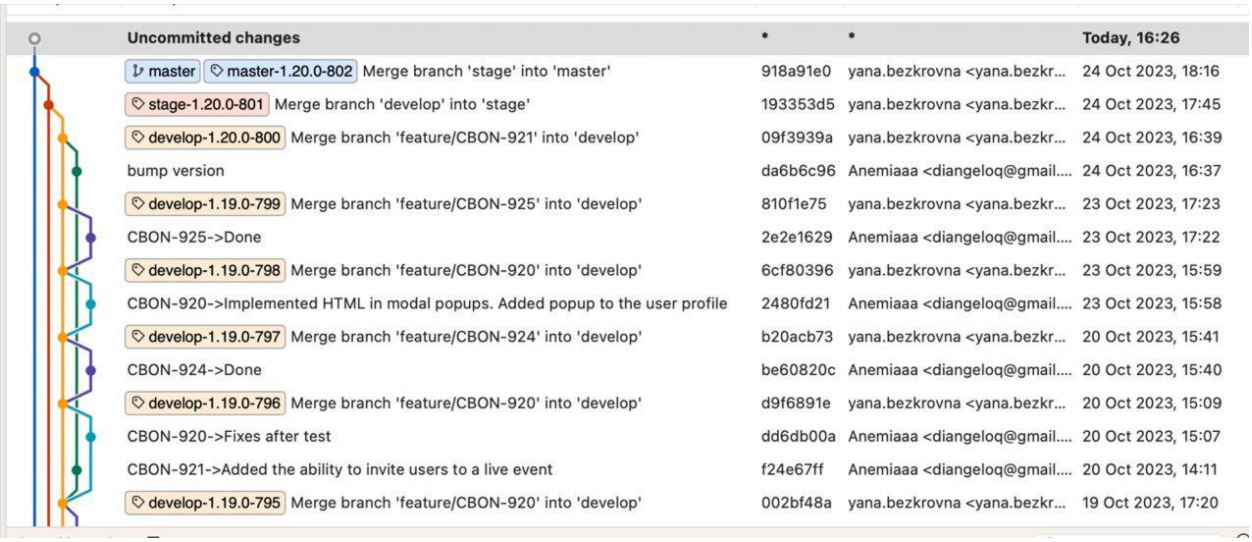

Рисунок 3.3 – Робочий простір GIT

Після завершення розробки розробники передають свій код в процес CI/CD (Continuous Integration/Continuous Deployment), який автоматизує процеси тестування та розгортання програмного продукту. Цей процес відбувається на серверах, які розташовані в Києві, забезпечуючи низьку латентність та швидкий доступ.

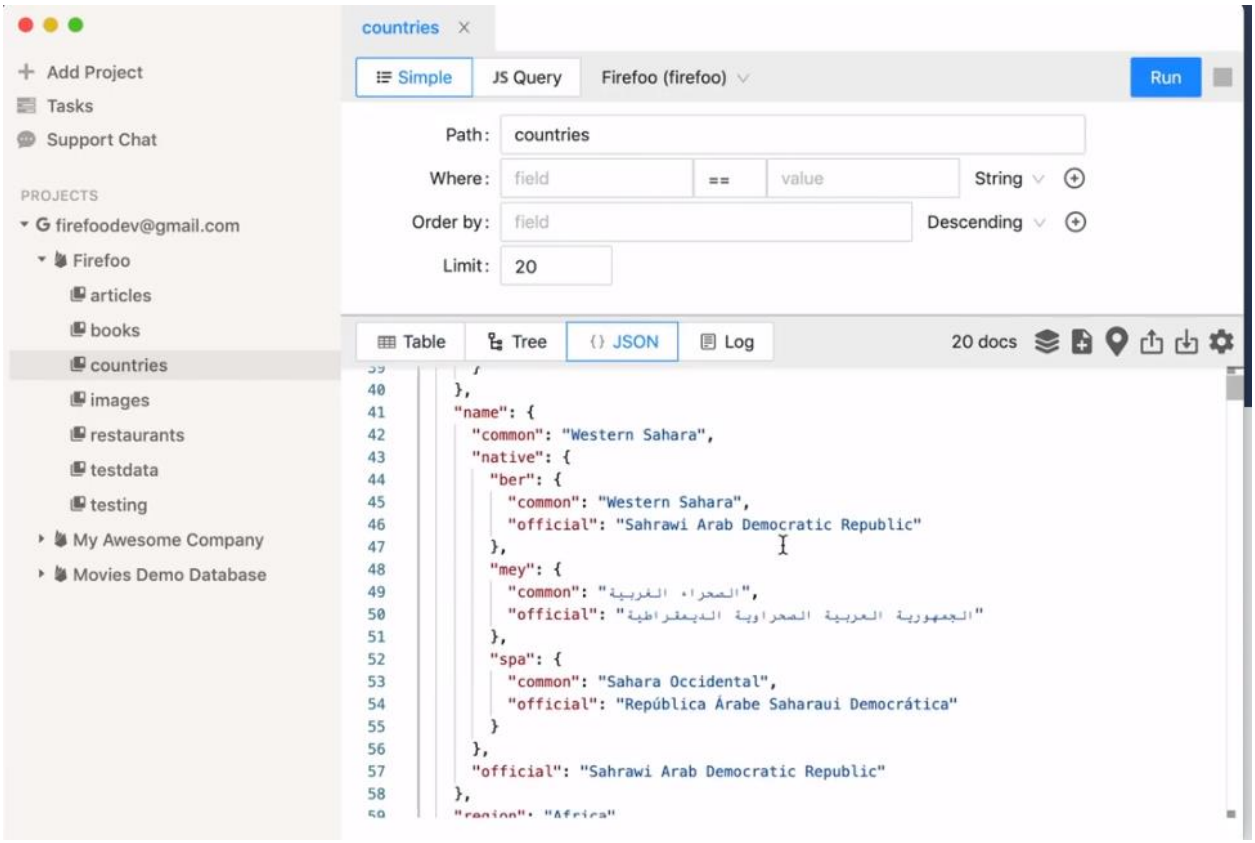

Рисунок 3.4 – Робочий простір FireBase

CI/CD включає в себе автоматичну збірку коду за допомогою хмарного середовища де розташовується код, такого як Firebase – це комплексний набір інструментів і послуг, що пропонуються як платформа Backend-as-a-Service (BaaS), що дозволяє розробникам легко створювати, запускати та розширювати мобільні та веб-додатки (рис.3.4). Під час збірки виконуються різні тести, які допомагають виявити помилки та забезпечують якість коду. Після успішної збірки програмний продукт готується до автоматичного розгортання на серверах, щоб забезпечити актуальну версію для користувачів.

Після успішного проходження тестування програма готова для об'єднання (merge) у головну гілку – master у системі контролю версій Git. Цей процес позначає завершення розробки нового функціоналу або виправлення помилок і визначає точку релізу

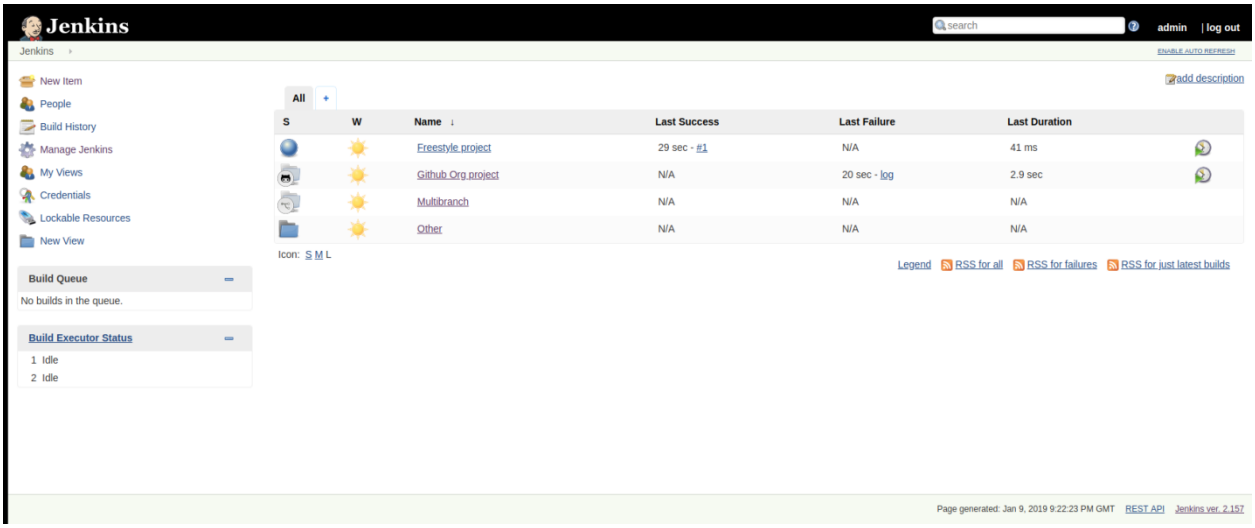

Рисунок 3.5 – Робочий простір Jenkins

Далі вступає в дію Jenkins, який автоматизує процес збірки релізу. Jenkins автоматично стягує код з гілки master, компілює його, виконує тести та готує збірку для виробництва (рис.3.5).

Після успішної збірки релізу весь результат, включаючи оновлений програмний код та зміни, готові для показу та використання замовнику. Цей підхід дозволяє забезпечити замовника завжди актуальним та стабільним програмним продуктом, а також реагувати на його вимоги швидко та ефективно.

3.2 Релізація backend частини проекту

Розглянемо код з backend-частини проекту. Backend відповідає за обробку бізнес-логіки, взаємодію з базою даних та надання необхідних даних frontend-частині та іншим клієнтам. Проаналізуємо деталі реалізації backend, щоб краще зрозуміти, як відбувається обробка запитів, взаємодія з базою даних та інші ключові аспекти функціональності серверної частини проекту.

Проаналізуємо фрагмент коду, пов'язаний з паролем, який зберігається у файлі reset_password.dart. В даному випадку, зокрема, нас цікавить функціонал введення паролю користувачем (рис.3.6).

У фрагменті коду, де відбувається скидання пароля (reset_password) використовується архітектура «Massive View Controller» (масивний контролер).

```dart
import 'package:cbone/const/color_constants.dart';
import 'package:cbone/providers/api/auth_provider.dart';
import 'package:cbone/services/routes/auth_router/auth_router.dart';
import 'package:cbone/services/routes/auth_router/auth_router_models.dart';
import 'package:cbone/ui/auth_flow/shared/auth_app_bar.dart';
import 'package:cbone/ui/auth_flow/shared/auth_button.dart';
import 'package:cbone/ui/auth_flow/shared/auth_field.dart';
import 'package:cbone/ui/auth_flow/shared/auth_titile.dart';
import 'package:cbone/ui/shared/loadable.dart';
import 'package:flutter/material.dart';
import 'package:flutter_gen/gen_l10n/app_localizations.dart';
import 'package:provider/provider.dart';

class ResetPassword extends StatefulWidget {
  const ResetPassword({super.key});

  @override
  State<ResetPassword> createState() => _ResetPasswordState();
}

class _ResetPasswordState extends State<ResetPassword> {
  final _formKey = GlobalKey<FormState>();

  String? _email;
  String? _error;

  void _processReset() {
    final provider = context.read<AuthProvider>();
    final state = _formKey.currentState;

    if (state?.validate() == true) {
      state?.save();
      provider.resetPassword(email: _email!).then(
        (value) {
```

Рисунок 3.6 – Код файлу reset_password

Даний код написаний на мові програмування Dart і використовує фреймворк Flutter для розробки мобільних додатків. Розглянемо код починаючи:

import 'package:cbone/const/color_constants.dart';: модуль (бібліотеку) з іменем color_constants.dart, який містить константи кольорів.

import 'package:cbone/providers/api/auth_provider.dart';: імпортує модуль auth_provider.dart, який, містить реалізацію провайдера для автентифікації.

import 'package:cbone/services/routes/auth_router/auth_router.dart';: імпортує модуль auth_router.dart, який має стосування до маршрутизації для автентифікації.

Import 'package:cbone/services/routes/auth_router/ auth_router_models.dart';: імпортує модуль auth_router_models.dart, який, містить моделі для автентифікаційного маршруту.

import 'package:cbone/ui/auth_flow/shared/auth_app_bar.dart';: імпортує модуль auth_app_bar.dart, який містить власний виджет для створення заголовку сторінки автентифікації.

import 'package:cbone/ui/auth_flow/shared/auth_button.dart';: імпортує модуль auth_button.dart, який, визначає власний виджет для кнопок автентифікації.

import 'package:cbone/ui/auth_flow/shared/auth_field.dart';: імпортує модуль auth_field.dart, який, містить власний виджет для текстових полів автентифікації.

import 'package:cbone/ui/auth_flow/shared/auth_titile.dart';: імпортує модуль auth_title.dart, який містить власний виджет для відображення заголовків сторінки автентифікації.

import 'package:cbone/ui/shared/loadable.dart';: імпортує модуль loadable.dart, який, визначає віджет для відображення індікатора завантаження.

import 'package:flutter/material.dart';: імпортує модуль material.dart з пакету Flutter, який містить основні компоненти інтерфейсу користувача.

import 'package:flutter_gen/gen_l10n/app_localizations.dart';: імпортує модуль для локалізації, згенерований автоматично з файлів локалізації.

import 'package:provider/provider.dart';: імпортує модуль provider.dart для роботи з пакетом Provider, який використовується для управління станом додатку.

class ResetPassword extends StatefulWidget { ... }: визначає клас ResetPassword, який розширює StatefulWidget і представляє сторінку для скидання паролю.

final _formKey = GlobalKey<FormState>();: створює глобальний ключ форми для валідації полів у формі.

String? _email;: оголошує змінну _email для збереження введеного електронного адресу.

String? _error;: оголошує змінну _error для збереження повідомлення про помилку.

void _processReset() { ... }: оголошує функцію _processReset, яка буде викликатися при спробі скидання паролю.

void _processReset() {...}: метод _processReset, який викликається при натисканні кнопки скидання паролю. У цьому методі виконується наступне:

a. Отримання екземпляра AuthProvider через context.read<AuthProvider>().

b. Отримання поточного стану форми за допомогою _formKey.currentState.

c. Перевірка валідності форми..

```
        (value) {
          final params = ConfirmCodeArguments(_email!);
          Navigator.of(context).pushNamed(
            AuthRouteKeys.confirmResetPassword,
            arguments: params,
          );
        },
      ).catchError(
        (error) {
          setState(() {
            _error = error.message;
          });
        },
      );
    }
  }

  @override
  Widget build(BuildContext context) {
    final l10n = AppLocalizations.of(context);
    final styles = Theme.of(context).textTheme;

    return Scaffold(
      appBar: AuthBar(context: context, l10n: l10n, styles: styles),
      backgroundColor: ColorConstants.surfaceBG,
      body: Loadable(
        providers: [context.watch<AuthProvider>()],
        child: Padding(
          padding: const EdgeInsets.fromLTRB(16, 24, 16, 0),
          child: SafeArea(
            child: SingleChildScrollView(
              child: Form(
                key: _formKey,
                child: Column(
                  children: [
                    AuthTitile(titile: l10n?.reset_password_title ?? ''),
                    const SizedBox(height: 10),
                    Text(
                      l10n?.reset_password_instruction ?? '',
                      style: styles.bodyText2,
                    ),
                    const SizedBox(height: 32),
                    _emailField(l10n),
                    const SizedBox(height: 32),
                    AuthButton(
                      onPresed: () => _processReset(),
                      titile: l10n?.reset_password_button ?? '',
                    ),
                    const SizedBox(height: 16),
                  ],
```

Рисунок 3.7 – Код файлу reset_password

Якщо форма валідна, тоді: збереження стану форми.

Виклик функції resetPassword з AuthProvider для скидання паролю і передача електронної пошти.

Якщо скидання паролю відбулося успішно, створення параметрів для екрану підтвердження скидання паролю (ConfirmCodeArguments), та перехід на цей екран за допомогою Navigator.of(context).pushNamed (рис.3.7).

Якщо виникла помилка під час скидання паролю, вона обробляється через catchError. Помилка (error message) зберігається в _error і викликається setState, щоб оновити відображення.

@override Widget build(BuildContext context) { ... }: це перевизначений метод build, який буде викликаний системою Flutter для побудови і відображення віджета.

final l10n = AppLocalizations.of(context);: отримання локалізації за допомогою AppLocalizations.of(context) та збереження його в змінній l10n.

final styles = Theme.of(context).textTheme;: Отримання текстових стилів з теми за допомогою Theme.of(context).textTheme та збереження їх в змінній styles.

return Scaffold(...);: повертає новий віджет Scaffold, який є основним структурним елементом для побудови сторінок в Flutter.

appBar: AuthBar(context: context, l10n: l10n, styles: styles),: додає панель додатку (appBar), використовуючи AuthBar. Передає контекст, локалізацію та стилі.

backgroundColor: ColorConstants.surfaceBG,: встановлює колір тла сторінки, використовуючи ColorConstants.surfaceBG.

body: Loadable(...),: встановлює тіло сторінки як Loadable віджет, що ймовірно відповідає за відображення завантаження.

Padding(...),: додає відступи для вмісту сторінки.

SafeArea(...),: гарантує, що вміст сторінки не буде врізатися в область безпеки (наприклад, верхню панель статусу або нижню панель навігації).

SingleChildScrollView(...),: забезпечує прокрутку вмісту, якщо вміст не поміщається на екрані.

Form(...),: використовується для створення форми, ідентифікованої за допомогою глобального ключа _formKey.

Column(...),: розміщає дочірні віджети у стовпчиковому порядку.

AuthTitle(...),: додає заголовок сторінки, інформацію про скидання паролю.

Text(...),: додає текст інструкції з скидання паролю.

_emailField(l10n),: викликає власний метод _emailField для створення поля вводу електронної пошти.

AuthButton(...),: додає кнопку для ініціації процесу скидання паролю.

const SizedBox(height: 16),: додає пропуск між елементами зазначеної висоти.

Це загальний огляд того, як побудована сторінка і які віджети використовуються для створення інтерфейсу користувача.

В наступному вивченні коду ми зосередимо увагу на файлі auth_provider.dart. Цей файл виконує важливу роль у проекті, функціонуючи як обгортка над API для Dart.

Ця обгортка дозволяє здійснювати зручні запити до сервера, взаємодіючи з ним через відповідні методи та функції. Розглянемо його структуру та основні функції для здійснення авторизації, скидання пароля та інших операцій, пов'язаних із користувачем (рис.3.8).

```
import 'package:cbone/providers/api/base_model.dart';
import 'package:cbone/services/network_service.dart';
import 'package:cbone/services/shared_preferences_wrapper.dart';
import 'package:flutter/material.dart';

class UserID {
  int id;

  UserID(this.id);
}

class AuthResult {
  late TokenModel tokenModel;
  late String notification;

  AuthResult(this.tokenModel, this.notification);
  AuthResult.fromJson(Map<String, dynamic> json) {
    tokenModel = TokenModel.fromJson(json);
    notification = json['notification']!;
  }
}

class TokenModel {
  late String token;
  late String refreshToken;

  TokenModel(this.token, this.refreshToken);
  TokenModel.fromJson(Map<String, dynamic> json) {
    token = json['token']!;
    refreshToken = json['refreshToken']!;
  }
}
```

Рисунок 3.8 – Код файлу auth_provider

Код містить декілька класів, які представляють моделі для роботи з авторизацією та токенами в рамках додатку. Розглянемо кожен клас окремо:

Клас UserID представляє об'єкт, який містить ідентифікатор користувача (id).

Клас AuthResult використовується для представлення результату авторизації. Він містить екземпляр TokenModel та рядок повідомлення (notification).

Конструктор AuthResult приймає екземпляр TokenModel та повідомлення та ініціалізує відповідні поля.

Метод fromJson призначений для конвертації об'єкта з формату JSON. Він приймає об'єкт типу Map<String, dynamic> та ініціалізує відповідні поля.

Клас TokenModel представляє об'єкт, який містить токен та оновлюваний токен.

Конструктор TokenModel ініціалізує об'єкт з переданими значеннями токену та оновлюваного токену.

Метод fromJson також слугує для конвертації об'єкта з формату JSON.

```
class AuthProvider extends BaseModel {
  NetworkService service;
  BuildContext context;

  AuthProvider(this.service, this.context);

  void set({required TokenModel token}) {
    service.addBasicAuth(token.token);
  }

  Future<bool> save({required TokenModel token}) {
    service.addBasicAuth(token.token);
    return SharedPreferencesWrapper.setToken(token);
  }

  TokenModel? token() {
    return SharedPreferencesWrapper.getToken();
  }

  Future<bool> removeToken() {
    service.addBasicAuth('');
    return SharedPreferencesWrapper.setToken(null);
  }

  Future<void> changePassword({
    required String email,
    required String password,
    required String repeatPassword,
  }) {
    startLoading();
    final request = NetworkRequest(
      type: NetworkRequestType.post,
      path: '/api/user/reset-password/change',
      data: NetworkRequestBody.formData(
        {
          'email': email,
          'passwordNew': password,
          'passwordNewRepeat': repeatPassword
        },
      ),
    );
```

Рисунок 3.9 – Код файлу auth_provider

Клас AuthProvider розширює клас BaseModel, є базовим класом для роботи зі станом додатку (рис.3.9).

Змінні service та context представляють екземпляри класів NetworkService та BuildContext відповідно. Вони використовуються для роботи з мережею та контекстом побудови інтерфейсу.

Конструктор приймає екземпляри NetworkService та BuildContext і ініціалізує відповідні змінні.

Метод set встановлює токен для основної аутентифікації в сервісі мережі.

Метод save встановлює токен для основної аутентифікації в сервісі мережі та зберігає його в SharedPreferences.

Метод token повертає збережений токен з SharedPreferences.

Метод removeToken видаляє токен для основної автентифікації в сервісі мережі та видаляє його з SharedPreferences.

Метод changePassword використовується для зміни паролю користувача. Він виконує мережевий запит, який передає інформацію про електронну адресу, новий пароль та підтвердження нового паролю на сервер. Метод також викликає методи startLoading і stopLoading, які, служать для відображення індікатора завантаження під час виконання операції (рис.3.10).

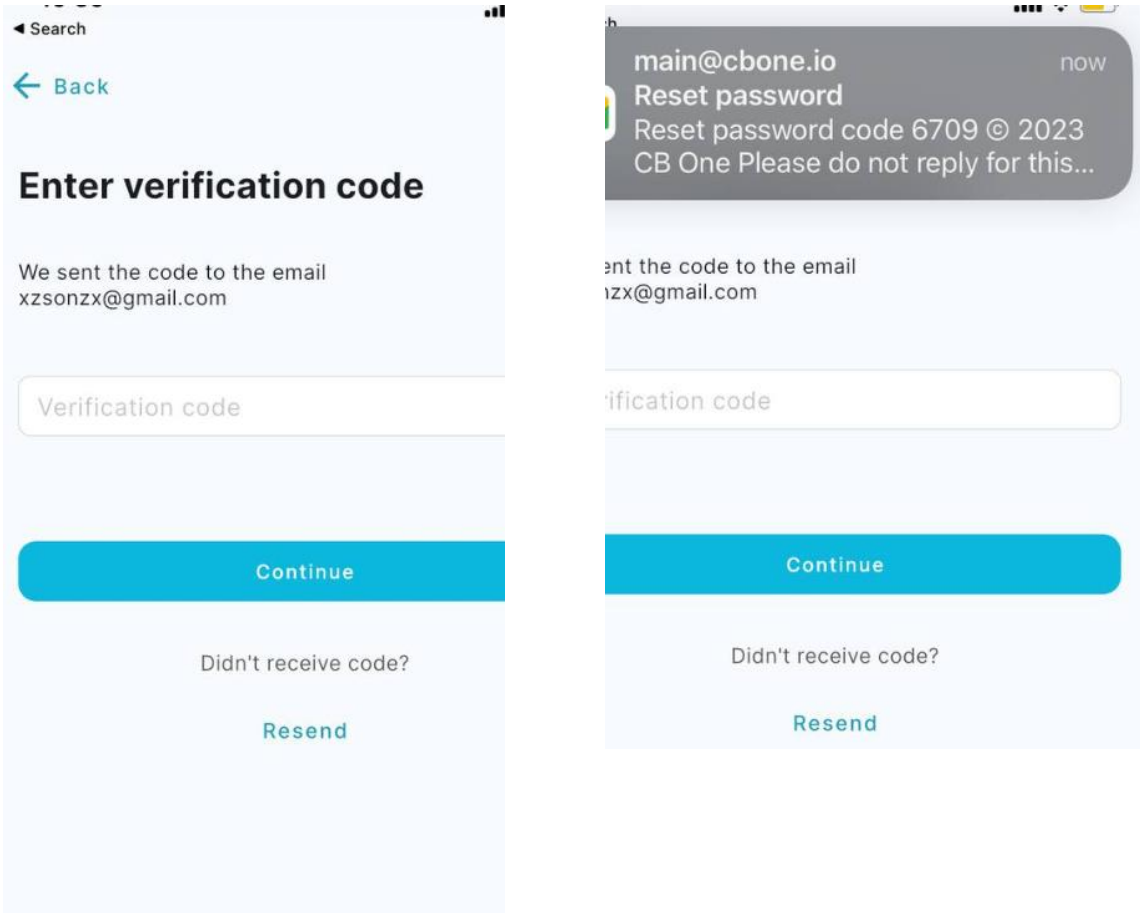

Рисунок 3.10 – Скидання паролю у додатку

Метод відповідає за ініціацію процесу скидання паролю. Він виконує HTTP POST-запит на шлях /api/user/reset-password з обов'язковим

параметром email. Метод також викликає startLoading перед виконанням запиту та stopLoading після його завершення (рис.3.11).

Метод відповідає за повторення відправлення коду підтвердження. Він виконує HTTP POST-запит на шлях /api/user/resend-code з обов'язковим параметром email. Також викликає startLoading перед виконанням запиту та stopLoading після його завершення.

Метод використовується для підтвердження скидання паролю після отримання коду підтвердження. Він виконує HTTP POST-запит на шлях /api/user/reset-password/confirm з обов'язковими параметрами email та code. Метод викликає startLoading перед виконанням запиту та stopLoading після його завершення.

```
Future<void> resetPassword({required String email}) {
  startLoading();
  final request = NetworkRequest(
    type: NetworkRequestType.post,
    path: '/api/user/reset-password',
    data: NetworkRequestBody.formData(
      {'email': email},
    ),
  );

  return service.execute(request, (data) => {}).whenComplete(stopLoading);
}

Future<void> resendCode({required String email}) {
  startLoading();
  final request = NetworkRequest(
    type: NetworkRequestType.post,
    path: '/api/user/resend-code',
    data: NetworkRequestBody.formData(
      {'email': email},
    ),
  );

  return service.execute(request, (data) => {}).whenComplete(stopLoading);
}

Future<void> confirmResetPassword({
  required String email,
  required String code,
}) {
  startLoading();
  final request = NetworkRequest(
    type: NetworkRequestType.post,
    path: '/api/user/reset-password/confirm',
    data: NetworkRequestBody.formData(
      {
        'email': email,
        'code': code,
      },
    ),
  );

  return service.execute(request, (_) => {}).whenComplete(stopLoading);
}
```

Рисунок 3.11 – Код файлу auth_provider

У даному фрагменті коду з файлу auth_router описана вся система навігації в додатку. Цей файл визначає маршрути та поведінку переходів між екранами, що є ключовим елементом в структурі будь-якого додатку. Розглянемо цей код для більшого розуміння того, як відбувається навігація в додатку (рис.3.12).

```
case AuthRouteKeys.emailAuth:
  return nativePageRoute(
    settings: setting,
    builder: (BuildContext context) {
      return MultiProvider(
        providers: [
          ChangeNotifierProvider(
            create: (context) => AuthProvider(service, context),
          ),
          ChangeNotifierProvider(
            create: (context) => SocialProvider(service, context),
          ),
          ChangeNotifierProvider(
            create: (context) => UserProvider(service, context),
          )
        ],
        child: const EmailLogn(),
      );
    },
  );

case AuthRouteKeys.confirmEmail:
  return nativePageRoute(
    settings: setting,
    builder: (BuildContext context) {
      return MultiProvider(
        providers: [
          ChangeNotifierProvider(
            create: (context) => AuthProvider(service, context),
          ),
          ChangeNotifierProvider(
            create: (context) => UserProvider(service, context),
          )
        ],
        child: ConfirmationEmail(
          params: setting.arguments as ConfirmCodeArguments,
        ),
      );
    },
  );
```

Рисунок 3.12 – Код файлу auth_router

У цьому коді ми бачимо два випадки (case) в блоку switch для різних ключів ролі в системі навігації. В кожному випадку відбувається навігація на відповідний екран, який представлений власним віджетом. Розглянемо це докладніше:

AuthRouteKeys.emailAuth.

Перевіряється, чи ключ ролі відповідає «emailAuth».

Якщо умова виконується, викликається функція nativePageRoute, яка повертає екземпляр PageRoute.

Для екрана «emailAuth» викликається MultiProvider, який обгортає декілька ChangeNotifierProvider. Кожен ChangeNotifierProvider відповідає за постачання екземплярів різних постачальників даних (наприклад, AuthProvider, SocialProvider, UserProvider).

Відображається віджет EmailLogn().

AuthRouteKeys.confirmEmail.

Перевіряється, чи ключ ролі відповідає «confirmEmail».

Якщо умова виконується, також викликається функція nativePageRoute, яка повертає екземпляр PageRoute.

Для екрана «confirmEmail» також викликається MultiProvider, який обгортає два ChangeNotifierProvider (для AuthProvider та UserProvider).

Відображається віджет ConfirmationEmail, якому передаються параметри ConfirmCodeArguments з settings.arguments.

У цих випадках nativePageRoute служить для визначення типу маршруту та створення екземпляра PageRoute. Використання MultiProvider дозволяє зручно обгортати декілька постачальників даних для передачі їх віджету, який буде відображений на відповідному екрані.

3.3 Релізація frontend частини проекту

Зараз ми розглянемо код з фронтенд-частини проекту. Фронтенд – це та частина програмного забезпечення, яка відповідає за візуальний інтерфейс користувача та взаємодію з ним. У нашому випадку ми сконцентруємося на коді, який відповідає за клієнтську сторону додатка, використовуючи мову програмування або фреймворк для створення веб-сайтів або мобільних додатків (рис.3.13).

Зараз проаналізуємл код з файлу auth_app_bar, який відповідає за створення та налаштування верхнього бару (App Bar) для сторінок авторизації в нашому додатку. У цьому файлі реалізовані функції для створення та налаштування бару, включаючи можливість автоматичного додавання кнопки «Назад» та локалізацію текстових ресурсів.

```
import 'package:flutter/material.dart';
import 'package:flutter_gen/gen_l10n/app_localizations.dart';
import 'package:cbone/const/color_constants.dart';

class AuthBar extends AppBar {
  AuthBar({
    super.key,
    required BuildContext context,
    required AppLocalizations? l10n,
    required TextTheme styles,
    List<Widget>? actions,
    bool needBackButton = true
  }) : super(
          shadowColor: ColorConstants.surfaceBG,
          elevation: 0,
          backgroundColor: ColorConstants.surfaceBG,
          leadingWidth: 110,
          actions: actions,
          automaticallyImplyLeading: false,
          leading: needBackButton ? _backButton(l10n, styles, context) : const SizedBox(),
        );

  static IconButton? _backButton(
    AppLocalizations? l10n,
    TextTheme styles,
    BuildContext context,
  ) {
    if (ModalRoute.of(context)?.isFirst == true) {
      return null;
    }

    return IconButton(
      icon: Row(
        children: [
          Icon(
            Icons.arrow_back,
            size: 27,
            color: ColorConstants.onSurfacePrimary,
          ),
          const SizedBox(width: 4),
          Text(
            l10n?.back ?? '',
            style: styles.button?.merge(
              TextStyle(
                fontSize: 14,
                color: ColorConstants.onSurfacePrimary,
              ),
        ],
      ),
      onPressed: () {
        Navigator.of(context).pop();
```

Рисунок 3.13 – Код файлу auth_app_bar

Цей код стосується створення та налаштування верхнього бару (App Bar) для сторінок авторизації в додатку. Розглянемо більш детально:

import 'package:flutter/material.dart';: цей рядок імпортує основний пакет Flutter для роботи з віджетами та інтерфейсом користувача.

import 'package:flutter_gen/gen_l10n/app_localizations.dart';: рядок імпортує клас AppLocalizations, який використовується для локалізації текстових ресурсів у додатку.

import 'package:cbone/const/color_constants.dart';: рядок імпортує константи кольорів з іншого файлу для використання в коді.

class AuthBar extends AppBar {: визначає новий клас AuthBar, який розширює клас AppBar.

AuthBar({ ... }) : super(...);: конструктор класу, який приймає кілька параметрів та передає їх до конструктора батьківського класу AppBar.

shadowColor: ColorConstants.surfaceBG,: встановлює колір тіні бару.

elevation: 0,: знімає тінь з бару, встановлюючи його висоту на 0.

backgroundColor: ColorConstants.surfaceBG,: встановлює фоновий колір бару за допомогою констант кольорів.

leadingWidth: 110,: встановлює ширину елемента "leading" (зазвичай лівий елемент) на 110.

actions: actions,: додає елементи дій (кнопки або інші віджети) до бару.

automaticallyImplyLeading: false,: вимикає автоматичне відображення лівого елемента бару.

leading: needBackButton ? _backButton(l10n, styles, context) : const SizedBox(),: встановлює лівий елемент бару як кнопку "Back", якщо needBackButton дорівнює true.

static IconButton? _backButton(...) { ... }: статичний метод, який повертає кнопку "Back" для лівого елемента бару.

if (ModalRoute.of(context)?.isFirst == true) { return null; }: перевіряє, чи поточна сторінка є першою в стеку навігації, і якщо так, то повертає null, щоб приховати кнопку "Back".

onPressed: () { Navigator.of(context).pop(); },: встановлює функціонал для кнопки "Back", який викликає Navigator.of(context).pop() для повернення на попередню сторінку.

Text(l10n?.back ?? '', style: styles.button?.merge(...),): виводить текст кнопки "Back" із врахуванням локалізації та стилів тексту.

Цей код використовується для створення та налаштування бару для сторінок авторизації, де важливою особливістю є можливість автоматично додавати кнопку «Back» на бар, якщо це необхідно, та локалізація текстових ресурсів.

Наразі ми докладно розглянемо вміст файлу color_constants.dart. Цей файл містить визначення різних кольорових констант, які використовуються в додатку (рис.3.14). Використання такого централізованого підходу дозволяє легко керувати кольорами в програмі та забезпечити консистентність їх використання (рис.3.15).

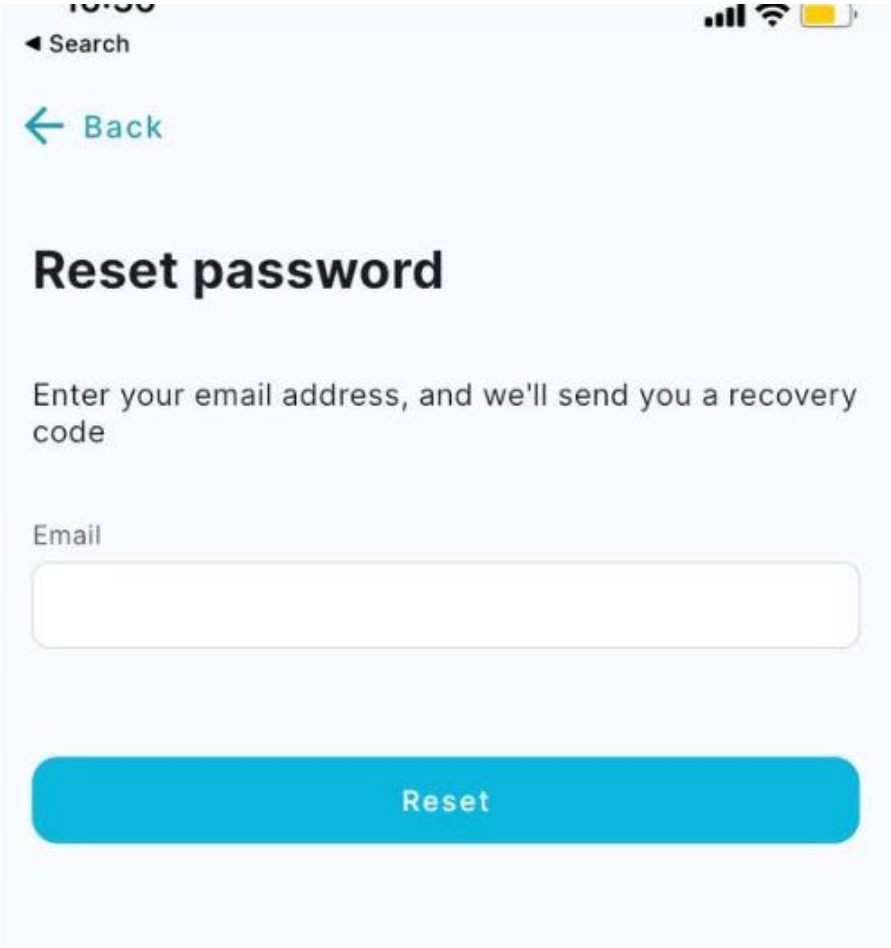

Рисунок 3.14 – Приклад кольорових констант

```
  static Color yellow = hexToColor('#FFCF24');
  static Color lightBlue = hexToColor('#40DDFF');
  static Color aquamarine = hexToColor('#C2F3FE');
  static Color green = hexToColor('#00CA2C');
  static Color backgroundControllerVideoRecord = hexToColor('#BDBDBD');

  static List<Color> liveEventGradient = [
    hexToColor('#FE5858'),
    hexToColor('#E1AFFF'),
  ];
  static List<Color> defaultEventGradient = [
    hexToColor('#27CEF3'),
    hexToColor('#BAAFFF'),
  ];

  static List<Color> pastEventGradient = [
    hexToColor('#B1B1B1').withOpacity(0.4),
    hexToColor('#838383').withOpacity(0.4),
  ];

  static List<Color> bottomGradientVideo = [
    hexToColor('#191D23').withOpacity(0),
    hexToColor('#191D23').withOpacity(0.4),
    hexToColor('#191D23').withOpacity(0.7),
    hexToColor('#191D23'),
  ];

  static List<Color> bottomGradientAudio = [
    hexToColor('#FFFFFF').withOpacity(0),
    hexToColor('#FFFFFF').withOpacity(0.4),
    hexToColor('#FFFFFF').withOpacity(0.7),
    hexToColor('#FFFFFF'),
  ];
}
```

Рисунок 3.15 – Код файлу color_constants

Код містить функцію hexToColor та абстрактний клас ColorConstants.

1. hexToColor – це функція, яка перетворює рядок з кодом кольору у відповідний об'єкт Color. Вона перевіряє, чи рядок має правильний формат (чи починається з '#', а далі йдуть шість або вісім шістнадцяткових символів) та конвертує його у числове значення.
2. ColorConstants – це абстрактний клас, який містить константи для різних кольорів у додатку. Кожне поле класу представляє конкретний колір і має відповідне значення згідно із шістнадцятковим кодом. Також визначені кольорові градієнти для різних частин додатку.

Цей підхід дозволяє централізовано управляти кольорами у всьому додатку та швидко змінювати їх в майбутньому, якщо це необхідно.

```
import 'package:cbone/const/color_constants.dart';
import 'package:cbone/providers/api/base_model.dart';
import 'package:flutter/material.dart';

class Loadable extends StatelessWidget {
  final Widget child;
  final bool forceLoad;
  final List<BaseModel> providers;
  const Loadable({
    Key? key,
    required this.child,
    required this.providers,
    this.forceLoad = false,
  }) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return Stack(
        children: (providers.any((element) => element.isLoading) || forceLoad)
            ? [
                child,
                const CBOneSpinner(),
                Container(
                  color: Colors.transparent,
                  height: double.infinity,
                  width: double.infinity,
                ),
              ]
            : [child]);
  }
}

class CBOneSpinner extends StatelessWidget {
  const CBOneSpinner({
    Key? key,
  }) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return Center(
        child: Container(
      height: 72,
      width: 72,
      decoration: BoxDecoration(
          borderRadius: BorderRadius.circular(12),
          color: ColorConstants.surfaceNeutral,
          boxShadow: [
            BoxShadow(
                color: Colors.black.withOpacity(0.08),
```

Рисунок 3.16 – Код файлу loadable

Проаналізуємо код з файлу loadable, який включає в себе віджети Loadable та CBOneSpinner. Ці віджети використовуються для створення індикації завантаження в інтерфейсі користувача.

Віджет Loadable призначений для оточення основного вмісту додатку та відображення індикатора завантаження, коли здійснюється взаємодія з сервером або іншими джерелами даних. Також, ми детально розглянемо внутрішній віджет CBOneSpinner, який представляє собою стилізований індикатор завантаження у вигляді кругового прогресу (рис.3.16).

Loadable – це віджет, який оточує дитинський віджет (child) та додає індикацію завантаження у випадку, якщо один з providers у списку BaseModel має статус завантаження (isLoading), або якщо forceLoad встановлено в true.

Stack використовується для розміщення декількох віджетів один над одним. У випадку, якщо хоча б один із BaseModel в providers має статус завантаження (isLoading) або forceLoad встановлено в true, відбувається відображення індикатора завантаження (CBOneSpinner), який накладається поверх дитячого віджету (child), і також додається прозорий контейнер, який залишається на весь екран.

CBOneSpinner – це віджет, який представляє собою індикатор завантаження у вигляді кругового прогресу.

CBOneSpinner використовує Container для стилізації зовнішнього вигляду, встановлюючи розміри, круглість країв, колір фону та ефект тіні. В середині контейнера розміщений CircularProgressIndicator, який представляє собою круговий індикатор завантаження.

Зараз розглянемо код з файлу auth_title, де реалізований віджет для відображення заголовку сторінки в автентифікаційному інтерфейсі. Віджет AuthTitile відповідає за створення блоку тексту, який відображає заголовок сторінки (рис.3.17).

```dart
import 'package:flutter/material.dart';

class AuthTitile extends StatelessWidget {
  final String? titile;

  const AuthTitile({Key? key, required this.titile}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    final styles = Theme.of(context).textTheme;
    return Container(
      padding: const EdgeInsets.only(bottom: 24),
      width: double.infinity,
      child: Text(
        titile ?? '',
        style: styles.headline5,
      ),
    );
  }
}
```

Рисунок 3.17 – Код файлу auth_titile

Код представляє собою віджет Flutter з назвою AuthTitile, який відповідає за відображення заголовка сторінки аутентифікації. Розглянемо його:

class AuthTitile extends StatelessWidget {: визначає новий клас AuthTitile, який розширює StatelessWidget.

final String? titile;: створюється змінна titile, яка представляє собою текст заголовка. Цей параметр обов'язковий і передається при створенні екземпляру віджету.

const AuthTitile({Key? key, required this.titile}) : super(key: key);: конструктор класу, де встановлюється обов'язковий параметр titile, а також викликається конструктор батьківського класу за допомогою super.

Widget build(BuildContext context) {: початок методу build, який повертає елемент інтерфейсу користувача (widget), який буде побудований.

final styles = Theme.of(context).textTheme;: отримання текстових стилів (font styles) з поточної теми додатку.

return Container(: початок конструкції Container, яка є контейнером для інших віджетів і надає можливість управління їхнім відображенням та розміщенням.

padding: const EdgeInsets.only(bottom: 24),: встановлення відступу знизу контейнера в 24 пікселі.

width: double.infinity,: контейнер розтягується на всю доступну ширину.

child: Text(: початок вкладеного віджету Text, який відповідає за відображення тексту.

titile ?? '',: виведення тексту titile. Оператор ?? використовується для перевірки на null і, у разі потреби, замінює його порожнім рядком.

style: styles.headline5,: встановлення стилів тексту, де headline5 - це один із стилів, отриманих з поточної теми.

Проаналізуємо код з файлу auth_button.dart, який містить клас AuthButton. Цей клас відповідає за створення кнопки для інтерфейсу автентифікації в мобільному додатку. Розглядатимемо кожен рядок коду для кращого розуміння його функціональності та забезпечення стильового оформлення кнопки відповідно до платформи (рис.3.18).

```dart
import 'package:cbone/const/color_constants.dart';
import 'package:flutter/material.dart';
import 'package:flutter_platform_widgets/flutter_platform_widgets.dart';

class AuthButton extends StatelessWidget {

  final String titile;
  final double? borderRadius;
  final Color? bgColor;
  final void Function()? onPresed;

  const AuthButton({
    Key? key,
    required this.titile,
    this.onPresed,
    this.bgColor,
    this.borderRadius
  })
      : super(key: key);

  @override
  Widget build(BuildContext context) {
    final styles = Theme.of(context).textTheme;

    return SizedBox(
      height: 40,
      width: double.infinity,
      child: PlatformTextButton(
        color: bgColor ??
          (onPresed != null
            ? ColorConstants.surfacePrimary
            : ColorConstants.surfacePrimary.withOpacity(0.3)),
        onPressed: onPresed,
        material: (context, platform) => MaterialTextButtonData(
          style: ButtonStyle(
            backgroundColor: MaterialStateProperty.all<Color>(
              bgColor ?? (onPresed != null
                  ? ColorConstants.surfacePrimary
                  : ColorConstants.surfacePrimary.withOpacity(0.3)),
            ),
            shape: MaterialStateProperty.all<RoundedRectangleBorder>(
              RoundedRectangleBorder(
                borderRadius: BorderRadius.circular(borderRadius ?? 12),
              ),
            ),
          ),
        ),
      ),
```

Рисунок 3.18 – Код файлу auth_button

Цей код реалізує віджет AuthButton, який представляє кнопку для інтерфейсу автентифікації.

import 'package:cbone/const/color_constants.dart';: імпорт констант кольорів для використання в оформленні.

import 'package:flutter/material.dart';: імпорт основного пакету Flutter для роботи з віджетами та іншими елементами інтерфейсу.

import 'package:flutter_platform_widgets/flutter_platform_widgets.dart';: імпорт пакету, який дозволяє використовувати платформоспецифічні віджети для Android та iOS.

class AuthButton extends StatelessWidget { ... }: оголошення класу AuthButton, який є безстанційним віджетом (віджетом, який не зберігає стан).

final String titile;: оголошення обов'язкового параметра titile – це текст, який відображатиметься на кнопці.

final double? borderRadius;: опціональний параметр borderRadius, який вказує радіус закруглення кутів кнопки.

final Color? bgColor;: опціональний параметр bgColor, який вказує колір фону кнопки.

final void Function()? onPresed;: опціональний параметр onPresed, який є функцією зворотного виклику та викликається при натисканні на кнопку.

const AuthButton({ ... }): конструктор класу AuthButton з обов'язковим та опціональними параметрами.

super(key: key);: виклик конструктора батьківського класу з ключем key.

Widget build(BuildContext context) { ... }: перевизначений метод build, який визначає віджет.

final styles = Theme.of(context).textTheme;: отримання стилів тексту з поточної теми.

return SizedBox(...);: повернення віджету SizedBox, який визначає розміри кнопки.

PlatformTextButton(...): аикористання платформоспецифічного текстового кнопкового віджету. Це дозволяє налаштовувати вигляд кнопки залежно від платформи (Android або iOS).

color: bgColor ?? ...: встановлення кольору фону кнопки на основі переданого bgColor або встановлення за замовчуванням.

onPressed: onPresed,: налаштування обробника подій для натискання кнопки.

child: Text(...): додавання тексту на кнопку з використанням стилів з поточної теми.

3.4 Завантаження готово проекту на платформи

Зараз розглянемо процес завантаження готового проекту на дві з найбільш визначальних платформ мобільних додатків – App Store та Google Play. Ця подія визначає крок до доступності та використання створеної програми для мільйонів користувачів, і у нашому дослідженні ми розглянемо ключові етапи та вимоги, які пов'язані з успішним публікуванням додатку на цих платформах.

Завантаження додатку на платформу AppStore.

1. Створили обліковий запис розробника та отримали доступ до App Store Connect. Без облікового запису розробника не зможемо подати заявку на розміщення додатка.

Вартість облікового запису становить 99 доларів на рік як для фізичних осіб, так і для організацій. Крім можливості публікації додатків, також отримуємо багато інших переваг, таких як можливість створення розширень Safari, перегляд розширених аналітичних даних про додатки, доступ до бета-версій програмного забезпечення від Apple та використання TestFlight (рис.3.19).

Тут же, після створення облікового запису розробника, надається доступ до App Store Connect, де можемо керувати своїми додатками та отримати всю інформацію про них [28].

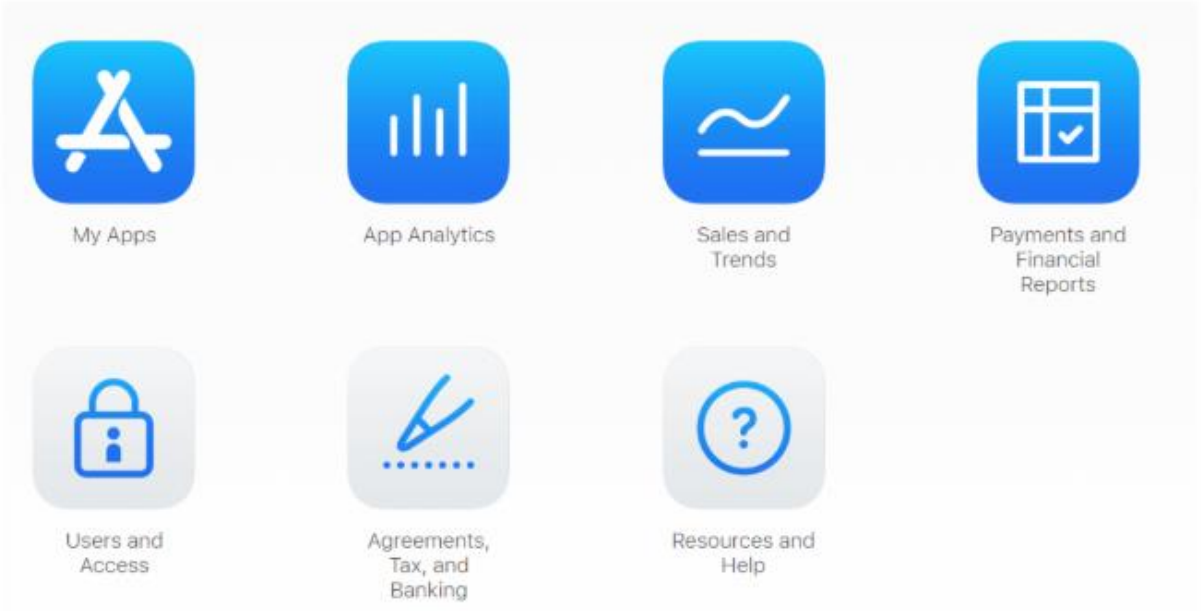

Рисунок 3.19 – Налаштування AppStore

Створення нового додатка та заповнення сторінки продукту зображено на рисунку 3.20.

Заповнили наступну інформацію на сторінці продукту:

Платформи – вказали, які платформи Apple підтримує ваш додаток.

Назва додатка – може містити до 30 символів. Це назва, яку користувачі побачать в App Store. Оскільки назва додатка перевіряється, вона повинна бути унікальною.

Основна мова – головна мова для інформації про додаток.

Ідентифікатор пакета (Bundle ID) – обрали ідентифікатори додатка, зареєстровані в обліковому записі розробника. Це значення є унікальним для кожного додатка в App Store. Ідентифікатор пакета повинен точно відповідати ідентифікатору пакета у файлі Info.plist вашого проекту Xcode (в розділі General > Identity вашої цілі).

SKU – абревіатура від Stock Keeping Unit (одиниця обліку товару). SKU – це необов'язкове поле і не видно для користувачів. Це може бути ідентифікатор, який ви використовуєте в своїй компанії або щось інше, що має для вас сенс.

Доступ користувача – чи має додаток бути доступним для всіх членів вашої організації в App Store Connect, чи лише для конкретних груп людей.

Заповнення всіх обов'язкових полів та натискання кнопки «Створити» перенаправить на сторінку продукту. Тут, в розділі «Інформація про додаток», додали інформацію для локалізації, що є дуже важливим для iOS-додатків, які націлені на різні регіони. Якщо не додати цю інформацію, буде використана основна мова, яку обрали на попередньому етапі [28].

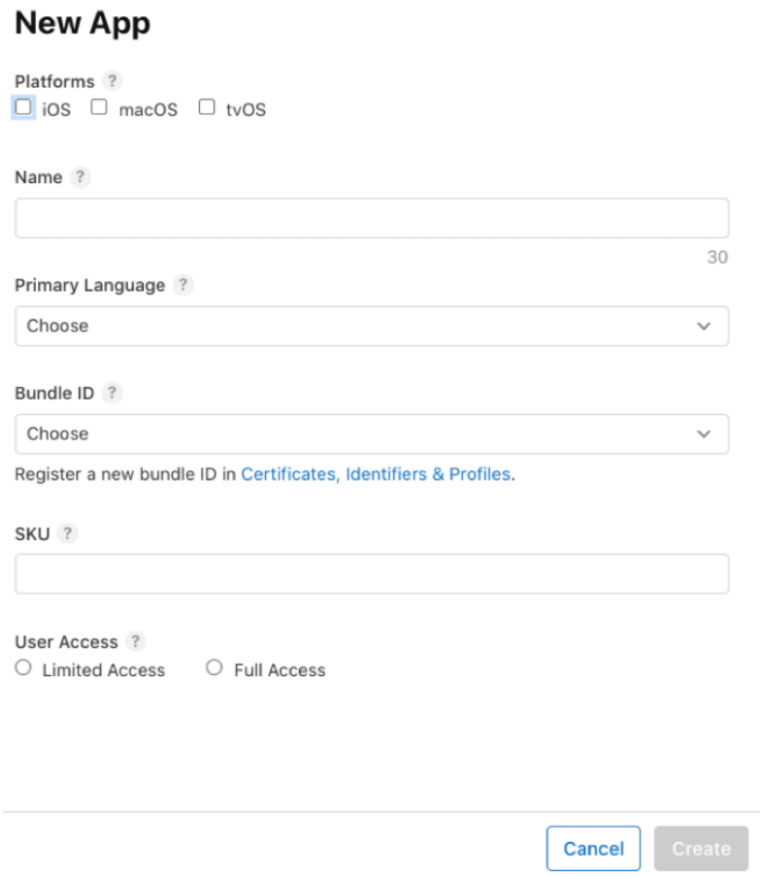

Рисунок 3.20 – Створення додатку

Якщо додаток є платним, потрібно вирішити питання ціноутворення в розділі «Ціноутворення та доступність». На відміну від маркетплейсу Google Play не можете встановити власну ціну в App Store [26]. Обрати відповідний рівень цін зі списку. Також можна керувати доступністю додатка для різних країн; це корисно при проведенні м'якого запуску продукту.

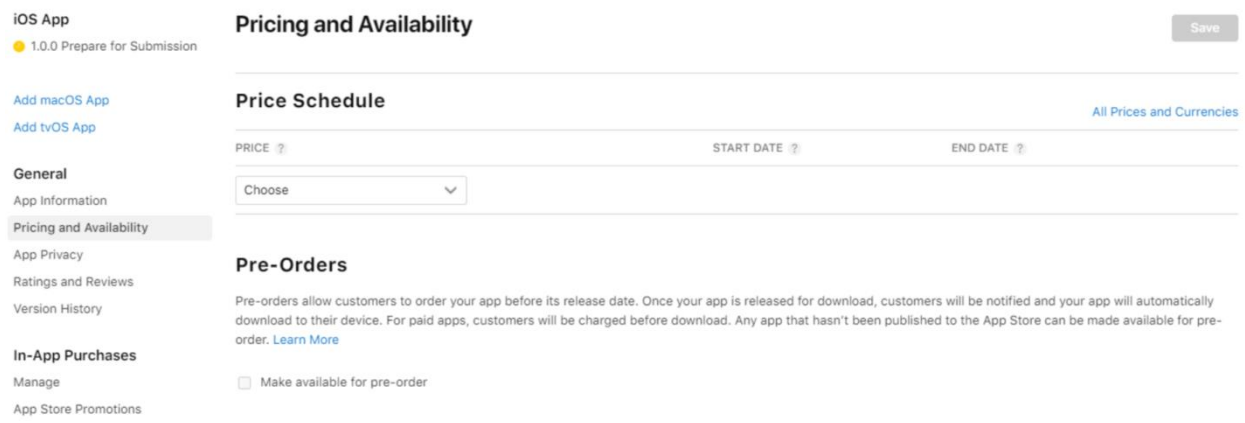

Рисунок 3.21 – Налаштування для додатку

Після реєстрації налаштували ролі, дозволи, сертифікати, ідентифікатори, профілі та інше [28]. Ознайомились з ролями в програмі розробника Apple для отримання додаткової інформації про ролі та дозволи. Іншим розробникам також потрібно отримати дозвіл на створення сертифікатів та профілів надання дозволів. Без цих дозволів розробники не зможуть вносити необхідні зміни (рис.3.22).

Отримавши необхідні дозволи, розробники можуть підписати свій додаток та вказати ідентифікатор та можливості додатка в розділі «Сертифікати, ідентифікатори та профілі».

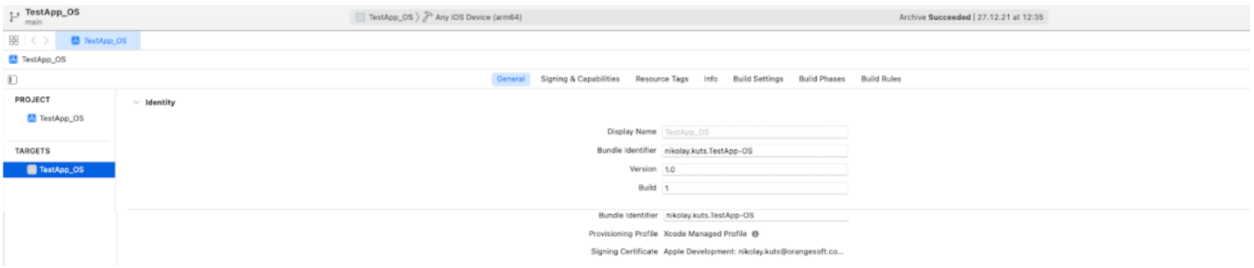

Рисунок 3.22 – Завантаження додатка

Далі розробники зібрали версію додатка та завантажити її (рис.3.20). Коли додаток завантажено до App Store Connect, потрібно відкрити розділ TestFlight та додати інженерів з якості до вашої команди Apple, щоб вони могли тестувати додаток та виявляти будь-які помилки.

Після подання додатка на розгляд, він потрапив в чергу на розгляд та сертифікацію від команди Apple. Процес перегляду може зайняти до двох тижнів. Також потрібно перейти до розділу «Встановити віковий рейтинг додатка» та завершити опитування. Apple автоматично встановить віковий рейтинг додатка на основі ваших відповідей. Подаємо додаток на розгляд. Він повинен відповідати всім вимогам Apple.

Завантаження додатку на платформу Google Play.

Для завантаження додатка в магазин Google Play важливий інструмент – це панель розробника. Консоль Google Play – це своєрідний центр управління, звідки розробники подають додатки для Android в магазин Google Play. Є єдиний збір у розмірі 25 доларів, за який розробник може відкрити обліковий запис, обладнаний функціями та можливостями управління. Після сплати цього збору можете безкоштовно подавати додатки в магазин Google Play (рис.3.23).

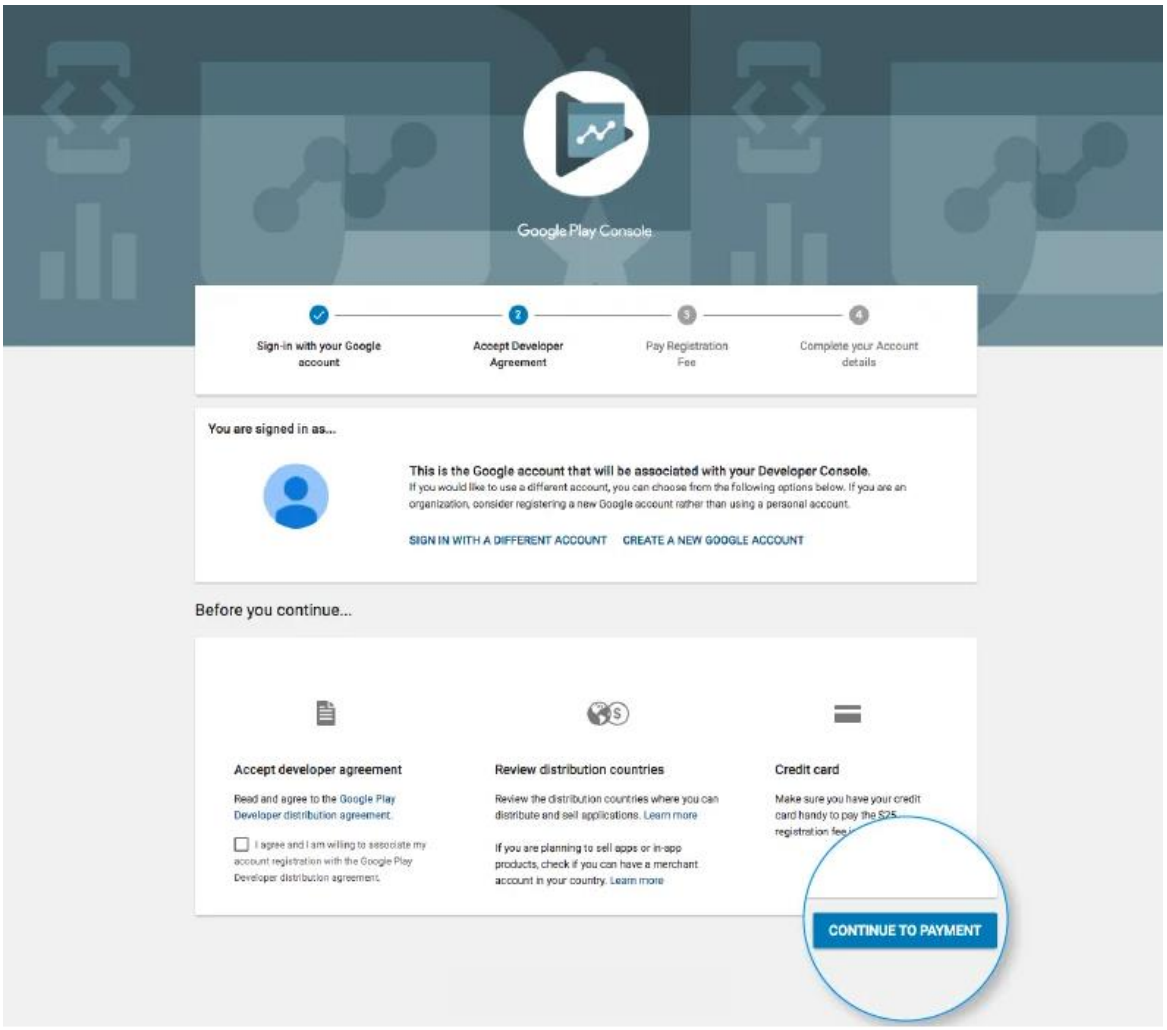

Рисунок 3.23 – Консоль розробника у Google play

Під час створення облікового запису заповнили всі обов'язкові дані, такі як ваше ім'я, країна та інші. Після подачі облікового запису на розгляд, затвердження може зайняти до 48 годин [29].

Якщо додаток, який завантажуємо в Google Play Store, підтримує внутрішні покупки, знадобиться торговий обліковий запис (рис.3.24). Щоб створити його, входимо в свій обліковий запис консолі Google Play

Торговий обліковий запис автоматично буде пов'язаний із обліковим записом консолі Google Play і дозволить управляти та переглядати продажі додатка.

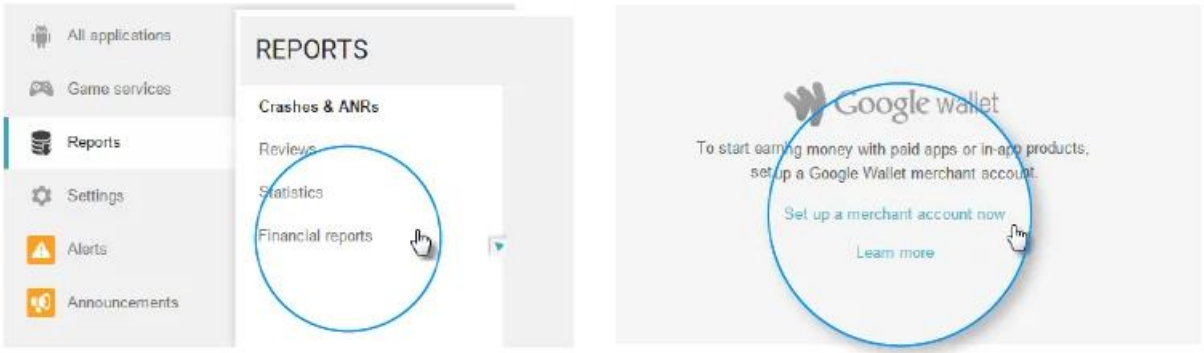

Рисунок 3.24 – Торговий обліковий запис

Після того, як увійшли до облікового запису розробника або видавця, кілька кроків, які потрібно виконати для публікації додатка в Google Play (рис.3.25):

1. У меню перейшли на вкладку «Всі додатки».
2. Обирали опцію «Створити додаток».
3. Зі списку обрали типову мову додатка.
4. Ввели назву свого додатка (її можна буде змінити пізніше).

Рисунок 3.25 – Створення додатка

Використали файли, такі як App bundle або APK, і підписали випуск додатка та завантажили їх у ваш додаток (рис.3.26).

Наступним кроком щодо того, як опублікувати Android-додатки в Google Play, це встановлення рейтингу для додатка [29]. «Без рейтингу», його можуть повністю видалити з магазину, тому важливо встановити рейтинг додатка.

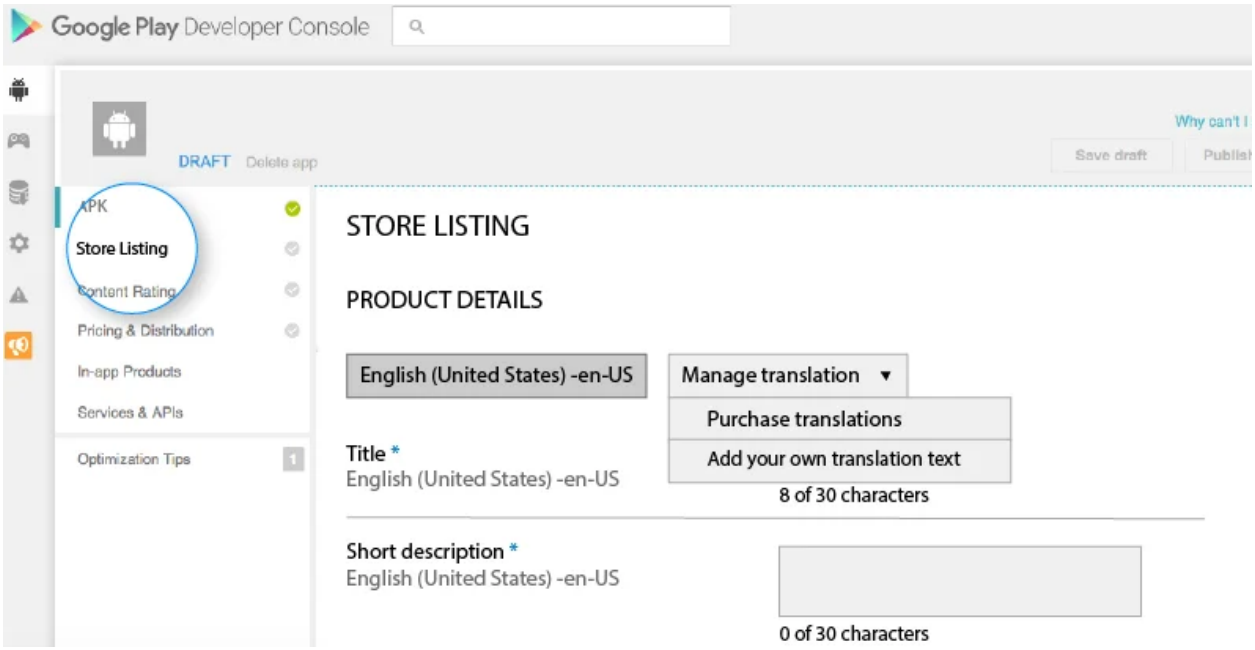

Рисунок 3.26 – Завантаження додатка

Повернулися до вкладки «Релізи додатків» і обрали «Керувати виробництвом», а потім «Редагувати реліз». Після цього клацніть «Огляд» і виберіть опцію «Розпочати випуск виробництва». Щоб завершити цей процес, обрали опцію «Підтвердити» – завантажили додаток на обліковий запис Google Play Store безкоштовно (рис.3.27).

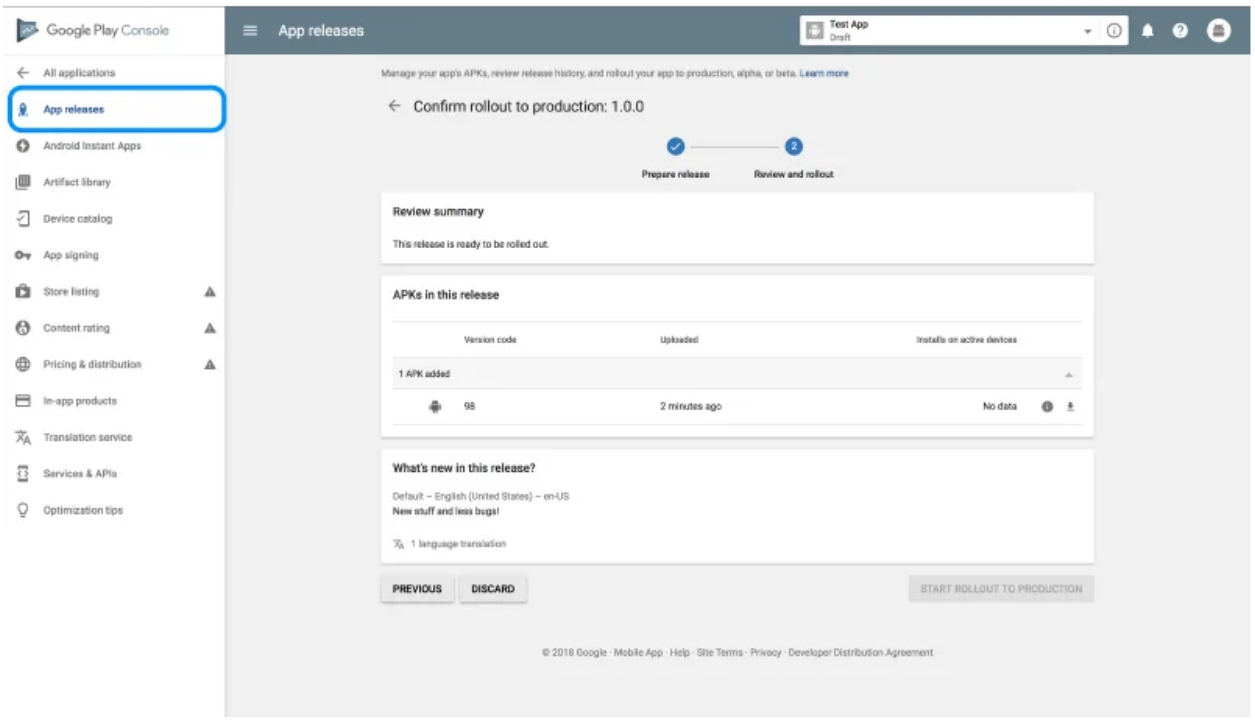

Рисунок 3.27 – Завантаження додатка

Після того, як ви завантажили додаток в Google Play Store безкоштовно, все, що залишається зробити, це просто зачекати, поки заявку на додаток затвердять [29]. Зазвичай це займає близько двох годин для перевірки

додатка але з оновленою політикою конфіденційності Google Play тепер це може зайняти години або навіть дні [29].

Висновки за розділом

Проектування програмного забезпечення включає кілька ключових етапів, які стосуються генерації, розробки та реалізації проекту. Важливо відзначити основні етапи роботи:

Генерація, розробка та реалізація проекту, описує процес створення проекту, включаючи в себе, ймовірно, аналіз вимог, визначення архітектури і вибір технологій для розробки. Реалізація backend частини проекту вказує на специфікації та реалізацію серверної частини програмного забезпечення. Це включає в себе обробку бізнес-логіки, взаємодію з базою даних та забезпечення серверної частини додатка.

Реалізація frontend частини проекту фокусується на розробці інтерфейсу користувача та його взаємодії з клієнтською частиною додатка. Це включає роботу з графічним інтерфейсом, взаємодію з backend та забезпечення зручного інтерфейсу для користувачів.

Завантаження готового проекту на платформи"описує процес завантаження готового програмного продукту на платформи. Це може включати в себе розгортання на веб-серверах чи платформах для мобільних додатків, таких як App Store та Google Play.

В цілому, цей розділ охоплює ключові етапи від ідеї до реалізації проекту, зосереджуючись на роботі як з backend, так і з frontend частинами програмного забезпечення.

.

ВИСНОВКИ

У кваліфікаційній роботі методи розробки програмного забезпечення на базі оптимальної архітектури.

Поставлені та виконані цілі кваліфікаційної роботи:

- описано структуру компанії IDAP;
- зроблено комплексний аналіз розділів компанії;
- дослідили проектування програмного забезпечення.

У результаті проведений огляд структури IDAP, визначені основні компоненти та їх взаємодія, що створює фундамент для подальшого детального розгляду. Ретельно описано frontend, визначено ключові елементи інтерфейсу. Backend розділ детально проаналізовано, зосереджуючись на обробці бізнес-логіки та взаємодії з базою даних. QA розділ визначає стратегії та методи тестування для забезпечення високої якості продукту. DevOps розділ розглядає інтеграцію розробки та операцій, зазначаючи інструменти для автоматизації та прискорення процесу впровадження. Комплексний аналіз розділів надає повний обзор структури та взаємодії компонентів IDAP, визначаючи ключові фактори, що впливають на якість та ефективність системи.

Проектування програмного забезпечення включає етапи генерації, розробки та реалізації проекту. Важливі кроки включають аналіз вимог, вибір архітектури і технологій. Backend реалізує серверну частину, обробляючи бізнес-логіку та взаємодію з базою даних. Frontend фокусується на інтерфейсі користувача. Описали процес завантаження готового проекту на платформи такі як App Store та Google Play. У результаті виконання роботи це надає огляд ключових етапів від ідеї до реалізації, з фокусом на роботі із backend і frontend частинами програмного забезпечення.

.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1) Аутсорсинг URL: https://sunone.com.ua/ru/articles-ru/chto-takoe-autsorsingovye-uslugi-prostymi-slovami// (дата звернення: 05.11.2023)
2) Аутсафінг URL: https://hurma.work/blog/autstafing-shho-cze-za-posluga-ta-chym-vidriznyayetsya-vid-autsorsyngu/ (дата звернення: 05.11.2023)
3) Продакт URL: https://medium.com/@productmanagementcommunity/ (дата звернення: 05.11.2023)
4) Продукти IDAP URL: https://idapgroup.com/ (дата звернення: 05.11.2023)
5) Спіральна динаміка URL: https://www.maxzosim.com/spiralna-dynamika-na-praktytsi-model-rozvytku/ (дата звернення: 05.11.2023)
6) Компанія IDAP URL: https://idapgroup.com/ (дата звернення: 05.11.2023)
7) Фронтенд URL: https://www.uplers.com/blog/top-skills-of-frontend-developers/ (дата звернення: 06.11.2023)
8) Фронтенд URL: https://www.developeronrent.com/hire-frontend-developers.html (дата звернення: 06.11.2023)
9) HTML URL: https://www.hostinger.com/tutorials/what-is-html (дата звернення: 06.11.2023)
10) CSS URL: https://developer.mozilla.org/en-US/docs/Learn/CSS/First_steps/What_is_CSS (дата звернення: 06.11.2023)
11) Kotlin URL: https://www.techtarget.com/whatis/definition/Kotlin (дата звернення: 06.11.2023)
12) Мова програмування C URL: https://builtin.com/software-engineering-perspectives/objective-c (дата звернення: 06.11.2023)
13) Мова програмування Swift URL: https://stripe.com/resources/more/what-is-swift (дата звернення: 06.11.2023)
14) Програмний каркас Flutter URL: https://www.adservio.fr/post/what-is-flutter-and-what-are-its-advantages (дата звернення: 06.11.2023)
15) Мова програмування Dart URL: https://dart.dev/ (дата звернення: 06.11.2023)
16) Як працює сервер URL: https://www.linkedin.com/pulse/how-do-web-servers-work-priyanka-kumari/ (дата звернення: 07.11.2023)
17) Як працює база даних URL: https://ducmanhphan.github.io/2020-01-19-How-relational-database-works/ (дата звернення: 07.11.2023)
18) Як працює SQL URL: https://kinsta.com/knowledgebase/what-is-mysql/ (дата звернення: 07.11.2023)
19) Як працює MongoDB URL:https://www.knowi.com/blog/mongodb-what-is-mongodb/ (дата звернення: 07.11.2023)
20) Як працює API URL: https://medium.com/@ama.thanu/what-is-an-api-how-does-it-work-f4ea552d741f (дата звернення: 07.11.2023)

21) Node js URL: https://www.freecodecamp.org/news/what-exactly-is-node-guide-for-beginners/ (дата звернення: 07.11.2023)
22) PHP URL: https://www.devopsschool.com/blog/what-is-php-and-how-php-works/ (дата звернення: 07.11.2023)
23) Що таке QA URL: https://testsigma.com/guides/qa-process/ (дата звернення: 07.11.2023)
24) Інструменти для тестування URL: https://theqalead.com/topics/what-are-test-management-tools/ / (дата звернення: 07.11.2023)
25) Мануал Apache URL: https://jmeter.apache.org/usermanual/build-web-test-plan.html (дата звернення: 07.11.2023)
26) Appilots URL: https://applitools.com/tutorials/ (дата звернення: 07.11.2023)
27) Devops URL: https://www.ansible.com/overview/how-ansible-works (дата звернення: 07.11.2023)
28) Як завантажити до AppStore URL: https://orangesoft.co/blog/how-to-submit-an-ios-app-to-the-app-store (дата звернення: 07.11.2023)
29) Як завантажити додаток до Google Play URL: https://appinventiv.com/blog/how-to-submit-app-to-google-play-store/ (дата звернення: 07.11.2023)