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)

Метод проектування веб-застосунків

(тема)

Виконав:

студент II курсу, групи СПм-20-1

Москаль В.Р
(прізвище, ініціали)

Спеціальність

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

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

Освітня програма

Системне програмування
(повна назва освітньої програми)

Керівник: ст. викл. Єрьоміна Н.С
(посада, прізвище, ініціали)

Допускається до захисту

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

2021 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-професійна
(освітньо-професійна або освітньо-наукова)

Освітня програма Системне програмування
(повна назва)

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

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

студенту Москалю В’ячеславу Романовичу
(прізвище, ім’я, по батькові)

1. Тема роботи Метод проектування веб - застосунків

затверджена наказом по університету від “ 05 ” листопада 2021 р. № 1657 Ст

2. Термін подання студентом роботи до екзаменаційної комісії 13 грудня 2021 р.

3. Вхідні дані до роботи 1) Патерни проектування
2) Бази даних
3) Front end
4) Тестування

4. Перелік питань, що потрібно опрацювати у роботі
1) Актуальність проблеми та постановка завдань дослідження
2) Архітектура веб-застосунку
3) Front-end частина застосунку
4) База даних
5) Методи тестування та оцінки продуктивності
6) Висновки

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (слайдів)

Слайд- презентація – 12 шт

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	Отримання завдання у керівника	09.11.20	
2	Аналіз проблеми та предметної області	10.11.21-16.11.21	
3	Аналіз методів архітектур веб-застосунків	17.11.21-21.11.21	
4	Аналіз методів розробки front end частини	22.11.21-26.11.21	
5	Аналіз баз даних та методів тестування	26.11.21-03.12.21	
6	Оформлення матеріалів кваліфікаційної роботи	04.12.21-06.12.21	
7	Подання кваліфікаційної роботи керівникові та	07.12.21-08.12.21	
	її попередній захист		
8	Подання кваліфікаційної роботи на рецензу-	10.12.21-11.12.21	
	Вання		

Дата видачі завдання 8 листопада 2021 р.

Студент _____
(підпис)

Керівник роботи _____ ст. викл. Єрьоміна Н.С.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка кваліфікаційної роботи: 69 с., 33 рис., 1 табл., 21 джерел.

ВЕБ-ЗАСТОСУНКИ, АРХІТЕКТУРА, БАЗА ДАНИХ, ПАТТЕРН, MVC, FRONT END, BACK END, МОВА ПРОГРАМУВАННЯ.

Метою кваліфікаційної роботи є аналіз сучасних методів розробки веб-застосунків.

У ході виконання кваліфікаційної роботи були розглянуті принципи та методи побудови веб-застосунків та проведений аналіз існуючих рішень який дозволить враховувати особливості кожного методу та обрати технологію що відповідає вимогам бізнесу.

ABSTRAC

Master’s thesis: 69 pages, 33 figures, 1 tables, 21 sources.

APPLICATION, ARCHITECTURE, DATABASE, PATTERN, MVC, FRONT END, BACK END, PROGRAMMING LANGUAGE.

The purpose of this certification work is the analysis of modern methods of web application development.

During the attestation work the principles and methods of building web applications were considered and the analysis of existing solutions which will allow to consider features of each method and to choose the technology meeting the requirements of business was carried out.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

TCP – мережева модель, що описує спосіб передання даних в цифровому вигляді (англ., Transmission ControlProtocol)

HTML – мова розмітки гіпертексту (англ., HyperText Markup Language)

CSS – формальна мова опису зовнішнього вигляду документу, написаного з використанням мови розмітки (Cascading Style Sheets)

IDE – інтегроване середовище розробки (англ., Integrated development environment)

JSON – запис об’єктів JavaScript (англ., JavaScript JavaScript Object Notation)

ПЗ - програмне забезпечення

ВСТУП

Проживаючи в цифровому світі, використання веб-застосунків суттєво спрощує життя. Застосування веб-застосунків відбувається у різноманітних сферах життя - від від сфери розваг до сфери бізнес рішень. Також, задля розвитку та закриті потреб бізнесу, використовується для реалізації багатьох різних маркетингових стратегій. Саме позитивний відгук про сервіс або товар, може збільшити вплив бізнесу на ринок в цілому. Веб-застосунок забезпечує надійність і також допомагає скласти позитивне враження та просування бренду компанії.

При створенні веб - сайту можна зіткнуться з різними проблемами, такими як: неправильно підібрана архітектура, неможливість масштабування та зміни бази даних, вразливості і так далі. Успіх і відповідно і якість веб-застосунку залежить від технологій для його розробки та платформи на якій будуть його розробляти. Його функціонал, масштабованість та складність обслуговування – ось що саме зумовлює життєздатність і конкурентоспроможність застосунку. Безпосередньо, це впливає так і на вартість розробки продукту.

Сьогодні існують різні архітектури, мови програмування , фреймворки, які допомагають створювати веб-застосунки. Таким чином, можливості сучасних технологій розробки дозволяють створювати веб- застосунки різної складності.

При створенні веб – застосунку, вибір методу розробки веб- застосунку самий важливий етап при його створенні. На це якраз впливають низка факторів, вплив швидкості інтернету на застосунок, таких як технічна оцінка розробників, потреба в доступі до інформації на пристрої.

Також, важливо правильно провести тестування. Саме, правильно підібраний набір видів тестування є актуальним та критичним для

забезпечення необхідної стабільності веб- застосунку.

Для вирішення більшості проблем, допомагає тестування, шляхом зменшення використання часу та грошей. Якщо продукт працює на 100% , то експоненціальна шкода не завдається бізнесу з точки зору витрат. Відповідно, приділяючи належної уваги тестування, витрати на технічне обслуговування будуть нижчі.

Метою роботи є аналіз сучасних методів розробки веб-застосунків, та розробка методу вибору архітектури та її створення.

1 АКТУАЛЬНІСТЬ ПРОБЛЕМИ ТА ПОСТАНОВКА ЗАВДАНЬ ДОСЛІДЖЕННЯ

1.1 Актуальність проблеми

Електронна комерція домінує у сфері покупок. Оскільки найбільшим веб-сайтом електронної комерції є Amazon, а Джефф Безос є найбагатшою людиною у світі зі значним відривом, неважко повірити, що веб-сайти електронної комерції становлять значну частину Інтернету в усьому світі. Електронна комерція справила революцію у тому, як ми робимо покупки. Більше не потрібно стояти у чергах, щоб доставити нові продукти. Сьогодні ми натискаємо пару кнопок і бажані продукти доставляються до нас на поріг.

Згідно результатів досліджень – кількість покупок на веб сайтах становить понад 26% населення світу, що робить електронну комерцію надзвичайно популярною [1].

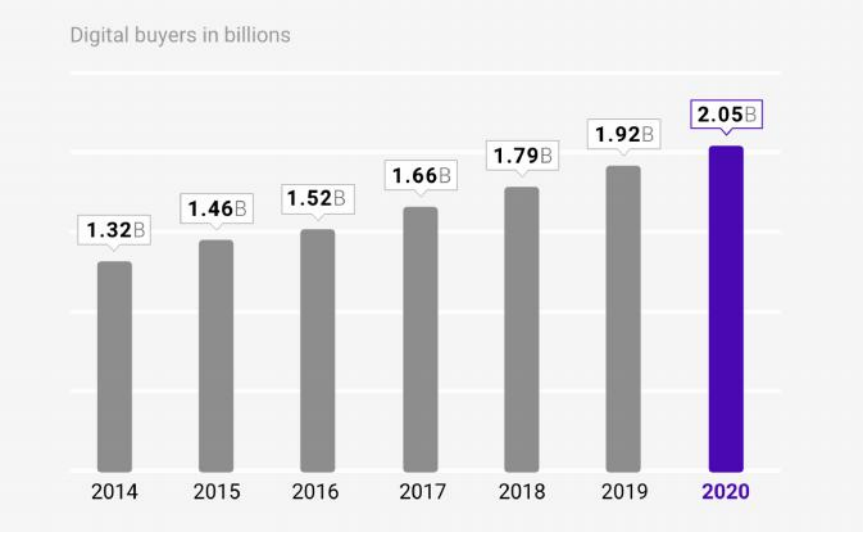

Рисунок 1.1 – Статистика кількість покупок користувачів у веб - застосунках

Також, було виявлено що кількість веб сторінок тільки збільшується (рисунок 1.2).

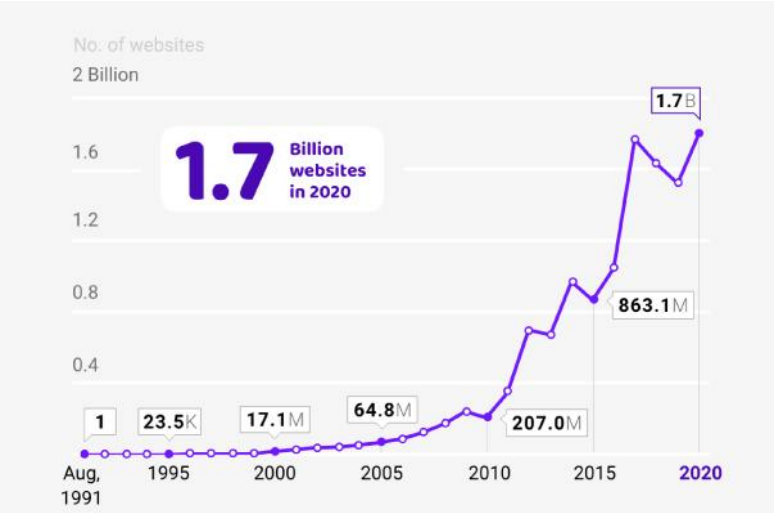

Рисунок 1.2 – Статистика збільшення кількості веб-застосунків

Це підтверджує факт, що конкуренція за увагу покупця та між бізнесами зростає. І, кожен потребує професійне створення та підтримання свого веб – застосунку.

Рисунок 1.3 – Процент відкриття сайтів згідно даним порталу Statista.com

Дані з 2009 по 2018 рік про глобальний трафік мобільних даних порталом Statista.com надало певну інформацію. 19,01 екзабайт на місяць – саме такий глобальний рух мобільних даних становив у 2018 році. По прогнозам експертів на 2022 рок очікується, що рух мобільних даних в усьому світі досягне 77,5 екзабайт, враховуючи, що на місяць загальний темп зростання в 46% (рисунок 1.3). Звіти підготованих та SMM-платформи Hootsuite відсоток генерування трафіку з мобільних пристроїв вже перевищив відсоток генерування з персональних комп’ютерів, через сервіс We Are Sotial .

За даними звітів Global Digital 2019, 68% жителів землі – власники мобільних телефонів, а 53% інтернет-трафіку припадає на мобільні телефони. Тож, зростання кількості смартфонів грає на користь бізнесу.

Важливо зазначити, що вибір ефективної архітектури та технологій для створення веб-застосунку – призведе до більшого прибутку бізнесу.

Також, було виявлено що відвідуваність сторінки буде покращуватись, на що впливає зменшення часу завантаження сторінок та взаємодія з користувачем. Тим паче, це збільшує отримати шанси лідируючі пункти у рейтину для пошукових системах. Якщо сервер працює повільніше, ніж дві секунди це призведе до зменшення кількості сканерів, які надсилає Google на сайт. Для прикладу, компанія Amazon виявила, що затримка на їх веб-застосунку в 100 мс коштує їм 1% продажів. Це вагомо може вплинути на обороти бізнесу та на його показники.

Причиною падіння трафіку на 20% у 2006 році при певних експерементах зі сторони Google, були призведені через виведення на головній сторінці 33 результати, замість 9. Час відгуку сторінок зріс з 400 до 900 мілісекунд [4].

Дані дослідницької групи Gomez зображені на рисунку 1.2. Він ретеально показує співвідношення секунди до дій на веб-застосунок. Як не дивно, навіть одна секунда часу має сильний вплив на потужність та

швидкість обробки запитів веб-застосунком [3].

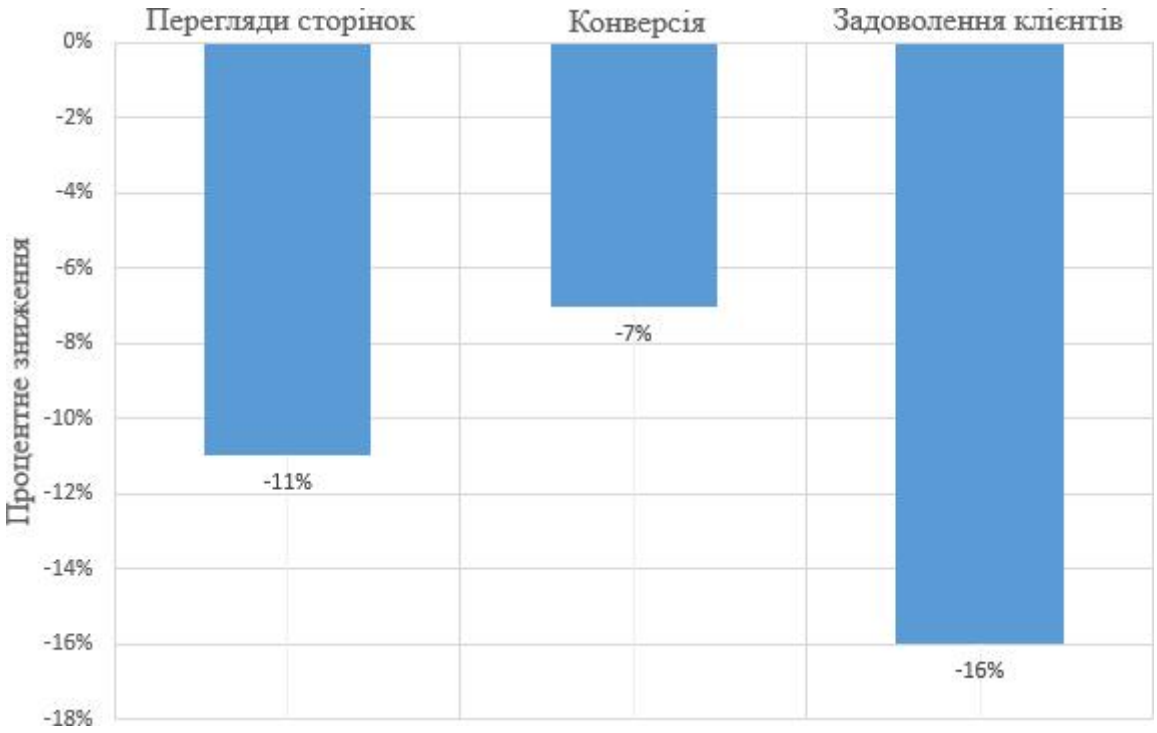

Рисунок 1.4 – Вплив часу відгуку на веб-застосунок

Користувачі продовжують отримувати доступ до веб-сайтів через смартфони та інші мобільні пристрої, що підкреслює що веб-застосунки залишаються актуальними. Згідно дослідженнь Gomez, 58% користувачів мобільних пристроїв очікують, що швидкість завантаження веб-застосунку на сайтах буде такою ж самою як на домашніх комп’ютерах. 61% користувачівзауважили, що вірогідність нового відвідування мобільного веб-сайту зменшиться при низькій його продуктивності [3].

1.2 Основні поняття та визначення

Наступні визначення описують термінологію, яка використовується в кваліфікаційній роботі. Визначення взяті з обраної літератури.

База даних (БД) це організована структура, яка призначена задля зберігання, зміни та обробки взаємозалежної інформації, дуже великих обсягів. Особисті дані клієнта, історія замовлень, каталог товарів – усе це об’єднання великої кількості даних, яке являє собою єдину базу та дає змогу для формування безлічі варіації групування інформації [6].

Front end – це клієнстька сторона інтерфейсу користувача по відношенню до програмно-аппаратної частини сервісу. Сюди відноситься все що бачить користувач, відкриваючи сторінку[7].

Back end – це програмно-аппаратна частини сервісу. Це набір інструкцій, за допомогою яких відбувається реалізація бізнес логіки веб – застосунку[7].

Веб-сервер – це комп’ютерна система, яка обробляє вхідні мережеві запити через HTTP та кілька інших пов’язаних протоколів [5].

Патерн проектування – це повторюване архітектурне рішення для створення та масштабування веб-застосунку.

1.3 Фактори для вибору методи розробки

Розробка веб-застосунку вимагає чіткий вибір технологій та методу розробки. І, необхідно зрозуміти по яким критеріям їх обирають на проект. Нижче перечислені важливі пункти для вибору технологій:

- розмір та складність проекту;
- швидкість розробки;
- вартість та доступність фахівців;
- тренд його розвитку;
- наявність детальної документації;
- вартість підтримки;
- вимоги до навантажень;
- вимоги до безпеки;

- можливості інтеграції з іншими рішеннями;

В цій роботі більш детально розглянемо декілька з них.

1.4 Огляд існіючих рішень

Вибір виграшної основи серед усіх підходів до розробки застосунків є складним завданням. Перше що розглянемо – це розмір та складність проекту. Проекти діляться на:

- прості;
- середні;
- складні.

Прості (візитки, лендінги, прості інтернети-магазини, прості застосування). Такі рішення зазвичай створюються в CMS або шаблонах.

Середні (складні інтернети-магазини і маркетплейси, портали національного масштабу, всілякі сервіси, просунуті застосування). Такі рішення зазвичай робляться на фраємворках.

Складні (величезні портали, соціальні мережі, інноваційні і нетипові рішення). Ядро таких проектів зазвичай розробляється на чистій (нативній) мові.

Також, важливі технології які будуть використовуватись в проекті. Їх можна виділити на три рівня абстрації:

- чиста мова;
- фраємворк;
- CMS.

Чиста мова. Facebook, Amazon, Digg, Linkedin – сервіси, якими користується більшість та являється явним прикладом зроблених сайтів на чистій мові з відвідуваністю в сотні мільйонів і мільярдів користувачів. Також, дані проекти потребують в створенні нових технології для себе, оскільки що вже існують їх не влаштовують.

Фраємворк — це середовище розробки для програміста з готовими правилами і інструментами. Фраємворк як допомагає прискорити розробку так накладає певні обмеження. На фраємворках робляться проекти середньої складності з відвідуваністю в мільйони.

CMS — це готове рішення, конструктор, завдякив якому можна зібрати потрібний проект. Його швидше не програмують, а набудовують.

Для створення успішного веб-застосунку необхідно підібрати стек технологій.

Стек технологій - це комбінація мов програмування, фреймворков і програмного забезпечення, використовуваних для створення застосунка. Коли ви плануєте наступний технологічний стек застосунка, ви повинні враховувати серверну і клієнтську сторони.

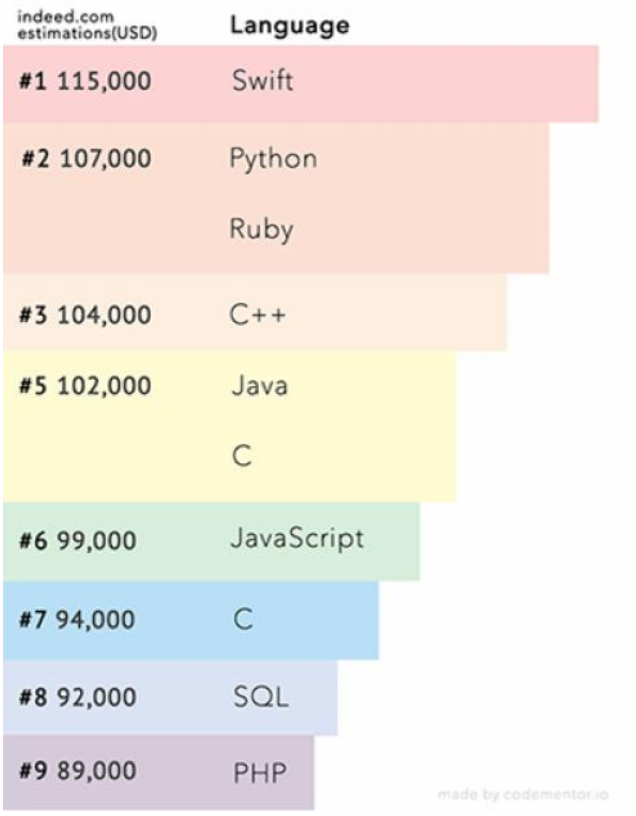

Рисунок 1.5 – Вартість розробників на різних мовах програмування

Вибір технологій залежить від запланованої архітектури проекту. Саме архітектор продумує основні блоки майбутнього сайту. Яка мова ляже до основу, чи буде він нативним або фреймворк, яку систему кешування вибрати, які бази даних, як все це буде зв’язано і так далі.

Один з найважливіших чинників вибору технології є вартість і доступність фахівців, тому що саме це — сама витратна частина в будь-якому проекті. Це детально продемонстровано на данному рисунку 1.5.

Також, вибираючи технологію, нам потрібно розуміти як технологія веде себе на ринку. Особливо, якщо мова йде про великий проект. Всі технології дуже швидко розвиваються, виходять все нові і нові версії. Мови сильно змінюються кожні 5-7 років, фреймворки — кожні 2-3 роки, а CMS — кожні 1-2 роки. Важливо вибрати не просто хорошу технологію сьогодні, а передбачити тренди розвитку так, щоб залишитися на коні і через декілька років. Інакше, доведеться переписувати проект, що завжди дуже проблематично. На рисунку 1.6 представлені тренди мов програмування.

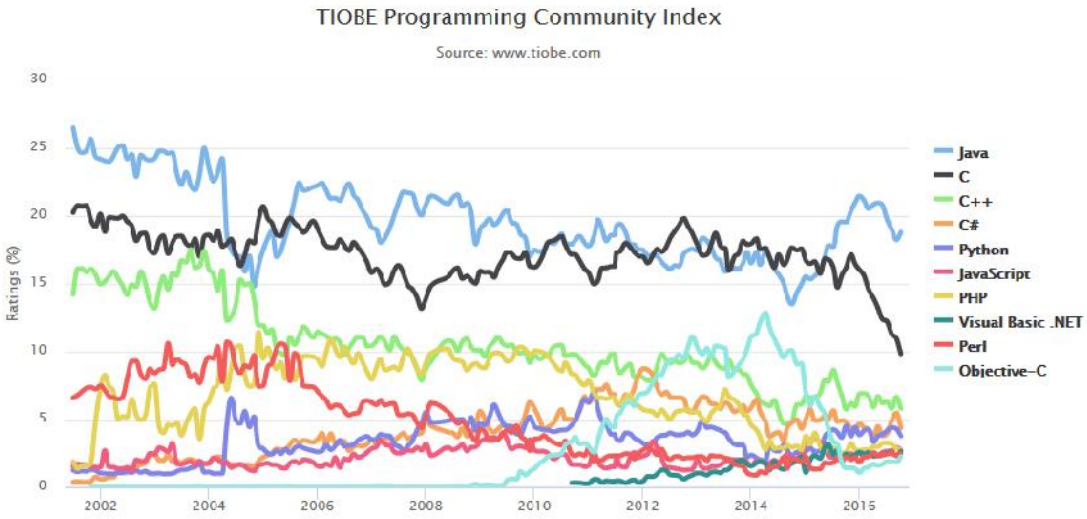

Рисунок 1.6 – Дослідження популярності мов програмування

У наступних розділах буде розглянута окремі технології що допомагають реалізувати веб-застосунок.

2 АРХІТЕКТУРА ВЕБ-ЗАСТОСУНКА

2.1 Аналіз патернів проектування

Патерни пропагують розділенню елементів системи та сприяють повторному використанню певного шаблону, надаючи рішення часто повторюваним проблемам. Коли певна проблема вирішена багатьма розробниками певним чином, і це відзначається як ефективним рішенням – це стає патерном проектування веб-застосунку.

Патерни описані використовуючи патерн-схему [11] . Вони мають в собі такі компоненти:

- контекст (ситуація що створює дану проблему);
- проблема;
- рішення (подана зі структурними елементами).

Вам необхідно правильно підібрати архітектуру веб-застосунку по даним причинам [9].

Команда розробників програмного забезпечення приймає перші рішення саме під час вибору моделі архітектури. Для цього етапу доступні тільки бізнес – вимоги. Перший набір рішень являється вирішальним в любому проекті, оскільки неоптимальні рішення можуть привезти до збільшення або втрати бюджету з боку клієнта.

Зацікавленим сторонам необхідно знати при хід робот над проектом. Вона діє як інстурмент візуальної коммунікації в команді. Вони будуть розуміти яку систему ви будуєте.

Патерни проектування полегчують повторне використання в інших проектах. Архітектура програмного забезпечення не обмежується архітектурним стилем, а часто являється комбінацією декількох, створюючих повну систему.

На вибір патерну впливає багато факторів. Ці фактори включають в себе можливості команди яка створює та реалізує веб-застосунок; досвід команди; інфраструктура та організаційні обмеження.

На даний момент можна виділити наступні патерни проектування [10]:

- клієнт/Сервер;
- model – view – controller;
- n - tier архітектура;
- шарова архітектура *(* Layered Architecture);
- сервісно-орієнтована архітектура;
- об’єктно-орієнтована архітектура;
- p2p (Peer – to – peer).

Першим розглянемо MVC.

2.2 MVC

Завдяки даному шаблону систему можна подылити на три частини: модель, представлення і контроллер. Таким чином, модифікація кожного компонента може здійснюватися незалежно.

Концепція шаблону модель–вигляд–контролер (MVC) була описана Трюгве Реенскаугом в 1978 році, що працював в науково-дослідному центрі "Xerox PARC" над ЯП "Smalltalk". Пізніше, Стів Бурбек реалізував шаблон в Smalltalk - 80. Остаточна версія концепції MVC була опублікована лише в 1988 році в журналі Technology Object. Згодом шаблон проектування став еволюціонувати. Напр., була представлена ієрархічна версія HMVC; MVA, MVVM.

Основна мета застосування цієї концепції полягає у відділенні бізнес-логіки(моделі) від її візуалізації(представлення, виду). За рахунок такого розділення підвищується можливість повторного використання коду.

Найбільш корисне застосування цієї концепції в тих випадках, коли користувач повинен бачити ті ж самі дані одночасно в різних контекстах і/або з різних точок зору.

Приклад даного патерну представлений на рисунку 2.1.

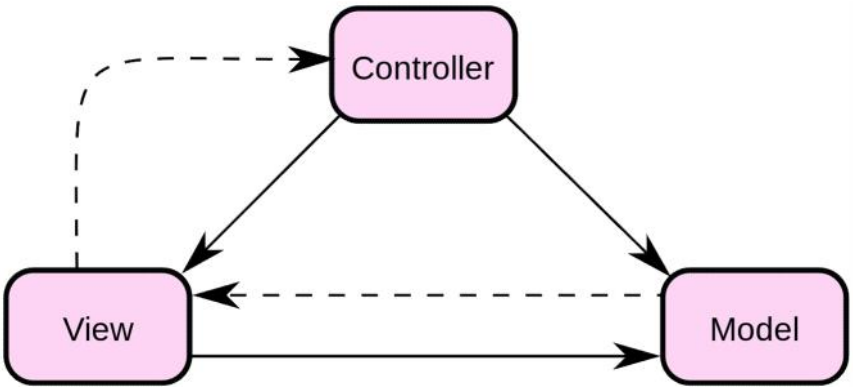

Рисунок 2.1 – Модель патерну MVC

У рамках архітектурного шаблону MVC програма поділяється на три окремі частини з розподілом функцій між компонентами. Модель (Model) відповідає за зберігання даних та їх структуру. Вигляд (View) відповідальний за інтерфейс програми. Контролер (Controller) керує компонентами, отримує сигнали у вигляді реакції на дії користувача та передає дані у модель.

Модель являється центральним компонентом та відображає поведінку застосунку, незалежну від інтерфейсу користувача. Модель має безпосереднє відношення до прямого керування даними, логікою і правилами застосунку.

Події, які зареєстровані, транслюються в різні запити, що спрямовуються компонентам моделі або об’єктам. Саме вони являються відповідальними за відображення необхідних даних. Відокремлення моделі від вигляду даних дозволяє незалежно використовувати різні компоненти для відображення інформації. Таким чином, вносячи зміни через контролер користувач до моделі даних, то інформація, подана одним або декількома візуальними компонентами, буде автоматично редагуватись відповідно до

змін, що відбулися.

Існує декілька видів даного патерну. Найбільш поширені:

- Model-View-Controller;
- Model-View-Presenter;
- Model-View-View Model.

Цей патерн являється популярним. Багато веб-застосунків та веб-фраєморків, такі як Spring и Rails використовують данний патерн. Розглянемо його переваги:

- використання даної модели пришвидчує розробку;
- команда розробників можуть представити користувачам декілька представлень;
- модель не форматує дані користувача, тому можуть використовувати цей патерн з любим інтерфейсом.

Також, розглянемо декілька недоліків даного підходу, наприклад:

- при використанні даного патерну, в коді з’являються нові шари програми, що складнює дану програму;
- як правило, даний патерн потребує використання декількох технологій, що збільшує час на його створення.

2.3 Мікросервіси

Програмна архітектура має не багато спільного з функціональними вимогами. Можна реалізувати набір сценаріїв (функціональних вимог до програми) з використанням будь-якої архітектури.

Звісно, архітектура також важлива, бо вона визначає так названі вимоги до якості обслуговування, відомі також як нефункціональні вимоги або атрибути якості.

Сьогодні все більше спеціалістів збігаються на тому, що при будуванні великої, складної програми слід замислитися над використанням

мікросервісів.

У мікросервісної архітектури є безліч визначень. Перші розуміють назву занадто буквально та затверджують, що сервіс повинен бути дуже малим, наприклад складатися зі 100 рядків коду. Інші вважають, що на розробку сервісу повинно відходити не більше двох тижнів.

Адріан Кокрофт, який раніше працював в Netflix, визначає мікросервісну архітектуру, як сервіс-орієнтовану, що складається з слабко пов’язаних елементів з обмеженим контекстом.

У книзі Мартіна Еббота та Майкла Фішера The Art of Scalabity описується практична тривимірна модель масштабування у вигладі кубу. Ця модель визначає три напрямки для масштабування програм: X, Y та Z. Модель представлена в рисунку 2.2.

Масштабування по осі X часто використовують в монолітних програмах. Принцип праці цього підходу такий: завантажуєтсья декілька прототипіва програми. Балансувальник розподіляє запити між однаковими екземплярами. Це відмінний спосіб покращити потужність та доступність програми.

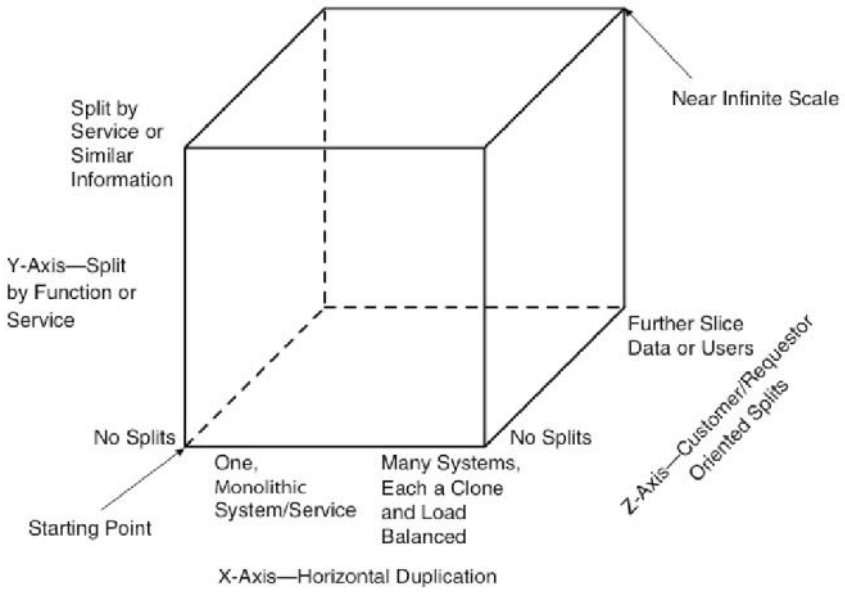

Рисунок 2.2 – Тривимірна модель масштабування

Масштабування по осі Z також передбачає запуск певних екземплярів монолітної програми, але в цьому випадку, на відміну від масштабування по осі X, кожний екземпляр відповідає за певну підмножину даних. Маршрутизатор, що виставлений спереду, задіє атрибут запиту, щоб направити його до відповідного екземпляра. Для цього, наприклад, можна використовувати поле userId.

В даному сценарії екземпляр програми відповідає за перелік користувачів. Маршрутизатор може перевірювати поле userId, що вказане в заголовку запиту авторизації, щоб обрати одну з ідентичних копій програми.

Масштабування за осями X та Z збільшує потужність та доступність програми. Але не один з цих підходів не вирішує проблем з ускладненням коду та процесу розробки. Щоб впоратися з ними, потрібно використати масштабування по осі Y, або функціональну декомпозицію (розподіл). Те, як це працює, представлено на рисунку 2.3 : монолітна програма розподіляється на окремі сервіси.

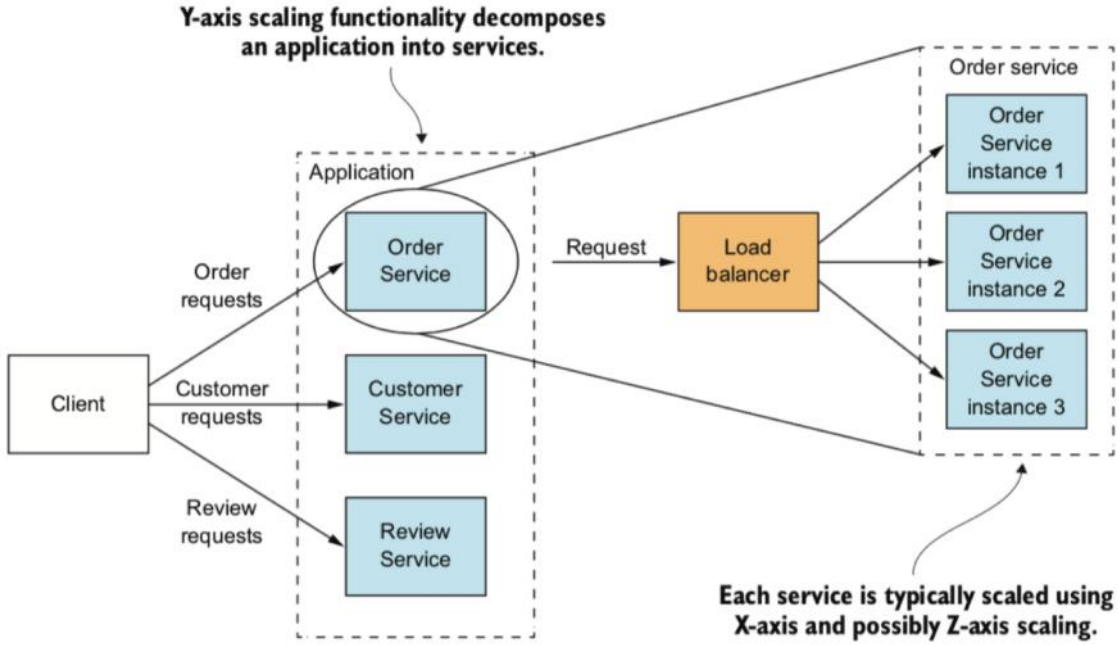

Рисунок 2.3 – Масштабування по осі Y

Це масштабування розподіляє програму на окремі сервіси. Вони відповідають за певну функцію та масштабуються по осі X і можливо, по осі Z.

Сервіс – це програма, що реалізує вузькоспеціалізовані функції. Наприклад це може бути управління клієнтами, управління запитами тощо. Сервіси масштабуються по осі X, деякі з них можуть використовувати також ось Z. Наприклад сервіс Order має декілька копій, навантаження на які балансуються.

Загальне визначення мікросервісної архітектури (мікросервісів) - це стиль проектування, що розподіляє програму на певні сервіси за різними функціями. Головне, що кожний сервіс має чіткий перелік пов’язаних між собою обов’язків.

Модульність являється невід’ємною частиною розробки компесних та складних програм. Сучасні програми дуже великі для розробки наодинці та складні для того, щоб в них змогла розібратися окрема людина. Програму потрібно розбивати на модулі, які розробляють різні люди.

В монолітному проекті модулі представляють собою поєднання концепцій мов програмування, таких як пакети Java, та ресурсів, що приймають участь в зборці, такі як JAR-файли.

Одиницею модульності в мікросервісній архітектурі являється сервіс. Сервіси мають API, які служать непроникним бар’єром. На відміну від пакетів в Java API неможливо обійти, щоб звернутися до внутрішнього класу. У перспективі це набагато спрощує підтримку модульності програми. Використання сервісів має й інші переваги, наприклад, кожний з них можна розгортати та масштабувати окремо.

Головна особливість мікросервісної архітектури це те, що сервіси між собою слабко пов’язані та взаємодіють лише через API. Слабка пов’язаність досягається за рахунок виділення кожному сервісу окремої бази даних. В інтернет -магазині сервіси Order та Customer могли б мати власні бази даних

з таблицями «Замовлення» та «Клієнти» відповідно. Структуру даних сервісу, не координуючи свої дії з розробниками інших сервісів, можна змінити на етапі розробки. На етапі виконання сервіси один від одного ізольовані, відповідно жодному з них, наприклад, не доведеться чекати якщо інший сервіс заблокував БД.

Мікросервіси піддаються критиці, що в данному підході немає нічого нового. Вказувалось,що це різновидність сервіс-орієнтованої архітектури (service-oriented architecture, SOA). Але,проводячи аналіз на вищому рівні існує деяка схожість. SOA та мікросервісна архітектура – це стилі проектування, які структурують систему в перелік сервісів. Але при більш детальному розгляді можна знайти відмінності (таблиця 2.1).

Таблиця 2.1 – Порівняння SOA та мікросервісів

Параметр	SOA	Мікросервіси
Міжсервісна взаємодія	SOAP – як приклад каналів, такі як сервісна шина підприємства, з використанням великовагових протоколів	Примітивні канали, яка являє собою прямою взаємодією між сервісами за допомогою протоколів, наприклад REST або Grpc
Данні	Глобальна модель даних та загальні БД	Розподілені моделі даних та БД для кожного сервісу
Типовий сервіс	Монолітний застосунок	Сервіс невеликий за розміром

SOA і мікросервісна архітектура загалом використовують різні технології. У програмах на основі SOA використовуються великовагові стандарти веб-сервісів збіжно до SOAP. Їм часто потрібна шина підприємства (Enterprise Service Bus, ESB) – smart канал, який інтегрує сервіси завдяки бізнес-логіки та коду для обробки повідомлень. Програми, що спроектовані у виді мікросервісів, зазвичай задіють легковагові технології з відкритим вихідним кодом. Сервіси взаємодіють через примітивні канали подібні до REST чи gRPC.

SOA і мікросервісна архітектура можуть по-різному використовувати данні. Так SOA-програми як правило мають глобальну модель даних та загальну БД, а в кожного мікросервіса є особиста база даних.

Варто звернути увагу на ще одну відмінність між двома архітектурами, яка полягає в розмірі сервісів. Зазвичай SOA використовується для інтеграції великих, складних, монолітних програм. В мікросервісній архітектурі сервіси не завжди являються дрібними, але майже завжди вони виявляються набагато меншими. Таким чином мікросервісні проекти частіше усього розподілені на десятки або сотні малих сервісів, а SOA-програми складаються з декількох великих частин.

Мікросервісна архітектура має наступні переваги:

- вона робить можливими безперервну доставку та розгортання великих, складних програм;
- сервіси виходять невеликими та простими в обслуговуванні;
- сервіси розгортаються незалежно один від одного;
- сервіси масштабуються незалежно один від одного;
- мікросервісна архітектура забезпечує автономність команд розробників;
- вона дозволяє експериментувати та впроваджувати нові технології; в неї краще ізольовані несправності.

Очевидно, що ідеальних технологій не існує та в мікросервісній архітектурі теж є ряд суттєвих недоліків та проблем.

Такі є основні недоліки та проблеми мікросервісної архітектури:

- важко підібрати відповідний набір сервісів;
- складність розподілу систем ускладнює розробку, тестування та розгортання;
- розгортання функцій, що охоплюють декілька сервісів, потребує ретельної координації;

- рішення про те, коли слід переходити на мікросервісну архітектуру, є нетривіальним.

Ще один недолік мікросервісної архітектури складається в тому, що при створенні розподілених систем виникають додаткові складності для розробників. Системи повинні використовувати звичайні методи. До того ж код повинен вміти впоратися з частковими перебоями та бути готовим до недоступності або високої латентності віддаленого сервісу.

Реалізація сервісів, що охоплюють декілька сервісів, потребує застосування незнайомих технологій. Кожний сервіс має особисту базу даних, що ускладнює реалізацію комбінованих транзакцій та запитів.

2.4 Процес переходу на мікросервіси

Перетворення монолітної програми в мікросервіс являється різновидом модернізації програмного забезпечення (ПЗ). Модернізація ПЗ включає в себе процес перекладу застарілого коду на нові архітектуру і стек технологію. Розробники десятиліттями займаються модернізацією своїх застосунків, а отриманий досвід, який вони накопичують за цей час, можна застосувати для міграції на мікросервісну архітектуру. Самий важливий урок полягає в тому, що не варто переписувати проект з нуля.

Є декілька проблем при переході на нову архітектуру. Перша – це зв’язність бізнес-логіки. При переході на мікросервіси це стає проблемою: увесь код досить жорстко пов’язаний, і складно розділити сервіси. Друга –це необіхдніть в додатковій кількості розробиків, які будуть інтегрувати нові технології та зв’язки між сервісами програми.

Замість того щоб починати з чистого аркуша, можна поступово трансформувати свій монолітний проект (рисунок 2.4), побудувавши нову програму. Вона складається з мікросервісів, які будуть працювати у зв’язці з монолітним кодом. З часом монолітна програма буде реалізовувати все

менше функцій до тих пір, поки повністю не зникне або ж не перетвориться в ще один мікросервіс. Це менш ризиковано, ніж спроба переписування з нуля.

Мартін Фаулер визначає цю стратегію як модернізація шаблоном «задушливої» програми.

Важлива перевага поступового переходу на мікросервісну архітектуру полягає в тому, що ви відразу ж бачите плоди своєї роботи. Це сильно відрізняється від переписування з нуля, про користь якого можна судити тільки по його закінченні. При поступовому рефакторінгу моноліту кожен новий сервіс можна писати за допомогою нового стека технологій і сучасного високошвидкісного процесу розробки і розгортання. Завдяки цьому ваша команда буде доставляти свій код все швидше.

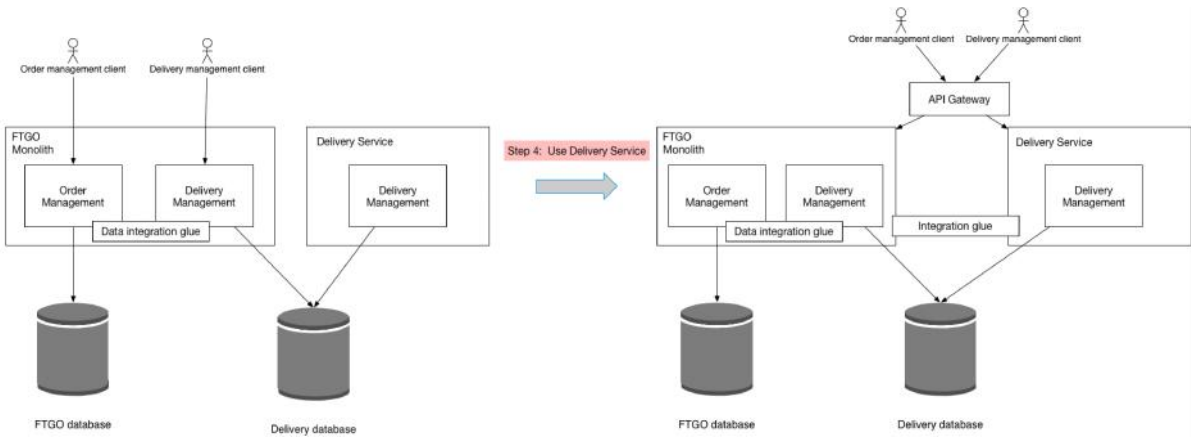

Рисунок 2.4 – Трансформація монолітної програми в мікросервісну

Єдине, без чого не можна обійтися - це процес розгортання з автоматичним тестуванням. Наприклад, якщо у вас всього кілька сервісів, вам не потрібні розвинені інструменти для розгортання та забезпечення спостережливості. З самого початку для пошуку послуг можна використовувати конфігурацію, вбудовану в вихідний код.

Існує три основні стратегії для «удушення» моноліту і поступової заміни його мікросервісамі:

а) реалізація нових можливостей у вигляді сервісів;

б) поділ рівня уявлення і внутрішніх компонентів;

в) розбиття моноліту шляхом оформлення функціональності у вигляді сервісів.

Перша стратегія запобігає подальший розвиток моноліту. Вона дозволяє швидко продемонструвати вигоду від використання мікросервісов, допомагаючи заручитися підтримкою керівництва. Реалізація нових можливостей у вигляді сервісів прискорює їх розробку. Це хороший спосіб швидко продемонструвати переваги мікросервісной архітектури. До того ж це уповільнює темпи розвитку моноліту. Дві інші стратегії розбивають моноліт на частини. Другий підхід може стати корисним у процесі рефакторінга, а без третього ви точно не обійдетеся, оскільки саме так функціональність переноситься з моноліту в «задушливу» програму.

Альтернативна стратегія більше піклується про планування: модулям програми призначається рейтинг відповідно до того, яку користь планується отримати від їх вилучення. Існує дві причини, чому витяг сервісу може бути корисним:

а) прискорення розробки. Її прискорення шляхом можна збільшити через вилучення її в сервіс, якщо згідно з планом якась частина програми буде активно розвиватися протягом наступного року;

б) рішення проблем з продуктивністю, масштабованість та надійністю. Якщо певна частина вашого застосування ненадійна або має проблеми з продуктивністю або масштабністю, буде корисно перетворити її в сервіс.

3 FRONT END ЧАСТИНА ЗАСТОСУНКУ

3.1 Поняття Front-end розробки

Даний напрям займається питаннями, які стосуються клієнтського інтерфейсу. Абсолютно все, що користувач бачить (шрифт, оформлення тощо) на моніторі та з чим він може взаємодіяти, відноситься до області впливу frontend. Організація роботи сервера, реалізація логіки веб- продукту, рішення інших завдань, які приховані від очей юзера, - сфера компетенції бекенд. Щоб додаток функціонував ефективно, важливо забезпечити грамотний розподіл функцій усередині команди з урахуванням специфіки кожного напряму. Розглянемо аспекти які включать в себе взаємодію з користувачем веб-застосунку:

- створення естетично привабливої картинки, де кожен елемент знаходиться на своєму місці;
- розробка простого, зрозумілого кінцевому користувачеві функціоналу для реалізації призначення додатка;
- отримання клієнтських запитів з відправкою у бекенд і формуванням відповідного відгуку.

В розробці веб-застосунків front-end і back-end визначаються як розділення відповідності між рівнем презентації (front-end) та бізнес логікою (back-end) веб-застосунку або фізичною інфраструктурою таобладнанням. Зазвича, моделю клієнт-сервер клієнтом вважають front-end, а сервер вважається backend. Відмітимо особливості архітектури front-end які необхідно обговорити [12; 13].

UX (User Experience) являється важливим фактором при виборі продукту користувачами. Саме для кроссплатформеної розробки стек JavaScript технологій використовується (один і той же веб-застосунок без

перешкод запускається як на Android, так на iOS і в браузері).

В ході роботи frontend- програміст стикається з необхідністю співпраці з фахівцями різних галузей. Текстовий, графічний контент, верстання і фронтенд нерозривно пов’язані. Тому він повинен взаємодіяти з копирайтерами, дизайнерами, маркетологами, прагнучи грамотно об’єднати усі блоки в єдине ціле і змусити їх злагоджено працювати. Від цього значною мірою залежить комерційний успіх проекту.

Проте головні інструменти, які потрібні йому в роботі, зводяться до трьох позицій:

- HTML;
- CSS;
- JavaScript.

Інструмент, що дозволяє зробити взаємодію з користувачем живим, динамічним. Ця мова програмування обробляє інформацію, що поступає від юзера :

- кліки мишею;
- натиснення клавіш;
- переміщення курсора;
- передає запити на сервер.

Фронтенд це у тому числі дизайн сайту, тому універсальний код, використовуваний для зовнішнього відображення контенту, потрібний. Він відповідає за формування розміру, кольору, стилю шрифту, створення фону, розміщення блоків на сторінці, переформатування документів для друкарської версії або для читання.

Застосована мова розмітки, яка дозволяє сформувати правильне верстання на сайті. З його допомогою текстовий контент розбивається на розділи, абзаци, списки, в структуру сторінки впроваджуються таблиці, малюнки, графіки, заголовки і так далі. Правильне застосування HTML дасть можливість кожному елементу знаходитися в потрібному місці для

органічного сприйняття користувачем.

3.2. Підходи та вимоги до Front-end розробки

В сучасному підході до створення інтерфейсу для кінцевого користувача - front-end, як і в цілому кожну структуру майже будь-якого забезпечення, представляється у вигляді ієрархічної структури. На верхньому рівні завжди знаходяться блоки, кожен з яких має якусь свою реалізацію, яка дозволяє створювати взаємодію з іншими блоками. В кожній з них виділяються модулі, які, в свою чергу, складаються з методів. Кожна частина данної ієрархії має свій спосіб коммунікації та взаємодії, представлені як вхідні і вихідні параметри веб-застосунку.

Сучасна архітектура Front-end базується на декількох принципах [12; 14]:

- потік даних займає центральне місце в архітектурі;
- компонентний підхід;
- архітектура на основі компонентів.

Масштаби та складність середовища, в якій виконується front-end, бізнес логіка визначається як ключовою. За данними, більша частина розробки витрачається на пошук необхідної інформації та рефакторинг коду. В даному випадку, важливою умовою для будь-якої архітектури є знаходження необхыдного нам коду та його саме швидкість його знаходження. Саме необхідне – це знайти відповідальну частину бізнес логіки. Надзвичайно важливо розуміти, вносячи якусь нову функціональність в систему, необхыжно прогнозувати як вона вплине на майбутню працездатність та безпосередню ефективність програмного забезпечення.

Розуміння на кожному етапі механізму переміщення даних по архітектурі програмного забезпення, являється єдиним спосібом для здійснення такого роду завдання. Тому, побудування любого Front-end фраємворку сьогодні базується на цьому принципі, на чіткому розподілі на

модуль State (зберігання і маніпуляція з даними) і на модуль View (відображення даних).

В приклад можемо привести такий фреймворк як Angular I який використовує принцип двонаправленого потоку даних, що створює неконтрольовані процеси при передачі даних. Поява патерну проектування Flux, зумовила використання більш надійного принципу односпрямованого потоку даних. По таким причиам, був створений Angular II, який коригувався як раз принципом односпрямованністю, як Flux.

Компонентний підхід є чітким наслідком першого пункту про потік даних. Традиційні архітектури front end застосунків спрямовані на горизонтальний розподіл функціональних забов’язань. Будь- який блок коду, що інкапсулює якусь одну "pure responsibility" логіку, може бути як причиною, що призвела врахування першого принципу.

Елементи тіла документу призначені для управління відображенням інформації в програмі інтерфейсу користувача. В данному випадку, компонентний підхід дозволяє перевикористовувати і тестувати окремі блоки веб-застосунку, не призводивши до видозмін його архітектури.

Беручи до уваги сучасні підходи, компоненти повинні бути:

- незалежні;
- слабкозв’язані;
- перевикористовані.

Елементи–контейнери ніякої дії браузеру не викликають, а використовують тільки для логічного ділення HTML документу на частини.

Компоненти можуть бути досить складні в середині, але мають бути прості зовні. Тут мається на увазі, що інтерфейс будь компоненти повинен бути настільки простим, щоб процес підключення компоненти до батьківського блоку проходив без побічних ефектів.

У контейнери (між початковим і закриваючим тегами) размещают інші елементи, якими потрібно управляти спільно. Такими елементи створюються

за допомогою тегів div і span. За допомогою тега div створюється блоковий контейнер, який розпочинається з нового рядка, і елемент, що слідує за нього, також розпочинатиметься з нового рядка. А за допомогою тега створюється рядковий елемент, який розміщується в тому ж самому рядку, що і текст, що оточує його. Основними атрибутами контейнерів є id і class. Слід пам’ятати, що рядкові контейнери можуть включати тільки інші рядкові контейнери, а блокові контейнери можуть включати як рядкові, так і блокові контейнери. Елементи-контейнери дозволяють застосовувати правила каскадних таблиць стилів(CSS) до вмісту HTML документу.

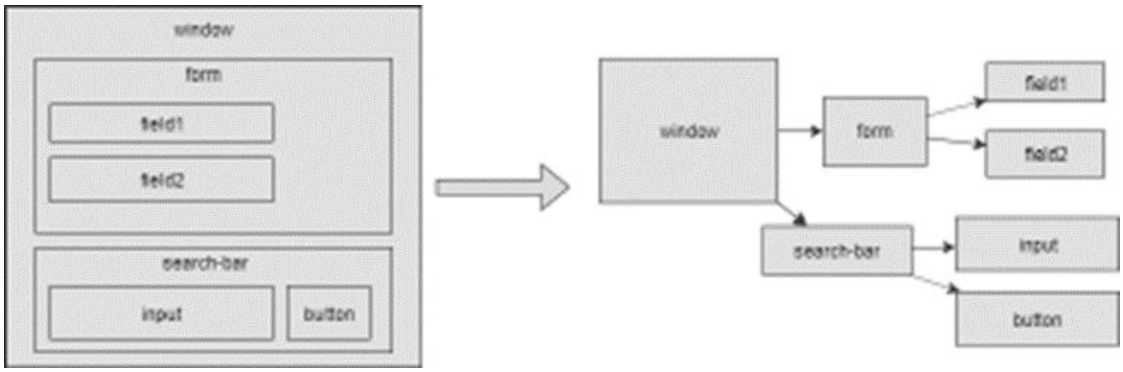

Рисунок 3.1 – Приклад декомпозиції View

Певне оновлення DOM є досить складним за участі ресурсів та часу. Тому в такий момент Frontend комьюніті дійшло до ідеї створення певної технології, яка б виконувала мінімальну для DOM кількість дій по переутворенні певних конкретних DOM елементів при зміні даних веб-застосунку, в певний State. Результатом таких змін стала поява певних підходів, які буде розглянуто далі.

Проблеми взаємозв’язку даних (Model) і уявлення (View), являються основними для DOM структури. Тому, Data-binding між Model і View у поєднанні з компонентним підходом використання є ідеальним вирішенням проблеми взаємозв’язку даних (Model) і уявлення (View). View можна уявити

функцією моделі, яка являється саме View та поля даних, які потрібні їй для відтворення необхідних їй компонентів. Як приклад, фреймворк Angular використовує низку шаблонів, в React JSX – мова, що є сумішшю HTML і JavaScript мов.

Одна з вимог є можливість архітектурного рішень будь яким чином стежити за тим, які поля стату інтерфейсу були оновлені, видалені чи додані – саме це включає в себе оповіщення про зміну даних (Change detection). Перевірка в нескінченному циклі через певний період часу, які поля були змінені, представлена як можливість в Angular і реалізована через метод брудної 20 перевірки (dirty checking). Тому, оновлення необхідних полів стану інтерфейсу у фреймворку React відбувається через незмінні структури даних і за допомогою подієвої шини відбувається оновлення. В майбутньому Vue3 планується використання об’єкта певного слою (proxy) для реалізації data checking (перевірка даних на зміну).

Оновлення DOM. Перемальовування DOM можливе після виявлення зміни стану, на основі яких необхідно виконати певні дії. Саме для цього у React використовується технологія VirtualDOM.

Отже, ключовими вимогами для розробки front-end веб-застосунку є можливість додавання нових функції з мінімальними впливами або зовсім без впливів на його внутрішню структуру інтерфейсу та потік даних [14; 15].

Модифікованість, ремонтопридатність та масштабованість – основні якості розширюваної програмної системи.

Поділ функціональної архітектури відбувається таким чином, щоб зміна системи передбачала би найменшу кількість змін в найменшій кількості можливих елементів визначається як модифікованість.

Ремонтопридатність, що визначено як термін Соммервіллом, полягає в створенні системи, якій необхідно враховувати введення нових додаткових вимог без ризику додавання новостворенних помилок.

Масштабованість, що являється здатністю системи до розширення у

вибраному режимі, без великих змін в її архітектурі. Масштабована система може обробляти більше даних з мінімальним впливом на продуктивність.

Модульність, тобто поділ програмної системи на кілька незалежних модулів, які можуть самостійно виконувати одне чи багато завдань. Ці модулі можуть функціювати як основні елементи для всього ПЗ. Модулі створюються з окремих задач. Переваги модуляції включають покращену ремонтопридатність та функціональність програмного забезпечення модулів, бажаний рівень абстракції, високу згуртованість, багаторазове використання та підвищену безпеку.

Архітектура на основі компонентів – це функціонал в якому кожен компонент має необхідність створення для взаємодії з іншими компонентами та їх повторне використання у веб - застосунку.

Формування даних вимог створювались саме з точки зору розробників та внутрішнього влаштування веб-застосунку. І що важливо, front-end представляє собою частину ПЗ, яку бачить користувач. Саме через неї він отримує уявлення про роботу данного застосунку, тому до цієї частини ще висуваються окремі вимоги.

Правильна структуризація веб-сторінки. (X)HTML є основою вебсайтів, на основі якої здійснюється пошукова оптимізація, тому життєва важливо встановити правильну структуру документа для класів та ідентифікаторів, які забезпечать стиль та взаємодію.

Саме погана семантична структура веб-сторінки досить часто є джерелом багатьох помилок в роботі front-end.

Унікальний візуальний стиль веб-сторінок, який також впливає на структурованість – є важливим аспектом стилю є перевірка в декількох браузерах та написання стислого, лаконічного коду, який є специфічним, але загальним і одночасно відображає якомога більше рендерів.

Кросплатформеність. Необхідно передбачати можливість того, що користувачі будуть використовувати різні браузери та різні комп’ютери, що

може суттєво вплинути на відображення веб-сторінок.

Юзабіліті – характеристика, яка оцінює взаємодію програмного забезпечення з користувачем. Взаємодія застосунка з людиною повинна бути зрозумілою, зручною, достатньо простою для сприйняття.

Продуктивність. Для створення веб-застосунків, що швидко завантажуються, розмітка, стилі та JavaScript повинні бути масштабованими. Необхідно використовувати скорочувати розміри сторінок там, же це можливо.

Виходячи з цього front-end застосунок повинен відповідати таким критеріям:

- у веб-застосунку має використовуватися лише попередньо визначена кількість загальних макетів;
- для кожного випадку використання, вже використовувані елементи взаємодії з користувачем повинні бути однаковими;
- у кожному випадку використання, розміщення елементів взаємодії з користувачем має бути однаковим;
- розміри елементів інтерфейсу користувача повинні бути однаковими для кожного випадку використання;
- стан та відгуки про взаємодію користувачів повинні відображатися однаково для кожного випадку використання;
- реалізація необхідна для усіх використовуваних елементів користувача, використовуючи лише попередньо визначений колір схем.

3.3 Огляд сучасних систем Front-end розробки

HTML, CSS, JavaScript – залишається основним набором при розробці front-end сторони веб-застосунку.

Markup Language. Розмітка документа (веб-сторінки) побудована саме на Markup Language. Також, вона неправомірно використовується для

управління способом відображення вмісту веб-сторінок на екрані монітора або при виведенні на принтер, що суттєво порушує певні вимоги до веб-застосунків [16].

До вагомого збільшення розміру підсумкового документа призводить застосування складного форматування до створення html-документа, хоча для написання досить простої сторінки HTML вистачає. Саме в даних випадках починають використовувати CSS – Cascading Style Sheets (каскадні таблиці стилів). В цілому, це окремий файл, що виконує роль певного шаблону, застосовуваного для форматування тексту, таблиць та інших елементів в документі HTML. Фізичний файл CSS можна коннектити до різних сторінок веб-застосунку. Оскільки команди CSS виконуються безпосередньо на комп'ютері користувача, то CSS його можна використовувати на сервері без обмежень.

До основних переваг використання HTML можна виднести простоту засвоєння та широкі можливості вибору низки програмних редакторів для написання коду. До значного збільшення розміру підсумкового документа призводить саме застосування складного форматування до створення html-документу, хоча створюється досить проста сторінка HTML. Тому в таких випадках краще застосовувати CSS – Cascading Style Sheets (каскадні таблиці стилів). В цьому випадку, за допомогою спеціальної макромови один раз виставляються зміни сторінки, що цілком може бути окремим файлом.

Але, для сторінок та якісної картинки не вистачає динаміки (випадаючі меню, анімація). Для цього використовують мови написання скриптів. Однією з стандартних мов є JavaScript.

JavaScript являється мовою програмування, для використання в складі сторінок HTML для створення більших можливостей. Була розроблена фірмою Netscape від великої корпорації Sun. Ця мова значно розширює можливості html-документа, створеного з використанням цієї технології. Цікаво, що розширення можливостей віддбувається у вигляді додання

декількох рядків коду JavaScript, що інтегрується в файл HTML.

Інтерпретатор JavaScript, вбудований до браузера, сприймає і скрипт і сам HTML-код як єдиний документ, обробляючи і ті, і інші дані одночасно. При налаштуванні та використанні технології JavaScript не вимагається встановлення та налаштування на сервері будь-яких додаткових модулів.

Веб-фреймворк – інструмент та технологія, що значно полегшує процес створення, рефакторингу та запуску веб-застосунків.

Для полегшення створення та комбінації різних частин великого веб-застосунку, фреймворк може містити в собі також допоміжні програми, деякі бібліотеки коду, скрипти та загалом все, що полегшує створення. Що цікаво, не обов'язково при створенні об'ємного програмного продукту. Побудова кінцевого продукту відбувається саме на основі конкретного API.

Наявність стандартної структури представлена як одна з основних переваг при використанні фреймворків.

Бібліотеки являються набором попередньо написаних фрагментів коду, що застосовуються для реалізації та створення основних особливостей JavaScript. Фрагмент можна легко інтегрувати в існуючий код проекту за необхідності.

Отже, бібліотеки не являються універсальною машиною для створення усього веб – застосунку. Він представлений як спеціалізований інструмент для конкретних потреб кодування.

Angular – фреймворк призначений для створення динамічних (dynamic), односторінкових та прогресивних веб-застосунків (SPA), він сумісний з більшістю поширених редакторів коду. Отримавши найвищу оцінку за свою здатність перелаштовувати документи на базі HTML в динамічний контент, при чому тільки після свого випуску на ринок.

Vue.js додаткова структура з відкритим кодом для SPA. Він дозволяє приєднувати компоненти до проекту та використовувати модель розробки на основі компонентів. Vue.js показовий приклад бібліотеки, яка по більшій мірі

є фреймворком. Абсолютною вимогою до застосуванню Vue.js - це знання HTML та CSS. В свою чергу, він пропонує не малу кількість шаблонів проектування. Vue визнається розміромбез великої структури документів та синтаксисом на основі HTML.

Ember.js є основою для SPA, мобільних та настільних застосунків. Він використовує шаблон model-view-view-model (MVVM). У загальному розумінні це JS-каркас, який працює за MVC-шаблоном розподілу коду. При цьому Ember.js легко інтегрується і може працювати з бібліотеками Handlebars і jQuery.

Bootstrap Framework, який є вільним набором інструментів для створення сайтів і веб-застосунків. Він включає в себе HTML і CSS шаблони оформлення для типографіки, веб-форм, кнопок, міток, блоків навігації та інших компонентів веб-інтерфейсу, включаючи JavaScript-розширення.

Bootstrap використовує найсучасніші напрацювання в області CSS і HTML, тому часто необхідно бути уважним при підтримці старих браузерів які не підтримують деякі функції [16].

Основною перевагою Bootstrap є економія часу при розробці, оскільки він дає змогу використовувати шаблони і класи дизайну. Масштабування в динамічних макетах Bootstrap відбувається для різних пристроїв та роздільних здатностей екрану без будь-яких змін в розмітці. Це скорочує час оптимізації проекту.

Для всіх компонентів використовується єдиний стиль і шаблони за допомогою центральної бібліотеки. Дизайн і макети веб-сторінок узгоджуються один з одним.

Mozilla Firefox, Google Chrome, Safari, Internet Explorer, Microsoft Edge і Opera – це ті браузери на яких Bootstrap коректно відображається. Це визначається його високою сумісністю з усіма основними браузерами.

Особливістю Twitter Bootstrap є відкритий вихідних код і безкоштовне розповсюдження.

React – це бібліотека з відкритим кодом для створення динамічних інтерфейсів користувача, створена та розроблена Facebook. Вона застосовується для створення веб-застосунків з декількома динамічними компонентами. Він в його основі лежить JavaScript та JSX, розширення мови PHP у Facebook. React дозволяє будувати багаторазові HTML-елементи для front-end. До React також входить React Native, спеціалізована міжплатформена система мобільного розвитку.

React характеризується простотою в освоєнні, лаконічністю синтаксису, можливістю створення і використання VirtualDOM, за допомогою якого розвантажуються високонавантажені застосунки. За допомогою React розробники створюють окремі компоненти, здатні до переносу з одного проекту в інший.

jQuery, у свою чергу, спрямований на контроль HTML-документів. Він має простий API для управління подіями та дизайном анімації в браузерах. Окрім цього, jQuery застосовується для маніпулювання DOM і також служить інструментом розробки плагінів. Він також оснащений більш легкою бібліотекою крос-браузерів, інтерфейсом jQuery. Це надає можливість використовувати дану технологію для мобільної основи jQuery Mobile та для побудови графічного інтерфейсу. Воно являється важливим критерієм при створенні веб-застосунків в данний час.

Даний фреймворк дозволяє ефективно і зручно працювати з будь-яким з елементів DOM, подіями, використовувати технологію Ajax, створювати складні візуальні ефекти і завжди мати під рукою величезну кількість JS-плагінів для створення користувацьких інтерфейсів дозволяє JavaScript-бібліотека jQuery. За допомогою даного фреймворка веб-розробникам вдається надати сайту динамічність.

D3.js - бібліотека, керована даними для візуалізації даних. Прив’язуючи тимчасові дані до DOM та здійснюючи керовані даними зміни в документі, бібліотека дозволяє керувати даними та робити динамічні візуалізації даних.

Він може підтримувати та обробляти великі набори даних та динамічні відповіді на взаємодію та анімацію. Функціональний стиль D3 дозволяє повторно використовувати код і працювати з CSV та HTML.

4 БАЗИ ДАНИХ

4.1 Види баз даних

Бази даних представляють собою певний набір даних, які пов’язані спільною ознакою або властивістю, та являються впорядкованими.

В свою чергу, системою баз даних – це комп’ютерізована система зберігання записів. Тобто це бази даних в якій розглядаються декілька суб’єктів : СУБД, апаратура та користувачі.

DBMS (Database Management System) – система управління базами цих "СУБД" є організованою сукупністю програмних і лінгвістичних даних, які включають схеми, таблиці, запити, звіти, перегляди і інше. По суті DBMS - це програма, яка дозволяє взаємодіяти з базами даних як абстрактним об’єктом (без необхідності писати запити).

До основних функцій DBMS відносять:

- управління даними, таких як створення, відображення, редагування та видалення інформації;
- управління буфером оперативної пам’яті;
- управління транзакціями.

Буферізація даних в оперативній пам’яті – спосіб збільшення швидкості звернення користувача до будь-якого елементу в базі.

Транзакція – це послідовність операцій над БД. Якщо транзакція успішна – СУБД фіксує успіх (COMMIT), та змінення зберігаються. У іншому випадку (ROLLBACK) – вся транзакція відмінюється, а база повертається в стан, який був до початку виконання транзакції.

Транзакція є чіткою послідовністю операцій з використовуваною базою даних, де представлена система управління розглядається в якості єдиного цілого. Якщо транзакція успішно виконується - система фіксує зміни, які були нею зроблені. В зовнішній пам’яті або ніякі з вказаних змін не будуть

впливати на стану структури БД. Вона необхідна потрібно для того, щоб забезпечити підтримку логічної цілісності зазнаеної БД. Варто відмітити, що підтримка правильного ходу механізму транзакцій є обов’язковою умовою навіть при застосуванні розрахованих на одного користувача СУБД, призначення і функції яких значно відрізняються від інших типів систем. В свою чергу, об’єднання великої кількості даних в єдину базу дає змогу для формування безлічі варіантів групування інформації — особисті дані клієнта, історія замовлень, каталог товарів та будь-що інше.

Відзначимо, що головною перевагою БД є швидкість внесення (insert) та використання потрібної інформації. Алгоритми, які використовуються для баз даних, легко знаходять запрошені дані користувачем. Сам час знаходження залежить від типу бази, що ми розглянемо далі. В середньому, для пошуку необхідно всього за декілька секунд. В базі, як і у багатьох структурах інформації даних, існує певний взаємозв’язок інформації: зміна в одному рядку може спричинити зміни в інших рядках. Саме це допомагає працювати з інформацією з більшою швидкість та меншим часом відгуку программи[20].

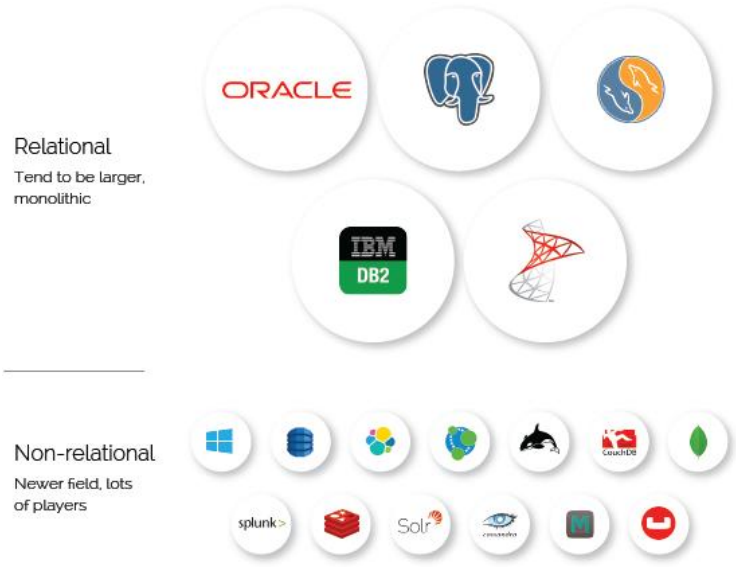

Рисунок 4.1 – Різновиди баз даних

Наразі, при створенні веб-застосунків розглядають 2 основних типів баз даних: реляційні та нереляційні бази. На рисунку 4.1 представлені основні представники данних типів. Підхід у цих базах – один, але компанії ефективно оптимізували свій продукт. Вибір самих баз переважно залежить від особистих переваг розробника та проектної команди.

Ієрархічна база даних (рисунок 4.2) – вид бази, де дані представлені у структурі, що нагадує дерево. Тобто у сутності можуть бути батьківські та дочірні елементи. Але об’єкт з якого все починається завжди один. Як приклад - файлова система комп’ютера. Вона являється чітким приклад даної бази. Бази такого типу оптимізовані під зчитування, тобто вміють ефективно знаходити (search) та вибирати необхідні дані.

Ієрархічна модель бази даних повинна мати лише одного з батьків для кожного дочірнього вузла, але батьківські вузли можуть мати більше однієї дитини. Багато батьків не допускаються. Це головна різниця між ієрархічною та мережевою моделлю баз даних. Розглядаючи дерево, перший вузол данної структури називається кореневим вузлом. Коли потрібно отримати дані, тоді все дерево переміщується, починаючи з кореневого вузла. Представлена модель представляє собою відносини один до багатьох.

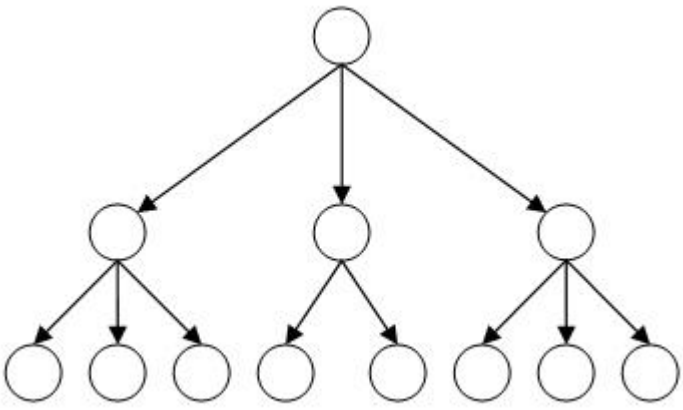

Рисунок 4.2 – Структура ієрархічної БД

На рисунку 4.2 можна побачити, що з самого зверху знаходиться «батько або кореневий елемент», а нижче «дочірні» елементи. Елементи, що знаходяться на одному рівні – «сусіди».

Перевагами ієрархічної бази є:

- дані шкидко отримуються за рахунок явних та ієрархічних зв’язків;
- чітка лінія між командуванням та повноваженями в самій базі.

Недоліки такої бази:

- певні предметні області не підпадають під представлення в даній структурі;
- зміна модельної структури тягне за собою зміну програмної релазіції у веб-застосунку.

Розглянемо, модифікацію ієрархічної моделі. В мережних базах, у порівнянні з ієрархічною, може бути як декілька дочірніх, так і декілька елементів вище по ієрархії [17].

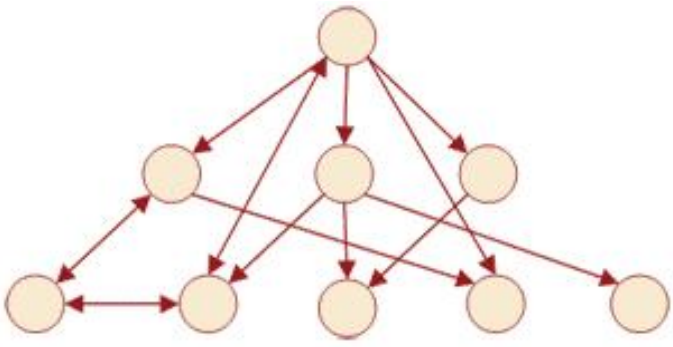

Рисунок 4.3 – Структура мережної БД

Переваги мережної БД:

- легкість реалізації. Усі зв’язки визначаються на етапі проектування;
- швидкий доступ до данних БД;
- ефективна реалізація за показниками витрат пам’яті і оперативності.

Недоліками мережної БД є:

- потрібна зміна ПЗ, при зміні інформації;
- важкість варіантів взаємозв’язку та особливостей реалізації.

Прикладами мережних БД являються:

- Norsk-Data SYBAS;
- NCR IDM-9000.

Реляційна база даних - це сукупність відносин, що містять усю інформацію, яка повинна зберігатися у базі даних. Таким чином, реляційна база даних являє собою перелік таблиць, необхідних для зберігання всіх даних. Таблиці реляційних баз даних логічно пов’язані між собою. Записи в таблицях не повторюються. Унікальність записів забезпечується завдяки первинному ключу. Він містить в собі набір полів, які однозначно визначаючих запис. Як приклад, розглянемо на рисунку 4.4, взаємозв’язок таблиці предметів які викладються в університеті та студентами.

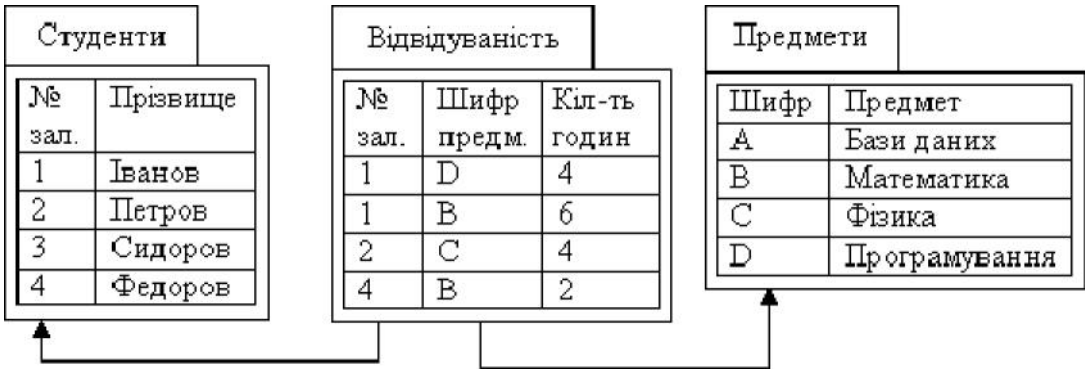

Рисунок 4.4 – Структура реляційної БД

Саме зберігання об’єктів всередині даних баз даних відображаються та представлені у вигляді двовимірних таблиць. Додатковою особливістю являється те, що кількість стовбців фіксована. Таким чином, структура відома заздалегідь, але число рядків в реляційних базах не обмежена. Таблиця – це, всього на всього, засіб збереження інформації, набір таблиць може бути пов’язаний логічно і цей набір називають база даних. А завдяки такому поняттю, як СУБД – система управління базами даних - ми і управляємо цими самими таблицями.

Проектування схеми БД повинно вирішувати завдання мінімізації дублювання даних, спрощення та безпосереднє прискорення процедур їх обробки та оновлення. При неправильно спроектованої схемою БД можуть виникнути аномалії модифікації даних. При виникненні подібних ситуації, рішенням являється нормалізація відносин.

Однак у технології роботи з сховищами даних може використовуватися зворотний прийом - денормалізація відносин з метою збільшення швидкості виконання запитів до дуже великим обсягам архівних даних. У такій великій структурі все ж таки можна визначити переваги:

- зручність та простота відображення інформації;
- лаконічне відображення данних;
- незалежність даних, завдяки якої при змінах в структурі бази, це призводить до мінімальних змін в прикладних програмах.

Недоліки, на які необхідно звернути увагу:

- низька швидкість в операції з’єднання;
- важкість розробки архітектури;
- великий об’єм займаної пам’яті.

Прикладами РСУБД є:

- MySQL;
- Microsoft SQL Server;
- SQLite
- Oracle 12c;
- PostgreSQL.

4.2 Бази даних типу NoSQL

NoSQL бази даних – це підхід до баз даних, де збереження інформації з гнучкою моделлю даних. В данному випадку, вона не являється структурованою в табличному вигляді.

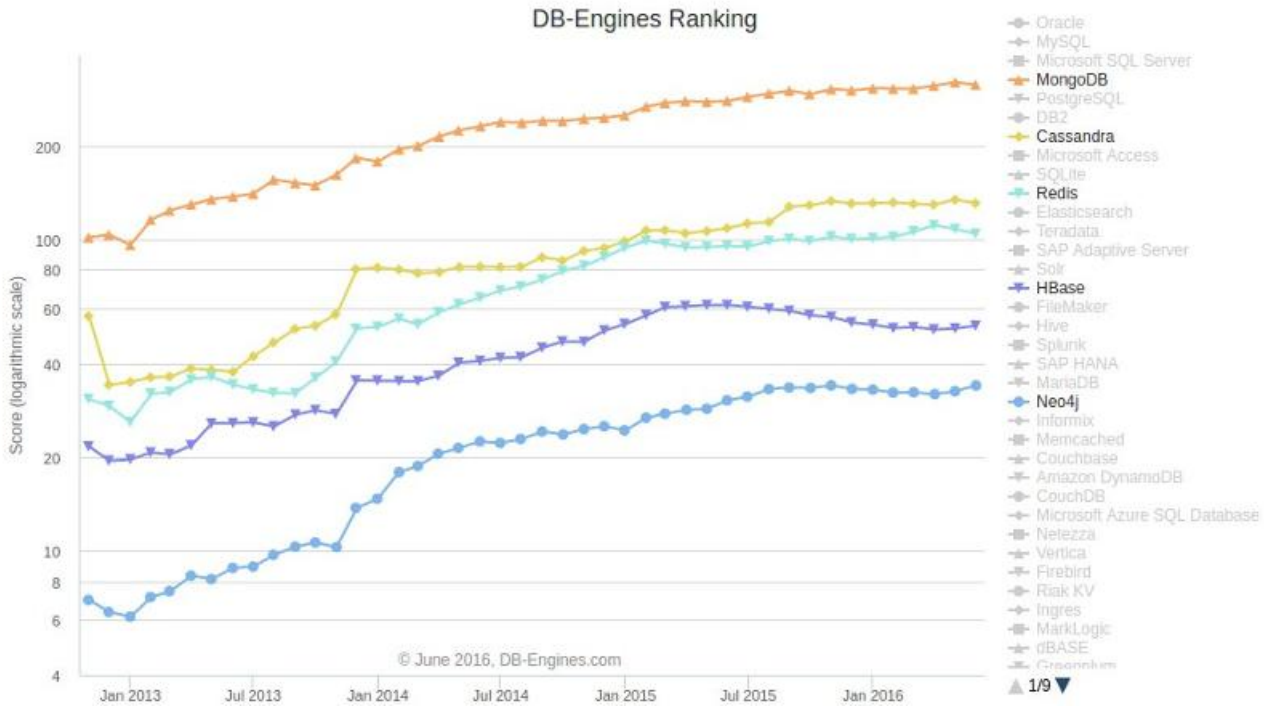

Рисунок 4.5 – Діаграма популярності NoSQL баз

Найвідоміші можна побачити на рисунку 4.6.

Рисунок 4.6 – Перелік нереляційних БД

Називаються вони так не тільки тому що, відрізняються від реляційних підходів до проектування. Дана база зберігає дані як ASCII-файли. А самі

файли, замість SQL-запитів використовувались shell-скрипти, які являється частиною командного інтерпретатору та мають розширеня «*.sh».

Саме вони використовуються в Unix системах. Це зумовило активний зріст, який можна побачити на рис. 4.5. Використання таких баз, а виробники почали розробляти свої особисті, і у наш час їх існує дуже багато. Також, бази даних NoSQL базуються на документах, парах ключ-значення, графіках або стовпцях.

На сьогоднішній день існує чотири основних типи NoSQL СУБД:

- сховище «ключ-значення» або хеш-таблиця, що містить ключ та його значення для кожної пари інформації. Наприклад: Riak, Amazon DynamoDB, Oracle NoSQL Database, Berkeley DB, MemcacheDB;

- колонне сховище – інформація з кожної колонки зберігається у кожному блоці. Наприклад: Google Big Table, HBase, Cassandra, ScyllaDB, Apache Accumulo;

- сховище на основі графів - мережева база даних, яка для збереження та відображення інформації використовує ребра та вузли. Наприклад: InfoGrid, Neo4j, AllegroGraph, Blazegraph, Amazon Neptune, OrientDB, InfiniteGraph та інші;

- документо-орієнтоване сховище – збереження інформації досягається завдяки тегованим елементам. Вони корисні для управління вмістом та обробки даних мобільних застосунків. Вони широко використовуються разом з JSON та JavaScript. Приклад: CouchDB, Couchbase, MongoDB, eXist, Berkeley DB XML.

Розглядаючи базу даних типу «ключ-значення», варто звернути увагу на те, що ключ може бути синтетичним або таким, що автоматично генерувався, а значення може бути представлено або рядком, або JSON, або BLOB (Binary Large Object, великий двійковий об’єкт) і т.д. Такі бази даних використовують хеш-таблицю, в якій знаходиться унікальний ключ і будь-яке значення прив’язне до цього ключа.

Також існує поняття блоку (bucket) – це логічна група ключів, що не

групують дані фізично. Блоки між собою не пов’язані, тому у різні блоки можуть містити ідентичні ключі.

Продуктивність піднімається за рахунок кешируємих механізмів, що працюють на основі маппінгу. Тому, щоб прочитати значення, потрібно знати як ключ, так і блок, оскільки насправді ключ є хеш-кодом (блок + ключ). Модель «ключ-значення» не містить нічого складного, так як реалізувати її простіше простого [18].

5 МЕТОДИ ТЕСТУВАННЯ ТА ОЦІНКИ ПРОДУКТИВНОСТІ

5.1 Тестування веб-застосунку

Веб тестування – це спосіб перевірити, чи відповідає фактичний програмний продукт певним вимогам, забезпечити відсутність дефектів перед користування кінцевими користувачами. Він включає в себе виконання програмних та системних компонентів за допомогою мануальних або автоматизованих інструментів для оцінки працездатності.

В даному розділі ми розглянемо чотири рівня тестування, які допомагають перевірити наявну поведінку та ефективність тестування програмного забезпечення. Вони зображені на рисунку 5.1.

Рисунок 5.1 – Рівні тестування

5.2 Види тестування

Систематичне тестування застосування, що розробляється, завдяки доступності концептуальної моделі і можливості перетворення моделі для генерування коду. Основна увага перенесена з перевірки конкретних web додатків до оцінки правильності роботи генерації коду. Якщо можна гарантувати, що генератор коду створює правильну реалізацію для усіх правильно складених концептуальних схем (тобто, комбінацій моделюючих конструкцій), що мають сенс, то тоді тестування web застосувань зводиться до більше вирішуваного завдання перевірки правильності опису концептуальних схем.

Модульне тестування, або юніт-тестування (англ. unit testing) – це найменша частина тестування системи або програми, яку можна скомпілювати, завантажити та виконати. Цей вид тестування допомагає протестувати кожен модуль окремо. Метою даного рівня є тестування кожної частини ПЗ, відокремивши її. Також, існує модель для автоматичної оцінки якості:

- статичний аналіз концептуальних схем(виконується під час компіляції). Грунтується на виявленні в концептуальній схемі шаблонів проектування і на їх автоматичній оцінці відносно атрибутів якості записаних у вигляді правил;

- динамічний (під час виконання) збір даних про використання веб-застосунку для автоматичного аналізу та порівнянн з навігацією заданою концептуальною схемою. Він заключається в автоматичній перевірці та аналізі вдосконаленого web- журналу, який доповнює звичайний HTTP журнал.

Системне тестування визначається як тестування системи яке проводиться на повній та інтегрованій системі. Воно дозволяє перевірити відповідність системи та співвіднести з вимогами до веб-застосунку. Перевіряється загальна взаємодія компонентів та передбачає тестування

навантаження, продуктивності, надійності та безпеки. Системне тестування використовується як фінальне тестування для перевірки відповідності системи специфікації. Воно оцінює як функціональну, так інефункціональну потребу в тестуванні [19]:

- вихідні посилання;
- внутрішні посилання;
- якірні посилання;
- тестові форми.

Функціональні тести базуються на функціях і особливостях, а також взаємодії з іншими системами, і можуть бути представлені на всіх рівнях тестування: компонентному або модульному (Component / Unit testing), інтеграційному (Integration testing), системному (System testing) і приймальному (Acceptance testing).

Тестові форми включають:

- перевірку сценаріїв форм, чи працюють належним чином.
- перевірку, чи заповнюються значення за замовчуванням;
- після подання, дані у формах подаються до бази даних або пов’язуються з діючою електронною адресою;
- форми оптимально відформатовані для кращої читабельності. Тестові файли cookie (сеанси) видаляються або після очищення кеш-пам’яті, або після закінчення терміну їх дії.

Надалі, розглянемо процесс та особливості перевірки практичності. Це процес, за допомогою якого вимірюються необхідні характеристики взаємодії людини і комп’ютера в системі та визначаються слабкі місця для виправлення:

- простота використовування веб-застосунку;
- навігація;
- суб’єктивне задоволення користувачів.

Тест навігації означає, як користувач переглядає веб-сторінки, різні елементи керування, такі як кнопки, рамки або як користувач використовує посилання на сторінках для перегляду різних елементів сайту.

Тестування зручності використання включає наступне:

- веб-сайт повинен бути простим у користуванні;
- надані інструкції повинні бути дуже чіткими;
- відповідність наданих інструкцій з їх призначенням.

Зміст повинен бути логічним і простим для розуміння. Зміст повинен бути осмисленим. Усі прив’язні текстові посилання повинні працювати належним чином.

При веб-тестуванні слід протестувати серверний інтерфейс. Це виконується шляхом перевірки правильності спілкування. Слід перевірити сумісність сервера із програмним, апаратним забезпеченням, мережею та базою даних. Основними інтерфейсами є:

- веб-сервер та інтерфейс сервера застосунків;
- сервер застосунків та інтерфейс сервера баз даних.

Потрібно перевірити, чи всі взаємодії між цими серверами виконуються, і помилки обробляються належним чином. Якщо база даних або веб-сервер повертає повідомлення про помилку для будь-якого запиту сервером застосунків, тоді сервер застосунків повинен ловити та відображати ці повідомлення про помилки користувачам.

Тестування на сумісність – це тестування що проводиться на основі контексту програми:

- сумісність браузера з програмним забезпеченням;
- сумісність з пристроями, такими як ноутбук, мобільний телефон тощо.

Тест на сумісність браузерів: однаковий веб-сайт у різних браузерах має відображатися по-різному. Потрібно перевірити, чи правильно відображається веб-застосунок в браузерах і чи JavaScript, AJAX автентифікація працює нормально.

Тестування продуктивності - тестування, яке з’ясовує та випробовує компоненти системи в певній ситуації, яка потенційно може статись вже при функціюванні веб-застосунку.

Його необхідно застосувати для аналізу при масштабованості веб-застосунку або для порівняння продуктивності. Данний процесс проводиться у середовищі сторонніх продуктів, наприклад як сервери та проміжне програмне забезпечення.

Навантаження на веб-застосунок виділяється по критеріям:

- кількість користувачів за раз;
- перевірка навантаженняна веб-застосунок при пікових значеннях системи та безпосередньо її поведінки;
- масштабний обсяг даних, до яких користувач має можливість отримати доступ.

Веб-застосунок повинен стримувати велике навантаження. Тестування веб-продуктивності повинно включати:

- тестування веб-навантаження;
- тестування динамічного навантаження;
- знаходження точки насичення;
- стрес тест.

Тестування веб-навантаження: потрібно чітко розуміти скільки користувачів зможуть отримати доступ до однієї сторінки, чи зможе система витримувати певний час пікового навантаження і так далі. Перелік навантажень на веб-застосунок достатньо великий: він повинен обробляти велику кількість одночасних запитів користувачів, вхідні дані від користувачів, одночасне підключення до БД тощо.

Використання програмних засобів у контексті веб розгортання являється імітацією роботи веб-застосунку при певних обставин та при певних навантаженнях на нього.

У кількісному тестуванні розглядаються низка показників - як час відгуку, тоді як якісне тестування стосується більше на данний момент масштабованості, стабільності та сумісності. Слово «продуктивність», у більшісті з людей одразу ассоціюється як визначення про швидкість. Задля перевірки часу відгуку,яким є абсолютно необхідними в наші дні. Саме це вимагає не просто перегляду всіх ваших посилань, для переконання, що вони працюють необхідним чином[19]. Ознака, що під час виробничого тестування працює справно, не являється критерієм, що це буде так, коли на веб-застосунок буде направлений трафік. Для данного тесту буде виконуватися тестування у звичайних умовах використання веб-застосунку. Для демонстрації даного прикладу, є зображення на рисунку 5.2.

Рисунок 5.2 – Тестування в звичайних умовах

Поліпшення взаємодії з користувачами та збільшення доходу може бути викликано тестуванням продуктивності веб-сайту або програми, яка в свою чергу дозволяє виявляти проблеми та покращувати загальну ефективність.Саме тестування продуктивності може виявити багато загальних проблем, наприклад, вузькі місця в программі. Переривання трафіку через обмежену потужність називається вузьким місцем. Часто, при раптовій збільшенню трафіка – і виникають описані вузькі місця можуть виникнути, наприклад, якщораптово зростає трафік, який не можуть витримати сервери. І

вона являється лише одна з багатьох проблем, яку можна обійти, коли веб-сайт не являється масштабованим.

Тестування апаратного чи програмного забезпечення на його стабільність в умовах рпевного навантаження визначається як процес стрес-тестування. Метою данного методу являється знайти числову точку, коли система буде мати поломку (з точки зору кількості запитів користувачів, серверу тощо) та чітко пов’зану з цим обробку помилок. Опишемо даний процес: веб-застосунок піддається стресу протягом певного періоду часу, щоб дізнатися про кількість навантаження,яке він може витримати. Саме використання стрес-тестування є визначає межі, при якій система, програмне чи апаратне забезпечення почне показувати некоректність своєї роботи[19].

Побачити, як система реагує на несподіване зростання та падіння навантаження кінцевого користувача являється метою динамічного навантаження. Тестування Spike допомагає визначити зменшенн продуктивності системи при раптовому великому навантаженні на неї.

Для динамічного зростання пропускної здатності сервера використовують хмарні платформи, такі як AWS, Azure. На рисунку 5.4 зображений графік тестування динамічного навантаження.

Рисунок 5.4 – Графік динамічного навантаження

Підтримання певної системи необхідним для визначення кількості користувачів та транзакцій, яке буде підтримувати та відповідати цілям ефективності.

Для ефективним управлінням майбутніми навантаженнями нам необхідно зрозуміти, скільки додаткових ресурсів (наприклад, ємність процесора, ємність диска, використання пам’яті або пропускна здатність мережі) необхідна для підтримки використання у подальшому майбутньому. Тестування на витривалість, в основному, проводиться за шляхом перевантаження даної системи, або за рахунок зменшення певних системних ресурсів та оцінки наслідків.

ВИСНОВКИ

В ході виконання робити був проведений аналіз предметної області, а саме, актуальності веб-застосунків та архітектурних рішень.

Був проаналізований ринок. Пошукові движки почали знижувати у видачі сайти, які не оптимізовані під смартфони. Тож розробка веб-застосунків нині залишається одним з затребуваніших завдань в сфері інформаційних технологій. Зростає потреба в розробці якісного і відповідного сучасним тенденціям продукту для бізнес рішень.

Був проведений порівняльний аналіз існуючих патернів для розробки веб-застосунків, а також були виявлені їх основні недоліки та переваги.

Також був проведений огляд мов програмування та баз даних для веб-застосунків.

Кожна мова програмування та бази даних мають свої переваги й недоліки, які враховуються бізнесом та виконавцем при виборі технології для розробки веб-застосунку. У числі найбільш значущих критеріїв - терміни і вартість розробки і супроводу, відповідність завданню, безпека та перспективність. Було запропоновано метод вибору технології розробки веб-застосунку оптимального для бізнесу.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Total number of Websites [Електронний ресурс] – Режим доступу до ресурсу: https://www.internetlivestats.com/total-number-of-websites/.

2. Global digital population as of January 2021 [Електронний ресурс] – Режим доступу до ресурсу: https://www.statista.com/statistics/617136/digital-population-worldwide/.

3. Why Web Performance Matters: Is Your Site Driving Customers Away? / Gomez / The Web Performance Devision of Compuware, available at: https://www.axity.com/wp-content/uploads/2020/08/201110_why_web_performance_ matters.pdf

4. Octo Perf [Еклектронний ресурс] / OctoPerf – режим доступу: www/ URL:https://octoperf.com/blog/2015/06/17/response-time-e-commerce/

5. Web server [Еклектронний ресурс] / Wikipedia – режим доступу: www/ URL: https://en.wikipedia.org/wiki/Web_server/ − 03.11.2020 р. – назв. з екрану.

6. Що таке база даних? [Електронний ресурс] – Режим доступу до ресурсу: http://apeps.kpi.ua/shco-take-basa-danykh.

7. Що таке backend та frontend? [Електронний ресурс] – Режим доступу до ресурсу: https://otus.ru/nest/post/943/.

8. Патерни та архітектура веб-додатків [Електронний ресурс] – Режим доступу до ресурсу: https://folkprog.net/patterny-i-arhitektura-veb-prilozhenij/.

9. 10 Most Popular Software Architectural Patterns [Електронний ресурс] – Режим доступу до ресурсу: https://nix-united.com/blog/10-common-software-architectural-patterns-part-1/.

10. How to Design a Web Application [Електронний ресурс] – Режим доступу до ресурсу: https://www.educative.io/blog/how-to-design-a-web-application-software-architecture-101.

11. Architectural patterns [Електронний ресурс] – Режим доступу до

ресурсу:https://www.ou.nl/documents/40554/791670/IM0203_03.pdf/30dae517-691e-b3c7-22ed-a55ad27726d6

12. Polillo R. Quality Models for Web [2.0] Sites: A Methodological Approach and a Proposal. [Электронный ресурс]. URL: http://gplsi.dlsi.ua.es/congresos/qwe11/fitxers/QWE11_Polillo.pdf. (дата звернення: 06.03.2020).

13. Sharma A., Kumar R., Grover P.S. Estimation of Quality for Software Components: An empirical approach. ACM SIGSOFT Software Engineering Notes, 2008, vol. 33, iss. 6, pp. 1–10.

14. Лавріщева К.М. Програмна інженерія.–К.– 2008.–319 с.

15. Моделі менеджменту при розробці програмних продуктів [Электроний ресурс]. URL: http://staratel.com/iso/InfTech/ DesignPO/index.html

16. Дакетт Джон. HTML и CSS. Розробка та створення веб-сайтів / Джон Дакетт [пер. с англ. М.А. Райтмана]. М.: Издательство «Э», 2013. 480 с.

17. Majors C. Database Reliability Engineering [Електронний ресурс] / Charity Majors – Режим доступу до ресурсу: https://book-erix.info/perevod-knigi-database-reliability-engineering-102764/.

18. Нереляційні бази даних NoSQL [Електронний ресурс] – Режим доступу до ресурсу: https://docs.microsoft.com/ru-ru/azure/architecture/data-guide/big-data/non-relational-data.

19. Software testing [Еклектронний ресурс] / Wikipedia – режим доступу: www/ URL:https://en.wikipedia.org/wiki/Software_testing/ − 15.09.2020 р. – назв з екрану.

20. Що таке база даних? [Електронний ресурс] – Режим доступу до ресурсу: http://apeps.kpi.ua/shco-take-basa-danykh.

21. Moskal , V., & Yeromina, N. (2021). METHODS OF DEVELOPING WEB APPLICATIONS. Збірник наукових праць SCIENTIA. вилучено із https://ojs.ukrlogos.in.ua/index.php/scientia/article/view/15870