Міністерство освіти і науки України

Харківський національний університет радіоелектроніки

Факультет комп’ютерної інженерії та управління
(повна назва)

Кафедра електронних обчислювальних машин
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА

Пояснювальна записка

Рівень вищої освіти другий (магістерський)

Архітектурна модель Full-Stack проекту

(тема)

Виконав:

студент II курсу, групи СПм-22-4

Комарчев А.В.
(прізвище, ініціали)

Спеціальність

123 «Комп’ютерна інженерія»
(код і повна назва спеціальності)

Тип програми освітньо-наукова
(освітньо-професійна або освітньо-наукова)

Освітня програма

Системне програмування
(повна назва освітньої програми)

Керівник: доц. Філімончук Т.В.
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри ЕОМ _____ Коваленко А.А.
(підпис) (прізвище, ініціали)

2024 р.

Харківський національний університет радіоелектроніки

Факультет комп’ютерної інженерії та управління

Кафедра електронних обчислювальних машин

Рівень вищої освіти другий (магістерський)

Спеціальність 123 «Комп’ютерна інженерія»
(код і повна назва)

Тип програми освітньо-наукова
(освітньо-професійна або освітньо-наукова)

Освітня програма Системне програмування
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
“____” _____ 20__ р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

студенту Комаричеву Андрію Валерійовичу
(прізвище, ім’я, по батькові)

1. Тема роботи Архітектурна модель Full-Stack проекту

затверджена наказом по університету від “ 01 ” квітня 2024 р. № 257 Ст

2. Термін подання студентом роботи до екзаменаційної комісії 15 червня 2024 р.

3. Вхідні дані до роботи 1) фреймворк Flutter; 2) фреймворк FastAPI; 3) база даних PostgreSQL; 4) застосунок Docker.

4. Перелік питань, що потрібно опрацювати у роботі

1) переваги та недоліки Full-Stack застосунків;
2) застосування технологій Flutter, FastAPI, PostgreSQL та Docker для розробки проєктів;
3) аналіз існуючої моделі розробки проєктів;
4) створення покращеної моделі розробки проектів;
5) висновки.

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (слайдів) Слайд-презентація – 12 слайдів

6. Консультанти розділів роботи (заповнюється за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Аналіз проблеми та огляд існуючих рішень	02.04.24-08.04.24	
2	Вибір та обгрунтування методики дослідження	09.04.24-16.04.24	
3	Вибір інструментальних засобів	17.04.24-22.04.24	
4	Розробка моделі Full-Stack проєкту	23.04.24-06.05.24	
5	Проведення експериментів	07.05.24-23.05.24	
6	Оформлення матеріалів кваліфікаційної роботи	24.05.24-03.06.24	
7	Подання кваліфікаційної роботи керівникові та	04.06.24-07.06.24	
	її попередній захист		
8	Подання кваліфікаційної роботи на рецензу-	08.06.24-12.06.24	
	вання		

Дата видачі завдання 01 квітня 2024 р.

Студент _____
(підпис)

Керівник роботи _____ доц. Філімончук Т.В.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка кваліфікаційної роботи: 81 с., 12 рис., 1 дод., 24 джерела.

FULL-STACK ЗАСТОСУНОК, МОДУЛЬНА АРХІТЕКТУРА, МОДЕЛЬ ПРОЄКТУ, FLUTTER, FASTAPI, DOCKER, POSTGRES

Метою кваліфікаційної роботи є дослідження наявної моделі створення Full-Stack проєкту та створення більш гнучкої та маштабованої моделі. У ході виконання кваліфікаційної роботи було проведено аналіз наявних моделей створення Full-Stack застосунків та технологій, які доступні для використання, запропоновано більш новітню та гнучку модель, проаналізовано можливості її використання в застосунках різної складності.

Об'єкт дослідження: архітектурна модель Full-Stack проєкту.

ABSTRACT

Master’s thesis: 81 pages, 12 figures, 1 appendice, 24 sources.

FULL-STACK APP, MODULAR ARCHITECTURE, PROJECT MODEL, FLUTTER, FASTAPI, DOCKER, POSTGRES

The purpose of the qualification work is to study the existing model for creating a Full-Stack project and create a more flexible and scalable model.

In the course of the qualification work, an analysis of existing models for creating Full-Stack applications and technologies that are available for use was conducted, a newer and more flexible model was proposed, and the possibilities of its use in applications of varying complexity were analyzed.

Object of study: the architectural model of a Full-Stack project.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

СУБД – система управління базами даних

ACID – атомарність, послідовність, ізольованість, стійкість (англ., Atomicity, Consistency, Isolation, Durability)

AD – автоматизація розгортання (англ., Automated Deployment)

API – інтерфейс програмування застосунків (англ., Application Programming interface)

BL – бізнес-логіка (англ., Business Logic)

CF – клієнтська частина (англ., Client Frontend)

CLI – інтерфейс командного рядка (англ., Command Line Interface)

CS – хмарні сервіси (англ., Cloud Services)

DB – база даних (англ., Database)

DO – оркестрація даних (англ., Data Orchestration)

DRY – не повторюй себе (англ., Don’t Repeat Yourself)

IL – логіка взаємодії (англ., Interaction Logic)

JWT – Json Веб Токен (англ., JSON Web Tokens)

KB – кешування (англ., Caching)

MA – модуль моніторингу та аналітики (англ., Monitoring and Analytics)

ML – машинне навчання (англ., Machine Learning)

MLog – модуль моніторингу та журналювання подій (англ., Monitoring and Logging)

MP – інтеграція мобільних платежів (англ., Mobile Payments)

MS – мікросервісна архітектура (англ., Microservice Architecture)

SM – модуль безпеки (англ., Security Module)

TM – модуль тестування (англ., Testing Module)

UI – інтерфейс користувача (англ., User Interface)

ВСТУП

Архітектурна модель Full-Stack проєкту, особливо коли йдеться про мобільний проєкт, що використовує Flutter для розробки клієнтської частини, Python FastAPI як бекенд-фреймворк, PostgreSQL як систему управління базами даних та Docker для контейнеризації та оркестрації, є важливою темою для дослідження у сфері програмної інженерії. Ця інтеграція технологій дозволяє створити ефективний, масштабований та гнучкий Full-Stack проєкт.

Архітектурна модель такого проєкту передбачає розробку з використанням сучасних підходів та інструментів, що забезпечують швидкість та якість кінцевого продукту. Використання Docker допомагає у стандартизації розгортання додатків та забезпечує їх легке та швидке розгортання в різноманітних середовищах.

Розглянемо задачи, які слід вирішити у кваліфікаційній роботі:

- детально вивчити та описати ключові архітектурні шаблони, які використовуються у сучасних Full-Stack проєктах;
- розглянути структурні особливості та підходи до побудови Full-Stack архітектури, включаючи розділення відповідальностей між клієнтською та серверною частинами;
- вивчити та описати основні особливості та переваги використання Flutter для розробки кросплатформних мобільних додатків;
- оглянути ключові характеристики та переваги Python FastAPI як інструменту для створення швидких та ефективних бекенд-сервісів;
- вивчити особливості та переваги використання PostgreSQL як надійної та гнучкої СУБД для Full-Stack розробки;
- проаналізувати практики роботи з даними, безпеку та можливості оптимізації;
- описати переваги використання Docker у контексті розробки та розгортання Full-Stack проєктів;

- дослідити методики контейнеризації додатків та їх інтеграції з іншими сервісами та інфраструктурою;

- визначити кращі практики та патерни інтеграції між фронтенд, бекенд та базою даних;

- розробити стратегію ефективної взаємодії між різними частинами системи;

- оцінити можливості масштабування додатку та його здатність витримувати збільшене навантаження, а також розглянути потенціал для розширення функціональності та інтеграції з іншими системами та сервісами;

- розглянути роботу серверної частини;

- розглянути доступні технології для створення мобільних Full-Stack застосунків.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

Сьогодні архітектурна модель Full-Stack проєкту, особливо у мобільному секторі, стає все більш значущою та впливовою у процесі розробки програмного забезпечення. Full-Stack розробку можна розглядати як комплексну систему, яка охоплює всі аспекти додатку – від інтерфейсу користувача до баз даних. Це забезпечує повний цикл розробки, дозволяючи створювати більш інтегровані та ефективні додатки, які краще відповідають потребам користувачів.

Ідея Full-Stack розробки виникла з необхідності більш швидкої та гнучкої розробки програмних продуктів, здатних швидко адаптуватися до змінних вимог ринку та користувачів. У цьому контексті Full-Stack розробник – це професіонал, який володіє різноманітними технологіями та платформами: від фронтенду до бекенду. Важливість Full-Stack розробників зростає разом із потребою у комплексних рішеннях, що охоплюють усі етапи створення програмного забезпечення.

Сучасна архітектурна модель Full-Stack проєкту в сегменті мобільної розробки використовує інноваційні технології, такі як Flutter для розробки мобільних додатків, Python FastAPI для створення серверних додатків, Postgres як систему управління базами даних та Docker для контейнеризації та розгортання програм. Цей підхід визначає новий етап у розробці програмного забезпечення, де інтеграція та масштабування відіграють ключову роль.

Розвиток Full-Stack розробки відповідає глобальним тенденціям у IT-індустрії, де велика увага приділяється автоматизації, оптимізації процесів та підвищенню продуктивності. Подібно до IoT, де об'єкти взаємодіють один з одним для створення кращих рішень для кінцевих користувачів, Full-Stack проєкти інтегрують різноманітні технології та підходи для створення цілісних та ефективних додатків.

Full-Stack розробка має потенціал революціонізувати сферу розробки

програмного забезпечення, пропонуючи комплексні рішення, що охоплюють усі аспекти програмного продукту. Це сприяє створенню нових стандартів у розробці мобільних та веб-додатків, де інтеграція, безпека, швидкість та досвід користувача стають ключовими факторами успіху.

Прогнозується, що кількість проєктів, які вимагають розробників з Full-Stack компетенціями, буде лише зростати. Це обумовлено постійно зростаючими вимогами до функціональності програмних продуктів та необхідністю їхньої швидкої адаптації до змінних потреб користувачів та ринку.

Інноваційні платформи та фреймворки, такі як Flutter та FastAPI, сприяють популяризації Full-Stack розробки, оскільки дозволяють розробникам ефективніше створювати та тестувати додатки. Особливо це стосується мобільних додатків, де Flutter дозволяє швидко створювати високоякісні кросплатформні додатки, а FastAPI – це сучасний, високопродуктивний веб-фреймворк для створення API з Python 3.7+, який спрощує розробку бекенду.

Використання Postgres як системи управління базами даних забезпечує надійне та ефективне зберігання даних. Docker, у свою чергу, відіграє важливу роль у контейнеризації та розгортанні додатків, роблячи процес розробки, тестування та впровадження більш гнучким та ефективним.

Таким чином, Full-Stack розробка стає невід'ємною частиною сучасної IT-індустрії, оскільки вона забезпечує створення гнучких, адаптивних та ефективних додатків. Інтеграція різних технологій та платформ у межах одного проєкту дозволяє досягти нових рівнів продуктивності та інтерактивності, що, у свою чергу, веде до створення кращих та більш інноваційних рішень для кінцевих користувачів.

1.1 Огляд актуальності мобільної розробки

Мобільні телефони стали невід'ємною частиною життя людей у всьому світі, надаючи можливість підтримувати зв’язок із родиною, колегами та

отримувати доступ до електронної пошти. Вони значно розширили свої функції: сучасні пристрої не лише забезпечують дзвінки, але й дозволяють зберігати дані, робити фотографії, а також використовувати їх як рації.

Коли мобільні телефони тільки з'явилися, вони були великими, дорогими та вимагали додаткових блоків для транспортування. Якість зв’язку залишала бажати кращого, і працювали вони лише в місцях з сильним сигналом. Однак із розвитком технологій використання мобільних телефонів стало набагато простішим, а якість зв’язку значно покращилася завдяки впровадженню супутникових та бездротових мереж.

Смартфони, що мають розширені обчислювальні можливості, з’явилися на ринку наприкінці 90-х років. Проте справжню популярність вони здобули з виходом iPhone від Apple у 2007 році, який запропонував зручний сенсорний інтерфейс та віртуальну клавіатуру. Перший Android-смартфон був представлений у 2008 році, і відтоді індустрія смартфонів неухильно розвивається і зростає [1].

У 2023 році прогнозується поставка близько 1,48 мільярда смартфонів по всьому світу. На кінець 2020 року майже половина населення світу володіла смартфонами, і оскільки багато людей мають більше одного пристрою, кількість підписок на смартфони перевищує кількість користувачів (рисунок 1.1). У 2021 році було приблизно 6,4 мільярда підписок на смартфони, а до 2026 року очікується їхнє зростання до 7,5 мільярдів. Лідерами ринку залишаються Apple і Samsung [2].

Операційні системи відіграють ключову роль у розробці додатків для Android та iOS. Вони надають користувачам всі можливості пристроїв, будь то традиційні програми, ігри, гібридні або веб-додатки. Від цих додатків залежить успіх пристроїв, виробників, платформ та всієї екосистеми загалом.

Сучасні смартфони схожі на кишенькові комп’ютери, що дозволяють не лише дзвонити та надсилати повідомлення, але й надсилати електронні листи, грати в ігри, читати новини та здійснювати відеодзвінки. Операційні системи або мобільні ОС, керують ресурсами та пам’яттю пристроїв під час

багатозадачності. Розвиток мобільних ОС приніс на смартфони розширені функції, які раніше були доступні лише на комп’ютерах. Вони є платформою для створення великої кількості додатків, кожен з яких має свою мету. Наприклад, програма погоди надає актуальні метеорологічні дані, новинні додатки відображають заголовки, а ігри дозволяють розважатися.

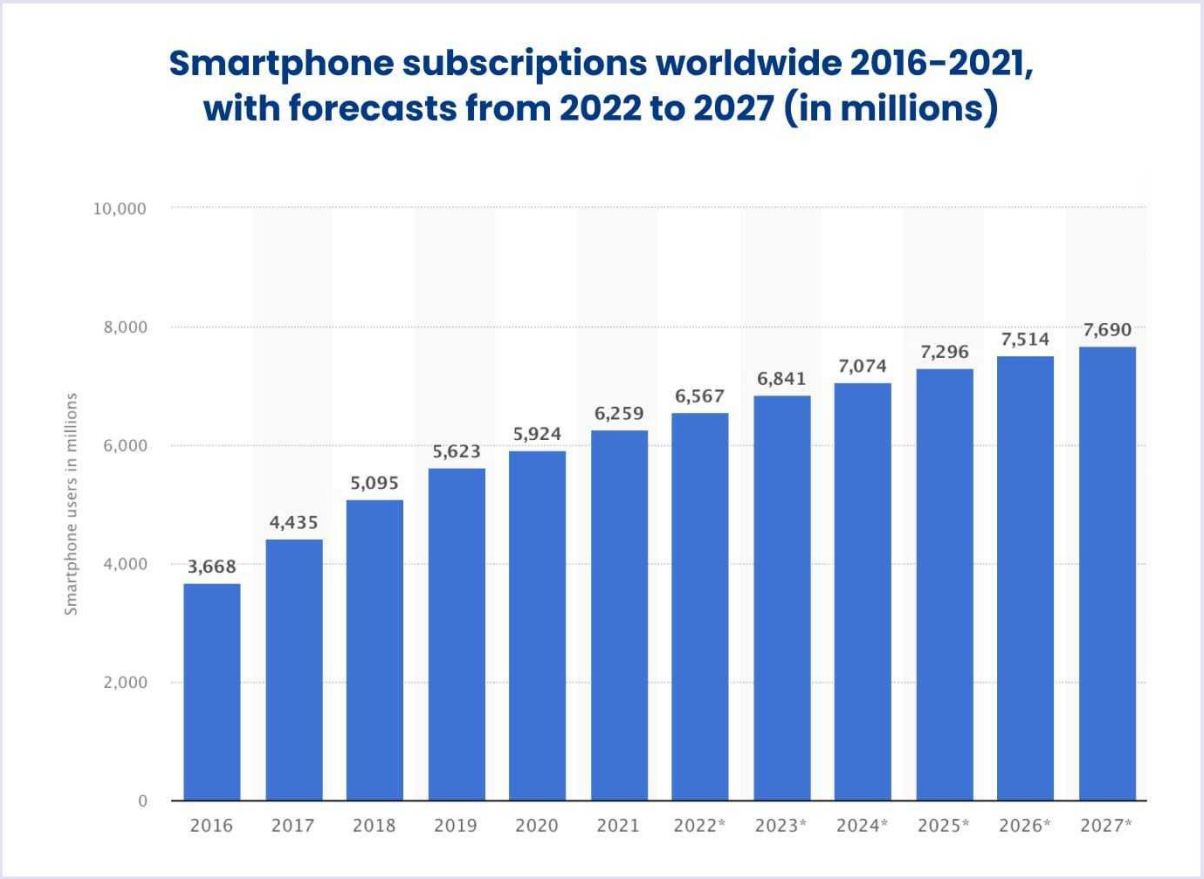

Рисунок 1.1 – Динаміка кількості підписок на смартфони 2016-2027рр.

Серед прикладів мобільних операційних систем можна назвати Apple iOS, Google Android та Windows Phone від Microsoft.

Мобільні додатки кардинально змінили ведення бізнесу. На 2020 рік близько 3,5 мільярда людей у світі володіли смартфонами [2]. Мобільні додатки стали найефективнішим способом доставлення інформації про продукт та підтримки лояльності клієнтів, подібно до того, як веб-сайти мали ключове значення десяток років тому.

Розробка мобільних додатків може здаватися складною для деяких компаній, але переваги виправдовують інвестиції. Це дозволяє постійно бути

на зв’язку з клієнтами. У середньому в США людина проводить близько 2,7 години на день зі смартфоном, і більшість цього часу присвячено додаткам [3].

Таким чином, мобільні додатки допомагають залучати клієнтів, підтримувати їхню лояльність та випереджати конкурентів, роблячи бізнес успішним у цифрову епоху.

1.2 Концепції та основні елементи Full-Stack розробки

Повний стек (Full-stack) означає створення комп'ютерного програмного забезпечення або додатків від початку до кінця. Розробка якісного мобільного застосунку розпочинається з визначення його цілей та напрямків розвитку для досягнення бажаного результату. Кожен застосунок проходить через кілька ключових етапів, починаючи з первинної ідеї. Після цього відбувається підготовча робота та аналітика, під час якої формується концепція та визначаються способи створення застосунку. На стадії аналізу початкова ідея трансформується в детальний план роботи, обирається набір технологій та методика розробки програмного забезпечення, які стануть основою для роботи команди розробників.

Модель мобільного додатку включає в себе структурований план або опис, який визначає архітектурні, функціональні та технічні елементи, необхідні для розробки мобільного застосунку. Вона може базуватися на різних методологіях, залежно від специфіки та цілей конкретного застосунку. Методологія розробки програмного забезпечення включає в себе набір методів, застосованих на різних етапах життєвого циклу застосунку, що забезпечують ефективність процесів розробки згідно з єдиною методологією. В IT-галузі існують різні ключові методики розробки мобільних застосунків, кожна з яких слугує певним цілям.

Сучасна модель мобільного застосунку містить такі компоненти: інтерфейс користувача (UI), який визначає взаємодію з користувачем; бізнес-логіку (BL), що виконує ключові функції додатку; базу даних (DB), яка

зберігає всю необхідну інформацію та інтерфейс програмування застосунків (API), що забезпечує комунікацію між клієнтською та серверною частинами.

Будь-який повноцінний Full-stack додаток будується за допомогою трьох основних частин: фронтенду, бекенду та бази даних. Фронтенд або клієнтська сторона, відповідає за взаємодію з користувачем та зазвичай розробляється з використанням таких технологій, як HTML, CSS та JavaScript. Бекенд або серверна сторона, керує бізнес-логікою, обробкою даних та виконанням запитів, використовуючи серверні мови програмування, такі як Python, Java або Node.js. База даних зберігає всю інформацію, необхідну для функціонування застосунку, і може бути реалізована за допомогою різних систем управління базами даних, таких як MySQL, PostgreSQL або MongoDB.

Архітектура Full-stack додатку включає взаємодію різних компонентів, таких як проміжне програмне забезпечення, інтерфейси користувача, структурні компоненти, бази даних та API. Це забезпечує злагоджену роботу всіх частин системи та їхню здатність ефективно виконувати свої функції. Архітектурна модель мобільного додатку також визначає, як дані передаються між різними компонентами, забезпечуючи безперебійну та швидку роботу застосунку. Для кращого розуміння наведено схему на рисунку 1.2.

Архітектура представленої системи складається з декількох взаємопов'язаних рівнів, які забезпечують її повноцінне функціонування, починаючи від взаємодії з користувачами та закінчуючи обробкою даних та генерацією рекомендацій. У центрі цієї системи розташовані клієнтські додатки, які можуть бути представлені як веб-додатки, так і мобільні додатки. Ці клієнтські додатки взаємодіють із системою через спеціалізовані API, що відіграють роль мосту між користувачами та внутрішньою логікою системи, розміщеною на сервері рекомендацій.

Сервер рекомендацій складається з двох шарів. Перший шар – це шар

додатку, де зосереджена логіка обробки запитів на рекомендації. Цей шар поділяється на онлайн та офлайн сервіси, які разом забезпечують швидку та ефективну обробку запитів. Онлайн сервіси відповідають за негайне реагування на запити користувачів, тоді як офлайн сервіси займаються більш глибоким аналізом даних та підготовкою складніших рекомендацій. Другий шар – це шар бази даних, який зберігає інформацію про дії користувачів та асоціативні правила, необхідні для генерації рекомендацій. Цей шар баз даних є критично важливим для забезпечення точності та актуальності рекомендацій.

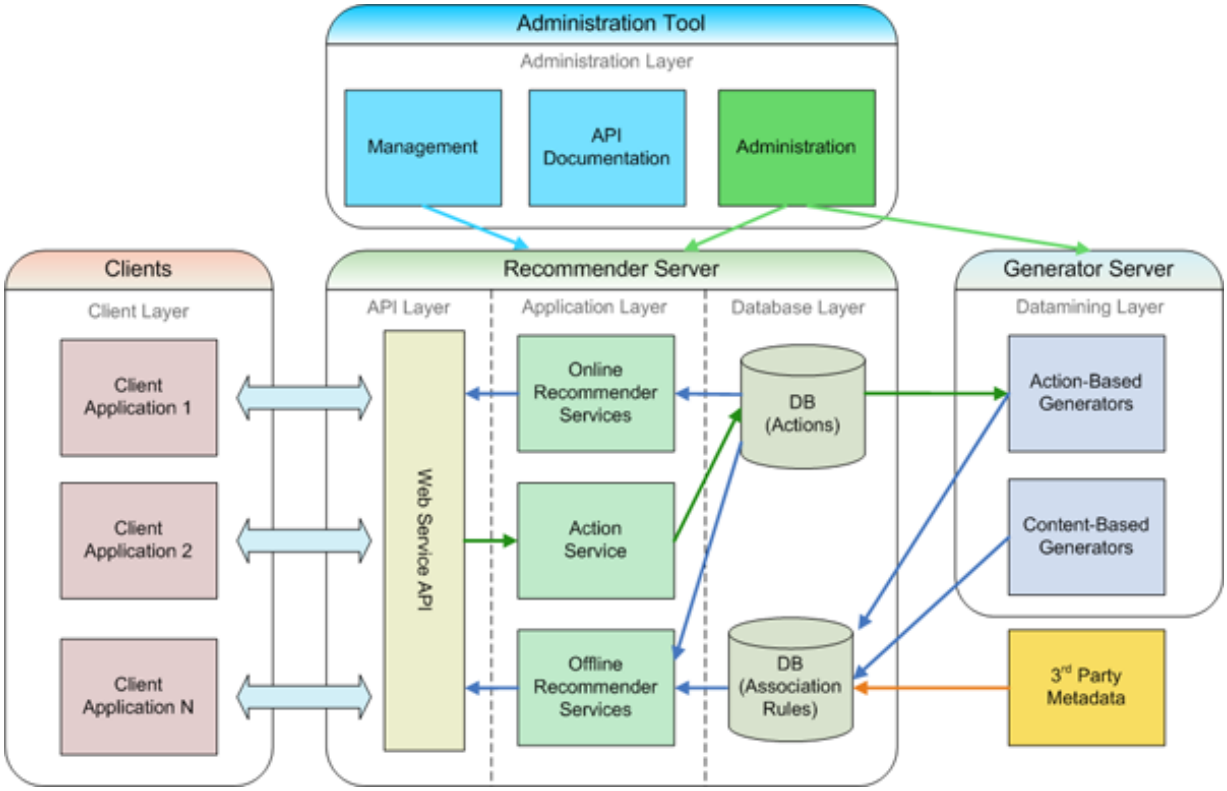

Рисунок 1.2 – Загальна структура функціонування Full-stack додатків

Крім цього, архітектура системи включає генераторний сервер, який займається датамайнінгом, використовуючи генератори, що базуються на діях та змісті. Генераторний сервер також може використовувати додаткові метадані, отримані від третіх сторін, для покращення якості рекомендацій. Це дозволяє системі бути більш адаптивною та точною, враховуючи ширший контекст та додаткову інформацію.

На верхівці архітектури розташований інструмент адміністрування, який включає інтерфейси для управління системою, моніторингу її стану, управління користувачами та надання документації API. Інструмент адміністрування є ключовим компонентом для ефективного управління та контролю над системою рекомендацій. Він дозволяє адміністраторам здійснювати моніторинг продуктивності системи, вносити необхідні корективи, управляти правами доступу користувачів та забезпечувати підтримку та оновлення системи. Завдяки цьому компоненту, система може функціонувати надійно та ефективно, забезпечуючи високий рівень обслуговування користувачів.

Архітектура веб-додатків зазвичай включає три ключові компоненти, кожен з яких відіграє критичну роль у функціонуванні системи. Перший компонент – клієнтський шар, також відомий як презентаційний шар. Він містить усі компоненти інтерфейсу користувача додатку, які відповідають за взаємодію з користувачем. Цей шар забезпечує відображення інформації та прийом введення від користувачів, створюючи зручний та інтуїтивно зрозумілий інтерфейс [4].

Другий компонент – шар додатку, на якому зосереджена бізнес-логіка. Це рівень, де виконується основний код додатку, який обробляє запити користувачів та відправляє їм результати. Логіка цього шару визначає, як дані обробляються, як реагувати на запити та як передавати дані між клієнтом та сервером.

Третій компонент – шар бази даних, що відповідає за зберігання та управління даними. Він містить усі дані додатку та надає необхідну інформацію на запити від шару додатку. База даних зберігає всю важливу інформацію та забезпечує її доступність для швидкого та ефективного використання.

Найкращі інструменти для повноцінної розробки веб-додатків можна класифікувати за шістьма фазами. Перша фаза охоплює розробку фронтенду, де використовуються такі технології, як HTML, CSS3, BootStrap, AngularJS,

ReactJS та jQuery. HTML є мовою розмітки, яка використовується для створення структури веб-сторінок, тоді як CSS3 визначає візуальний вигляд цих сторінок. BootStrap – це фреймворк для швидкого розроблення адаптивних та мобільно-оптимізованих веб-сторінок. AngularJS є JavaScript фреймворком для розробки односторінкових додатків, тоді як ReactJS – це бібліотека для створення інтерфейсів користувача. jQuery полегшує роботу з HTML-документами, обробку подій, анімацію та AJAX-запити.

Друга фаза стосується розробки бекенду, де часто використовуються мови програмування, такі як Python, PHP, Ruby та TypeScript. Python є високорівневою мовою програмування, яка широко застосовується для веб-розробки, наукових обчислень та штучного інтелекту. PHP – це популярна мова скриптів, особливо підходяща для веб-розробки. Ruby, відома своєю простотою та продуктивністю, є динамічною мовою програмування, а TypeScript, розроблена Microsoft, додає до JavaScript строгу типізацію. Також на цьому етапі використовуються веб-фреймворки, такі як Django для Python та Node.js для JavaScript, які дозволяють створювати масштабовані мережні додатки.

Третя фаза охоплює роботу з базами даних, де популярними є MySQL, PostgreSQL та MongoDB. MySQL є найпопулярнішою системою управління реляційними базами даних, PostgreSQL – потужною об'єктно-реляційною системою, а MongoDB – NoSQL базою даних, орієнтованою на документи.

Четверта фаза включає використання текстових редакторів, таких як Atom та Sublime Text. Atom – відкритий текстовий редактор від GitHub, а Sublime Text – швидкий та гнучкий редактор для коду, розмітки та прози.

П'ята фаза охоплює менеджери пакетів, такі як NPM та Grunt. NPM (Node Package Manager) – це система управління пакетами для JavaScript, яка дозволяє розробникам знаходити, ділитися та використовувати пакети коду для Node.js. Grunt – це засіб запуску JavaScript завдань. Він автоматизує завдання, що повторюються, такі як мініфікація, компіляція, тестування та лінтинг.

Остання, шоста фаза, включає контроль версій, де найчастіше використовується GitHub. GitHub – це веб-платформа для хостингу коду, яка використовує Git для контролю версій та співпраці, дозволяючи розробникам ефективно керувати проєктами будь-якого масштабу.

Кожна з цих технологій має свої унікальні особливості та переваги. HTML та CSS3 є основою для створення структури та стилю веб-сторінок, в той час як JavaScript бібліотеки та фреймворки, такі як AngularJS, ReactJS та jQuery, додають інтерактивність та динамічні елементи. Для бекенду, мови програмування, такі як Python, PHP та Ruby, разом з фреймворками Django та Node.js, дозволяють розробляти складні серверні додатки. Бази даних MySQL, PostgreSQL та MongoDB забезпечують зберігання та управління даними. Текстові редактори Atom та Sublime Text, менеджери пакетів NPM та Grunt, а також системи контролю версій, такі як GitHub, підтримують процес розробки, забезпечуючи ефективність та організованість робочого процесу.

Тепер, коли було розглянуто архітектуру повноцінних веб-додатків та технології, які необхідні для їх створення, можливо перейти до перегляду прикладів реальних додатків, які були створені з використанням різних технологічних стеків [5].

Серед найкращих повноцінних веб-додатків можна назвати декілька відомих платформ, розроблених за допомогою LAMP стека. Цей стек, який складається з Linux, Apache, MySQL та PHP дозволяє розробникам створювати веб-додатки від початку до кінця. Знамениті додатки, такі як WordPress, Facebook та Wikipedia, є яскравими прикладами успішного використання LAMP стеку. Кожен з цих додатків демонструє, як поєднання цих чотирьох компонентів може забезпечити стабільну та надійну роботу веб-платформ.

Інший популярний стек – MEAN, який включає MongoDB, ExpressJS, AngularJS та NodeJS, також дозволяє створювати потужні веб-додатки. Використовуючи JavaScript як у фронтенді, так і в бекенді, цей стек

забезпечує єдину мову програмування для всієї системи. Відомі додатки, такі як Flickr, Forbes та Youtube, були розроблені з використанням MEAN стека, демонструючи його здатність підтримувати складні та інтерактивні платформи.

Ще один популярний стек – Django, що поєднує JavaScript, Django, Python та MySQL. Цей стек використовується для створення веб-додатків, які відрізняються високою продуктивністю та надійністю. Такі відомі додатки, як The Washington Post, Instagram та Spotify були розроблені за допомогою Django стека, підкреслюючи його потужність та гнучкість у створенні масштабованих веб-додатків.

Стек Ruby on Rails, який включає Ruby on Rails, JavaScript та MySQL, також є дуже популярним серед розробників. Приклади таких додатків, як Airbnb та Github, демонструють, як цей стек може забезпечити високу продуктивність та масштабованість, обслуговуючи тисячі користувачів одночасно. Це особливо важливо для платформ, які повинні підтримувати великий обсяг трафіку та забезпечувати безперебійну роботу.

Кожен з цих технологічних стеків має свої переваги та особливості, що робить їх придатними для різних типів веб-додатків. Використання цих стеків дозволяє розробникам створювати потужні, масштабовані та надійні платформи, які відповідають сучасним вимогам та очікуванням користувачів. Вибір відповідного стека залежить від конкретних потреб проєкту та ресурсів, доступних для його реалізації.

2 ВИБІР СТЕКУ ІНСТРУМЕНТІВ ДЛЯ ОГЛЯДУ АРХІТЕКТУРИ

2.1 Огляд технологій для мобільної Full-Stack розробки

У розробці мобільних Full-stack додатків ключовим аспектом є вибір правильних технологій. Розглянемо основні мови програмування, фреймворки та UI бібліотеки, які використовуються для створення сучасних мобільних додатків.

Dart є мовою програмування, розробленою Google, яка використовується разом із фреймворком Flutter для створення високопродуктивних мобільних додатків для Android та iOS. Dart надає розробникам можливість створювати багатофункціональні додатки з плавною анімацією та швидкою реакцією на дії користувачів. Використання Dart разом із Flutter дозволяє розробникам створювати додатки з єдиною кодовою базою, що значно спрощує процес розробки та підтримки.

Мови програмування JavaScript та TypeScript є базовими для багатьох сучасних фреймворків, які використовуються для розробки як веб-, так і мобільних додатків. JavaScript, відома своєю динамічністю, є однією з найпопулярніших мов у світі, яка використовується для створення інтерактивних веб-сторінок та додатків. Вона дозволяє розробникам створювати багатофункціональні інтерфейси, які можуть працювати безперебійно на різних пристроях. TypeScript, що додає до JavaScript статичну типізацію, робить код більш передбачуваним та легким для підтримки. Це особливо корисно у великих проєктах, де важливо зберігати високу якість коду та мінімізувати помилки.

Swift та Kotlin є офіційними мовами для розробки додатків відповідно для iOS та Android. Swift, розроблена Apple, пропонує сучасний підхід до створення додатків для iOS, дозволяючи розробникам використовувати прості та інтуїтивно зрозумілі синтаксиси для створення багатофункціональних додатків. Мова Kotlin, розроблена JetBrains, стала

офіційною мовою для Android завдяки своїй сумісності з Java та покращеній безпеці. Kotlin дозволяє розробникам писати більш зрозумілий та безпечний код, що значно знижує кількість помилок [6].

Java, хоча і є однією з перших мов, що підтримується на платформі Android, до цього часу залишається популярною завдяки великій екосистемі бібліотек та інструментів. Вона забезпечує стабільну та надійну основу для створення різноманітних додатків, від простих до дуже складних.

Фреймворки та бібліотеки також відіграють важливу роль у розробці сучасних додатків. React Native, розроблений Facebook, дозволяє створювати мобільні додатки з використанням JavaScript та React. Цей фреймворк надає змогу розробникам писати більшу частину коду один раз та використовувати його для обох платформ – Android та iOS, зберігаючи при цьому нативну продуктивність та вигляд. Flutter, створений Google, використовує мову Dart та пропонує швидку розробку з гарячим перезапуском (hot reload). Flutter дозволяє створювати візуально багаті додатки з єдиною кодовою базою для Android та iOS, завдяки власному рендерингу інтерфейсу.

Xamarin, розроблений Microsoft, використовує C# та дозволяє розробляти кросплатформні додатки, які мають нативний вигляд та продуктивність. Цей фреймворк інтегрується з .NET екосистемою, що робить його привабливим для розробників, знайомих з цими технологіями. Ionic, що використовує веб-технології (HTML, CSS, JavaScript), дозволяє створювати гібридні додатки, які можуть працювати на будь-якій платформі. Ionic інтегрується з Angular, React або Vue.js, що дає розробникам гнучкість у виборі фреймворка.

Angular, Vue.js та Svelte також широко застосовуються для створення мобільних додатків, особливо у поєднанні з фреймворками на зразок Ionic. Angular надає потужну структуру для побудови складних додатків, Vue.js славиться своєю простотою та гнучкістю, а Svelte пропонує унікальний підхід до створення високопродуктивних компонентів.

UI бібліотеки також є важливими інструментами для розробки

сучасних інтерфейсів. Material-UI та Ant Design є популярними бібліотеками для React, які надають готові компоненти для створення сучасних інтерфейсів користувача. Material-UI дотримується дизайнерських принципів Material Design від Google, тоді як Ant Design більше орієнтована на бізнес-додатки. Bootstrap, одна з найвідоміших бібліотек для створення адаптивних веб-додатків, надає готові компоненти та сіткову систему, що спрощує розробку інтерфейсів. Semantic UI пропонує гнучкі та легко налаштовувані компоненти з використанням природної мови, що полегшує розробку та розуміння коду. Tailwind CSS є утилітарним CSS-фреймворком, який дозволяє швидко створювати унікальні дизайни без написання додаткових стилів. Tailwind популярний серед розробників, які цінують високий контроль над виглядом додатку та його адаптивністю.

Розробка мобільного Full-stack додатку вимагає використання різних бекенд-технологій, які забезпечують логіку, обробку даних та взаємодію з базами даних. Вибір відповідних технологій для бекенду залежить від вимог щодо продуктивності, масштабованості та підтримки. У цьому контексті важливо розглянути основні мови програмування, фреймворки та бази даних, які є популярними серед розробників.

Python є однією з найпопулярніших мов для бекенд-розробки завдяки своїй простоті та потужній екосистемі. Вона використовується для створення веб-додатків, API та обробки даних. Python дозволяє швидко та ефективно розробляти складні додатки, що робить його популярним вибором для багатьох розробників.

JavaScript, зокрема через Node.js, забезпечує можливість використання однієї мови для фронтенду та бекенду, що сприяє зручності та єдності в розробці додатків. Node.js славиться своєю асинхронною природою, що робить його підходящим для додатків з високим навантаженням та швидкою обробкою запитів.

Java є стабільною та продуктивною мовою, яка широко використовується у великих корпоративних додатках. Завдяки потужній

екосистемі бібліотек та інструментів, Java забезпечує високу продуктивність та надійність, що робить її популярним вибором для розробки складних систем.

C# у поєднанні з платформою .NET Core забезпечує високу продуктивність та кросплатформність. Це популярний вибір для корпоративних додатків та систем з високими вимогами до безпеки та продуктивності, що дозволяє створювати масштабовані та надійні рішення.

Ruby з фреймворком Ruby on Rails, який забезпечує швидку розробку та зручність завдяки принципам конвенції над конфігурацією та DRY (Don't Repeat Yourself). Ця мова та фреймворк дозволяють розробникам швидко створювати веб-додатки, що відповідають сучасним вимогам.

PHP залишається популярним вибором для веб-розробки завдяки своїй простоті та широкому використанню. Він особливо підходить для додатків, що взаємодіють з контент-менеджмент системами (CMS), забезпечуючи ефективну та швидку розробку веб-сайтів.

Go (Golang), розроблений Google, відомий своєю швидкістю та ефективністю. Це відмінний вибір для мікросервісів та додатків з високим навантаженням, де важлива швидка обробка запитів та висока продуктивність.

Kotlin стає все більш популярним для бекенд-розробки завдяки своїй сумісності з Java та сучасним можливостям мови. Вона дозволяє використовувати існуючі бібліотеки Java та пропонує покращену безпеку та зручність використання.

Популярні фреймворки для цих мов також грають важливу роль у розробці бекенду. Django та Flask для Python пропонують різні підходи до розробки. Django забезпечує комплексний підхід з вбудованими адміністративними інструментами, тоді як Flask надає легкість та гнучкість для створення менш структурованих додатків. Express.js та NestJS є основними виборами для Node.js. Express.js є мінімалістичним фреймворком, що забезпечує гнучкість та простоту, тоді як NestJS пропонує більш

структурований підхід з використанням TypeScript та модульної архітектури.

Spring Boot для Java пропонує потужний набір інструментів для створення масштабованих додатків. Він забезпечує високу продуктивність та гнучкість завдяки мікросервісній архітектурі. ASP.NET Core для C# забезпечує високу продуктивність та безпеку, роблячи його ідеальним для корпоративних додатків. Ruby on Rails відомий своєю швидкістю розробки та зручністю завдяки чіткій структурі та великій кількості готових рішень. Laravel для PHP є сучасним фреймворком з елегантним синтаксисом та багатим набором функцій для швидкої розробки додатків. Gin для Go є високопродуктивним фреймворком, що забезпечує ефективну обробку запитів з мінімальними витратами. Ktor для Kotlin пропонує легкість та гнучкість для створення сучасних серверних додатків з використанням корутин для асинхронної обробки.

Бази даних є важливим аспектом бекенд-розробки. Реляційні бази даних, такі як PostgreSQL, MySQL, SQLite та Microsoft SQL Server, забезпечують надійність та продуктивність для додатків з потребою у складних запитах та транзакціях. Вони широко використовуються для зберігання структурованих даних та забезпечують високу продуктивність і надійність. NoSQL бази даних, такі як MongoDB, Firebase Firestore, CouchDB та Cassandra, забезпечують гнучкість та масштабованість для додатків з великими обсягами даних та потребою у швидкій обробці. Вони особливо корисні для додатків, що працюють з неструктурованими даними. Neo4j є основною графовою базою даних, що використовується для складних реляційних запитів, таких як соціальні мережі та рекомендаційні системи.

Вибір конкретних технологій для бекенду залежить від специфіки проєкту, вимог до продуктивності, масштабу та інфраструктури. Кожна з цих технологій має свої переваги та недоліки, і важливо обрати ті, які найкраще підходять для досягнення цілей мобільного Full-stack додатку, який розробляється. Таким чином, ретельний аналіз потреб проєкту та вимог до продуктивності дозволить обрати найефективніші інструменти для успішної

реалізації додатку.

Інструменти та сервіси DevOps відіграють критичну роль у розробці мобільних Full-stack додатків, оскільки вони забезпечують безперервну інтеграцію, розгортання, управління версіями, а також моніторинг та логування. Використання цих технологій допомагає автоматизувати та оптимізувати процес розробки, зменшуючи час від написання коду до його впровадження в продуктивне середовище.

Одним з основних інструментів для контейнеризації додатків є Docker. Він дозволяє ізолювати та запускати додатки у відокремлених середовищах, що забезпечує повторюваність та консистентність, незалежно від того, де запускається додаток – на локальній машині розробника чи у хмарі. Завдяки Docker, розробники можуть створювати контейнеризовані додатки, які легко переміщувати між різними середовищами без змін у коді.

Для оркестрації контейнерів зазвичай використовується Kubernetes. Цей інструмент забезпечує автоматизацію розгортання, масштабування та управління контейнеризованими додатками. Kubernetes дозволяє легко керувати складними системами, розподіляти навантаження та забезпечувати високодоступність додатків. Завдяки цьому, навіть великі системи можуть бути ефективно керовані та масштабовані відповідно до потреб користувачів.

Docker Compose полегшує роботу з багатоконтейнерними додатками, дозволяючи визначати та запускати декілька сервісів за допомогою простих YAML-файлів. Це робить процес налаштування та запуску локальних середовищ розробника швидким та зручним, що значно спрощує життя розробникам та прискорює процес розробки.

Інструменти для безперервної інтеграції та розгортання (CI/CD) також є ключовими елементами DevOps. Jenkins є потужним інструментом для CI/CD, який забезпечує автоматизацію різноманітних етапів розробки, таких як збірка, тестування та розгортання додатків. Завдяки численним плагінам, Jenkins може інтегруватися з іншими інструментами та сервісами, що робить його дуже гнучким та потужним.

GitLab CI/CD, вбудована система в платформу GitLab, дозволяє автоматизувати процеси інтеграції та розгортання безпосередньо з репозиторіїв GitLab. Це забезпечує тісну інтеграцію з системою управління версіями та дозволяє легко налаштовувати пайплайни для розробки, що сприяє більш ефективній та злагодженій роботі команди розробників.

Travis CI та CircleCI також є популярними CI/CD сервісами, які забезпечують простоту налаштування та інтеграцію з GitHub. Вони дозволяють автоматизувати процеси збірки, тестування та розгортання додатків з використанням YAML-конфігурацій, що робить їх дуже зручними для розробників.

GitHub Actions надає потужний інструмент для автоматизації робочих процесів безпосередньо в GitHub, дозволяючи налаштовувати CI/CD пайплайни та автоматизувати різні завдання на основі подій у репозиторії. Це робить робочі процеси більш інтегрованими та зручними для команд, що використовують GitHub як основну платформу для розробки.

Системи управління версіями є невід'ємною частиною DevOps процесів. Git є основною системою управління версіями, що дозволяє відслідковувати зміни в коді, працювати з гілками та злиттями, а також співпрацювати з іншими розробниками. Платформи для хостингу Git-репозиторіїв, такі як GitHub, GitLab та Bitbucket, надають додаткові інструменти для управління проєктами, інтеграції з CI/CD, код-рев'ю та багатьох інших аспектів розробки [7].

Інструменти для моніторингу та логування також є невід'ємною частиною DevOps. Prometheus є системою моніторингу та оповіщення, яка збирає та зберігає метрики в реальному часі, дозволяючи відстежувати стан додатків та інфраструктури. Grafana використовується для візуалізації даних моніторингу, що зібрані Prometheus або іншими системами, дозволяючи створювати інформативні дашборди, які допомагають розробникам та адміністраторам відстежувати продуктивність та виявляти проблеми.

ELK Stack (Elasticsearch, Logstash, Kibana) є потужним рішенням для

збору, обробки та аналізу логів. Elasticsearch забезпечує індексацію та пошук, Logstash обробляє та трансформує дані, а Kibana надає інструменти для візуалізації та аналізу. Це рішення є дуже корисним для розробників, які потребують ефективного інструменту для роботи з великими обсягами логів [8].

Splunk є комерційним рішенням для аналізу даних та логів, що надає потужні інструменти для моніторингу, пошуку та аналізу великих обсягів даних. Це робить його ідеальним вибором для великих організацій, що потребують надійного та ефективного інструменту для аналізу даних.

Sentry спеціалізується на моніторингу помилок та виключень в реальному часі, дозволяючи розробникам швидко виявляти та виправляти проблеми у своїх додатках. Завдяки Sentry, розробники можуть покращити якість своїх додатків, забезпечуючи безперебійну роботу та швидке реагування на помилки.

Таким чином, інструменти та сервіси DevOps дозволяють забезпечити ефективну та надійну розробку, розгортання та підтримку мобільних фуллстак додатків. Вони зменшують час на ручні процеси та забезпечують високу якість кінцевого продукту, роблячи розробку більш інтегрованою, автоматизованою та ефективною.

У процесі розробки мобільних Full-stack додатків API та інтеграції є критично важливими компонентами, що забезпечують взаємодію фронтенду та бекенду, а також інтеграцію з іншими сервісами та системами. Основними технологіями для реалізації цього є REST API, GraphQL, WebSockets, а також аутентифікація та авторизація.

Одним з найбільш поширених способів організації взаємодії між клієнтом та сервером є створення REST API. Для цього розробники використовують різні фреймворки, які забезпечують зручний та ефективний спосіб обробки запитів. Наприклад, Django REST Framework є потужним інструментом для створення RESTful API на основі фреймворка Django. Він надає широкий набір інструментів для серіалізації даних, обробки запитів та

керування доступом, що значно спрощує розробку складних бекенд-додатків. Flask-RESTful, як розширення для Flask, також полегшує створення REST API, відомий своєю легкістю та гнучкістю, дозволяючи швидко створювати та налаштовувати API з мінімальними витратами. Express.js для Node.js є одним з найпопулярніших фреймворків для створення веб-додатків та API, надаючи простий та потужний інструментарій для обробки HTTP-запитів та маршрутизації. Spring Boot для Java дозволяє створювати масштабовані та продуктивні REST API, забезпечуючи високий рівень абстракції та автоматизації [9].

GraphQL, як сучасна технологія для запитів даних, надає клієнтам можливість запитувати саме ті дані, які їм потрібні, з мінімальними накладними витратами. Один з найпопулярніших інструментів для роботи з GraphQL – Apollo Client/Server, який забезпечує зручний та потужний інтерфейс для створення та використання GraphQL API, включаючи клієнтську та серверну частини. Relay, який спеціалізується на інтеграції GraphQL з React, забезпечує ефективне управління станом та оптимізацію запитів, дозволяючи створювати високопродуктивні додатки з мінімальним навантаженням на мережу. Hasura, як платформа, автоматично генерує GraphQL API на основі існуючої бази даних, що значно скорочує час розробки та забезпечує високий рівень гнучкості при роботі з даними.

WebSockets забезпечують двосторонню комунікацію в реальному часі між клієнтом та сервером, що є особливо важливим для додатків, які вимагають миттєвих оновлень, таких як чати або системи сповіщень. Socket.io для Node.js є популярною бібліотекою, яка забезпечує простий інтерфейс для роботи з WebSockets, дозволяючи створювати реальні час оновлення та інтерактивні додатки з мінімальними зусиллями. Django Channels розширює можливості Django, додаючи підтримку протоколів, що використовуються для асинхронних додатків, таких як WebSockets, дозволяючи створювати реальні час інтерактивні додатки на основі Django. SignalR для ASP.NET Core є бібліотекою для створення реальних час веб-

додатків в реальному часі, автоматично керуючи підключеннями, включаючи відновлення з'єднань та масштабування.

Забезпечення безпеки є одним з найважливіших аспектів у розробці мобільних додатків, тому технології аутентифікації та авторизації займають важливе місце. OAuth є відкритим стандартом для аутентифікації, який дозволяє користувачам надавати доступ до своїх ресурсів на інших сайтах без передачі паролів, забезпечуючи високий рівень безпеки та зручності для користувачів. JWT (JSON Web Tokens) є компактним URL-безпечним засобом представлення вимог між двома сторонами, що використовується для передачі інформації, яка може бути перевірена та довірена, що робить його ідеальним для аутентифікації та авторизації. Auth0 є сервісом, що забезпечує аутентифікацію та авторизацію як сервіс, підтримуючи широкий спектр стандартів безпеки, включаючи OAuth, OpenID та SAML, що дозволяє розробникам легко інтегрувати безпечні механізми аутентифікації у свої додатки. Firebase Authentication, частина платформи Firebase від Google, забезпечує прості інструменти для аутентифікації користувачів через різні методи, включаючи електронну пошту, паролі, соціальні логіни та багато іншого, спрощуючи процес реалізації аутентифікації та забезпечуючи високий рівень безпеки.

У сучасній розробці програмного забезпечення системи управління залежностями відіграють ключову роль, забезпечуючи ефективне керування пакетами та бібліотеками, необхідними для розробки додатків. Один із найпопулярніших менеджерів пакетів для екосистеми Node.js – npm. Цей інструмент дозволяє розробникам керувати залежностями та пакетами для JavaScript-додатків. Менеджер пакетів npm надає доступ до великої кількості бібліотек та інструментів, що значно спрощує процес встановлення та оновлення залежностей, дозволяючи розробникам легко інтегрувати нові функції та підтримувати актуальність проєктів.

Однак, альтернативою npm є yarn, який пропонує швидший та надійніший менеджмент пакетів для JavaScript. Yarn оптимізує процеси

встановлення, забезпечуючи стабільні версії пакетів через використання lock-файлів. Цей інструмент стає все більш популярним серед розробників, які шукають ефективні та швидкі рішення для управління залежностями. Yarn дозволяє зменшити час встановлення пакетів та забезпечує більш передбачувану інтеграцію нових бібліотек.

Для Python існує pip, який є стандартним менеджером пакетів для цієї мови програмування. Pip дозволяє встановлювати та керувати бібліотеками та пакетами для Python-додатків, забезпечуючи легкий доступ до широкого спектру інструментів та бібліотек через Python Package Index (PyPI). Це надає розробникам змогу швидко знаходити та інтегрувати необхідні ресурси, що сприяє швидкому розвитку проєктів.

У екосистемі Java одним з найпопулярніших інструментів для управління проєктами та залежностями є Maven. Maven використовує XML-файли для визначення конфігурацій проєкту та залежностей, що забезпечує автоматизовану збірку, тестування та розгортання додатків. Завдяки своєму структурованому підходу Maven допомагає розробникам легко керувати складними проєктами, забезпечуючи високу продуктивність та надійність.

Gradle, потужний інструмент для автоматизації збірки, підтримує різні мови програмування, включаючи Java та Kotlin. Gradle дозволяє визначати та керувати залежностями проєкту, забезпечуючи гнучкість та продуктивність завдяки своїй скриптовій мові на основі Groovy або Kotlin. Це робить його ідеальним вибором для великих та складних проектів, де потрібна висока ступінь налаштування та автоматизації.

Використання цих інструментів дозволяє забезпечити ефективний процес розробки, управління та підтримки мобільного Full-stack додатку. Кожен з цих менеджерів пакетів має свої унікальні особливості та переваги, які можуть бути корисними в залежності від специфічних вимог проєкту та досвіду команди розробників. Наприклад, npm та yarn забезпечують швидкий та надійний менеджмент пакетів для JavaScript, тоді як pip пропонує простоту та зручність для Python-розробки. Maven та Gradle, зі свого боку, надають

потужні інструменти для управління проєктами та автоматизації збірки у Java-екосистемі, забезпечуючи високу продуктивність та надійність розробки.

У розробці мобільних Full-stack додатків вибір правильних технологій є ключовим для успіху проекту. Для фронтенду та бекенду популярними є мови програмування JavaScript, TypeScript, Dart, Swift, Kotlin, Java, Python, C#, Ruby, PHP, Go та Kotlin. Вони забезпечують високу продуктивність, безпеку та гнучкість. Серед фреймворків виділяються React Native, Flutter, Xamarin, Ionic, Angular, Vue.js, Svelte, Django, Flask, Express.js, NestJS, Spring Boot, ASP.NET Core, Ruby on Rails, Laravel, Gin, та Ktor. Вони допомагають створювати ефективні та масштабовані додатки. Для управління залежностями використовуються npm, yarn, pip, Maven, та Gradle, які забезпечують зручність інтеграції бібліотек та стабільність проєктів. Інструменти DevOps, такі як Docker, Kubernetes, Jenkins, GitLab CI/CD, Travis CI, CircleCI, GitHub Actions, а також системи управління версіями та хмарні платформи, забезпечують ефективну автоматизацію та інтеграцію процесів розробки. Загалом, використання цих технологій дозволяє створювати якісні, надійні та масштабовані мобільні додатки, відповідні сучасним стандартам індустрії.

2.2 Вибір стеку технологій для прикладу Full-Stack архітектури

При виборі технологічного стеку для Full-Stack розробки мобільних проєктів важливо враховувати його специфіку, вимоги до продуктивності, масштабованості та безпеки. У цьому підрозділі аналізується популярний та сучасний набір технологій: Flutter, Python FastAPI, PostgreSQL та Docker, що використовується для розробки на платформах Android та iOS.

Flutter, відкритий фреймворк від Google, дозволяє розробникам створювати нативні інтерфейси для обох основних мобільних платформ з єдиної кодової бази, забезпечуючи швидкість та гнучкість у розробці. Ця технологія відзначається можливістю швидкого експериментування завдяки

функціям гарячого перезавантаження, що значно спрощує процес вдосконалення додатків.

Python FastAPI виділяється як сучасний фреймворк у сфері бекенд-розробки, дозволяючи ефективно створювати високопродуктивні API. Використання стандартних типів Python забезпечує чітку структуру, полегшуючи створення документації та валідацію даних.

PostgreSQL є одним з лідерів серед систем управління базами даних завдяки своїй високій надійності та гнучкості. Ця СУБД використовується в усьому світі, підтримуючи складні запити, транзакції та версіонування даних, що робить її ідеальним вибором для сучасних веб-додатків.

Docker, провідна технологія у сфері контейнеризації, змінює підходи до розгортання додатків, забезпечуючи їх портативність та консистентність незалежно від цільової платформи. Використання контейнерів дозволяє розробникам швидко розгортати та масштабувати додатки, а також забезпечує додаткову ізоляцію та безпеку [10].

Вибір технологічного стеку, що складається з Flutter, Python FastAPI, PostgreSQL та Docker для Full-Stack розробки мобільних додатків, є стратегічним рішенням, яке об'єднує швидкість розробки, ефективність, гнучкість та безпеку. Таке поєднання технологій надає розробникам потужний інструментарій для створення вражаючих, високопродуктивних мобільних додатків, які можуть задовольнити різноманітні потреби користувачів та бізнес-вимоги.

Flutter, відкритий фреймворк від Google, призначений для створення високоякісних нативних інтерфейсів на платформах iOS та Android з єдиною кодовою базою. Це означає, що за допомогою Flutter можна розробляти додатки, які виглядають консистентно та працюють плавно на обох платформах, значно зменшуючи зусилля та час, необхідний для розробки.

Архітектура фреймворку Flutter складається з кількох ключових компонентів, які разом створюють потужне середовище для розробки веб-додатків. Основою є платформа Dart, мова програмування, яка

використовується для написання додатків Flutter, що забезпечує розширені можливості. Двигун Flutter, переважно написаний на C++, забезпечує підтримку низькорівневого рендерингу через графічну бібліотеку Skia від Google або власний графічний шар "Impeller". Цей двигун також відповідає за інтеграцію з платформними SDK, що дозволяє створювати високопродуктивні та візуально привабливі інтерфейси.

Бібліотека Foundation, створена на Dart, містить базові класи та функції для побудови додатків, включаючи API для зв'язку з двигуном. У фреймворку є два набори віджетів, які відповідають мовам дизайну Material Design від Google та рекомендаціям Apple щодо інтерфейсу користувача iOS. Це дозволяє розробникам використовувати будь-який набір віджетів на будь-якій платформі, з можливістю автоматичної адаптації дизайну програми до поточної операційної системи за допомогою сторонніх пакетів.

Flutter підтримує інтеграцію з кількома IDE та редакторами коду через плагіни, включаючи IntelliJ IDEA, Android Studio, Visual Studio Code та Emacs. Крім того, Flutter дозволяє використовувати інструменти з командного рядку. На ринку також з'явилися інструменти, які використовують офіційний фреймворк Flutter IDE та пропонують кастомізовані конструктори графічного інтерфейсу користувача [11].

Основним елементом у програмі на Flutter є "віджет", який може містити в собі інші віджети та описує логіку, взаємодію та дизайн елементів інтерфейсу користувача. Віджети можуть бути без стану, оновлюючись лише при зміні вхідних даних, або зі станом, з можливістю оновлення через метод setState(). Ця система віджетів, що самостійно рендеряться, дозволяє Flutter малювати інтерфейси користувача безпосередньо на піксельному рівні, на відміну від інших кросплатформних інструментів. Окрім цього, існує можливість безпосереднього малювання на полотні, що іноді використовується для імплементації ігрових двигунів у Flutter. Така можливість дозволяє розробникам створювати складні інтерактивні додатки та ігри, виходячи за рамки стандартних інтерфейсів користувача.

Flutter також пропонує гаряче перезавантаження (hot reload), що дозволяє розробникам одразу бачити зміни у коді без необхідності повного перезапуску додатку. Це значно спрощує процес розробки та дозволяє швидше експериментувати з дизайном та функціоналом.

Завдяки цій гнучкій архітектурі та розширеним можливостям, Flutter надає розробникам потужний інструментарій для створення високопродуктивних, візуально привабливих мобільних та веб-додатків. Разом з Dart's Pub менеджером пакетів та репозиторієм програмного забезпечення, Flutter створює ефективне середовище для швидкої розробки та розповсюдження додатків, дозволяючи спільноті розробників ділитися та використовувати корисні плагіни та бібліотеки.

Крім того, Flutter має велику та активну спільноту розробників, що постійно доповнює екосистему фреймворку новими плагінами, бібліотеками та інструментами. Це робить Flutter одним із найбільш привабливих інструментів для розробки мобільних додатків на сьогодні [12]. Завдяки цьому фреймворку, розробники можуть створювати додатки з високою продуктивністю, чудовою візуальною привабливістю та зручністю у використанні, що є важливими факторами для успіху в сучасному мобільному середовищі.

FastAPI – це сучасний та високопродуктивний веб-фреймворк, який призначений для створення API з використанням Python 3.6+ та стандартних типів Python. Цей фреймворк дозволяє розробникам легко створювати масштабовані веб-додатки завдяки вбудованій підтримці асинхронної обробки запитів, що значно підвищує ефективність та швидкість роботи додатків.

Однією з ключових переваг FastAPI є його інтеграція з Pydantic для валідації даних та автоматичного створення схем. Це означає, що розробники можуть використовувати стандартні типи Python для визначення параметрів, відповідей та тіл запитів, а FastAPI автоматично відобразить ці дані на JSON-схеми та документацію для вашого API. Така можливість значно спрощує

роботу з моделями та запитами, роблячи процес розробки більш ефективним та передбачуваним.

FastAPI також автоматично створює інтерактивну документацію за допомогою Swagger UI та ReDoc. Це означає, що розробники отримують зручний інструмент для тестування API прямо у браузері. Така документація дозволяє миттєво бачити, як виглядають і працюють ендпоінти, що значно полегшує процес розробки та тестування.

Крім того, FastAPI підтримує управління залежностями на всіх рівнях: для окремих ендпоінтів, для цілого додатку або навіть для окремих параметрів запиту. Це дає можливість легко інтегрувати такі залежності, як бази даних, токени аутентифікації чи інші сервіси, що робить розробку більш організованою та ефективною.

Безпека є важливим аспектом будь-якого веб-додатку, і FastAPI не залишається осторонь. Він підтримує вбудовані безпекові схеми для аутентифікації та авторизації, включаючи OAuth2 з JWT токенами, що дозволяє легко побудувати безпечні додатки. Використання цих механізмів забезпечує захист додатків від несанкціонованого доступу та гарантує безпеку даних користувачів.

Використання FastAPI для серверної логіки надає розробникам потужний інструментарій для створення швидких, ефективних та легко масштабованих веб-додатків. Фреймворк забезпечує високу продуктивність розробки завдяки великій кількості інтегрованих функцій, які сприяють підвищенню ефективності роботи. Наприклад, автоматичне створення документації, валідація даних, управління залежностями та підтримка асинхронних запитів роблять FastAPI відмінним вибором для розробників, які прагнуть створювати надійні та продуктивні додатки.

Завдяки своїй гнучкості та потужним можливостям, FastAPI підходить для широкого спектра застосувань, від невеликих проектів до великих корпоративних систем. Його інтеграція з сучасними технологіями та стандартами робить його одним із найбільш привабливих фреймворків для

розробки веб-додатків на сьогоднішній день [10].

PostgreSQL, відомий також як Postgres, є відкритою системою управління реляційними базами даних (СУБД), яка не лише використовує, але й розширює мову SQL завдяки численним розширеним функціям. Ці функції дозволяють зберігати та масштабувати великі обсяги складних даних, що робить Postgres відмінним вибором для веб-додатків, які вимагають гнучкості та ефективності у роботі з даними.

Однією з ключових особливостей PostgreSQL є його висока надійність і стабільність. СУБД підтримує транзакції з властивостями ACID (атомарність, послідовність, ізольованість, стійкість), що гарантує надійність і цілісність даних навіть у випадку збоїв системи або інших несподіваних подій. Це забезпечує високий рівень довіри до даних та стабільну роботу додатків. Крім того, PostgreSQL підтримує розширені функції, такі як тригери, збережені процедури та в'ю, які дозволяють автоматизувати обробку даних та підвищувати ефективність виконання запитів. Ці функції надають розробникам можливість створювати складні логічні конструкції та ефективно керувати даними.

PostgreSQL також володіє потужними можливостями управління та оптимізації даних. Він підтримує різні типи індексування, включаючи B-tree, Hash, GiST, SP-GiST та GIN, які допомагають значно покращити швидкість виконання запитів. Індексація дозволяє зменшити час доступу до даних, що є критично важливим для додатків з високими вимогами до продуктивності. Крім того, PostgreSQL пропонує можливість реплікації даних та розподіленої обробки, що дозволяє створювати масштабовані та відмовостійкі архітектури. Це означає, що додатки, що розробляються, можуть легко обробляти зростаючі обсяги даних та витримувати високі навантаження без втрати продуктивності.

Завдяки підтримці геопросторових даних через розширення PostGIS, PostgreSQL є ідеальним вибором для розробки географічних інформаційних систем (ГІС). PostGIS додає підтримку географічних об'єктів до бази даних

PostgreSQL, що дозволяє виконувати складні просторові запити та аналіз. Це робить PostgreSQL незамінним інструментом для додатків, що працюють з географічними даними, такими як картографічні сервіси, системи навігації та аналізу просторових даних.

Більше того, PostgreSQL підтримує великий набір типів даних, включаючи JSON та XML, що робить його дуже гнучким рішенням для управління різноманітними типами даних. Підтримка JSON дозволяє зберігати та маніпулювати напівструктурованими даними, що є корисним для сучасних веб-додатків, які часто працюють з різноманітними форматами даних. Це також дозволяє інтегрувати PostgreSQL з іншими системами та сервісами, що використовують JSON для обміну даними.

Використання PostgreSQL як системи управління базами даних надає розробникам потужний інструмент для створення складних, багатофункціональних веб-додатків. PostgreSQL забезпечує високу продуктивність, надійність та гнучкість у роботі з даними, дозволяючи розробникам ефективно керувати великими обсягами інформації та створювати масштабовані рішення. Завдяки своїм розширеним можливостям та підтримці різноманітних типів даних, PostgreSQL є ідеальним вибором для проєктів будь-якого масштабу та складності, забезпечуючи стабільну та ефективну роботу додатків [13].

Docker – це набір продуктів у формі платформи як сервіс (PaaS), який використовує віртуалізацію на рівні операційної системи для доставки програмного забезпечення у контейнерах. Ці контейнери дозволяють додаткам ефективно працювати у різних середовищах ізоляції. Docker надає як безкоштовні, так і преміальні тарифи для своїх користувачів. Програмне забезпечення, яке розміщує контейнери, відоме як Docker Engine. Воно було вперше випущене у 2013 році та розроблено компанією Docker, Inc [14].

Docker автоматизує процес розгортання додатків у легковагових контейнерах, що дозволяє додаткам ефективно працювати в різних середовищах ізоляції. Контейнери ізольовані один від одного та містять

власне програмне забезпечення, бібліотеки та конфігураційні файли. Вони можуть взаємодіяти між собою через добре визначені канали. Оскільки всі контейнери використовують сервіси єдиної операційної системи, вони споживають менше ресурсів, ніж віртуальні машини.

Docker може використовувати різні інтерфейси для доступу до функцій віртуалізації ядра Linux. Він може упакувати додаток та його залежності у віртуальний контейнер, який може працювати на будь-якому комп'ютері з Linux, Windows або macOS. Це дозволяє додатку працювати у різноманітних середовищах, включаючи локальні, публічні або приватні хмари. Коли Docker працює на Linux, він використовує можливості ізоляції ресурсів ядра Linux, такі як cgroups та простори імен ядра, а також файлову систему з можливістю об'єднання, щоб дозволити контейнерам працювати в межах однієї інстанції Linux. Це допомагає уникати накладних витрат на запуск та підтримку віртуальних машин. Docker на macOS використовує віртуальну машину Linux для запуску контейнерів.

Легковагові контейнери Docker дозволяють одному серверу або віртуальній машині одночасно запускати кілька контейнерів. Аналіз 2018 року показав, що типовий випадок використання Docker передбачає запуск восьми контейнерів на одному хості, причому чверть аналізованих організацій запускали 18 або більше контейнерів на одному хості. Docker також може бути встановлений на одноплатний комп'ютер, наприклад, Raspberry Pi.

Підтримка ядра Linux для просторів імен забезпечує ізоляцію перегляду додатку оперативного середовища, включаючи дерева процесів, мережу, ідентифікатори користувачів та монтування файлових систем, тоді як cgroups ядра надають обмеження ресурсів для пам'яті та ЦП. Починаючи з версії 0.9, Docker включає власний компонент під назвою libcontainer для використання віртуалізаційних можливостей, які надаються безпосередньо ядром Linux, на додаток до використання абстрактних віртуалізаційних інтерфейсів через libvirt, LXC і systemd-nspawn.

Docker реалізує високорівневе API для надання легковагових контейнерів, які запускають процеси в ізольованому середовищі. Модель ліцензування Docker Engine підпадає під Apache License 2.0. Docker Desktop розповсюджує деякі компоненти, що ліцензуються за GNU General Public License. Файли Dockerfile можуть ліцензуватися за відкритою ліцензією самостійно. Область такої ліцензії обмежується лише Dockerfile та не поширюється на образ контейнера.

Програмне забезпечення як сервіс від Docker складається з трьох компонентів: програмного забезпечення, об'єктів та реєстрів. Програмне забезпечення включає в себе демон Docker, який управляє контейнерами Docker та обробляє об'єкти контейнерів, слухаючи запити, відправлені через Docker Engine API. Клієнтська програма Docker надає інтерфейс командного рядка (CLI), який дозволяє користувачам взаємодіяти з демонами Docker. Docker об'єкти включають різні сутності, використовувані для збирання додатку в Docker, такі як образи, контейнери та сервіси. Реєстри Docker є репозиторієм для образів Docker, до яких клієнти Docker підключаються для завантаження або завантаження образів [15]. Ілюстративний приклад докер-файлу наведено на рисунку 2.1.

```
1  ARG CODE_VERSION=latest
2  FROM ubuntu:${CODE_VERSION}
3  COPY ./examplefile.txt /examplefile.txt
4  ENV MY_ENV_VARIABLE="example_value"
5  RUN apt-get update
6
7  # Mount a directory from the Docker volume
8  # Note: This is usually specified in the 'docker run' command.
9  VOLUME ["/myvolume"]
10
11 # Expose a port (22 for SSH)
12 EXPOSE 22
```

Рисунок 2.1 – Ілюстративний приклад докер-файлу

Вибір стеку технологій для Full-Stack розробки мобільних додатків є стратегічно важливим рішенням, що визначає продуктивність,

масштабованість та безпеку майбутнього додатку. Аналіз популярного та сучасного набору технологій, що включає Flutter, Python FastAPI, PostgreSQL та Docker показує їхні переваги та взаємодоповнюючі можливості.

Flutter дозволяє створювати нативні інтерфейси для iOS та Android з єдиною кодовою базою, що значно спрощує та прискорює процес розробки. Завдяки функціям гарячого перезавантаження та гнучкій архітектурі, Flutter забезпечує високу продуктивність та візуальну привабливість додатків.

Python FastAPI виділяється як потужний інструмент для створення високопродуктивних та масштабованих API. Використання стандартних типів Python для валідації та автоматичне створення документації сприяють ефективній та передбачуваній розробці.

PostgreSQL є надійною та гнучкою системою управління реляційними базами даних, що підтримує складні запити та транзакції. Завдяки підтримці різних типів індексів та можливостям реплікації, PostgreSQL забезпечує високу продуктивність та масштабованість додатків.

Docker надає ефективні інструменти для контейнеризації, що забезпечують портативність та консистентність додатків. Використання Docker дозволяє швидко розгортати та масштабувати додатки, забезпечуючи додаткову ізоляцію та безпеку.

Використання цих технологій забезпечує розробникам потужний інструментарій для створення високопродуктивних, надійних та візуально привабливих мобільних додатків, здатних задовольнити різноманітні потреби користувачів та бізнес-вимоги і дає хороший приклад для розгляду Full-stack архітектури в мобільних додатках.

3 ОГЛЯД РОБОТИ СЕРВЕРНОЇ ЧАСТИНИ, ЇЇ МАСШТАБУВАННЯ І РОЗВИТОК. МЕТОДИ ОПТИМІЗАЦІЇ ТА ЗАБЕЗПЕЧЕННЯ БЕЗПЕКИ В FULL-STACK ПРОЕКТАХ

3.1 Серверна взаємодія в Full-Stack проектах

У розробці мобільних додатків на Full-stack сервер виконує ключову роль, слугуючи основою, яка підтримує різноманітні функціональні можливості та забезпечує безперебійну комунікацію між клієнтською частиною (мобільним додатком) та серверною частиною (бекендом). Це дозволяє мобільним додаткам функціонувати ефективно, обробляти складні запити користувачів та надавати відповідні дані у відповідь.

Коли користувач взаємодіє з мобільним додатком, відбувається обмін даними між клієнтом та сервером. Наприклад, при натисканні кнопки для отримання додаткової інформації, клієнтська частина надсилає HTTP-запит на сервер. Цей запит може включати різні операції, такі як отримання даних, подання форм або інші дії, які потребують серверної обробки. Сервер приймає запит, аналізує його зміст та виконує необхідні дії для обробки. Це може включати зчитування даних з бази даних, виконання обчислень або інші дії, визначені бізнес-логікою додатка. Після обробки запиту сервер формує відповідь, яка повертається на клієнтську частину для відображення користувачу [16].

Маршрутизація є важливим етапом обробки запитів на сервері. Сервер використовує механізм маршрутизації для визначення, який саме фрагмент коду повинен обробити конкретний запит. Маршрути задаються для зіставлення URL-адрес зі специфічними функціями або контролерами на бекенді, що дозволяє організувати логіку додатка та забезпечити чітке розподілення завдань. Кожен маршрут відповідає за обробку певного типу запитів, що дозволяє забезпечити правильне функціонування додатка. Таким чином, маршрутизація є ключовим аспектом архітектури серверної частини,

оскільки визначає, як різні частини додатка взаємодіють між собою та як користувачі отримують доступ до різних функцій додатка [17].

Проміжне програмне забезпечення виконує важливу роль у забезпеченні належної обробки запитів та відповідей на сервері. Middleware-функції розташовуються між запитом та відповіддю та можуть виконувати різні завдання, такі як логування, аутентифікація, валідація введення та обробка помилок. Логування дозволяє відслідковувати події та дії користувачів, що є корисним для діагностики та моніторингу роботи додатка. Аутентифікація забезпечує перевірку особи користувача перед наданням доступу до певних ресурсів або функцій, що підвищує безпеку додатка. Валідація введення гарантує, що дані, отримані від користувача, відповідають необхідним вимогам та форматам, запобігаючи помилкам та некоректним діям. Обробка помилок дозволяє коректно реагувати на виникнення різних збоїв та проблем під час роботи додатка, забезпечуючи стабільність та надійність його функціонування [18].

Сервер виконує обробку даних, що може включати виконання запитів до бази даних, проведення обчислень або трансформацію форматів даних. На цьому етапі застосовується бізнес-логіка, яка визначає, як додаток повинен функціонувати та реагувати на дії користувача. Обробка даних включає виконання складних алгоритмів, аналіз інформації та підготовку даних для подальшої передачі клієнту, що дозволяє забезпечити коректну та ефективну роботу додатку, відповідно до визначених вимог та очікувань користувачів. Бізнес-логіка також визначає правила та процеси, які необхідно виконувати для досягнення бажаних результатів у додатку [17].

Сервер взаємодіє з базою даних для читання або запису даних. Це включає виконання SQL-запитів (у випадку реляційних баз даних) або використання відповідних команд для NoSQL баз даних. Для спрощення взаємодії з базами даних часто використовуються інструменти ORM (Object-Relational Mapping), такі як Sequelize (для Node.js) або SQLAlchemy (для Python). Ці інструменти дозволяють розробникам працювати з базами даних

за допомогою об'єктів, що значно спрощує та прискорює процес розробки. Взаємодія з базою даних є критичним компонентом, оскільки забезпечує збереження та доступ до даних, необхідних для функціонування додатку. Крім того, оптимізовані запити до бази даних допомагають підвищити продуктивність додатку та забезпечити швидкий доступ до необхідної інформації.

Після обробки запиту сервер генерує відповідь, яка зазвичай має формат JSON або XML. Ця відповідь містить результати обробки запиту та необхідну інформацію для оновлення інтерфейсу користувача або надання зворотного зв'язку. Відповідь може включати дані, отримані з бази даних, результати обчислень або повідомлення про успішність виконання операції. Після цього відповідь надсилається назад на клієнт, де вона використовується для оновлення інтерфейсу або інформування користувача про результат виконання його дії. Генерація відповіді є важливим етапом у забезпеченні належного функціонування додатка, оскільки вона визначає, як користувач взаємодіє з системою та отримує необхідну інформацію [17].

Серверна частина у розробці мобільних додатків на Full-stack виконує критичну роль, забезпечуючи обробку запитів, маршрутизацію, використання проміжного програмного забезпечення, обробку даних, взаємодію з базою даних та генерацію відповідей. Кожен з цих аспектів є важливим для забезпечення коректного та ефективного функціонування додатку, що дозволяє задовольнити потреби користувачів та досягти бізнес-цілей. Завдяки правильній організації серверних процесів, розробники можуть створювати надійні та продуктивні додатки, які відповідають сучасним вимогам та стандартам [13].

Загальна схема взаємодії в Full-stack проєктах наведена на рисунку 3.1. У розробці мобільних додатків на Full-stack сервер виконує ключову роль, забезпечуючи комунікацію між клієнтською та серверною частинами додатка. Сервер обробляє запити користувачів, виконує маршрутизацію, використовує проміжне програмне забезпечення для аутентифікації та

обробки помилок, здійснює взаємодію з базою даних та генерує відповіді, що повертаються на клієнтську частину.

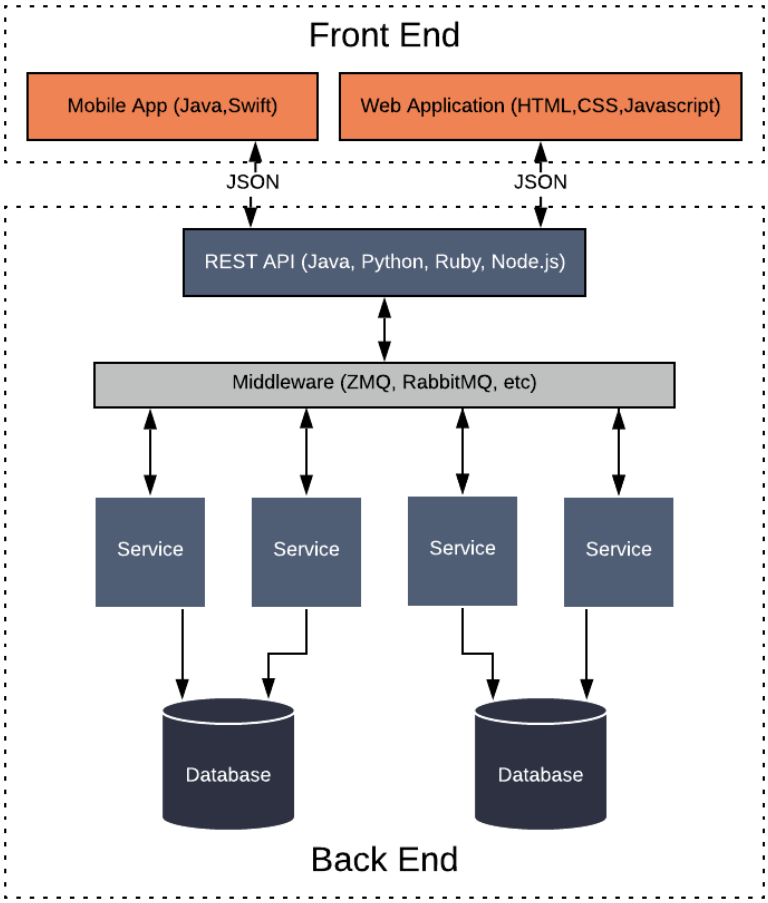

Рисунок 3.1 – Загальна схема серверної взаємодії в Full-Stack проєктах

Правильна організація серверних процесів дозволяє створювати надійні, ефективні та продуктивні додатки, що відповідають сучасним вимогам та стандартам, забезпечуючи стабільну роботу та задоволення потреб користувачів.

3.2 Розвиток та масштабування серверної частини Full-Stack проєкту

Розглянемо набір з двох компонентів: додатка та бази даних (рисунок 3.2). Це базове рішення, яке показує, що багато веб-додатків працюють саме за такою схемою. Основними перевагами цього підходу є простота використання та впровадження. Проте серед недоліків варто

зазначити обмежені можливості масштабування та забезпечення стабільної роботи.

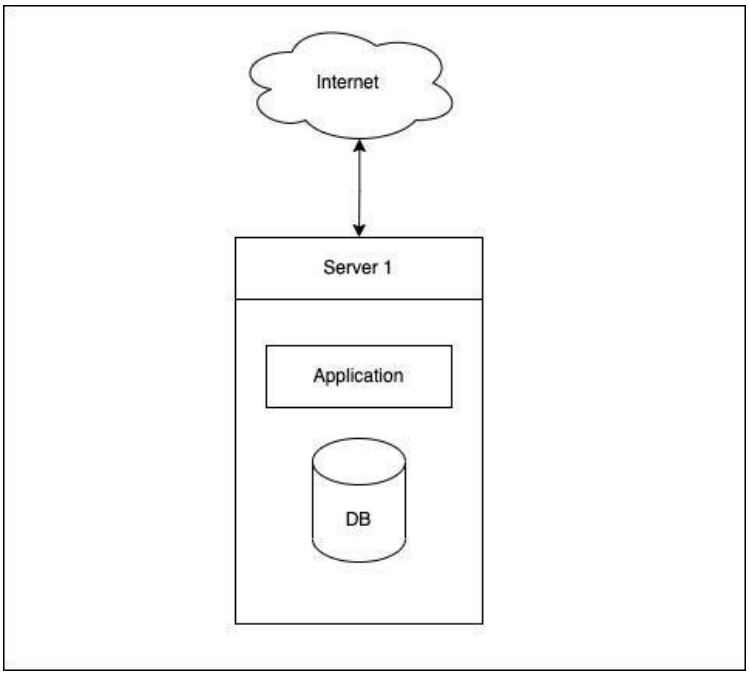

Рисунок 3.2 – Схематичний приклад роботи

Маштабування та відмовостійкість є ключовими аспектами для подальшого розвитку архітектури будь-якого додатку.

Масштабування та відмовостійкість є ключовими аспектами для подальшого розвитку архітектури будь-якого додатку. Кешування – це метод зберігання копій даних або результатів обчислень для їхнього швидкого використання в майбутньому. У веб-додатках цей процес дозволяє зберігати сторінки, об'єкти, запити до баз даних та інший часто використовуваний контент.

Зазвичай у веб-додатках ми стикаємося з процесами додавання та отримання інформації, де отримання даних відбувається набагато частіше, ніж їх додавання. Сервер може зберігати кеш статичних або динамічних елементів, що зменшує час на обробку запитів, дозволяючи серверу негайно надавати кешовані дані замість виконання складних обчислень щоразу.

Розглянемо приклад з блогом: новий пост додається щотижня. Коли підписники бачать оновлення, вони заходять на сайт, і для показу статті

системі потрібно кожного разу звертатися до бази даних та рендерити сторінку для кожного відвідувача. З підвищенням популярності блогу зростає і кількість читачів, а отже, збільшується і кількість запитів до сервера. Це може призвести до перевантаження бази даних або сервера. У такій ситуації впровадження кешування може значно покращити продуктивність, зменшивши навантаження на сервер (рисунок 3.3) [19].

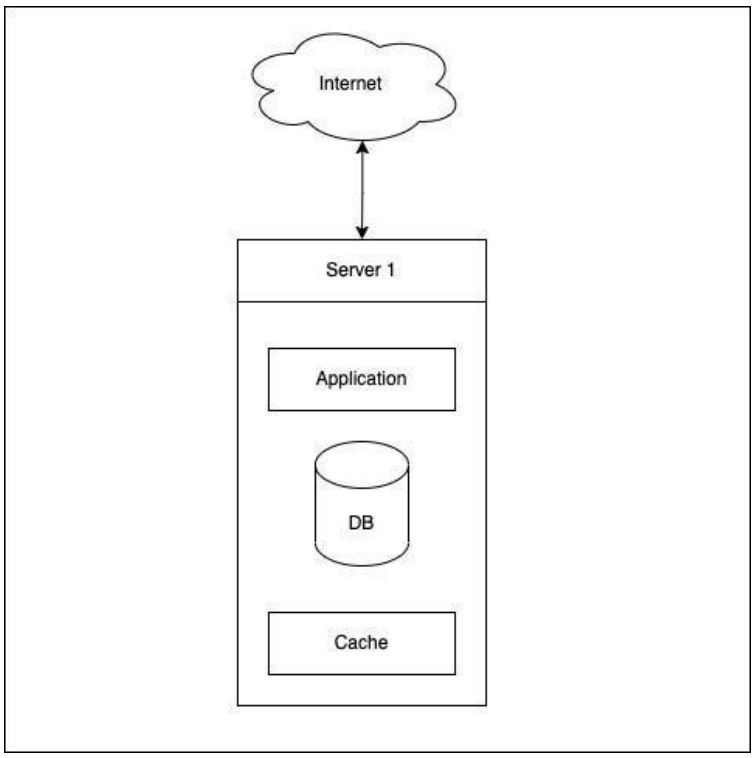

Рисунок 3.3 – App + DB + Cache

Кешування є технікою, яка широко застосовується та дозволяє зберігати вже оброблені версії сторінок, що забезпечує можливість надання сторінки безпосередньо з кешу під час подальших запитів, без звернення до бази даних. Цей підхід скорочує час реакції сервера та зменшує обсяг мережного трафіку, оскільки ресурси з кешу стають доступними для користувачів швидше. Впровадження кешування вимагає реалізації додаткових функцій у програмі, зокрема перевірки наявності актуальних даних у кеші та їх оновлення у разі змін у вже закешованому контенті.

Хоча проєкт може функціонувати без системи кешування, використання цієї техніки підвищує його здатність витримувати високе

навантаження та збільшує загальну стійкість додатку.

Інший важливий аспект для веб- та мобільних додатків – це зберігання мультимедійного контенту. Наприклад, у контексті блогу, користувач, ймовірно, захоче доповнювати свої статті зображеннями чи відео. Цей контент необхідно десь зберігати. На початковому етапі можливим варіантом є використання локального диску сервера. Однак, дисковий простір може бути дорогим, а з часом місця може стати замало, що вимагатиме переїзду на новий сервер і, відповідно додаткових зусиль [19].

Тому, якщо розробник планує зберігати значну кількість медіафайлів або дозволите користувачам завантажувати власні матеріали, кращим рішенням може бути використання зовнішніх сервісів для зберігання даних. Багато хмарних сервісів пропонують рішення для об’єктного сховища, такі як Amazon S3 або Google Cloud Storage. Альтернативою може бути налаштування власного сховища за допомогою таких інструментів, як MinIO (рисунок 3.4).

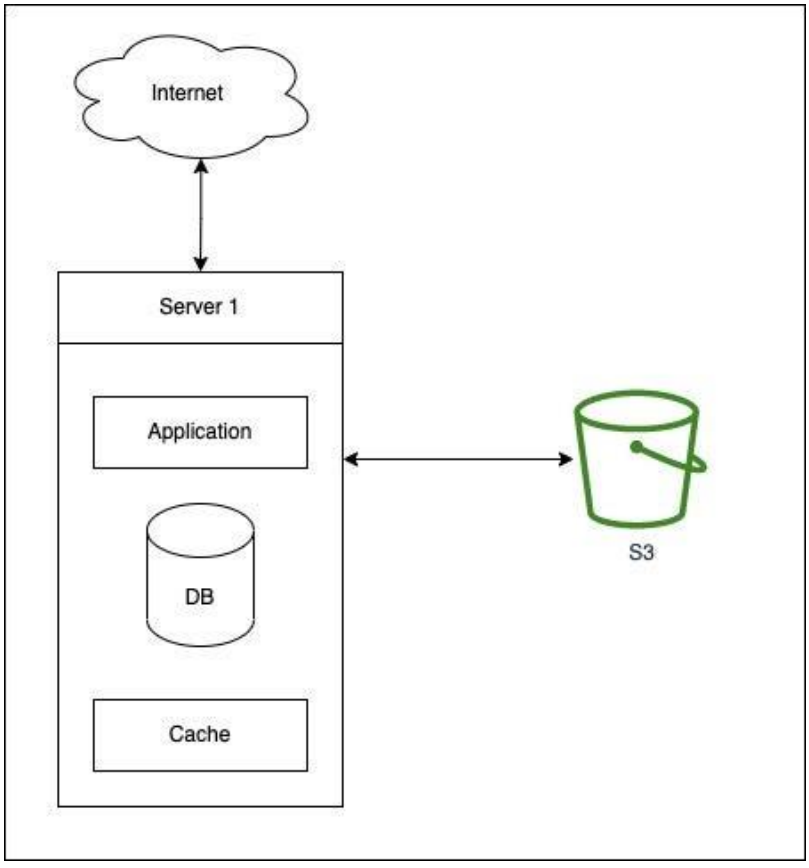

Рисунок 3.4 – App + DB + Cache + S3

Наступним етапом у вдосконаленні архітектури може стати розділення бази даних та системи кешування на окремі сервери або групи серверів (рисунок 3.5).

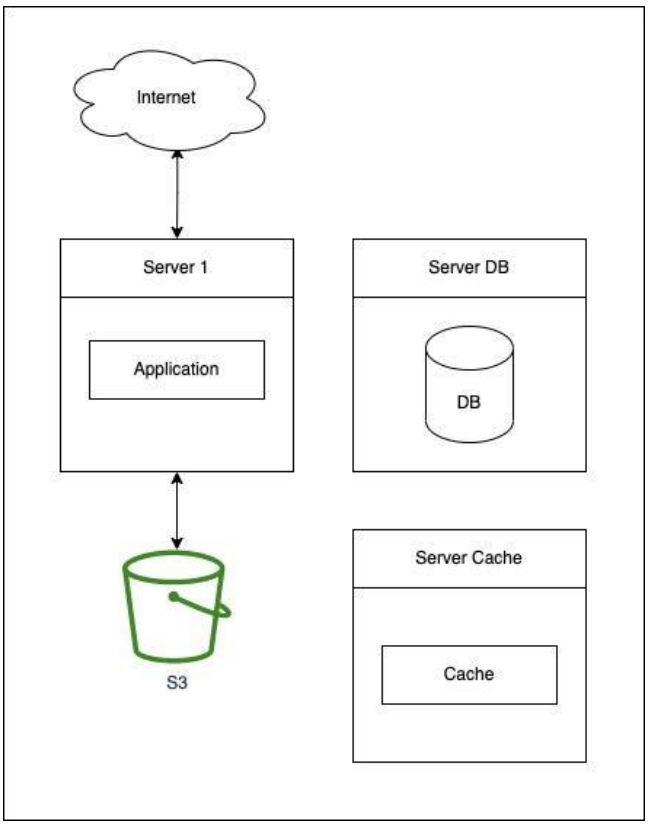

Рисунок 3.5 – Відокремлення бази даних та кешу в окремий сервер або сервери

Розглянемо більш докладніше тему масштабування. Як відомо, розподіл компонентів дозволяє здійснювати їх масштабування незалежно один від одного. У контексті масштабування існують два основні підходи.

Вертикальне масштабування передбачає збільшення або зменшення ресурсів, доступних програмі [19]. Це означає, що при зростанні вимог до ресурсів підвищуються характеристики існуючого сервера (рисунок 3.6).

Сервер може зіткнутися з обмеженнями за параметрами, такими як CPU, пам'ять або дисковий простір. Через це вертикальне масштабування не є безмежним. Коли досягаються максимальні можливості сервера, необхідно придбати новий із більшими ресурсами та перенести рішення на нього. Тому слід заздалегідь планувати стратегію масштабування.

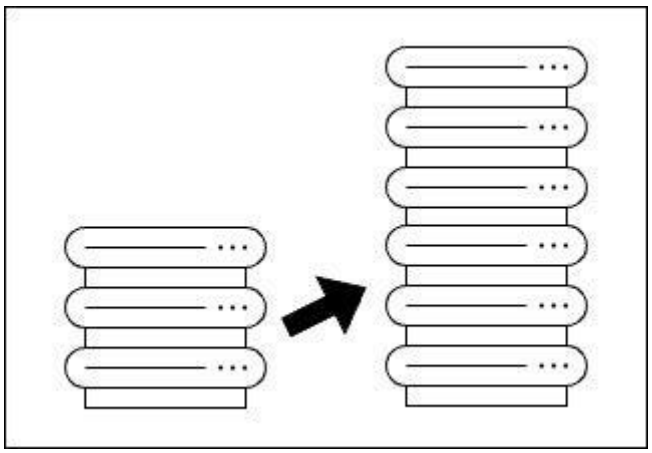

Рисунок 3.6 – Вертикальне масштабування

Горизонтальне масштабування полягає в додаванні нових одиниць або серверів до існуючої системи, тим самим розширюючи загальний пул доступних ресурсів (рисунок 3.7).

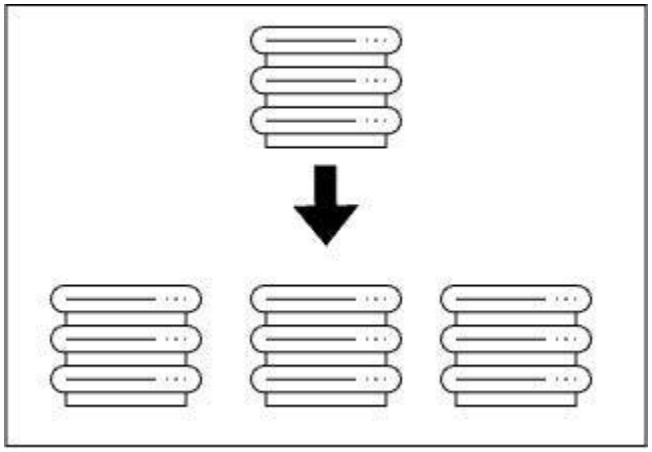

Рисунок 3.7 – Горизонтальне масштабування

Вибір розділення компонентів в архітектурі надає гнучкість у розширенні кількості серверів для додатків та баз даних відповідно до потреб. Балансування навантаження є критичною частиною структури веб-додатків, оскільки це дозволяє рівномірно розподіляти трафік між серверами. У цьому контексті важливо обговорити значення балансування навантаження для підтримки продуктивності та надійності додатків.

Балансування навантаження означає розподіл інтернет-трафіку та робочих навантажень між декількома серверами або ресурсами. Це допомагає оптимізувати використання ресурсів та знизити час відгуку на

запити, підвищуючи тим самим продуктивність додатку та швидкість відповіді користувачам. У стандартній конфігурації можливо розподіляти сервери додатків та вносити їх адреси в налаштування DNS для доступності користувачам. Проте це може бути незручно, оскільки вимагає постійного оновлення DNS при додаванні нових серверів. Цей метод може бути достатнім для деяких систем, але він не пропонує гнучкості для швидкого розгортання чи скорочення ресурсів додатку.

Основні функції балансування навантаження включають підвищення продуктивності шляхом рівномірного розподілу запитів між серверами, що знижує час відповіді на запити користувачів, забезпечення високої доступності додатку шляхом автоматичного перенаправлення трафіку на інші сервери у випадку збоїв одного з них, а також масштабування ресурсів, дозволяючи додавати або видаляти сервери з системи для збалансування навантаження.

Серед алгоритмів балансування навантаження можна виділити Round-robin, який рівномірно розподіляє запити між серверами, тобто вагове розподілення, де сервери з вищою вагою обробляють більше запитів та вибір сервера з найменшим навантаженням для забезпечення рівномірного розподілу роботи. Вибір та налаштування методу балансування навантаження залежить від специфіки та потреб додатку. Обравши відповідний алгоритм, можна оптимізувати роботу додатку, забезпечити його стабільну та ефективну роботу навіть під час піків навантаження.

Реалізація балансування навантаження в архітектурі додатку є важливим кроком у забезпеченні його масштабованості та відмовостійкості. Це дозволяє не тільки покращити загальну продуктивність, але й забезпечити більшу надійність за рахунок розподілу ризиків та навантаження між кількома серверами. Таким чином, навіть у разі виходу з ладу одного сервера, система продовжує працювати, забезпечуючи безперервний доступ до додатку для користувачів.

Оптимізація архітектури за допомогою балансування навантаження

також допомагає у гнучкому управлінні ресурсами, дозволяючи легко адаптуватися до змін у запитах користувачів та обсягах даних. Це забезпечує краще використання інфраструктури та оптимізацію витрат. В результаті виконання всіх зазначених операцій з оптимізації можна досягти ефективної та стабільної архітектури додатку (рисунок 3.8).

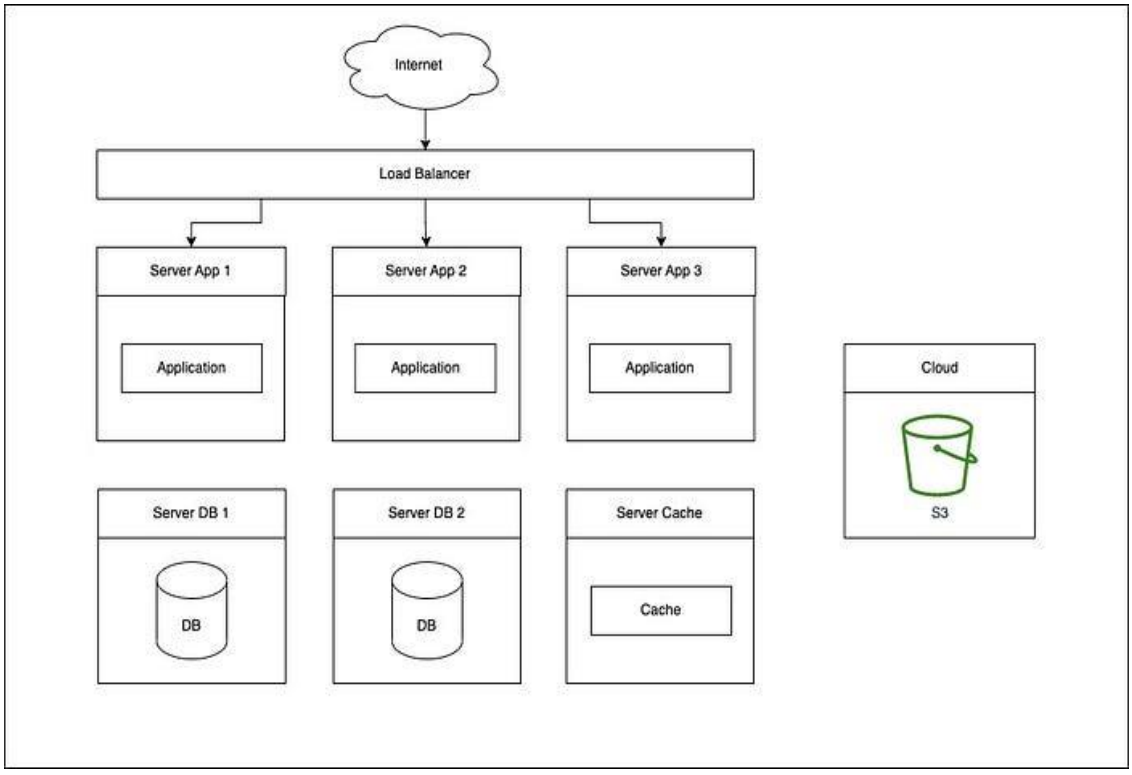

Рисунок 3.8 – Розвинена архітектурна модель Full-stack додатку

За допомогою обраної архітектури з'являється можливість збільшувати або зменшувати потужність системи та окремих її компонентів. У випадку збою одного з серверів навантаження автоматично перерозподіляється між рештою працюючих серверів. Для обробки асинхронних задач наступний крок у розвитку архітектури додатку може включати впровадження систем черг. Це сприяє підвищенню продуктивності, масштабованості та оперативності додатку. Використання черг дозволяє відокремлювати важкі або тривалі завдання від основного потоку обробки, що забезпечує більш плавну та швидшу роботу додатку.

Застосування черг дає можливість відкласти виконання ресурсомістких

задач, таких як відправлення електронних листів або обробка медіафайлів до моменту, коли система буде менш завантажена. Це дозволяє додатку швидше реагувати на запити користувачів, покращуючи їх досвід користування. Для асинхронної обробки можна використовувати різні інструменти та платформи, зокрема системи управління чергами повідомлень. Вони допомагають ефективно розподіляти завдання між робочими процесами, забезпечуючи асинхронне виконання важких операцій.

Такий підхід не лише підвищує продуктивність додатку, але й дозволяє легко адаптувати систему до змінних обсягів роботи завдяки можливості додавати або видаляти обробники завдань відповідно до поточних потреб. Загалом, архітектура розвиненого Full-stack додатку наведена на рисунку 3.9.

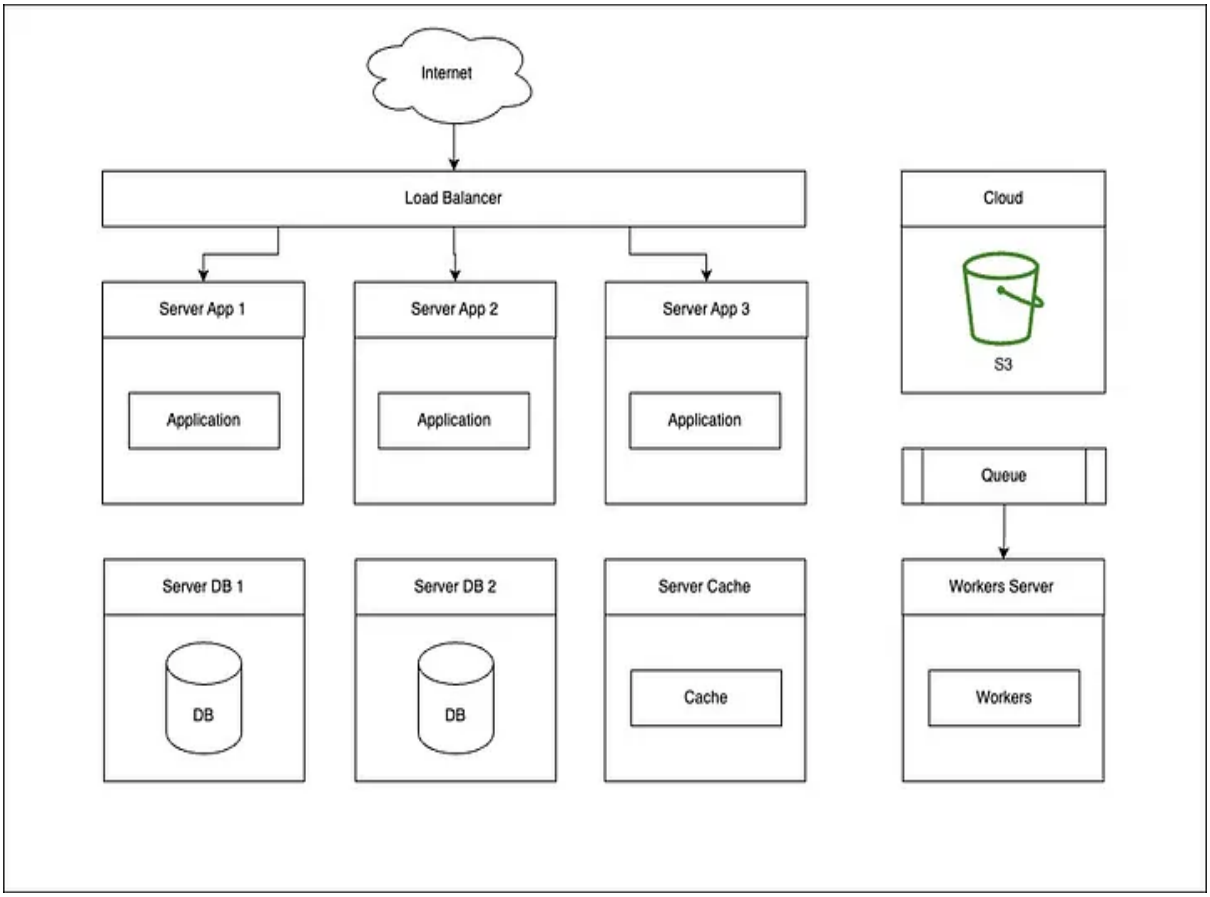

Рисунок 3.9 – Ідеальна архітектурна модель Full-stack додатку

Розвиток та масштабування серверної частини Full-Stack проєкту включає важливі аспекти, такі як використання кешування, зберігання мультимедійного контенту, вертикальне та горизонтальне масштабування, а

також балансування навантаження. Кешування дозволяє швидко надавати користувачам закешовані дані, знижуючи навантаження на сервери. Для зберігання великих обсягів мультимедійного контенту доцільно використовувати зовнішні хмарні сховища.

Масштабування може здійснюватися вертикально, тобто шляхом збільшення ресурсів існуючих серверів, або горизонтально, додаючи нові сервери до системи. Балансування навантаження забезпечує рівномірний розподіл трафіку між серверами, що підвищує продуктивність та надійність додатку.

Запровадження систем черг для обробки асинхронних задач дозволяє ефективно розподіляти навантаження та покращувати оперативність додатку, що дозволяє відкладати виконання важких завдань, забезпечуючи швидкий відгук на запити користувачів. Загалом, комплексний підхід до оптимізації архітектури забезпечує ефективність, стабільність та гнучкість Full-Stack додатку, що є ключовими для його успішного функціонування.

3.3 Захист даних і безпека в Full-Stack проєктах

У динамічному середовищі розробки повностекових додатків, де програмне забезпечення функціонує як на клієнтській, так і на серверній стороні, питання безпеки займає ключове місце. Кожен етап процесу розробки – від інтерфейсу користувача до баз даних – вносить свої виклики та можливі вразливості, які потребують уваги розробників для забезпечення захисту від потенційних загроз. Далі розглянемо основні аспекти безпеки, які необхідно врахувати при створенні додатків у повному стеці розробки.

Перший аспект, який варто взяти до уваги, це перевірка та очищення вхідних даних для захисту інтерфейсу користувача. Неналежно оброблені дані можуть стати джерелом багатьох проблем, включаючи атаки типу XSS та SQL-ін'єкцій. Важливо забезпечити ретельну перевірку та санітацію даних, що вводяться користувачами, для запобігання ін'єкцій та іншим видам зловмисних дій. Використання бібліотек та фреймворків для валідації

введених даних, таких як Validator.js для JavaScript або Marshmallow для Python, допомагає запобігти потенційним загрозам [1].

Автентифікація та авторизація також є критично важливими для контролю доступу до додатку. Автентифікація – це процес перевірки особи користувача, тоді як авторизація визначає, які дії користувач може виконувати на основі його ролі та привілеїв. Використання сучасних методів автентифікації, таких як JSON Web Tokens (JWT), OAuth, Passport.js або Firebase Auth, дозволяє ефективно управляти доступом до додатку та забезпечити безпеку даних користувача. Ці інструменти допомагають налаштовувати різні рівні доступу, перевіряти користувачів та захищати додаток від несанкціонованого доступу.

Для забезпечення захищеного зв'язку між клієнтською частиною Flutter та сервером FastAPI, використання HTTPS є необхідним. Це забезпечує шифрування даних, переданих між клієнтом та сервером, захищаючи їх від атак типу "людина посередині". Крім того, Docker може бути налаштований для запуску служб у безпечній мережі з використанням SSL-сертифікатів для шифрування трафіку. Це забезпечує додатковий рівень захисту, гарантуючи, що дані залишаються конфіденційними і не можуть бути перехоплені або змінені під час передачі [1].

Захист та шифрування даних є важливими компонентами загальної безпеки додатку. Захист даних включає впровадження політик та процедур, що запобігають несанкціонованому доступу, модифікації або видаленню даних. Шифрування даних перетворює їх у нечитабельну форму, яку можна розшифрувати лише за допомогою секретного ключа, доступного лише уповноваженим особам. Використання бібліотек, таких як bcrypt для хешування паролів, crypto для шифрування даних або MongoDB Encryption для захисту даних у базі даних, допомагає забезпечити високий рівень безпеки даних у додатку [1].

Захист від атак типу CSRF (Cross-Site Request Forgery) є критично важливим для збереження цілісності сеансів користувачів. Впровадження

захисних маркерів (tokens) та інших заходів допоможе запобігти несанкціонованим діям від імені користувачів. Це включає використання CSRF-токенів, які додаються до форм та запитів, що виконуються користувачем, і перевіряються сервером для підтвердження легітимності запиту.

Керування сеансами включає заходи безпеки для обмеження доступу до сеансів користувачів та захисту від перехоплення. Важливо регулярно оновлювати ключі сеансу, встановлювати ліміти часу на активність користувачів та використовувати безпечні методи зберігання сеансів. Це дозволяє знизити ризик викрадення сеансів та забезпечити безпечне зберігання даних користувачів.

Безпека API є важливим аспектом захисту мобільних додатків. Вона включає використання автентифікаційних токенів для перевірки користувачів та обмеження доступу до необхідних функцій. Це зменшує ризик несанкціонованого доступу до внутрішніх ресурсів та забезпечує захист від потенційних атак. Крім того, впровадження правил доступу, таких як rate limiting, допомагає захистити API від перевантажень та зловживань.

Захист баз даних вимагає застосування строгих мір контролю доступу, шифрування даних та захисту від SQL-ін'єкцій. Це включає використання параметризованих запитів, які запобігають ін'єкціям шляхом ізоляції команд SQL від введених користувачами даних. Крім того, впровадження ролей та привілеїв для користувачів бази даних забезпечує контроль доступу до даних і обмежує можливості потенційних зловмисників.

Обробка помилок та ведення журналів є важливими для забезпечення безпеки додатків. Обробка помилок включає виявлення та обробку винятків, надання користувачеві відповідного зворотного зв'язку та можливостей для відновлення. Журналювання включає запис і зберігання інформації про помилки та події, що відбуваються в додатку. Це допомагає виявляти незвичайну або зловмисну активність, запобігати атакам та реагувати на них. Використання інструментів для ведення журналів, таких як Winston, Morgan,

Sentry або Bugsnag, дозволяє швидко виявляти, діагностувати та виправляти проблеми безпеки [2].

Регулярний аудит та тестування безпеки додатків та інфраструктури є обов'язковими для виявлення слабких місць та вразливостей. Тестування передбачає перевірку роботи додатка на відповідність вимогам та стандартам безпеки, тоді як аудит включає аналіз коду, конфігурації та залежностей на предмет потенційних ризиків. Використання інструментів, таких як Jest, Mocha, Chai або NPM Audit, дозволяє виявляти та виправляти недоліки безпеки до того, як вони можуть бути використані зловмисниками.

Забезпечення безпеки в повностекових додатках є критично важливим і включає різноманітні заходи на всіх етапах розробки. Це включає перевірку та очищення вхідних даних для запобігання атакам, таких як XSS та SQL-ін'єкції, автентифікацію та авторизацію для контролю доступу, використання HTTPS для захищеного зв'язку, а також шифрування даних для захисту від несанкціонованого доступу. Важливим є також захист від CSRF-атак, керування сеансами, безпека API, захист баз даних та обробка помилок і ведення журналів. Регулярний аудит та тестування безпеки допомагають виявляти та усувати вразливості, забезпечуючи надійний та безпечний додаток, що відповідає сучасним стандартам безпеки та захищає користувачів від потенційних загроз. Використання цих методів та інструментів створює комплексний підхід до безпеки, який гарантує стабільність та захищеність додатків у динамічному середовищі розробки.

3.4 Архітектура безпеки в Full-Stack проєктах та її особливості

Для початку потрібно визначити поняття архітектури безпеки. Архітектура безпеки є стратегічним підходом до проєктування та створення захисної інфраструктури в організації. Цей підхід включає в себе аналіз процесів, методів контролю та інших систем з метою усунення ризиків, пов'язаних із захистом даних. Архітектура безпеки охоплює різноманітні аспекти, включаючи політики безпеки, управління ризиками та визначення

контрольних заходів. Це стратегічне планування є важливим для захисту в різних сферах, таких як мережна безпека, безпека програмного забезпечення та захист бізнес-даних.

Метою архітектури мережної безпеки є захист організаційної мережі за допомогою інструментів, таких як брандмауери та системи виявлення вторгнень. Вона забезпечує безпеку передачі даних та захист від зовнішніх загроз, що можуть призвести до несанкціонованого доступу або втрати даних. Водночас, архітектура безпеки програмного забезпечення зосереджується на забезпеченні безпечного кодування та впровадженні ефективних механізмів аутентифікації. Це включає захист від атак типу SQL-ін'єкцій, міжсайтових скриптів (XSS) та інших видів загроз, що можуть бути використані для компрометації додатків.

Архітектура захисту інформації в компанії інтегрує заходи безпеки з бізнес-стратегіями, об'єднуючи зусилля по захисту людей, процесів та технологій. Це забезпечує гармонійне поєднання технологічних рішень з організаційними заходами для досягнення найвищого рівня захисту. Такий підхід дозволяє забезпечити, щоб заходи безпеки підтримували загальні стратегічні напрямки компанії та сприяли досягненню її бізнес-цілей [20].

Існує кілька типів архітектур безпеки, кожна з яких займається різними аспектами захисту. Це включає мережну безпеку, безпеку хмарних та бездротових мереж, безпеку кінцевих точок, а також комплексну безпеку всього підприємства. Наприклад, архітектура хмарної безпеки зосереджується на захисті даних та додатків, розміщених у хмарних середовищах, тоді як архітектура безпеки кінцевих точок включає заходи для захисту пристроїв, що використовуються працівниками для доступу до корпоративних ресурсів.

Основні компоненти архітектури безпеки включають встановлення нормативної бази, управління безпекою, оцінку ризиків, керування ідентифікацією та доступом, шифрування, реагування на інциденти та навчання з питань безпеки. Встановлення нормативної бази передбачає

розробку та впровадження політик та процедур, що встановлюють стандарти безпеки та забезпечують їх дотримання. Управління безпекою включає моніторинг та контроль за дотриманням цих стандартів, а також розробку заходів для зниження ризиків та реагування на інциденти безпеки.

Оцінка ризиків включає постійний аналіз та моніторинг потенційних загроз, що дозволяє організації оперативно реагувати на зміни у загрозовому ландшафті. Це допомагає вчасно виявляти вразливості та розробляти ефективні заходи для їх усунення або зменшення впливу потенційних атак. Керування ідентифікацією та доступом дозволяє ідентифікувати користувачів і регулювати їх доступ до важливої інформації та ресурсів, забезпечуючи, що лише уповноважені особи мають можливість доступу до конфіденційних даних.

Шифрування використовується для захисту даних від несанкціонованого доступу, особливо під час їх передачі або зберігання. Це гарантує, що навіть у разі витоку інформації, вона залишиться незрозумілою для неуповноважених осіб. Реагування на інциденти безпеки включає швидке виявлення та вирішення порушень безпеки, мінімізуючи їх наслідки та допомагаючи відновити нормальну роботу систем. Ефективні механізми реагування допомагають запобігти подальшим порушенням та забезпечують безперервність бізнес-процесів.

Важливою частиною архітектури безпеки є навчання та освітні програми для працівників, що сприяють підвищенню обізнаності з питань безпеки серед усіх співробітників. Це включає навчання кращим практикам, ознайомлення з політиками безпеки та тренінги з виявлення та реагування на потенційні загрози. Підвищення рівня обізнаності сприяє створенню культури безпеки в організації, де кожен працівник відчуває особисту відповідальність за захист даних та систем [19].

Приклади архітектур безпеки включають такі фреймворки, як TOGAF, SABSA, Zachman Framework, NIST Cybersecurity Framework та ISO/IEC 27001. Кожен з цих фреймворків пропонує різні підходи та методології для

інтеграції заходів безпеки в структуру організації. TOGAF (The Open Group Architecture Framework) надає детальний підхід до проєктування архітектури підприємства, включаючи аспекти безпеки. SABSA (Sherwood Applied Business Security Architecture) фокусується на бізнес-орієнтованій безпеці, тоді як Zachman Framework забезпечує структурований підхід до управління інформаційними системами. NIST Cybersecurity Framework та ISO/IEC 27001 пропонують стандарти та керівні принципи для побудови ефективної системи управління безпекою.

Наявність архітектури безпеки має вирішальне значення для зменшення ризиків, пов'язаних із інформаційними активами організації, та забезпечення комплексного захисту від кіберзагроз та несанкціонованого доступу. Це також дозволяє досягти відповідності до бізнес-цілей, адаптуючи заходи безпеки так, щоб вони підтримували загальні стратегічні напрямки компанії. Конфіденційність, цілісність та доступність (CIA triad) є основними принципами, які архітектура безпеки прагне захистити, використовуючи різні заходи, такі як шифрування та автентифікація для забезпечення безпечного доступу та захисту від несанкціонованих дій.

Архітектура безпеки має чітку структуру, якої слід притримуватися для забезпечення ефективного захисту. Основою цієї архітектури є розробка та впровадження різноманітних політик та процедур, які встановлюють стандарти безпеки, необхідні для забезпечення захищеності даних та систем. Ці нормативи допомагають у формуванні міцної структури безпеки, що охоплює як технічні, так і організаційні аспекти.

У рамках архітектури безпеки особлива увага приділяється управлінню та контролю доступом, що дозволяє ідентифікувати користувачів та регулювати їх доступ до важливої інформації та ресурсів. Це забезпечує, що лише уповноважені особи мають можливість доступу до конфіденційних даних, тим самим зменшуючи ризики витоку або несанкціонованого доступу.

Значною частиною архітектури є також управління ризиками, яке передбачає постійний аналіз та моніторинг потенційних загроз. Це дозволяє

організації оперативно реагувати на зміни у загрозовому ландшафті, виявляти вразливості та розробляти ефективні заходи для їх усунення або зменшення впливу потенційних атак.

Ключовим елементом є також шифрування, яке використовується для захисту даних від несанкціонованого доступу, особливо під час їх передачі або зберігання. Шифрування гарантує, що навіть у разі витоку інформації вона залишиться незрозумілою для неуповноважених осіб, оскільки дані будуть перетворені в нечитабельну форму за допомогою секретного ключа. Це робить шифрування критично важливим елементом для захисту чутливої інформації під час її зберігання та передачі, забезпечуючи конфіденційність даних.

Реагування на інциденти безпеки є важливою складовою архітектури безпеки, оскільки надає можливість швидко ідентифікувати та вирішувати безпекові порушення, мінімізуючи їх наслідки. Ефективні механізми реагування допомагають відновити нормальну роботу систем та запобігти подальшим порушенням. Це включає створення планів реагування на інциденти, визначення ролей та відповідальностей, а також проведення регулярних тренувань для підготовки персоналу до можливих надзвичайних ситуацій.

Важливою частиною архітектури безпеки є також навчання та освітні програми для працівників, що сприяють підвищенню обізнаності з питань безпеки. Це включає навчання кращим практикам, ознайомлення з політиками безпеки та тренінги з виявлення та реагування на потенційні загрози. Підвищення рівня обізнаності сприяє створенню культури безпеки в організації, де кожен працівник відчуває особисту відповідальність за захист даних та систем. Це дозволяє зменшити ризик людських помилок, що часто стають причиною безпекових інцидентів.

Архітектура безпеки є критично важливим аспектом для захисту інформаційних активів організації, забезпечуючи комплексний підхід до захисту від кіберзагроз та несанкціонованого доступу. Вона включає в себе

стратегічне планування та впровадження різноманітних політик та процедур, що охоплюють перевірку та очищення вхідних даних, автентифікацію та авторизацію, використання HTTPS, шифрування даних, захист від CSRF-атак, керування сеансами, безпеку API, захист баз даних, обробку помилок та ведення журналів, а також регулярний аудит та тестування. Важливими компонентами архітектури безпеки є управління ризиками, реагування на інциденти та навчання працівників, що сприяють створенню культури безпеки в організації. Використання фреймворків, таких як TOGAF, SABSA, Zachman Framework, NIST Cybersecurity Framework та ISO/IEC 27001, допомагає інтегрувати заходи безпеки в структуру організації, забезпечуючи надійний захист даних та підтримуючи стратегічні бізнес-цілі.

4 АНАЛІЗ І ОБГРУНТУВАННЯ ПОДАНОЇ МОДЕЛІ

Розробка якісного мобільного додатку починається з визначення його мети та напряму за яким слід рухатись, щоб досягти необхідного результату. Кожен додаток має низку етапів, і як правило, розробка починається з ідеї. Наступним кроком здійснюється аналіз предметної області, пошук концепції та шляхи його створення. На етапі аналізу ідея перетворюється на чіткий план дій, де здійснюється вибір відповідних технологій та методологій для подальшої розробки програмного забезпечення (ПЗ) – це так званий підхід або план, за яким команда розробників буде створювати та наповнювати додаток контентом. Тому вибір моделі розробки додатку визначає послідовність реалізацій задач, які було заплановано и сформульовано у технічному завданні, сформує терміни створення окремих частин мобільного додатку та у подальшому визначить його вартість [14].

Модель – це представлення об'єкта в будь-якій формі (математичній, фізичній, символічній), яка призначена для дослідження окремих частин об'єкту як окремо, так і в загальній роботі для отримання відповідей на запитання, які було сформульовано на початковому етапі. Якщо розглядати модель мобільного застосунку, то можливо сказати, що це структурований опис, який визначає архітектурні, функціональні та технічні елементи для подальшої розробки додатку. Модель може оперувати різними методологіями, які орієнтовано на створення різноманітних додатків [21].

Провівши аналіз різноманітних мобільних додатків можливо сказати, що модель буде залежити не лише від цілей даного застосунку, а від інструментарію, за допомогою якого він буде створюватися. Єдине, що можливо зрозуміти, що застосунок повинен мати інтуїтивний та зрозумілий інтерфейс користувача, виконувати закладені в нього функції за допомогою відповідної логіки (бізнес-логіки). У випадку оперування великими об’ємами даних мати сховища (наприклад, базу даних) та організовувати взаємодію між клієнтською та серверною частинами за допомогою API. Але ці складові

простішого застосунку. У випадку, коли застосунок розробляється під потреби великої кількості користувачів, слід враховувати багато факторів, які зроблять його не тільки корисним у застосуванні, а й простим і зручним. Також не слід забувати, що більшість застосунків включають в свій склад інформацію, яку треба захищати, тому що вона може зберігати фінансову цінність.

Розглянемо детальніше модель Full-Stack мобільного додатку та перелік технологій, які в ній може бути задіяно (вираз 4.1):

$$M = \{UI, CF, IL, BL, \{DB, CS\}, API, SM, MLog, MA, TM\}, \quad (4.1)$$

де UI – інтерфейс користувача (User Interface);

CF – клієнтська частина (Client Frontend);

IL – логіка взаємодії (Interaction Logic);

BL – бізнес-логіка (Business Logic);

DB – база даних (Database);

CS – хмарні сервіси (Cloud Services);

API – інтерфейс програмування застосунків (API);

SM – модуль безпеки (Security Module);

MLog – модуль моніторингу та журналювання подій (Monitoring and Logging);

MA – модуль моніторингу та аналітики (Monitoring and Analytics);

TM – модуль тестування (Testing Module).

Інтерфейс користувача (UI) є одним з найважливіших компонентів мобільного додатку, оскільки саме через нього користувач взаємодіє з програмою. Для розробки UI у даному проєкті використовується платформа Flutter, яка надає можливість створювати високоякісні, адаптивні та привабливі інтерфейси, що однаково добре функціонують як на iOS, так і на Android пристроях. Це дозволяє розробникам охопити широку аудиторію користувачів, забезпечуючи їм комфортний досвід взаємодії з додатком.

Для створення інтуїтивно зрозумілого та зручного UI необхідно

провести ретельний аналіз потреб цільової аудиторії. Це включає вивчення їхніх вимог та очікувань від додатку, а також дослідження найкращих практик у сфері дизайну користувацького досвіду (UX). Завдяки адаптивному дизайну, додаток буде сумісним з різними розмірами екранів та типами пристроїв, що забезпечить зручність взаємодії для всіх користувачів.

Клієнтська частина (CF) відповідає за відображення UI та забезпечує взаємодію з додатком на мобільних платформах. Основна мова програмування для Flutter – це Dart. Вона забезпечує високу продуктивність додатку, оптимізуючи використання ресурсів пристрою та знижуючи енергоспоживання, що дозволяє додатку швидко реагувати на дії користувача, забезпечуючи плавний та відзивчивий досвід взаємодії.

Одним з важливих аспектів клієнтської частини є її адаптація до різних мобільних платформ. Це включає налаштування UI та функціональності додатку відповідно до особливостей iOS та Android, що дозволяє забезпечити консистентний досвід користувача на різних пристроях.

Логіка взаємодії (IL) відповідає за обробку дій користувача та забезпечує зв’язок з сервером через API. У проєкті для реалізації цієї логіки використовується фреймворк Python FastAPI, який дозволяє швидко та ефективно обробляти запити від клієнта, перевіряти та обробляти введені дані, а також надсилати запити на сервер та отримувати відповіді. FastAPI забезпечує високу продуктивність та низьку затримку при обробці запитів, що є критично важливим для додатків, які працюють у режимі реального часу. Крім того, він підтримує асинхронні операції, що дозволяє ще більше підвищити ефективність обробки запитів.

Бізнес-логіка (BL), як правило, реалізує основні функції додатку, включаючи обробку запитів користувача, валідацію даних та виконання бізнес-процесів. Для її реалізації використовується мова Python, яка забезпечує гнучкість та легкість в реалізації складних бізнес-процесів, що може включати розрахунки, генерацію звітів, взаємодію з базою даних та інші критично важливі операції. Завдяки використанню Python, можливо

швидко та ефективно розробляти та впроваджувати нові функціональні можливості, а також легко масштабувати додаток для задоволення зростаючих потреб користувачів.

База даних (DB) забезпечує зберігання та організацію всіх даних, які необхідні для роботи додатку. У проєкті використовується потужна реляційна база даних PostgreSQL, яка дозволяє ефективно організувати зберігання даних, забезпечити швидкий доступ до них, а також захистити дані від несанкціонованого доступу та втрат. Завдяки своїм можливостям масштабування та високій продуктивності, PostgreSQL ідеально підходить для великих додатків, що обробляють великі обсяги даних. Крім того, вона підтримує розширені функції безпеки, що також дозволяє забезпечити надійний захист даних користувачів.

Хмарні сервіси (CS) можуть надати додаткові можливості для зберігання даних та виконання обчислень. Вони можуть використовуваться я самі по собі так і в звязці з базою даних. У проєкті використовується AWS (Amazon Web Services), що дозволяє зберігати великі обсяги даних у хмарі та обробляти їх у режимі реального часу. AWS забезпечує гнучкість та масштабованість, що дозволяє легко адаптувати додаток до зростаючих потреб користувачів. Використання хмарних сервісів також дозволяє знизити витрати на інфраструктуру та забезпечити високу доступність додатку, що є важливим аспектом для забезпечення безперебійної роботи.

Інтерфейс програмування (API) забезпечує взаємодію між клієнтською та серверною частинами додатку. У проєкті використовується FastAPI, який дозволяє реалізувати надійний обмін даними між клієнтом та сервером, обробку запитів та забезпечення захисту передачі даних через шифрування та автентифікацію. FastAPI підтримує сучасні стандарти безпеки, включаючи OAuth 2.0 для автентифікації та авторизації користувачів, а також SSL/TLS для шифрування даних під час їх передачі між клієнтом та сервером. Це забезпечує високий рівень захисту особистих даних користувачів та запобігає можливим загрозам. Використання цих стандартів допомагає підвищити

безпеку (SM) мобільного додатку.

Моніторинг та журналювання подій (ML) здійснюється за допомогою ELK Stack (Elasticsearch, Logstash, Kibana). Ці інструменти дозволяють аналізувати дані та виявляти можливі загрози, що дозволяє оперативно реагувати на інциденти безпеки та забезпечити надійний захист додатку.

Механізм аналітики та моніторингу (MA) дозволяє відстежувати продуктивність додатку, аналізувати активність користувачів та виявляти можливі проблеми. У проєкті використовуються інструменти Prometheus та Grafana, які забезпечують збір та аналіз даних у режимі реального часу. Prometheus дозволяє зберігати дані про продуктивність додатку, включаючи метрики використання ресурсів та активність користувачів. Grafana надає можливість візуалізації цих даних, що дозволяє легко виявляти тренди та проблеми, а також приймати обґрунтовані рішення щодо покращення продуктивності додатку.

Модуль тестування (TM) забезпечує надійність та стабільність роботи додатку. Також проєкт використовує різні інструменти для тестування, включаючи Unittest для юніт-тестування, Selenium для інтеграційного тестування, а також інструменти для тестування витривалості та безпеки. Юніт-тестування дозволяє перевірити окремі компоненти додатку на коректність роботи, інтеграційне тестування забезпечує перевірку взаємодії між компонентами, а тестування витривалості дозволяє виявити проблеми, пов'язані з великою кількістю запитів та високим навантаженням. Тестування безпеки дозволяє виявити можливі вразливості та забезпечити надійний захист додатку.

Використання моделі, яка наведена у виразі 4.1 була частково обґрунтована на фундаментальному рівні в науковій статті [22] і на даному етапі була опрацьована на прикладі всіх вищезазначених технологій.

При подальшому аналізі даної моделі також виріс новий варіант того, як її можна було б доповнити, але цей варіант буде валідним лише про необхідності розширення поточної моделі (вираз 4.2):

$$M' = \{UI, CF, IL, BL, \{DB, CS\}, API, SM, MLog, MA, TM, MS, DO, ML, KB, AD, MP\} \quad (4.2)$$

де: MS – мікросервісна архітектура (Microservice Architecture);

DO – оркестрація даних (Data Orchestration);

ML – машинне навчання (Machine Learning);

KB – кешування (Caching);

AD – автоматизація розгортання (Automated Deployment);

MP – інтеграція мобільних платежів Mobile Payments.

Один з ключових напрямків такого розширення (MS) – це перехід до мікросервісної архітектури, який передбачає розбиття додатку на менші, незалежні служби, які можуть бути розроблені, розгорнуті та масштабовані окремо. Наприклад, використання технологій Docker та Kubernetes дозволяє забезпечити контейнеризацію та оркестрацію мікросервісів, а Spring Boot можна застосувати для розробки мікросервісів на Java.

Щоб підвищити продуктивність та ефективність обробки даних, доцільно додати механізми оркестрації даних (DO) та кешування (ML). Зокрема, Apache Kafka може використовуватися для обробки потоків даних та забезпечення взаємодії між мікросервісами, а Redis – для кешування даних, які часто використовуються, що дозволить знизити навантаження на базу даних.

Інтеграція аналітичних інструментів та моделей машинного навчання (ML) значно підвищить можливості додатку в аналізі даних та передбаченні поведінки користувачів. Використання TensorFlow або PyTorch для розробки та впровадження моделей машинного навчання, а також Apache Spark для обробки великих даних та аналітики, дозволить додатку більш ефективно обробляти дані та надавати користувачам цінну інформацію.

Для покращення безпеки (SM) можна додати розширені механізми автентифікації, авторизації та моніторингу. Наприклад, Keycloak може бути використаний для керування автентифікацією та авторизацією користувачів,

а OWASP ZAP – для автоматизованого тестування безпеки додатку, що забезпечить високий рівень захисту від потенційних загроз.

Автоматизація процесів розгортання та інтеграції (AD) може значно підвищити ефективність розробки та розгортання додатку. Використання Jenkins або GitLab CI/CD для автоматизації процесів безперервної інтеграції та розгортання (CI/CD) забезпечить швидке та ефективне впровадження змін та нових функцій. Terraform можна застосувати для управління інфраструктурою як кодом (Infrastructure as Code), що дозволить знизити ризики, пов’язані з ручним налаштуванням інфраструктури, та забезпечить її повторюваність та надійність.

Додавання модуля для інтеграції мобільних платежів (MP) значно розширить функціональність додатку. Використання Stripe або PayPal SDK дозволить реалізувати безпечні та зручні платежі, що підвищить задоволеність користувачів та розширить можливості додатку.

Отже, новий архітектурний план може включати такі компоненти: мікросервісну архітектуру (MS) з використанням Docker, Kubernetes та Spring Boot; оркестрацію даних (DO) та кешування (KB) за допомогою Apache Kafka та Redis; розширену аналітику та машинне навчання (ML) з використанням TensorFlow, PyTorch та Apache Spark; розширені механізми безпеки (SM) з Keycloak та OWASP ZAP; автоматизацію розгортання (AD) з Jenkins, GitLab CI/CD та Terraform та модуль для роботи з мобільними платежами (MP) з інтеграцією Stripe або PayPal SDK.

Ці додаткові компоненти дозволяють забезпечити ще більшу гнучкість, масштабованість, продуктивність та безпеку моделі, роблячи її більш адаптованою до сучасних вимог ринку та потреб користувачів.

Варто зазначити, що запропонована модель є комплексною та може адаптуватися до вимог конкретного проєкту. Наприклад, деякі модулі можуть бути видалені через непотрібність та додані у майбутньому з масштабуванням проєкту.

ВИСНОВКИ

Архітектура мобільного Full-Stack застосунку на базі Flutter, FastAPI, Postgres та Docker представляє собою сучасне рішення для розробки високопродуктивних, масштабованих та гнучких додатків. Основні ключові аспекти цієї архітектури охоплюють різноманітні технології та інструменти, що забезпечують ефективну інтеграцію між клієнтською та серверною частинами, а також надійне зберігання та управління даними.

Flutter є відкритим фреймворком від Google, який дозволяє розробникам створювати високоякісні нативні інтерфейси для Android та iOS з єдиною кодовою базою, що значно знижує витрати на розробку та підтримку мобільних додатків, оскільки весь код пишеться один раз і використовується на обох платформах. Flutter надає потужні інструменти для створення анімацій, інтерактивних елементів та швидкої адаптації дизайну до різних розмірів екрану, що дозволяє розробляти візуально привабливі та функціональні додатки.

FastAPI – це сучасний високопродуктивний веб-фреймворк для створення API на основі Python. Він забезпечує швидку розробку та високу продуктивність завдяки використанню асинхронних викликів та автоматичної генерації документації. FastAPI підтримує валідацію даних на основі типів Python, що дозволяє розробникам створювати надійні та безпечні API.

PostgreSQL є однією з найпопулярніших систем управління базами даних, відомою своєю надійністю, продуктивністю та гнучкістю. Вона підтримує складні запити, транзакції та зберігання великих обсягів даних, що робить її ідеальним вибором для сучасних веб- та мобільних додатків.

Docker дозволяє ізолювати додатки в контейнери, забезпечуючи повторюваність та консистентність незалежно від середовища, в якому вони запускаються. Це полегшує процес розгортання та управління додатками, забезпечуючи їх портативність та безпеку. Docker також підтримує

масштабування додатків, дозволяючи легко додавати нові контейнери для обробки зростаючого навантаження.

Масштабування та відмовостійкість є критичними аспектами для забезпечення стабільної роботи додатка під час високих навантажень. Використання кешування, балансування навантаження та асинхронної обробки задач дозволяє ефективно розподіляти ресурси та знижувати навантаження на сервера, що забезпечує швидкий відгук додатка та підвищує задоволеність користувачів.

Забезпечення безпеки є важливим аспектом архітектури мобільного Full-Stack застосунку. Використання OAuth, JWT та інших методів аутентифікації та авторизації гарантує захист даних користувачів та обмеження доступу до важливих ресурсів. Інтеграція з іншими сервісами за допомогою REST API або GraphQL забезпечує гнучкість та розширюваність системи.

Архітектура мобільного Full-Stack застосунку на базі Flutter, FastAPI, Postgres та Docker забезпечує ефективну розробку, масштабованість та безпеку додатків. Використання цих технологій дозволяє створювати високоякісні мобільні додатки, які відповідають сучасним вимогам ринку та забезпечують відмінний досвід користувача.

В результаті аналізу різноманітних програмних продуктів було запропоновано та розроблено модель, яка поєднує в собі декілька сучасних технологій. Переваги розробленої моделі включають підвищену гнучкість завдяки мікросервісній архітектурі (MS), що дозволяє незалежно розробляти та розгортати окремі компоненти, а також вищу продуктивність через використання кешування (KB) та оркестрації даних (DO). Збагачена аналітика та можливість використання машинного навчання (ML) для передбачення поведінки користувачів забезпечать більш інтелектуальні функції додатку. Покращена безпека (SM) завдяки розширеним механізмам автентифікації, авторизації та моніторингу підвищить захист даних та додатку в цілому. Автоматизація розгортання (AD) дозволить швидко та ефективно

впроваджувати зміни та нові функції, що сприятиме більш гнучкому та стабільному розвитку додатку. Розширена функціональність завдяки інтеграції мобільних платежів (MP) відкриє нові можливості для монетизації та підвищить зручність для користувачів.

Проте головною перевагою розробленої моделі є те, що вона може бути спрощена задля того, аби підходити під потреби будь-якого проєкту, а її використання дозволить швидко розгорнути й у майбутньому легко та швидко маштабувати проєкт, доповнювати його функціоналом та підтримувати. Саме ці властивості роблять її універсальною та дають можливість використовувати для різноманітних проєктів з їх специфічними вимогами, обмеженнями та строками.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Xia F., Hsu C. H., Liu X., Liu H., Ding F., Zhang W. The power of smartphones // Multimedia Systems. 2015. Vol. 21. P. 87-101.

2. Panhale Mahesh. Beginning Hybrid Mobile Application Development. United States of America: Apress, 2015. 246 p.

3. Sheppard Dennis. Beginning Progressive Web App Development. United States of America: Apress, 2017. 286 p.

4. Murali P., Linke N. M., Martonosi M., Abhari A. J., Nguyen N. H., Alderete C. H. Full-stack, real-system quantum computer studies: Architectural comparisons and design insights // Proceedings of the 46th International Symposium on Computer Architecture. 2019. P. 527-540.

5. Mattila T. Building a complete full-stack software development environment. 2018. 112 p.

6. Singh C., Gaba N. S., Kaur M., Kaur B. Comparison of different CI/CD tools integrated with cloud platform // Proceedings of the 2019 9th International Conference on Cloud Computing, Data Science & Engineering (Confluence). IEEE, 2019. P. 7-12.

7. Uddin M. S., Hasan K. J., Tasnima R. A Comparative Analysis of the Rise of Git-Based Distributed Collaborative Hosting Platforms: Survey, Performance Test, and Comparison // Asian Journal of Engineering and Applied Technology. 2023. V. 12(2). P. 29-33.

8. Chhajed S. Learning ELK stack. Packt Publishing Ltd, 2015. 205 p.

9. Muhammad I.R.D., Paputungan I.V. Development of Backend Server Based on REST API Architecture in E-Wallet Transfer System // Jurnal Sains, Nalar, dan Aplikasi Teknologi Informasi. 2024. Vol. 3(2). P. 79-87.

10. Liao-Mäkinen S. Developing a full stack mobile application. 2023. 48 p.

11. Ngo C. J., Chang J., Chung S. Decreasing the barrier to entry for an open-source full-stack web development // Proceedings of the Conference on Information Systems Applied Research ISSN. 2021. Vol. 2167. P. 1508.

12. Deshpande M., Ray D., Dixit S., Agasti A. ShareInsights: An unified approach to full-stack data processing // Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data. 2015. P. 1925-1940.

13. Thielman G. Full-stack project containerization. 2022. 55 p.

14. Ключові методології розробки програмного забезпечення: робота команди зсередини. URL: https://wezom.com.ua/ua/blog/metodologija-razrabotki-programmnogo-obespechenija.

15. Сеспедес Гарсія Н.В., Сеспедес Гарсія П.Д. Моделі життєвого циклу розробки програмного забезпечення // Молодий вчений. 2023. №2 (114). С. 17-20. DOI: https://doi.org/10.32839/2304-5809/2023-2-114-4.

16. Gay S., Hole M. (1999). Types and Subtypes for Client-Server Interactions. In: Swierstra, S.D. (eds) Programming Languages and Systems. ESOP 1999. Lecture Notes in Computer Science, Vol 1576. P. 74-90.

17. Raducanu A. L., Carabas M., Barbulescu M., Tapus N., Suliman G. Client-server application with Android mobile phone technology // 2019 18th RoEduNet Conference: Networking in Education and Research (RoEduNet). IEEE, 2019. P. 1-6.

18. Balat V. Client-server Web applications widgets // Proceedings of the 22nd International Conference on World Wide Web. 2013. P. 19-22.

19. Zmerzlyi I. Архітектура веб додатків. Medium. URL: https://medium.com/@IvanZmerzlyi/архітектура-веб-додатків-ca4c82f75bcf.

20. Sheppard Dennis. Beginning Progressive Web App Development. United States of America: Apress, 2017. 286 p.

21. Розробка мобільних додатків від А до Я: повний гайд. URL: https://dan-it.com.ua/uk/blog/rozrobka-mobilnih-dodatkiv-vid-a-do-ja-povnij-gajd.

22. Бешта В.С., Комаричев А.В., Філімончук Т.В., Баранєй Д.І. Модель мобільного додатку, яка орієнтована на обробку даних // Системи управління, навігації та зв'язку. Збірник наукових праць. Полтава: ПНТУ, 2024. Випуск 3 (77). С. 43-47. doi: 10.26906/SUNZ.2024.2.135.