ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНФОРМАЦІЙНИХ СИСТЕМ ТА ТЕХНОЛОГІЙ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Дослідження та порівняння методів розробки веб-додатків на Angular та React»

на здобуття освітнього ступеня магістра
зі спеціальності 126 Інформаційні системи та технології
(код, найменування спеціальності)
освітньо-професійної програми Інформаційні системи та технології
(назва)

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ _____
(підпис) *Ім'я, ПРІЗВИЩЕ здобувача*

Виконав:
здобувач вищої освіти Сергій СТАДНИК
група ІСДМ-63

Керівник: Каміла СТОРЧАК
науковий ступінь, вчене звання д.т.н., професор

Рецензент:
науковий ступінь, вчене звання _____
Ім'я, ПРІЗВИЩЕ

Київ 2025

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Інформаційних систем та технологій

Ступінь вищої освіти Магістр

Спеціальність Інформаційні системи та технології

Освітньо-професійна програма Інформаційні системи та технології

ЗАТВЕРДЖУЮ
Завідувач кафедрою ІСТ

_____ Каміла СТОРЧАК
«_____» _____ 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Стадник Сергій Петрович _____
(прізвище, ім'я, по батькові здобувача)

1. Тема кваліфікаційної роботи: Дослідження та порівняння методів розробки веб-додатків на Angular та React

керівник кваліфікаційної роботи Каміла СТОРЧАК, д.т.н., професор,
(Ім'я, ПРІЗВИЩЕ науковий ступінь, вчене звання)
затверджені наказом Державного університету інформаційно-комунікаційних технологій від «15» 10.2024р. №320

2. Строк подання кваліфікаційної роботи «27» грудня 2024р.

3. Вихідні дані до кваліфікаційної роботи: науково-технічна література, результати досліджень у сфері веб-розробки, офіційна документація Angular та React, аналітичні дані з популярності технологій.

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

Аналіз сучасних підходів до розробки веб-додатків
Дослідження та порівняння ключових особливостей Angular та React за інженерними критеріями
Аналіз бізнес-критеріїв вибору між Angular та React для різних типів проектів
Формування практичних рекомендацій щодо вибору технології для розробки веб-додатків.

5. Перелік графічного матеріалу: *презентація*

1. Мета дослідження
2. Компоненти в React
3. Компоненти в Angular
4. Маршрутизація в React
5. Маршрутизація в Angular
6. Управління запитами та станом
7. Додаткові технічні аспекти
8. Загальне порівняння
9. Простота використання, поріг входу, швидкість розробки
10. Відомі приклади використання React та Angular
11. Опитування розробників про досвід і ставлення до фронтенд фреймворків
12. Кількість вакансій на Linkedin
13. ВИСНОВКИ

6. Дата видачі завдання «15» жовтня 2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз сучасних підходів до розробки веб-додатків	10.10-05.11.24	Виконано
2	Вивчення особливостей React і Angular, їх архітектури та можливостей	05.11-12.11.24	
3	Дослідження основних відмінностей у маршрутизації, управлінні станом, асинхронними запитами, підтримці SSR, інтеграції TypeScript та інструментах для тестування.	13.11-19.11.24	
4	Аналіз продуктивності та гнучкості React і Angular	20.11-25.11.24	
5	Огляд бізнес-аспектів і критеріїв вибору між React та Angular	27.11-03.12.24	
6	Формування висновків щодо вибору технології залежно від типу проекту	04.12-10.12.24	
7	Оформлення роботи: вступ, висновки, реферат	11.12-20.12.24	
8	Розробка демонстраційних матеріалів	21.12-29.12.24	

Здобувач вищої освіти _____ Сергій СТАДНИК
(підпис) *(Ім'я, ПРІЗВИЩЕ)*

Керівник
кваліфікаційної роботи _____ Каміла СТОРЧАК
(підпис) *(Ім'я, ПРІЗВИЩЕ)*

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня магістра: 90 стор., 5 табл., 17 джерел.

Мета дослідження: Аналіз та порівняння методів розробки веб-додатків за допомогою Angular та React з урахуванням технічних і бізнес-критеріїв для обґрунтованого вибору технології для різних типів проектів.

Об'єкт дослідження: Сучасні фреймворки для розробки веб-додатків.

Предмет дослідження: Особливості, переваги та недоліки Angular і React у контексті веб-розробки.

Короткий зміст роботи: У дослідженні проаналізовано сучасні тенденції веб-розробки та основні концепції популярних фреймворків. Проведено порівняння Angular і React з урахуванням їхніх архітектурних особливостей, продуктивності, популярності та бізнес-критеріїв, таких як швидкість розробки, доступність фахівців і вартість розробки. На основі аналізу запропоновано практичні рекомендації щодо вибору технології для різних типів веб-проектів. Отримані результати сприяють ефективнішому використанню сучасних фреймворків у комерційних проектах.

КЛЮЧОВІ СЛОВА: REACT, ANGULAR, ФРЕЙМВОРКИ, ВЕБ-РОЗРОБКА, АРХІТЕКТУРА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ, БІЗНЕС-КРИТЕРІЇ, UX/UI.

ABSTRACT

Text part of the master's qualification work: 90 pages, 5 tables, 17 sources.

The purpose of the work: To analyze and compare web application development methods using Angular and React, considering technical and business criteria, to justify the choice of technology for different types of projects.

Object of research: Modern frameworks for web application development.

Subject of research: Features, advantages, and disadvantages of Angular and React in the context of web development.

Summary of the work: The study analyzes current trends in web development and the main concepts of popular frameworks. A comparison of Angular and React is conducted, focusing on their architectural features, performance, popularity, and business criteria such as development speed, availability of specialists, and development costs. Based on the analysis, practical recommendations are proposed for selecting the appropriate technology for various types of web projects. The results contribute to the more efficient use of modern frameworks in commercial projects.

KEYWORDS: REACT, ANGULAR, FRAMEWORKS, WEB DEVELOPMENT, SOFTWARE ARCHITECTURE, BUSINESS CRITERIA, UX/UI.

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

ПОДАННЯ ГОЛОВІ ЕКЗАМЕНАЦІЙНОЇ КОМІСІЇ ЩОДО ЗАХИСТУ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Направляється здобувач Стадник С.П. до захисту кваліфікаційної роботи
(прізвище та ініціали)
за спеціальністю 126 Інформаційні системи та технології
(код, найменування спеціальності)
освітньо-професійної програми Інформаційні системи та технології
на тему: «Дослідження та порівняння методів розробки веб-додатків на Angular та React».

Кваліфікаційна робота і рецензія додаються.

Директор ННІІТ _____ Катерина НЕСТЕРЕНКО
(підпис) *(Ім'я, ПРІЗВИЩЕ)*

Висновок керівника кваліфікаційної роботи

Здобувач Стадник Сергій Петрович показав відмінну теоретичну та практичну підготовку, уміння володіти новими комп'ютерними технологіями, користуватися навчальною, довідковою і науково-технічною літературою. Працюючи над завданнями, які були поставлені керівником, проявив ініціативність, сумлінність та хист до наукової роботи.

Все це дозволяє оцінити виконану кваліфікаційну роботу здобувача Стадник С.П. на оцінку «відмінно» та присвоїти йому кваліфікацію магістр з інформаційних систем та технологій.

Керівник кваліфікаційної роботи _____ Каміла СТОРЧАК
(підпис) *(Ім'я, ПРІЗВИЩЕ)*
"___" _____20__ року

Висновок кафедри про кваліфікаційну роботу

Кваліфікаційна робота розглянута. Здобувач Стадник С.П. допускається до захисту даної роботи в Екзаменаційній комісії.

Завідувач кафедрою Інженерії програмного забезпечення автоматизованих систем
(назва)

_____ Каміла СТОРЧАК
(підпис) *(Ім'я, ПРІЗВИЩЕ)*

ВІДГУК РЕЦЕНЗЕНТА
на кваліфікаційну магістерську роботу

здобувача вищої освіти *Стадник Сергій Петрович*
на тему «*Дослідження та порівняння методів розробки веб-додатків на Angular та React*»

Актуальність.

Магістерська робота Стадника присвячена актуальному питанню порівняння підходів до розробки веб-додатків за допомогою популярних фреймворків Angular та React. Зростаюча потреба у високоякісних веб-додатках обумовлює важливість цього дослідження, оскільки воно може допомогти технічним лідерам і менеджерам обирати оптимальні інструменти для розробки комерційних продуктів.

Позитивні сторони.

1. Чітко структурований аналіз сучасних фреймворків Angular і React.
2. Комплексне порівняння інженерних критеріїв обраних технологій.
3. Ретельний аналіз бізнес-критеріїв розробки.
4. Практичні рекомендації, які можуть бути використані у реальних проектах.
5. Використання новітніх статистичних даних для обґрунтування висновків.

Недоліки.

1. Обмежений опис впливу зовнішніх технологічних трендів на вибір фреймворку.
2. Відсутність аналізу міжгалузевих аспектів використання Angular та React.
3. Недостатня увага до порівняння інтеграційних можливостей обох технологій із сучасними платформами, такими як AWS або Azure.
4. Відсутність порівняння з іншими сучасними фреймворками.
5. Бракує розгляду соціальних аспектів впливу технологій на суспільство.

Відзначені зауваження не впливають на загальну позитивну оцінку кваліфікаційної магістерської роботи.

Висновок: *кваліфікаційна магістерська робота заслуговує оцінку "Відмінно", а здобувач Стадник Сергій Петрович заслуговує присвоєння кваліфікації: магістр з інформаційних систем та технологій.*

Рецензент: _____ Вікторія ЖЕБКА
доктор технічних наук, професор *підпис*

ЗМІСТ

ВСТУП

Сучасна веб-розробка є однією з найдинамічніших сфер інформаційних технологій. Попит на високоякісні, масштабовані та інтерактивні веб-додатки постійно зростає. Вибір правильного фреймворку для розробки веб-додатків безпосередньо впливає на продуктивність команди розробників, час виведення продукту на ринок та якість кінцевого продукту. Angular і React є найбільш популярними інструментами у цій сфері, але мають різні підходи, архітектуру та сфери застосування. Дослідження та порівняння цих фреймворків дозволяє краще зрозуміти їх особливості, переваги та недоліки, що є важливим для вибору оптимального інструменту для розробки.

Проаналізувати та порівняти методи розробки веб-додатків за допомогою Angular та React, враховуючи технічні та бізнес-критерії, для надання рекомендацій щодо вибору відповідного фреймворку для конкретних типів проектів.

Завдання дослідження

1. Дослідити основні концепції та підходи до розробки веб-додатків.
2. Провести огляд Angular та React, включаючи їх архітектурні особливості, переваги та недоліки.
3. Здійснити порівняння фреймворків за технічними характеристиками та бізнес-критеріями.
4. Розробити рекомендації щодо вибору фреймворку для різних типів проектів.

Об’єкт дослідження

Об’єктом дослідження є сучасні фреймворки для розробки веб-додатків, а саме Angular та React. Ці два фреймворки є одними з найбільш популярних і широко використовуваних інструментів у веб-розробці, що дозволяє їх порівняти

за різними критеріями, такими як архітектура, гнучкість, продуктивність, інструменти та можливості інтеграції. Вивчення їх особливостей у контексті розробки веб-додатків дає можливість визначити, який з них краще підходить для конкретних типів проектів, а також допомагає виявити тенденції та новітні підходи у сучасній веб-розробці.

Предмет дослідження

Предметом дослідження є порівняння фреймворків Angular та React з точки зору їх архітектурних особливостей, підходів до управління станом, інтеграції з іншими технологіями, а також з урахуванням їх сильних та слабких сторін при розробці веб-додатків. Особливу увагу буде приділено таким аспектам, як масштабованість, гнучкість, продуктивність, а також підтримка великих команд розробників. Оцінюватиметься, наскільки ці фреймворки підходять для різних типів проектів, таких як корпоративні додатки, стартапи або платформи електронної комерції. Це дозволить не тільки зрозуміти технічні характеристики кожного фреймворку, але й вибрати найбільш ефективний варіант залежно від конкретних вимог і бізнес-цілей проекту.

Методи дослідження

У дослідженні застосовувалися кілька основних методів для досягнення об’єктивних і комплексних результатів.

- Аналітичні методи були використані для оцінки архітектурних особливостей Angular та React. Це дозволило виявити ключові принципи побудови додатків на кожному з фреймворків, а також визначити, які з них краще підходять для певних завдань. Аналітичний підхід дозволяє здійснити порівняння таких

аспектів, як управління станом, модульність, взаємодія з іншими бібліотеками та інструментами.

- Емпіричні методи використовувалися для тестування продуктивності фреймворків у реальних умовах. Включали в себе проведення тестів на швидкість рендерингу компонентів, час завантаження, а також аналіз реальних додатків, створених на Angular і React. Це дозволило порівняти ефективність цих фреймворків у різних сценаріях, включаючи високі навантаження, складні взаємодії з сервером та інтерактивні функціональності.
- Системний підхід використаний для створення цілісної картини порівняння технологій, враховуючи не тільки технічні, а й бізнес-критерії. Цей підхід допоміг встановити критерії вибору фреймворків в залежності від типу проекту, масштабів, бюджету та вимог до часу розробки. Системний підхід також включає оцінку наявних інструментів, плагінів та бібліотек для кожного фреймворку, що впливає на швидкість і якість розробки.

Таким чином, за допомогою зазначених методів було проведено детальне дослідження, яке дозволило не тільки проаналізувати характеристики Angular та React, а й дати практичні рекомендації щодо їх вибору для різних типів проектів.

Теоретична, методична та практична значущість отриманих результатів

Теоретична значущість

Результати дослідження сприяють поглибленню розуміння особливостей сучасних фреймворків Angular та React, а також їх архітектурних підходів до розробки веб-додатків. Проведене порівняння цих фреймворків з технічної та бізнес-орієнтованої точки зору дозволяє виявити ключові фактори, що впливають на вибір інструментів для розробки, такі як продуктивність, зручність для

розробників та адаптивність до різних типів проектів. Це дає можливість розвинути теоретичні уявлення про вибір технологій у контексті веб-розробки.

Методична значущість

Методичні рекомендації, які містяться у роботі, можуть бути використані для визначення критеріїв вибору фреймворків у навчальному процесі, допомагаючи студентам і фахівцям приймати обґрунтовані рішення при виборі технологій для конкретних завдань. Розроблені підходи до аналізу та порівняння технологій можуть стати основою для подальших досліджень в області веб-розробки, а також для впровадження в навчальні програми з веб-розробки.

Практична значущість

Практичні результати дослідження можуть бути безпосередньо застосовані в реальних проектах веб-розробки для вибору оптимального фреймворку залежно від специфіки проекту. Рекомендації щодо вибору між Angular та React допоможуть компаніям та розробникам заощадити час і ресурси, знижуючи ризики, пов’язані з неправильним вибором інструментів. Крім того, дослідження може бути корисним для покращення процесу розробки в командних проектах, оптимізації архітектури та покращення продуктивності розробки веб-додатків.

1 АНАЛІЗ СУЧАСНИХ ПІДХОДІВ ДО РОЗРОБКИ ВЕБ-ДОДАТКІВ

1.1 Основні концепції сучасних фреймворків для розробки веб-додатків

У цьому підрозділі розглянуто основні принципи та концепції, на яких базуються сучасні фреймворки для розробки веб-додатків. До таких фреймворків належать React, Angular, Vue та інші популярні рішення. Основні принципи забезпечують швидкість розробки, масштабованість і продуктивність веб-додатків. Нижче розглянуто ключові концепції, що лежать в основі сучасних фреймворків.

1.1.1 Компонентно-орієнтована архітектура

Компонентно-орієнтована архітектура є однією з ключових концепцій у сучасній розробці програмного забезпечення, особливо в області веб-розробки. Вона базується на використанні компонентів як основних будівельних блоків, які дозволяють структурувати додатки в зручний, модульний та повторно використовуваний спосіб.

Компонент — це ізольована, незалежна частина інтерфейсу користувача (UI), яка об'єднує логіку, структуру (шаблони) і стилі. Він виконує певну роль у додатку, наприклад, відображення кнопки, форми або складнішого функціоналу, такого як таблиця або модальне вікно. Компоненти забезпечують інкапсуляцію, тобто вони приховують внутрішню реалізацію і взаємодіють із зовнішнім світом через визначені інтерфейси (пропси, input/output параметри тощо).

1.1.1.1 Основні переваги компонентного підходу

- Модульність. Кожен компонент є окремим модулем, що виконує конкретну задачу. Це дозволяє розбивати складні інтерфейси на дрібніші частини, які легше розробляти, тестувати та підтримувати.
- Повторне використання. Один раз створений компонент можна використовувати в різних частинах додатку або навіть в інших проектах. Це значно знижує кількість дубльованого коду і підвищує ефективність розробки.
- Легке тестування. Завдяки ізольованості компонентів, їх можна тестувати незалежно один від одного. Це робить виявлення і виправлення помилок швидшим і простішим.
- Масштабованість. Компоненти дозволяють зберігати чітку структуру коду навіть у великих проектах, де кількість елементів інтерфейсу значно зростає.
- Спрощення підтримки. Якщо зміни потрібні в певній частині інтерфейсу, достатньо оновити лише відповідний компонент, без необхідності змінювати інший код.

1.1.2 Реактивність і динамічна зміна стану

Реактивність є однією з основних характеристик сучасних фреймворків, що забезпечує динамічну зміну стану користувацького інтерфейсу (UI) у відповідь на оновлення даних. Вона дозволяє створювати інтуїтивно зрозумілі, інтерактивні додатки, які миттєво реагують на дії користувача або зміни в системі.

Реактивність — це здатність програми автоматично оновлювати UI, коли змінюється стан додатка. Це досягається за допомогою зв'язку між даними (моделлю) і представленням (UI). Як тільки дані змінюються, відповідні частини інтерфейсу оновлюються автоматично, без необхідності розробнику вручну маніпулювати DOM (Document Object Model).

1.1.2.1 Основні принципи реактивності

- Спостереження за станом. Система відстежує зміни стану (даних) і визначає, які частини UI залежать від цих даних.
- Автоматичне оновлення. Коли стан змінюється, відповідні елементи інтерфейсу оновлюються автоматично. Це усуває необхідність ручного оновлення DOM, що робить код більш простим і зрозумілим.
- Оптимізація оновлень. Сучасні фреймворки використовують різні механізми, щоб мінімізувати кількість оновлень DOM, що покращує продуктивність додатків.

1.1.3 Шаблони у фреймворках: декларативний підхід до побудови інтерфейсу

Шаблони у фреймворках є ключовим інструментом для опису структури та вигляду користувацького інтерфейсу (UI). Вони дозволяють використовувати декларативний підхід, при якому розробник описує що потрібно відобразити, а не як це зробити. Такий підхід робить код більш зрозумілим, зручним для підтримки та читабельним.

Шаблони — це текстові конструкції, які описують структуру HTML у поєднанні з динамічними даними, логікою відображення (цикли, умови) та стилями. Завдяки шаблонам розробники можуть створювати динамічні, інтерактивні інтерфейси, які автоматично оновлюються у відповідь на зміни стану.

1.1.3.1 Основні можливості шаблонів

- Декларативний синтаксис. Дозволяє зосередитися на описі структури інтерфейсу, не турбуючись про внутрішню реалізацію оновлення DOM.
- Динамічні дані. Шаблони підтримують інтеграцію з даними додатку, що дозволяє автоматично оновлювати UI у відповідь на зміни стану.
- Цикли та умовні конструкції. Дають можливість створювати повторювані елементи (списки) та умовно відображати частини інтерфейсу.

– Вбудована логіка. Шаблони можуть включати прості логічні операції, наприклад, умовні вирази чи обробку подій.

1.1.4 Система модулів: організація коду через логічні групи

Модульність — це підхід до організації коду, при якому він розділяється на логічно згруповані частини (модулі), кожна з яких відповідає за певний функціонал програми. Завдяки модульності програмний код стає більш зрозумілим, легшим у підтримці, масштабуванні та повторному використанні.

Модуль — це ізольована частина програми, яка виконує конкретну задачу. Модулі можуть містити компоненти, функції, класи, сервіси, директиви, стилі тощо. Модулі зазвичай мають чітко визначені точки входу та виходу через механізми імпорту/експорту.

1.1.4.1 Переваги модульності

– Організованість коду. Розділення програми на окремі модулі допомагає уникнути плутанини в коді та зменшити залежності між різними частинами програми.
– Повторне використання. Один і той самий модуль можна використовувати у різних проектах або частинах програми.
– Масштабованість. Модульна структура спрощує додавання нових функціональних можливостей без суттєвих змін у наявному коді.
– Тестованість. Модулі ізолюють функціонал, що дозволяє легко тестувати їх незалежно один від одного.

1.1.5 Інструменти для маршрутизації

Маршрутизація — це процес управління переходами між різними частинами додатку (сторінками) у веб-застосунках. У сучасних фреймворках вона дозволяє створювати SPA (Single Page Applications, односторінкові додатки), де користувачі можуть переміщуватися між сторінками без повного перезавантаження браузера.

У SPA маршрутизація забезпечує:

- Плавні переходи між сторінками. Браузер не перезавантажує сторінку, а змінює її вміст динамічно.
- Підтримку глибоких посилань. URL змінюється відповідно до відображуваної сторінки, дозволяючи створювати закладки або ділитися посиланнями.
- Відповідність структурі URL. Маршрутизація допомагає відображати правильні компоненти чи сторінки відповідно до маршруту.

Переваги маршрутизації:

- Поліпшення продуктивності. Менше запитів до сервера, оскільки дані та сторінки завантажуються частинами.
- Кращий користувацький досвід. Перехід між сторінками відбувається швидше, що створює відчуття інтерактивності.
- Легка інтеграція з SEO. Деякі маршрутизатори підтримують серверний рендеринг, щоб покращити індексацію сторінок пошуковими системами.

1.1.6 Система управління станом

Управління станом є важливим аспектом розробки складних веб-додатків, оскільки воно забезпечує централізоване зберігання та обробку даних, знижуючи складність роботи з численними компонентами, що мають доступ до одного і того ж набору даних. Це особливо важливо для додатків з великим обсягом взаємодій, таких як інтерактивні інтерфейси, де різні частини програми повинні синхронізуватися та реагувати на зміни.

1.1.6.1 Основні аспекти управління станом

- Централізація даних: У великих додатках, де дані можуть змінюватися в різних компонентах, централізоване управління станом дозволяє зберігати всі дані в одному місці, що спрощує доступ і зменшує дублювання. Це

забезпечує уніфікований доступ до стану програми, що важливо для складних додатків.

- Глобальний стан: У великих застосунках існують елементи, які потребують доступу до одних і тих самих даних, наприклад, інформація про поточного користувача, налаштування або статуси запитів. Управління глобальним станом дозволяє зберігати ці дані в одному місці і робити їх доступними для всіх компонентів, які їх потребують.
- Реактивність: У сучасних веб-додатках важливо реагувати на зміни стану в реальному часі. Реактивне управління станом дозволяє автоматично оновлювати інтерфейс при зміні даних, не вимагаючи додаткових зусиль для вручну синхронізації даних між компонентами.

1.1.7 Побудова інтерфейсів із реактивними потоками. Реактивне програмування для асинхронних даних

Реактивне програмування є підходом, що дозволяє будувати додатки, орієнтуючись на потоки даних і події. Цей підхід є особливо корисним для обробки асинхронних даних, таких як запити до серверу, зміни стану чи користувацькі події. Реактивність дозволяє автоматично оновлювати інтерфейс користувача у відповідь на зміни в цих потоках даних, що значно спрощує роботу з асинхронними операціями і підвищує ефективність додатка.

1.1.7.8 Переваги реактивного підходу

- Простота управління асинхронними операціями: Реактивне програмування дозволяє більш елегантно працювати з асинхронними даними. За допомогою потоків даних можна автоматично обробляти зміни та синхронізувати події без необхідності вручну маніпулювати складним станом.
- Забезпечення передбачуваності: Завдяки використанню потоків та операторів для обробки даних, можна легко прогнозувати, як додаток буде реагувати на різні зміни, що робить розробку більш керованою і стабільною.

– Легкість інтеграції з іншими частинами додатка: Реактивні потоки дозволяють інтегрувати різні частини додатка, наприклад, зручну обробку HTTP-запитів з асинхронними подіями користувача, що підвищує взаємодію компонентів.

1.1.8 Інтегровані та сторонні інструменти

Сучасні фронтенд-фреймворки, такі як Angular і React, надають широкий спектр інтегрованих та сторонніх інструментів, які значно полегшують розробку, тестування, перевірку якості коду та автоматизацію процесів. Ці інструменти охоплюють різні аспекти розробки від ініціалізації проекту до інтеграції з іншими системами, а також забезпечують зручність під час тестування та підтримки високої якості коду.

1.1.8.1 Інструменти для побудови проектів

Одним з важливих аспектів розробки є налаштування середовища для роботи з проектом. Сучасні фреймворки зазвичай мають інтегровані інструменти для ініціалізації та побудови проектів, що значно спрощує старт розробки.

– Angular CLI (Command Line Interface) є потужним інструментом для автоматизації процесу створення, налаштування та побудови проектів на Angular. Він дозволяє створювати нові Angular проекти з нуля, генерувати компоненти, сервіси, модулі, а також автоматично конфігурувати збірку, перевірку, тестування, деплой та інші аспекти. Angular CLI має безліч команд, які дозволяють швидко інтегрувати різноманітні функції в проект, такі як ng serve, ng build, ng test, ng lint тощо. Цей інструмент допомагає спростити налаштування середовища для розробки та підтримує процес розробки на всіх етапах.
– Create React App (CRA) — це офіційний інструмент для створення нових React проектів, який автоматично налаштовує середовище для розробки та збірки. CRA надає простий шаблон, який включає налаштування для Babel,

Webpack, ESLint та інші основні інструменти для створення сучасного веб-додатка. Це дозволяє розробникам швидко почати роботу без необхідності вручну налаштовувати всі ці інструменти. Для запуску React додатка достатньо лише кількох команд, таких як npx create-react-app для створення проекту і npm start для запуску локального сервера.

- Vite — це сучасний інструмент для побудови проектів, який став популярним завдяки своїй швидкості та простоті налаштування. Vite використовує нові можливості браузерів, такі як ES-модулі, для швидкого старту розробки, миттєвого перезавантаження сторінки при зміні коду та оптимізації процесу збірки. Він підходить як для React, так і для інших фреймворків, таких як Vue.js, і дозволяє розробникам зосередитись на створенні функціональності, не витрачаючи час на конфігурацію інструментів. Завдяки своїй високій продуктивності, Vite отримав широке визнання серед розробників, які працюють із сучасними фреймворками.

1.1.8.2. Інструменти для тестування

Тестування є важливою частиною процесу розробки програмного забезпечення. Для забезпечення стабільності і якості додатків розробники використовують інструменти для автоматизованого тестування, такі як юніт-тести, інтеграційні тести та E2E (end-to-end) тести.

- Jest — це популярна бібліотека для тестування JavaScript-коду, яка зазвичай використовується для тестування React-додатків. Jest надає потужні можливості для юніт-тестування компонентів, функцій та інших частин додатка. Він також підтримує створення моків, стіблінг, а також зручний синтаксис для написання тестів. Jest має вбудовану підтримку для тестування асинхронних функцій і інтеграцію з іншими інструментами для тестування, такими як Enzyme або Testing Library. Завдяки своїй швидкості та простоті використання, Jest став стандартом у React спільноті.

- Karma — це тестовий запускник, який дозволяє виконувати тести в реальному браузері. Karma забезпечує інтеграцію з популярними бібліотеками для тестування, такими як Jasmine, Mocha, QUnit тощо. З його допомогою можна тестувати додатки в різних браузерах, що дозволяє перевірити крос-браузерну сумісність і точність роботи додатка в реальних умовах.
- Jasmine — це фреймворк для написання юніт-тестів для JavaScript, який дозволяє створювати тести із зрозумілим та читабельним синтаксисом. Jasmine зазвичай використовується в парі з Karma для запуску тестів у браузері. Завдяки своїй гнучкості та зручному синтаксису Jasmine є одним з найпопулярніших інструментів для тестування JavaScript коду.

1.1.8.3 Інструменти для статичного аналізу та перевірки коду

Статичний аналіз коду допомагає автоматично виявляти помилки, недоліки та неефективні практики в коді без необхідності виконувати програму. Інструменти для статичного аналізу коду дозволяють підтримувати високий рівень якості коду, що важливо для великих проектів з багатьма розробниками.

- ESLint це потужний інструмент для статичного аналізу JavaScript-коду, який дозволяє знаходити проблеми у коді, такі як стилістичні помилки, неправильне використання змінних, а також потенційні баги. ESLint також надає можливість налаштувати власні правила перевірки або використовувати стандартні конфігурації, такі як Airbnb або Google. В Angular та React додатках ESLint часто використовується для забезпечення узгодженості коду і його відповідності певним стилістичним стандартам.
- TSLint це інструмент для статичного аналізу TypeScript коду. Він допомагає забезпечити високу якість коду, використовуючи правила перевірки, специфічні для TypeScript, і підтримує численні конфігурації. Однак з часом TSLint був замінений на ESLint, який тепер підтримує

аналіз як JavaScript, так і TypeScript коду, зручно інтегруючи правила перевірки для обох мов.

1.1.9 Продуктивність і оптимізація

Продуктивність є важливою характеристикою будь-якого сучасного веб-додатка, особливо коли йдеться про великі та складні інтерфейси користувача, які можуть містити багато динамічних компонентів. Для забезпечення високої продуктивності фреймворки використовують різні механізми оптимізації, щоб зменшити час відгуку додатка, кількість рендерів та ефективно працювати з DOM.

1.2 Огляд фреймворків Angular і React

Angular і React є одними з найбільш популярних інструментів для розробки веб-додатків, проте вони значно різняться у своїй архітектурі, філософії розробки та інструментах, які вони пропонують. Вибір між цими двома фреймворками залежить від багатьох факторів, включаючи вимоги проекту, складність розробки та досвід команди.

Angular — це фреймворк для створення складних веб-додатків, який пропонує повну екосистему інструментів для розробки. Розроблений Google, Angular активно використовується для побудови односторінкових додатків (SPA) з високою продуктивністю та великими вимогами до масштабованості.

1.2.1 Основні характеристики Angular

– Цілісність фреймворка: Angular включає все необхідне для розробки веб-додатків, включаючи маршрутизацію, форми, HTTP-запити, а також потужну систему для управління залежностями через механізм Dependency Injection.

- Мови та інструменти: Angular написаний на TypeScript, що додає строгу типізацію та інші можливості для покращення підтримки коду, як-от автодоповнення та відловлювання помилок на етапі компіляції.
- RxJS: Angular активно використовує бібліотеку RxJS для обробки асинхронних операцій, що дозволяє розробникам працювати з потоками даних через концепцію "реактивного програмування".
- Продуктивність та масштабованість: Завдяки своїй архітектурі, Angular ідеально підходить для великих корпоративних проектів, де важлива підтримка масштабованості та структурованості коду.

React — це бібліотека для побудови інтерфейсів користувача, яка була розроблена Facebook і стала однією з найбільш популярних технологій для створення динамічних веб-додатків. На відміну від Angular, React пропонує набагато менше "з коробки", дозволяючи розробникам вибирати інші інструменти для виконання різних завдань, таких як маршрутизація або управління станом.

1.2.2 Основні характеристики React

- Гнучкість: React не нав'язує архітектурних рішень, що дає розробникам більшу свободу у виборі бібліотек та інструментів для маршрутизації (наприклад, react-router) або управління станом (наприклад, Redux, Recoil чи MobX).
- Використання компонентів: Всі елементи в React є компонентами, що полегшує повторне використання коду та створення модульних і інтерактивних інтерфейсів.
- JSX: React використовує JSX, що дозволяє писати компоненти у форматі, який схожий на HTML, але при цьому інтегрується з JavaScript-кодом. Це дозволяє створювати динамічні інтерфейси та інтерактивні елементи без необхідності переписувати код на кількох рівнях.

– Продуктивність: Завдяки віртуальному DOM, React оптимізує оновлення інтерфейсу, що дозволяє досягати високої продуктивності при зміні великої кількості елементів на сторінці.

1.2.3 Загальне порівняння React та Angular

Рівень абстракції: Angular є більш цілісним фреймворком, який надає більш структуровану розробку додатків за допомогою вбудованих інструментів, таких як маршрутизація, форми та інші, тоді як React є лише бібліотекою для створення інтерфейсів користувача, залишаючи багато рішень на вибір розробників.

Типізація: Angular використовує TypeScript, що допомагає уникнути багатьох помилок на етапі компіляції завдяки строгій типізації. React може працювати як з JavaScript, так і з TypeScript, що дає розробникам більшу гнучкість.

Навчання та спільнота: React має велику спільноту та безліч навчальних ресурсів, завдяки своїй популярності. Angular, хоча і менш популярний за React, забезпечує добре документовану підтримку з численними офіційними керівництвами.

Підходи до тестування: Angular має вбудовану підтримку тестування, що є важливою частиною фреймворка. У React тестування також підтримується, але залежить від сторонніх бібліотек, таких як Jest та React Testing Library.

Angular і React — це потужні інструменти, кожен з яких має свої переваги та недоліки. Angular підходить для великих та складних проектів, де важлива сувора структура та вбудовані інструменти. React ж є більш гнучким рішенням, яке дозволяє розробникам створювати інтерфейси з мінімальними обмеженнями і вибирати власні інструменти для інших аспектів розробки.

Цей підрозділ дозволяє зрозуміти основні відмінності між Angular та React та дає підстави для вибору одного з цих фреймворків залежно від потреб конкретного проекту.

1.3 Порівняння популярності Angular і React у сучасному IT-середовищі

React і Angular є двома з найбільш популярних інструментів для розробки веб-додатків. Хоча обидва активно використовуються в сучасному IT-середовищі, їх популярність та сфери застосування відрізняються.

- Популярність у спільноті. React має значну перевагу за кількістю користувачів і активністю спільноти. У GitHub React налічує понад 231 тис. зірок, тоді як Angular — близько 97 тис. Це свідчить про більшу залученість і підтримку спільноти React. React також вважається більш бажаним серед розробників: у 2023 році 32% опитаних розробників хотіли працювати з ним, у порівнянні з 13% для Angular
- Корпоративне використання. React широко використовується компаніями, такими як Facebook, Instagram, Netflix, Airbnb і Shopify, завдяки його гнучкості та легкості у використанні. Angular популярний у великих корпоративних додатках, наприклад, Gmail, YouTube TV, Microsoft Office Web і банківських платформах на зразок PayPal
- Навчання та вхідний поріг. React має більш плавну криву навчання, оскільки він фокусується на побудові інтерфейсу і не потребує знання складніших концепцій, як, наприклад, у Angular (з його TypeScript, залежностями та багаторівневою архітектурою). Це робить React більш привабливим для новачків у JavaScript
- Стабільність і еволюція. Angular забезпечує більш структуровану архітектуру та стабільність у великих проектах завдяки строгим правилам і інструментам, таким як Angular CLI. React, як бібліотека, пропонує більшу гнучкість, однак це може створювати складнощі для новачків через різноманіття додаткових бібліотек і часті оновлення екосистеми

– Тренди та рейтинги. У рейтингах, як-от Stack Overflow Developer Survey 2023, React випереджає Angular у категоріях "використання" та "задоволеність". Більша популярність React зумовлена його простотою, продуктивністю та широким вибором інструментів для інтеграції, таких як React Router чи Redux

React домінує в категорії популярності завдяки своїй гнучкості та дружності до розробників. Angular залишається фаворитом у великих корпоративних проектах, де важлива структурованість і строгість архітектури. Вибір між ними залежить від вимог вашого проекту та готовності команди працювати з відповідним інструментом.

2 ПОРІВНЯННЯ МЕТОДІВ РОЗРОБКИ ВЕБ-ДОДАТКІВ У REACT ТА ANGULAR ЗА ІНЖЕНЕРНИМИ КРИТЕРІЯМИ

2.1 Компоненти в React та Angular

Компоненти є основним будівельним блоком обох фреймворків, але підходи до їх створення, структурування та керування дещо відрізняються. У цьому розділі розглянуто порівняння підходів до реалізації компонентів у React та Angular, зокрема:

- Структура компонента
- Життєвий цикл компонента
- Підхід до шаблонів (Templates)
- Стилізація компонентів
- Комунікація між компонентами
- Розширення компонентів (HOC, Directives)

2.1.1 Структура компонента

У React компонент — це JavaScript-функція або клас, яка повертає елементи JSX, які, в свою чергу, описують структуру UI. Структура компонента у React зазвичай виглядає так:

```
import { useState } from 'react'

function MyComponent() {
   const [count, setCount] = useState(0)
    return (
       <div>
           <p>Count: {count}</p>
           <button onClick={() => setCount(count + 1)}>Increment</button>
```

```
        </div>
    )
}

export default MyComponent
```

- JSX використовується для опису структури компонента.
- Логіка та UI тісно пов'язані, оскільки компоненти обробляють свою власну логіку й рендеринг.
- React використовує функціональні компоненти (з хуками) або класові компоненти для більш складних сценаріїв.

В Angular компонент складається з трьох основних частин:

- HTML-шаблон (Template)
- CSS-стилі (Styles)
- TypeScript-клас (Class)

```
import { Component } from '@angular/core';

@Component({
 selector: 'app-my-component',
 template: `
   <div>
     <p>Count: {{ count }}</p>
     <button (click)="increment()">Increment</button>
   </div>
 `,
 styles: ['button { color: blue; }']
})
export class MyComponent {
 count = 0;

 increment() {
   this.count++;
 }
}
```

– Angular компоненти поєднують HTML-шаблон, CSS та TypeScript клас у одному файлі або у розділених файлах.
– Шаблони в Angular використовують власний синтаксис, наприклад {{ count }} для прив'язки даних.
– Класова структура є більш традиційною, що дозволяє легко додавати методи та властивості.

2.1.2 Життєвий цикл компонента

React має декілька методів життєвого циклу, доступних через класи або хуки:

– Класові компоненти: життєвий цикл базується на методах, таких як componentDidMount(), shouldComponentUpdate(), componentWillUnmount() тощо.
– Функціональні компоненти: для роботи з життєвим циклом використовуються hooks, такі як useEffect() для виконання побічних ефектів, подібних до componentDidMount та інших.

```
useEffect(() => {
   // код, який виконується при монтуванні компонента
   return () => {
       // код, який виконується при розмонтуванні компонента
   }
}, [])
```

Angular має більш структурований підхід до життєвого циклу з використанням спеціальних хуків, таких як:

– ngOnInit(): викликається після ініціалізації компонента.
– ngOnChanges(): викликається при зміні властивостей компонента.
– ngOnDestroy(): викликається перед знищенням компонента.

```
ngOnInit() {
 console.log('Component initialized');
}
```

Angular також надає більш вбудовану підтримку для взаємодії з циклом життя компонентів через декоратори та інтерфейси.

2.1.3 Підхід до шаблонів (Templates)

У React шаблони створюються за допомогою JSX, що дозволяє використовувати JavaScript безпосередньо всередині шаблону. Це дозволяє створювати більш інтерактивні компоненти та гнучко маніпулювати даними в процесі рендерингу.

```
return (
   <div>
     <h1>{userName}</h1>
     {isLoggedIn ? <button>Logout</button> : <button>Login</button>}
   </div>
 );
```

JSX є надбудовою над JavaScript, тому працює як візуальна репрезентація компонентів, але в ньому також можна вбудовувати логіку для обробки подій, умовних рендерів тощо.

В Angular шаблони більш декларативні і використовують власний синтаксис для прив'язки даних, обробки подій та умовного рендерингу. Наприклад:

```
<div>
 <h1>{{ userName }}</h1>
 <button (click)="login()">{{ isLoggedIn ? 'Logout' : 'Login' }}</button>
</div>
```

Angular використовує двосторонню прив'язку даних через синтаксис [(ngModel)] і дозволяє обробляти події через (event).

2.1.4 Стилізація компонентів

Стилізація компонентів є важливим аспектом у розробці веб-додатків, і React та Angular пропонують різні підходи для досягнення цієї мети. Обидва фреймворки мають потужні інструменти для стилізації, але їх підходи значно відрізняються, зокрема в контексті модульності, організації стилів та використання додаткових технологій, таких як SASS, SCSS та LESS.

У React є кілька способів для стилізації компонентів, що дозволяє розробникам вибирати найбільш зручний підхід залежно від потреб проекту:

– Зовнішні CSS файли. Найбільш традиційний спосіб стилізації в React — це використання зовнішніх CSS-файлів, які імпортуються безпосередньо в компонент. Такий підхід дозволяє застосовувати глобальні CSS класи до всього додатка або специфічні CSS класи до окремих компонентів. У цьому випадку клас my-component визначено у файлі MyComponent.css, і він буде застосовуватись до елемента у компоненті.

```
// MyComponent.jsx
import './MyComponent.css'

function MyComponent() {
   return (
       <div className="my-component">
           <h1>Hello, React!</h1>
       </div>
   )
}
```

– CSS Modules. React також підтримує CSS Modules, що дозволяє локалізувати стилі кожного компонента і уникнути конфліктів імен класів. У цьому випадку класи CSS є унікальними для кожного модуля, і їх не потрібно додавати вручну. Це забезпечує ізольованість стилів на рівні компонентів. У цьому прикладі стилі з MyComponent.module.css будуть застосовуватись

тільки до цього компонента, що дозволяє уникнути конфліктів з іншими класами на сторінці.

```
// MyComponent.module.css
.myComponent {
   color: blue;
}
 // MyComponent.jsx
import styles from './MyComponent.module.css'
 function MyComponent() {
   return (
       <div className={styles.myComponent}>
           <h1>Hello, React with CSS Modules!</h1>
       </div>
   )
}
```

– CSS-in-JS. Завдяки технологіям CSS-in-JS стилі можна визначати безпосередньо в JavaScript коді, що дозволяє мати додаткову гнучкість, зокрема для умовної стилізації. Одна з найбільш популярних бібліотек для цього — це styled-components, яка дозволяє створювати стилізовані компоненти. CSS-in-JS підхід дозволяє використовувати JavaScript для додавання логіки в стилі, а також забезпечує масштабованість і зручність для великих додатків, оскільки дозволяє створювати динамічні стилі на основі стану або пропсів компонента.

```
import styled from 'styled-components'

const Button = styled.button`
 background-color: blue;
 color: white;
`

function MyComponent() {
   return (
       <div>
```

```
            <Button>Click me</Button>
        </div>
    )
}
```

– SASS/SCSS та LESS. React також підтримує SASS/SCSS та LESS, що дозволяє використовувати функціональність цих препроцесорів, таких як вкладеність, змінні, міксини тощо, для стилізації компонентів.

```
// styles.module.scss
$primary-color: blue;

.button {
 background-color: $primary-color;
 color: white;
}
```

```
import styles from './styles.module.scss'

function MyComponent() {
    return (
        <button className={styles.button}>Click me</button>
    )
}
```

Angular надає більш інтегрований підхід до стилізації, де стилі є частиною кожного компонента. Angular підтримує різні методи стилізації, включаючи CSS, SCSS, SASS, LESS та локальні стилі для компонентів.

– Зовнішні стилі та локальні стилі. Кожен Angular компонент може містити локальні стилі, що визначаються в опціях компонента через декоратор @Component. Це дозволяє стилізувати лише конкретний компонент, не впливаючи на інші частини додатка. У цьому прикладі стилі визначені безпосередньо в компоненті, і вони не впливають на інші компоненти в додатку. Це допомагає уникнути проблем з глобальними стилями.

```
import { Component } from '@angular/core';

@Component({
 selector: 'app-my-component',
 template: `
   <div class="my-component">
     <h1>Hello, Angular!</h1>
   </div>
 `,
 styles: [`
   .my-component {
     color: blue;
   }
 `]
})
export class MyComponent {}
```

– Використання зовнішніх файлів CSS або SCSS. Як і в React, в Angular можна використовувати зовнішні файли для стилізації компонентів, як через імпорти у компоненті, так і через глобальні стилі. Цей підхід дозволяє відокремлювати логіку компонента від його стилів і використовувати більш комплексні стилізації для більших проектів.

```
@Component({
 selector: 'app-my-component',
 templateUrl: './my-component.component.html',
 styleUrls: ['./my-component.component.css']
})
export class MyComponent {}
```

– SASS/SCSS та LESS. Angular також підтримує SASS, SCSS та LESS для стилізації. Для цього потрібно налаштувати відповідний препроцесор у вашому проекті. Для SCSS: Angular автоматично підтримує SASS/ SCSS через CLI, тому достатньо просто створити SCSS файли і вказати їх у налаштуваннях компонента. У цьому випадку Angular буде обробляти SCSS файли та компілювати їх у CSS при збірці додатка.

```
@Component({
 selector: 'app-my-component',
 templateUrl: './my-component.component.html',
 styleUrls: ['./my-component.component.scss']
})
export class MyComponent {}
```

2.1.5 Комунікація між компонентами

У React комунікація між компонентами відбувається через props та state:

- Props використовуються для передачі даних від батьківського компонента до дочірнього.
- State використовується для збереження внутрішнього стану компонента, але може бути переданий в інші компоненти через колбеки.

В Angular для передачі даних використовуються:

- @Input() та @Output() декоратори для обміну даними між батьківським та дочірнім компонентами.
- Angular також підтримує служби (services) для спільного доступу до даних між різними компонентами.

2.1.6 Розширення компонентів (HOC, Directives)

React використовує Higher-Order Components (HOC) для розширення функціональності компонентів. HOC — це функції, які приймають компонент і повертають новий компонент з додатковою логікою.

```
function withLogging(Component) {
   return function WrappedComponent(props) {
       console.log('Rendering', Component.name)
       return <Component {...props} />
   }
}
```

Angular використовує директиви для розширення HTML елементів, додаючи їм нову поведінку. Директиви можуть бути структурними (наприклад, *ngIf) або атрибутивними (наприклад, ngClass).

```
@Directive({
 selector: '[appHighlight]'
})
export class HighlightDirective {
 constructor(el: ElementRef) {
   el.nativeElement.style.backgroundColor = 'yellow';
 }
}
```

2.2 Маршрутизація

Маршрутизація є ключовим аспектом будь-якого веб-додатка, дозволяючи перемикатись між сторінками або розділами додатка без повного перезавантаження. React та Angular підходять до реалізації маршрутизації по-різному, відображаючи загальний підхід фреймворків до організації додатків.

У React маршрутизація реалізується через сторонні бібліотеки, найпопулярніша з яких — React Router. Ця бібліотека надає розробникам потужний та гнучкий набір інструментів для роботи з маршрутизацією. Основні її можливості:

– Декларативний підхід: У React маршрути визначаються як компоненти, що робить їх легко зрозумілими і зручними для інтеграції з іншими компонентами.

```
import { BrowserRouter as Router, Route, Routes } from 'react-router-dom'

import About from './About'
import Home from './Home'

function App() {
   return (
       <Router>
           <Routes>
```

```
                <Route path="/" element={<Home />} />
                <Route path="/about" element={<About />} />
            </Routes>
        </Router>
    )
}
```

– Динамічні маршрути: React Router дозволяє створювати динамічні маршрути з використанням параметрів.

```
<Route path="/user/:id" element={<UserProfile />} />
```

– Lazy Loading: Завдяки інтеграції з функцією React.lazy, React Router підтримує динамічне завантаження компонентів, що покращує продуктивність великих додатків.
– Навігація через API: Навігація може бути здійснена програмно за допомогою хука useNavigate.

```
import { useNavigate } from 'react-router-dom'

function Login() {
    const navigate = useNavigate()

    const handleLogin = () => {
        // Логіка автентифікації
        navigate('/dashboard')
    }

    return <button onClick={handleLogin}>Login</button>
}
```

Переваги React Router:

– Легка інтеграція в існуючий код.
– Гнучкість у налаштуванні.
– Підтримка сучасних можливостей, таких як маршрутизація вкладених компонентів, редіректи та захищені маршрути.

Однак, оскільки маршрутизація у React реалізується через сторонні бібліотеки, початківці можуть потребувати часу для освоєння додаткових концепцій.

Angular має вбудовану систему маршрутизації, яка є частиною його ядра. Це забезпечує більш інтегрований підхід і значно полегшує налаштування маршрутизації для складних додатків. Основні можливості:

– Маршрутизація через RouterModule. Angular використовує RouterModule, який експортує функціонал маршрутизації. Маршрути описуються у вигляді конфігураційних об'єктів.

```
import { NgModule } from '@angular/core';
import { RouterModule, Routes } from '@angular/router';
import { HomeComponent } from './home/home.component';
import { AboutComponent } from './about/about.component';

const routes: Routes = [
 { path: '', component: HomeComponent },
 { path: 'about', component: AboutComponent }
];

@NgModule({
 imports: [RouterModule.forRoot(routes)],
 exports: [RouterModule]
})
export class AppRoutingModule {}
```

– Вкладені маршрути. Angular підтримує вкладені маршрути, що дозволяє створювати складні ієрархії сторінок.

```
const routes: Routes = [
 { path: 'dashboard', component: DashboardComponent, children: [
   { path: 'profile', component: ProfileComponent },
   { path: 'settings', component: SettingsComponent }
 ]}
];
```

– Захищені маршрути (Guards). Angular дозволяє створювати механізми для обмеження доступу до маршрутів через Route Guards.

```
import { Injectable } from '@angular/core';
import { CanActivate } from '@angular/router';

@Injectable({
 providedIn: 'root'
})
export class AuthGuard implements CanActivate {
 canActivate(): boolean {
   return !!localStorage.getItem('token'); // Приклад перевірки автентифікації
 }
}
```

Guard підключається до маршруту через параметр canActivate:

```
{ path: 'dashboard', component: DashboardComponent, canActivate: [AuthGuard] }
```

– Lazy Loading модулів. Angular дозволяє завантажувати модулі додатку динамічно, що корисно для оптимізації продуктивності великих проектів.

```
{ path: 'admin', loadChildren: () => import('./admin/admin.module').then(m =>
m.AdminModule) }
```

Переваги Angular Router:

– Глибока інтеграція в фреймворк.
– Зручна конфігурація через типізовані об'єкти.
– Вбудовані засоби для безпеки та оптимізації.

Таблиця 2.1

Порівняння маршрутизації у React та Angular

Критерій	React Router	Angular Router
Підхід до маршрутизації	Декларативний, базується на компонентах	Конфігураційний, інтегрований у фреймворк
Вбудованість	Потребує додаткової бібліотеки	Вбудований у фреймворк
Вкладені маршрути	Підтримуються	Підтримуються
Захищені маршрути	Ручна реалізація через хук або функції	Вбудовані Guards
Lazy Loading	Через React.lazy	Підтримується на рівні модулів
Навігація програмно	useNavigate	Вбудований сервіс Router

2.3 Управління асинхронними запитами

У сучасних веб-додатках одна з найважливіших задач — це робота з асинхронними запитами до сервера. Зазвичай, ми відправляємо HTTP-запити на сервер для отримання чи відправлення даних, і після цього працюємо з отриманими результатами. У цьому підрозділі розглянемо, як реалізовано управління асинхронними запитами в React та Angular.

2.3.1 Управління асинхронними запитами в React

2.3.1.1 Fetch API

Fetch API — це вбудоване рішення в JavaScript для виконання HTTP-запитів. Це більш низькорівневе рішення порівняно з axios, але надає багато можливостей. Основна відмінність у тому, що fetch не має внутрішньої підтримки перехоплювачів і деяких зручних особливостей, що є у axios.

Приклад використання Fetch API в React:

```
import { useEffect,useState } from 'react'

const App = () => {
   const [data, setData] = useState([])
    useEffect(() => {
       fetch('https://api.example.com/data')
           .then(response => response.json())
           .then(data => setData(data))
           .catch(error => console.error('Error fetching data:', error))
   }, [])
    return (
       <div>
           {data.length > 0 ? (
               <ul>
                   {data.map(item => (
                       <li key={item.id}>{item.name}</li>
                   ))}
               </ul>
           ) : (
               <p>Loading...</p>
           )}
       </div>
   )
}

export default App
```

2.3.1.2 Axios

Axios — це популярна бібліотека для виконання HTTP-запитів, яка дає зручний інтерфейс для взаємодії з сервером і підтримує такі важливі можливості,

як обробка помилок, перехоплювачі запитів і відповідей, підтримка JSON, налаштування тайм-аутів і багато іншого. У React її можна використовувати для відправлення запитів у функціональних компонентах або в хуках.

Приклад використання Axios в React:

```
import { useEffect,useState } from 'react'
import axios from 'axios'

const App = () => {
   const [data, setData] = useState([])
    useEffect(() => {
       axios.get('https://api.example.com/data')
           .then(response => {
               setData(response.data)
           })
           .catch(error => {
               console.error('Error fetching data:', error)
           })
   }, [])
    return (
       <div>
           {data.length > 0 ? (
               <ul>
                   {data.map(item => (
                       <li key={item.id}>{item.name}</li>
                   ))}
               </ul>
           ) : (
               <p>Loading...</p>
           )}
       </div>
   )
}

export default App
```

2.3.1.2 TanStack Query

TanStack Query (раніше відомий як React Query) — це потужна бібліотека для управління асинхронними запитами та кешування даних. Вона автоматизує багато

завдань, пов'язаних із завантаженням і зберіганням асинхронних даних, включаючи кешування, рефетчінг, синхронізацію даних, а також обробку помилок.

Переваги TanStack Query:

- Кешування відповідей для запитів, що значно зменшує навантаження на сервер.
- Автоматичне оновлення даних, коли вони змінюються.
- Автоматичне повторне завантаження даних, коли компонент знову відображається.
- Можливість зручного оброблення помилок і станів завантаження.

Приклад використання TanStack Query в React:

```
import { useQuery } from '@tanstack/react-query'

const fetchData = async () => {
   const response = await fetch('https://api.example.com/data')
   if (!response.ok) throw new Error('Network response was not ok')
   return response.json()
}

const App = () => {
   const { data, error, isLoading } = useQuery(['data'], fetchData)

   if (isLoading) return <p>Loading...</p>
   if (error) return <p>Error: {error.message}</p>

   return (
       <div>
           <ul>
               {data.map(item => (
                   <li key={item.id}>{item.name}</li>
               ))}
           </ul>
       </div>
   )
}

export default App
```

2.3.2 Управління асинхронними запитами в Angular

У Angular також є кілька рішень для роботи з асинхронними запитами. Основними є HttpClient та бібліотеки для роботи з RxJS.

2.3.2.1 HttpClient

HttpClient — це сервіс, наданий Angular, який використовує RxJS для управління асинхронними запитами. Це дуже потужний інструмент, оскільки він дозволяє використовувати всі переваги RxJS, включаючи обробку потоків даних, відписки та трансформацію запитів. Приклад використання HttpClient в Angular:

```
import { Component, OnInit } from '@angular/core';
import { HttpClient } from '@angular/common/http';

@Component({
 selector: 'app-root',
 template: `
   <div *ngIf="loading">Loading...</div>
   <ul *ngIf="data">
     <li *ngFor="let item of data">{{ item.name }}</li>
   </ul>
 `,
})
export class AppComponent implements OnInit {
 data: any[] = [];
 loading = true;
 constructor(private http: HttpClient) {}

 ngOnInit() {
   this.http.get('https://api.example.com/data').subscribe(
     (response: any) => {
       this.data = response;
       this.loading = false;
     },
     (error) => {
       console.error('Error fetching data:', error);
       this.loading = false;
     }
   );
 }
}
```

2.3.2.2 RxJS

RxJS — це бібліотека для роботи з асинхронними подіями, яка активно використовується в Angular. Вона дозволяє обробляти потоки даних, комбінувати їх, обробляти помилки, таймери та інші асинхронні операції. Приклад використання RxJS з HttpClient в Angular:

```
import { Component, OnInit } from '@angular/core';
import { HttpClient } from '@angular/common/http';
import { catchError, map } from 'rxjs/operators';
import { of } from 'rxjs';

@Component({
 selector: 'app-root',
 template: `
   <div *ngIf="loading">Loading...</div>
   <ul *ngIf="data">
     <li *ngFor="let item of data">{{ item.name }}</li>
   </ul>
 `,
})
export class AppComponent implements OnInit {
 data: any[] = [];
 loading = true;

 constructor(private http: HttpClient) {}

 ngOnInit() {
   this.http.get('https://api.example.com/data').pipe(
     map((response: any) => response),
     catchError(error => {
       console.error('Error fetching data:', error);
       this.loading = false;
       return of([]);
     })
   ).subscribe((response: any) => {
     this.data = response;
     this.loading = false;
   });
 }
}
```

2.3.3 Порівняння React і Angular у контексті асинхронних запитів

React:

- В React ми часто використовуємо axios або fetch, щоб зробити запити до сервера.
- З бібліотеками, як TanStack Query, React значно спрощує управління асинхронними запитами завдяки автоматичному кешуванню та повторному запитуванню даних.

Angular:

- У Angular стандартне рішення — це використання HttpClient, що працює на основі RxJS.
- Angular активно підтримує RxJS для роботи з потоками даних, що дозволяє обробляти складні асинхронні операції з мінімальною кількістю коду.
- Angular надає більш структурований підхід до обробки запитів, включаючи вбудовану обробку помилок, з'єднання кількох запитів та комбінування потоків через оператори RxJS.

2.4 Управління станом (state management)

Управління станом є однією з ключових тем у розробці веб-додатків, оскільки стан визначає, як додаток реагує на зміни даних. В обох фреймворках (React і Angular) існують власні підходи та інструменти для керування станом, але кожен з них має свої особливості, переваги та недоліки.

2.4.1 Підходи до управління станом у React

2.4.1.1 Локальний стан в компонентах.

В React найпростіший спосіб керувати станом — це використання локального стану в компонентах за допомогою хуків (useState). Цей підхід добре підходить для невеликих додатків або компонентів, де стан обмежений певною частиною UI.

```
const [count, setCount] = useState(0);
return <button onClick={() => setCount(count + 1)}>{count}</button>;
```

2.4.1.2 Контекстний API

Для більш масштабних додатків, де потрібно передавати стан через багато компонентів на різних рівнях, React пропонує Context API. Це дозволяє глобально управляти станом без необхідності передавати пропси на кожному рівні.

Спочатку потрібно створити Context в React і передавати стан через Provider.

```
import { createContext, useState } from 'react'

// Створюємо контекст для теми
const ThemeContext = createContext()

function App() {
   // Локальний стан для теми
   const [theme, setTheme] = useState('light')

   return (
   // Значення theme передається через ThemeContext.Provider
       <ThemeContext.Provider value={{ theme, setTheme }}>
           <Component />
       </ThemeContext.Provider>
   )
}

export default App
```

Пояснення:

- createContext() — створює новий контекст. Це дозволяє вам передавати значення через дерево компонентів без необхідності передавати пропси вручну на кожному рівні.
- useState() — використовується для керування станом, тут для зберігання поточної теми.

– ThemeContext.Provider — компонент, який "передає" значення (в нашому випадку, стан теми та функцію для її зміни) всім нащадкам, які підписалися на цей контекст.

Щоб використовувати дані з контексту всередині компонента потрібно підключитися до Context та отримати дані в компоненті, який знаходиться всередині Provider.

```
import { useContext } from 'react'

import { ThemeContext } from './App' // Імпортуємо наш контекст

function Component() {
   // Використовуємо useContext для доступу до значень з контексту
   const { theme, setTheme } = useContext(ThemeContext)

   return (
       <div>
           <h1>Current theme: {theme}</h1>
           <button onClick={() => setTheme(theme === 'light' ? 'dark' :
'light')}>
       Toggle Theme
           </button>
       </div>
   )
}

export default Component
```

Пояснення:

– useContext(ThemeContext) — хук, який дозволяє вам підключитися до контексту та отримувати значення з нього (у нашому випадку це theme і setTheme).
– setTheme() — функція, яка була передана через контекст, і вона використовується для зміни стану теми.

– ThemeContext — це контекст, який ми створили в першому прикладі, та який надає значення теми та функцію для її зміни через дерево компонентів.

Використання Context API дозволяє зручно передавати дані, не вдаючись до багатошарової передачі пропсів. Застосування useContext у компонентах робить доступ до глобального стану зручним та інтуїтивно зрозумілим. Цей підхід ідеально підходить для невеликих і середніх додатків, де потрібно обмінюватися даними між кількома компонентами на різних рівнях дерева без потреби передавати пропси вручну. Однак, для великих і складних додатків може бути корисно поєднувати Context API з іншими рішеннями для управління станом, такими як Redux або MobX, щоб зберегти масштабованість.

2.4.1.3 Зовнішнє управління станом

Для більш складних додатків React часто використовує зовнішні бібліотеки для керування станом, такі як Redux, MobX, TanStack Query або інші.

Redux — це одна з найбільш популярних бібліотек для управління станом у React. Вона дозволяє створювати store, що містить глобальний стан додатку.

У React для того, щоб надати доступ до глобального стану через Redux, потрібно використовувати Provider з бібліотеки react-redux. Цей компонент надає доступ до Redux store для всіх дочірніх компонентів.

```
import React from 'react'
import ReactDOM from 'react-dom'
import { Provider } from 'react-redux'

import App from './App'
import { store } from './store' // Це конфігурація Redux Store

ReactDOM.render(
   <Provider store={store}>
       <App />
   </Provider>,
   document.getElementById('root'),
)
```

У цьому випадку компонент Provider забезпечує доступ до store для всіх дочірніх компонентів, включаючи ті, що використовують useDispatch та useSelector для роботи з станом Redux.

Redux має чітку архітектуру на основі actions, reducers та dispatch.

```
// Створення дії та редюсера
const increment = () => ({ type: 'INCREMENT' })
const counterReducer = (state = 0, action) => {
   switch (action.type) {
   case 'INCREMENT':
       return state + 1
   default:
       return state
   }
}

// Використання Redux у компоненті
import { useDispatch, useSelector } from 'react-redux'

function Counter() {
   const count = useSelector(state => state)
   const dispatch = useDispatch()

   return (
       <div>
           <p>Count: {count}</p>
           <button onClick={() => dispatch(increment())}>Increment</button>
       </div>
   )
}
```

Для спрощення роботи з Redux використовують Redux Toolkit (RTK) — це офіційно підтримуваний набір інструментів Redux. Він зменшує кількість шаблонного коду, спрощує написання редюсерів і налаштування middleware.

```
// Створення slice з Redux Toolkit
import { configureStore,createSlice } from '@reduxjs/toolkit'

const counterSlice = createSlice({
   name:         'counter',
   initialState: 0,
   reducers:     {
       increment: state => state + 1,
   },
})

const store = configureStore({
   reducer: {
       counter: counterSlice.reducer,
   },
})

export const { increment } = counterSlice.actions

// Використання в компоненті
import { useDispatch, useSelector } from 'react-redux'

function Counter() {
   const count = useSelector(state => state.counter)
   const dispatch = useDispatch()

   return (
       <div>
           <p>Count: {count}</p>
           <button onClick={() => dispatch(increment())}>Increment</button>
       </div>
   )
}
```

Redux Toolkit дозволяє зменшити обсяг коду, який потрібно писати для конфігурації Redux, і автоматично налаштовує корисні опції, такі як redux-thunk для асинхронних операцій.

MobX — це ще одна бібліотека для управління станом, яка використовує принципи реактивного програмування. MobX автоматично відслідковує залежності і реагує на зміни стану. Він значно простіший в налаштуванні, ніж Redux, але може бути менш передбачуваним при великих і складних додатках.

Для MobX використовується провайдер, який надає доступ до стану MobX. Зазвичай, якщо ви використовуєте MobX, то створюєте провайдер, який надає контекст для стану, і обгортаєте додаток в цей провайдер.

```
import { StrictMode } from 'react'
import { Provider } from 'mobx-react' // використовуємо провайдер MobX
import { createRoot } from 'react-dom/client'

import App from './App'
import { store } from './store' // конфігурація MobX Store


createRoot(document.getElementById('root')).render(
   <StrictMode>
       <Provider store={store}>
           <App />
       </Provider>
   </StrictMode>,
)
```

У MobX немає необхідності використовувати екшени або редюсери. Стан автоматично спостерігається, і зміни стають доступними через реактивний підхід.

```
import { makeAutoObservable } from 'mobx'
import { observer } from 'mobx-react'

class CounterStore {
   count = 0

   constructor() {
       makeAutoObservable(this)
   }

   increment() {
       this.count++
```

```
    }
}

const counterStore = new CounterStore()

const Counter = observer(() => (
    <div>
        <p>Count: {counterStore.count}</p>
        <button onClick={() => counterStore.increment()}>Increment</button>
    </div>
))

export default Counter
```

TanStack Query (раніше відомий як React Query) — це бібліотека для роботи з асинхронними запитами, кешуванням, синхронізацією даних та обробкою серверних запитів. Вона значно спрощує управління запитами на сервер, не вимагаючи окремого збереження стану у Redux чи MobX.

Для TanStack Query не потрібно використовувати спеціальний провайдер для кожного конкретного запиту. Однак, необхідно обгорнути додаток в QueryClientProvider, щоб забезпечити доступ до QueryClient, який керує кешуванням та іншими аспектами асинхронних запитів.

```
import { StrictMode } from 'react'
import { QueryClient, QueryClientProvider } from '@tanstack/react-query'
import { createRoot } from 'react-dom/client'
import App from './App'

// Створюємо новий екземпляр клієнта TanStack Query
const queryClient = new QueryClient()

createRoot(document.getElementById('root')).render(
    <StrictMode>
        <QueryClientProvider client={queryClient}>
            <App />
        </QueryClientProvider>
    </StrictMode>,
)
```

Тут QueryClientProvider надає доступ до QueryClient для всіх дочірніх компонентів, що використовують useQuery або інші хуки TanStack Query для виконання асинхронних запитів.

TanStack Query забезпечує автоматичне кешування, повторне завантаження, синхронізацію та багато інших корисних функцій для роботи з серверними запитами без необхідності вручну управляти станом.

```
import { useQuery } from '@tanstack/react-query'

async function fetchData() {
   const response = await fetch('/api/data')
   return response.json()
}

function DataDisplay() {
   const { data, isLoading, error } = useQuery(['data'], fetchData)

   if (isLoading) return <div>Loading...</div>
   if (error) return <div>Error occurred</div>

   return <div>Data: {JSON.stringify(data)}</div>
}
```

2.4.2 Підходи до управління станом у Angular

2.4.2.1 Локальний стан

В Angular також можна керувати станом локально в компонентах через властивості компонентів. Проте Angular не має вбудованого механізму для роботи зі станом, як у React.

```
export class AppComponent {
 count = 0;

 increment() {
   this.count++;
 }
}
```

2.4.2.2 Сервіси для глобального стану

Основним способом управління глобальним станом в Angular є використання сервісів. Сервіси дозволяють зберігати і поширювати стан серед різних компонентів додатку через Dependency Injection (DI).

```
@Injectable({
 providedIn: 'root'
})
export class CounterService {
 private count = 0;

 increment() {
   this.count++;
 }

 getCount() {
   return this.count;
 }
}
```

2.4.2.3 RxJS і управління станом

Angular тісно інтегрований з RxJS, що дозволяє використовувати Observables для керування станом. Це дає змогу обробляти асинхронні зміни стану та легко управляти потоками даних.

```
import { BehaviorSubject } from 'rxjs';

export class CounterService {
 private countSubject = new BehaviorSubject(0);
 count$ = this.countSubject.asObservable();

 increment() {
   this.countSubject.next(this.countSubject.value + 1);
 }
}
```

2.4.2.4 NgRx

NgRx — це популярна бібліотека для керування глобальним станом в Angular, яка базується на патерні Redux. Вона використовує store, який зберігає стан додатку, а зміни стану виконуються через actions та reducers.

```
// Використання NgRx
import { Store } from '@ngrx/store';

constructor(private store: Store<{ count: number }>) {}

increment() {
 this.store.dispatch({ type: '[Counter] Increment' });
}
```

2.4.3 Порівняння управління станом в React та Angular

Гнучкість та Структурованість:

- React дає більше гнучкості при виборі інструментів для управління станом. Це дозволяє вибирати бібліотеки відповідно до конкретних потреб.
- Angular має більш структурований підхід, пропонуючи власні інструменти для управління станом через сервіси і інтеграцію з RxJS.

Інтеграція з зовнішніми бібліотеками:

- В React зовнішні бібліотеки для управління станом, такі як Redux, дуже популярні і добре документовані, але вимагають додаткового налаштування.
- В Angular використання бібліотек, таких як NgRx, є стандартною практикою для великих проектів, що може здаватися більш організованим, але також вимагає додаткових зусиль для освоєння.

Реактивність:

- Angular використовує RxJS для управління потоками даних, що надає потужний набір інструментів для асинхронних операцій.
- React через Redux або Context API також підтримує реактивність, але не має такої прямої інтеграції з реактивними потоками, як в Angular.

React пропонує велику гнучкість у виборі інструментів для управління станом, дозволяючи розробникам використовувати різні підходи, такі як локальний стан, Context API або сторонні бібліотеки, як Redux. Це підходить для проектів, де важлива свобода вибору і спрощене керування станом для невеликих додатків.

Angular має більш структурований підхід до управління станом, з використанням сервісів і RxJS для реактивного програмування. Це робить Angular кращим вибором для великих та складних проектів, де необхідна сувора організація і підтримка великої кількості асинхронних операцій.

2.5 Підтримка SSR (Server-Side Rendering)

Server-Side Rendering (SSR) — це технологія, яка дозволяє рендерити веб-сторінки на сервері перед їх відправкою клієнту. Це значно покращує продуктивність додатків, скорочує час завантаження сторінок і сприяє підвищенню SEO. У цьому розділі розглядаються особливості реалізації SSR у React та Angular.

2.5.1 Особливості використання SSR у React

React підтримує SSR за допомогою бібліотеки ReactDOMServer, яка дозволяє рендерити компоненти в HTML на сервері. Основними етапами роботи є:

- Серверний рендеринг: HTML генерується на сервері за допомогою функції renderToString, після чого надсилається клієнту.

– Гідратація: На стороні клієнта React додає інтерактивність до вже відображеного HTML. Цей процес дозволяє повністю передати управління додатком React після початкового серверного рендерингу.

Для спрощення реалізації SSR часто використовується фреймворк Next.js, який забезпечує зручні функції, як-от:

– Автоматичний серверний рендеринг: Використання методів getServerSideProps для динамічного рендерингу сторінок.
– Статичний рендеринг: Використання getStaticProps для попередньо згенерованих сторінок, що покращує продуктивність.
– Динамічне імпортування: Завантаження компонентів лише за необхідності для зменшення розміру початкового бандлу.

Ці можливості роблять Next.js ефективним інструментом для створення додатків із високими вимогами до SEO і продуктивності.

2.5.2 Особливості використання SSR у Angular

Angular підтримує SSR через бібліотеку Angular Universal, яка інтегрується у фреймворк для рендерингу сторінок на сервері. Основні аспекти реалізації SSR в Angular:

– Рендеринг на сервері. Використання Angular Universal для генерування готового HTML.
– Гідратація. Після отримання сторінки клієнт підключає бізнес-логіку до отриманого HTML.
– Робота з даними. Сервер під час рендерингу може виконувати запити до API та передавати підготовлені дані клієнту.

Angular Universal також сприяє покращенню SEO завдяки створенню оптимізованих HTML-сторінок, що полегшує їх індексацію пошуковими системами.

Обидва інструменти сприяють покращенню користувацького досвіду та SEO, але вибір залежить від специфіки проекту та вимог команди.

Таблиця 2.2

Порівняння SSR в React та Angular

Характеристика	**React (Next.js)**	**Angular (Universal)**
Легкість налаштування	**Next.js** спрощує налаштування SSR.	Необхідно використовувати **Angular Universal** для налаштування.
Підтримка SEO	Забезпечує покращене SEO завдяки SSR.	Angular Universal також підтримує SEO.
Гідратація	Використовує гідратацію для зв'язку з клієнтом.	Так само, як у React, використовується гідратація.
Вбудована підтримка	**Next.js** має вбудовану підтримку SSR.	Angular Universal вимагає додаткової конфігурації.
Підтримка асинхронних запитів	Легко інтегрується з асинхронними запитами через **getServerSideProps**.	Angular має механізм для асинхронних запитів під час SSR.

2.6 Продуктивність веб-додатків у Angular та React

Продуктивність є критичним аспектом при виборі фреймворку для розробки веб-додатків. Оскільки React і Angular мають різні підходи до рендерингу та обробки даних, їх продуктивність можна оцінювати за кількома важливими параметрами. У цьому розділі розглянемо, як кожен з фреймворків забезпечує високу продуктивність і як можна оптимізувати додатки для досягнення найкращих результатів.

2.6.1 Рендеринг та оновлення DOM

React використовує віртуальний DOM (Virtual DOM) для оптимізації рендерингу. Віртуальний DOM є легким у порівнянні з реальним DOM і дозволяє React обчислювати мінімальні зміни між старим і новим станом компонента. Коли стан компонента змінюється, React оновлює лише ту частину DOM, яка змінилася, що знижує навантаження на браузер і покращує продуктивність.

Реактивна модель оновлення також допомагає React швидко відображати зміни, не переробляючи всі елементи на сторінці.

Angular використовує свій механізм change detection, що дозволяє відслідковувати зміни в компонентах і оновлювати DOM. Angular також оптимізує рендеринг через стратегічне використання zone.js, який допомагає відслідковувати зміни в асинхронних процесах і автоматично виконувати необхідні оновлення DOM.

Однак Angular має певні проблеми з продуктивністю на великих і складних додатках, оскільки йому необхідно перевіряти всі компоненти на зміни під час кожного циклу детекції. Для оптимізації Angular використовує OnPush change detection strategy, що дозволяє зменшити кількість перевірок, але вимагає більш ручного управління.

2.6.2 Завантаження та ініціалізація

React підтримує динамічне імпортування компонентів за допомогою React.lazy та Suspense. Це дозволяє зменшити початковий розмір бандлу і завантажувати тільки необхідні компоненти в момент їх використання. Це особливо корисно для великих додатків, де користувачі можуть завантажувати тільки необхідні частини програми, а не весь код відразу.

Angular також підтримує lazy loading, що дозволяє завантажувати модулі тільки тоді, коли вони потрібні. Це знижує початковий час завантаження сторінки, оскільки Angular завантажує тільки те, що необхідно для поточної маршрутизації. Завантаження модулів можна організувати через router configuration, що дозволяє реалізувати lazy loading для окремих частин додатка.

2.6.3 Оптимізація продуктивності

Методи оптимізація продуктивності у React:

- React.memo — це інструмент оптимізації, який запобігає повторному рендерингу компонента, якщо його пропси залишилися незмінними..
- useMemo і useCallback — це React-хуки, які дозволяють зберігати обчислені значення або функції між рендерами, запобігаючи їх повторному перерахунку, якщо залежності не змінилися.
- Code splitting — за допомогою бібліотеки Webpack можна розбивати додаток на менші частини, що дозволяє завантажувати лише необхідні частини коду.

Методи оптимізація продуктивності у Angular:

- OnPush Change Detection — дозволяє зменшити кількість перевірок на зміни, обмежуючи їх лише до компонентів, де дійсно відбуваються зміни.
- trackBy в ngFor — дозволяє Angular відслідковувати елементи списку та ефективніше їх оновлювати без необхідності перерендерити весь список.

– Lazy loading — розвантаження модулів і компонентів, щоб зменшити час завантаження додатку.

2.6.4 Продуктивність при великій кількості елементів

React ефективно працює з великими списками завдяки використанню бібліотек, таких як react-window або react-virtualized. Ці бібліотеки дозволяють рендерити лише видимі елементи в списку, що значно знижує навантаження на браузер.

Angular також має вбудовані оптимізації для рендерингу великих списків. Для подібних задач можна використовувати cdk-virtual-scroll-viewport, що дозволяє відображати тільки ті елементи списку, які знаходяться в межах видимої частини екрану.

2.6.5 Продуктивність при серверному рендерингу

React забезпечує підтримку SSR через бібліотеку ReactDOMServer, що дозволяє рендерити компоненти на сервері перед відправленням їх на клієнт. Це знижує час першого завантаження та покращує SEO. Однак для того, щоб підтримувати продуктивність на високому рівні, необхідно уважно налаштовувати кешування та обробку запитів на сервері.

Angular надає власне рішення для SSR через Angular Universal, яке дозволяє виконувати рендеринг Angular-додатків на сервері. Це також дозволяє покращити SEO і зменшити час завантаження додатка, але потребує додаткових налаштувань для підтримки серверного рендерингу.

2.6.6 Продуктивність мобільних додатків

React Native дозволяє створювати мобільні додатки з використанням React. Оскільки React Native має доступ до рідних компонентів, продуктивність мобільних додатків часто є вищою, ніж у додатків, створених за допомогою веб-технологій.

Angular не має прямої підтримки для мобільних додатків, але з допомогою Ionic можна створювати гібридні мобільні додатки. Однак, у порівнянні з React Native, продуктивність таких додатків може бути нижчою через додаткові рівні абстракції.

2.7 Використання TypeScript у React та Angular

TypeScript — це надбудова над JavaScript, яка розширює можливості мови за рахунок таких функцій, як статична типізація, підтримка сучасних стандартів JavaScript, інтерфейси, розширені можливості роботи з класами, і багато іншого. Його основною перевагою є покращена безпека коду за допомогою типів, що дозволяє уникати багатьох помилок на етапі компіляції замість виконання коду.

2.7.1 Основні аспекти використання TypeScript в React

– Типізація компонентів. У React можна визначати типи для функціональних компонентів, їх пропсів та стану. Це дозволяє легко перевіряти типи на етапі розробки, уникати неочікуваних помилок при передачі даних у компоненти.

```
interface MyComponentProps {
 name: string;
 age: number;
}

const MyComponent: React.FC<MyComponentProps> = ({ name, age }) => {
 return <div>{name}, {age}</div>;
};
```

– Типізація стану. Типізація стану дає змогу визначити, які типи даних можна зберігати в стейті. Це особливо важливо, коли стан включає складні структури, такі як об'єкти або масиви.

```
const [user, setUser] = useState<{ name: string; age: number }>({ name: '',
age: 0 });
```

– Типізація функцій та подій. TypeScript дозволяє точніше визначати типи функцій, особливо при роботі з подіями або асинхронними функціями.

```
const handleClick = (event: React.MouseEvent<HTMLButtonElement>) => {
 console.log(event);
};
```

– Підтримка для бібліотек. TypeScript у React надає підтримку для бібліотек, таких як Redux, React Router і інших, завдяки наявності типів для цих бібліотек.

Переваги використання TypeScript в React:

– Більш ефективний пошук помилок на етапі розробки.
– Підвищення якості коду завдяки чітким інтерфейсам для компонентів і стану.
– Легкість у підтримці великих кодових баз.

2.7.2 Основні аспекти використання TypeScript в Angular

– Вбудованість у фреймворк. TypeScript є основною мовою для розробки в Angular, що дозволяє інтегрувати статичну типізацію без необхідності додаткової конфігурації.
– Типізація компонентів. Як і в React, компоненти в Angular визначаються за допомогою класів, але з додаванням типізації, що дозволяє чітко визначати інтерфейси для вхідних та вихідних даних.

```
import { Component, Input } from '@angular/core';

interface User {
 name: string;
 age: number;
}

@Component({
 selector: 'app-user',
```

```
 template: `<div>{{ user.name }}</div>`,
})
export class UserComponent {
 @Input() user: User;
}
```

– Сервіси та ін'єкція залежностей. TypeScript дозволяє чітко типізувати сервіси та їх методи, що полегшує роботу з ін'єкцією залежностей, надаючи розробникам можливість передбачати, які сервіси будуть передані.

```
@Injectable({
 providedIn: 'root',
})
export class UserService {
 getUser(): User {
   return { name: 'John', age: 30 };
 }
}
```

– Модулі та маршрути. Angular також використовує TypeScript для типізації модулів і маршрутів, що робить додатки більш безпечними і легшими для масштабування.
– Реактивні форми та управління станом. Для обробки форм і управління станом у Angular використовується реактивний підхід, що добре підтримується через типізацію за допомогою TypeScript.

Переваги використання TypeScript в Angular:

– Типізація всіх елементів фреймворку, зокрема компонентів, сервісів, модулів.
– Вбудована підтримка з самого початку, що робить інтеграцію TypeScript простішою.
– Легше масштабувати проект, оскільки всі частини додаються з типами з самого початку.

Таблиця 2.3

Порівняння використання TypeScript в React і Angular

Аспект	**React**	**Angular**
Початкове налаштування	Необхідно додавати налаштування TypeScript.	TypeScript є вбудованим.
Типізація компонентів	Типізація через функціональні компоненти.	Типізація через класи компонентів.
Типізація стану	Типізація стейту за допомогою useState.	Типізація через сервіси та ін'єкцію залежностей.
Типізація сервісів	За допомогою окремих файлів і типів.	Вбудована підтримка через ін'єкцію залежностей.
Інтеграція з бібліотеками	Розширена підтримка для зовнішніх бібліотек.	Інтеграція з Angular бібліотеками та сервісами.

2.8 Тестування у React та Angular

Тестування є важливою частиною розробки веб-додатків, оскільки воно допомагає виявити помилки на ранніх етапах і забезпечує стабільність роботи. React та Angular пропонують різні підходи до тестування, кожен із яких має свої особливості, інструменти та рівень інтеграції з фреймворком.

2.8.1 Тестування у React

React, як бібліотека, не включає інструменти для тестування «з коробки». Проте існує багато сторонніх бібліотек, які доповнюють екосистему React для різних типів тестування. Ось основні:

Jest

- Основний інструмент для юніт-тестування компонентів та логіки.
- Підтримує функції, як-от знімкове тестування (snapshot testing), для перевірки, чи HTML, який рендериться компонентом, залишився незмінним.
- Має інтеграцію з React Testing Library для перевірки поведінки компонентів.

React Testing Library

- Орієнтована на тестування компонентів через взаємодії користувача (натискання, введення тексту, навігація).
- Фокусується на доступності (ARIA-атрибути) і моделюванні реальної роботи програми.

Cypress

- Для end-to-end тестування, симулює дії користувача, тестує додаток у браузері.
- Часто використовується для інтеграційних та функціональних тестів.

Enzyme

– Раніше популярна бібліотека для тестування компонентів, але зараз поступово втрачає актуальність на користь React Testing Library.

2.8.2 Тестування у Angular

Angular, як фреймворк, включає потужний набір інструментів для тестування:

TestBed

– Основний API для тестування компонентів і сервісів.
– Імітує роботу Angular-застосунків у тестовому середовищі, забезпечуючи точну емуляцію залежностей.

Karma

– Використовується для запуску тестів у браузері.
– Забезпечує автоматичне виконання тестів під час змін у коді.

Protractor

– Інструмент для end-to-end тестування, побудований на WebDriver.
– Станом на сьогодні його рекомендують замінювати на Cypress або Playwright через припинення підтримки.

Jasmine

– Інтегрована бібліотека для написання тестів у Angular.
– Підтримує функції, як-от асерти, мокування та перевірка поведінки функцій.

2.8.2 Порівняння React та Angular у тестуванні

Таблиця 2.3

Порівняння React та Angular у тестуванні

Критерій	React	Angular
Інструменти для тестування	Jest, React Testing Library, Cypress, Puppeteer	TestBed, Karma, Jasmine, Protractor
Підхід	Гнучкість у виборі бібліотек, легка інтеграція	Вбудовані рішення для комплексного тестування
Тестування компонентів	Через віртуальне DOM та користувацьку взаємодію	Емуляція залежностей і реального середовища
E2E тестування	Cypress, Playwright	Пропонують Cypress або Playwright

2.9 Мобільна розробка у React та Angular

Мобільна розробка є важливим аспектом сучасних додатків, оскільки користувачі часто використовують мобільні пристрої. React і Angular пропонують різні підходи до розробки мобільних додатків, включаючи адаптивний дизайн і інтеграцію з фреймворками для створення нативних рішень.

2.9.1 Мобільна розробка у React

React забезпечує широкий вибір інструментів для мобільної розробки:

- React Native. Фреймворк для створення нативних додатків для iOS та Android з використанням єдиної кодової бази. Підтримує нативний UI та доступ до платформних API.
- Expo. Інструменти для швидкого запуску та тестування React Native-додатків без складної конфігурації.
- PWA (Progressive Web Apps). Прогресивні веб-додатки, які виглядають і працюють як нативні, створюються з використанням сучасних підходів і бібліотек, що забезпечують підтримку PWA.

2.9.2 Мобільна розробка у Angular

Angular також має кілька потужних рішень для мобільної розробки:

- Ionic Framework. Фреймворк для створення кросплатформених мобільних додатків із використанням Angular та веб-технологій. Підтримує доступ до нативних API через Capacitor або Cordova.
- NativeScript. Дозволяє створювати нативні додатки з Angular, використовуючи нативні компоненти для UI.
- PWA: Angular підтримує прогресивні веб-додатки завдяки Angular Service Worker та зручним інструментам CLI.

2.9.3 Порівняння підходів мобільної розробки у React та Angular

React пропонує більший вибір для створення нативних рішень (React Native, Expo), тоді як Angular орієнтується на інтеграцію з існуючими фреймворками (Ionic, NativeScript). Обидва підходи підтримують створення PWA, що дозволяє використовувати веб-додатки як мобільні.

Таблиця 2.4

Порівняння аспектів мобільної розробки у React та Angular

Характеристика	React (React Native)	Angular (Ionic, NativeScript)
Продуктивність	Висока, завдяки використанню нативних компонентів.	Помірна, залежить від використання веб-технологій.
Нативні можливості	Повний доступ до нативних API через модулі.	Потребує додаткових бібліотек, таких як Capacitor.
Легкість навчання	Простий для розробників, знайомих із React.	Потребує глибшого розуміння Angular.
Кросплатформеність	Підтримка iOS та Android через одну кодову базу.	Також підтримує різні платформи.
Спільнота	Велика, завдяки популярності React Native.	Значна, особливо серед Angular-розробників.

2.10 Гнучкість і можливість компонування

2.10.1 Гнучкість у React

React забезпечує високу гнучкість завдяки своїй компонентній архітектурі, яка дозволяє легко створювати багаторазово використовувані компоненти. Кожен компонент є незалежною одиницею, що може взаємодіяти з іншими через props та state. Завдяки цьому розробники можуть створювати складні інтерфейси,

розбиваючи їх на дрібніші компоненти, що легко тестуються, модифікуються та повторно використовуються.

- Компонентний підхід: React орієнтований на функціональні компоненти, які є простими та зрозумілими для розробника. Компоненти можна поєднувати і вкладати один в одного, утворюючи гнучку ієрархію.
- Hooks: React надає можливість використовувати хуки (наприклад, useState, useEffect), які дозволяють ефективно керувати станом всередині компонентів, а також створювати власні хуки для повторного використання логіки між компонентами.
- Context API: Для обміну станом між компонентами, які знаходяться на різних рівнях ієрархії, React використовує Context API. Це забезпечує ще більшу гнучкість у компонентному побудові і управлінні станом.
- Styled Components та CSS-in-JS: React активно підтримує стильові компоненти за допомогою бібліотек, таких як styled-components або emotion. Це дозволяє стилізувати компоненти на рівні JS, забезпечуючи більшу модульність і гнучкість у використанні стилів.

2.10.2 Гнучкість в Angular

Angular також підтримує компонентний підхід, однак у порівнянні з React він має дещо жорсткішу архітектуру, яка включає декілька стандартних шаблонів і обов'язкових структур. Однак ця структура допомагає забезпечити консистентність і масштабованість додатків.

- Компонентний підхід: В Angular кожен компонент має свій шаблон, стилі та логіку. Це дозволяє зручно структурувати додаток, роблячи компоненти легкою одиницею для повторного використання.
- Directives: Angular надає можливість додавати поведінку до елементів DOM через директиви. Директиви можуть змінювати вигляд або поведінку елементів, забезпечуючи більшу гнучкість.

- Dependency Injection (DI): Angular широко використовує ін'єкцію залежностей для розв'язування залежностей між компонентами. Це дозволяє створювати більш гнучкі та розширювані компоненти, які не залежать від конкретних реалізацій інших частин додатку.
- NgModules: Структура проекту в Angular базується на використанні модулів, які об'єднують різні компоненти, сервіси та інші ресурси. Це дозволяє групувати компоненти за їх функціональністю, полегшуючи керування складними додатками.

2.10.3 Можливості компонування

2.10.3.1 Реюзабельність компонентів

В обох фреймворках компонентний підхід дає можливість створювати модульні компоненти, які можна багаторазово використовувати в різних частинах додатку.

- Створення функціональних компонентів в React дозволяє легко їх перевикористовувати. Завдяки хукам та контексту, компоненти можуть бути легко адаптовані для різних завдань. Наприклад, компонент кнопки, який приймає стилі та функцію обробника події через props, можна використовувати в різних місцях додатку без змін.
- В Angular також компоненти можна повторно використовувати в різних частинах додатку. Завдяки директивам та модулям, компоненти можуть бути легко змінені або доповнені новими функціональностями. Кожен компонент має свою власну логіку, шаблон і стилі, що робить їх більш незалежними і самодостатніми.

2.10.3.2 Спільне використання стейту

У React спільне використання стану між компонентами здійснюється за допомогою таких інструментів як Context API, Redux або TanStack Query. Вони

дозволяють організувати централізоване зберігання стану та забезпечують ефективну взаємодію між різними частинами додатку.

В Angular для управління станом зазвичай використовуються NgRx або BehaviorSubject через RxJS. Це дозволяє організовувати асинхронні запити та управляти станом додатку з використанням потоку даних, що забезпечує зручне компонування.

2.10.3.3 Інтеграція з іншими бібліотеками

Як React, так і Angular мають велику екосистему додаткових бібліотек, які дозволяють розширювати можливості компонентів. У React часто використовуються бібліотеки для стилізації компонентів, управління глобальним станом, маршрутизації тощо. У Angular екосистема включає різноманітні бібліотеки для роботи з формами, інтернаціоналізацією, маршрутами та іншими аспектами розробки.

2.11 Архітектурна організація проекту

Архітектурна організація проекту є одним із ключових аспектів при розробці веб-додатків. Вибір архітектурного підходу визначає, як будуть структуровані файли, компоненти, як буде здійснюватися взаємодія між різними частинами системи, і як буде зберігатися стан додатку. У цьому розділі розглянемо підходи до архітектурної організації проекту в React та Angular, порівнюючи їх за різними критеріями.

2.11.1 Архітектура проекту в React

React дозволяє розробникам вибирати між різними архітектурними підходами, залежно від складності проекту та вимог до нього. React не нав'язує

конкретної організації структури проекту, тому кожен розробник або команда може вирішити, як саме буде організовано їх додаток.

2.11.1.1 Компонентний підхід

React базується на компонентному підході, де основним елементом є компонент, що має свою логіку, вигляд і часто свої стилі. Компоненти можуть бути:

- Функціональними: Використовуються для представлення UI і логіки в простому вигляді, де відсутнє зберігання стану всередині компонента, за винятком використання хуків.
- Класовими: Старіший підхід, який використовує класи для створення компонентів. Стан і життєвий цикл обробляються через методи класів.

Структура компонентів дозволяє легко розділяти додаток на невеликі, самодостатні одиниці, що можна повторно використовувати.

2.11.1.2 Розподіл відповідальностей

У React застосовується принцип єдиної відповідальності, що означає, що кожен компонент має відповідати тільки за одну частину логіки чи UI. Це дозволяє забезпечити кращу підтримуваність і тестованість коду. Основні стратегії включають:

- Presentational and Container Components: Розподілення компонентів на ті, що відповідають тільки за відображення даних (presentational), і ті, що займаються бізнес-логікою та управлінням даними (container).
- Hooks: Встановлення логіки управління станом і ефектами поза компонентами у власні хуки для повторного використання.

2.11.1.3 Управління станом

В React для організації глобального стану можна використовувати різні стратегії, в залежності від складності проекту:

- Context API: Для невеликих додатків можна використовувати контекст для збереження глобальних даних, таких як налаштування користувача або мова.
- Redux: Для більш складних проектів часто використовують Redux для централізованого зберігання стану. Redux Toolkit значно спрощує процес налаштування Redux.
- TanStack Query: Для асинхронних запитів даних з API, можна використовувати TanStack Query, який спрощує роботу з даними, автоматизує кешування та оновлення UI.

2.11.1.4 Підхід до стилізації

В React існує багато способів стилізувати компоненти, серед яких:

- CSS-in-JS: Бібліотеки, такі як styled-components і emotion, дозволяють писати CSS безпосередньо в JavaScript, зберігаючи стильові налаштування в контексті компонентів.
- CSS Modules: Для більшої ізоляції стилів використовуються CSS-модулі, де кожен файл CSS стає локальним для компонента.

2.11.2 Архітектура проекту в Angular

Angular забезпечує більш структурований підхід до організації проекту, ніж React. Він слідує принципам MVVM (Model-View-ViewModel) або MVC (Model-View-Controller), надаючи чітке розмежування між бізнес-логікою і UI.

2.11.2.1 Компонентний підхід

Як і в React, основними одиницями в Angular є компоненти. Кожен компонент складається з трьох частин:

- Шаблон (HTML): Визначає зовнішній вигляд компонента.
- Клас (TS): Логіка компонента, що обробляє дані і керує їх відображенням.
- Стилі (CSS): Стилі, що застосовуються до шаблону компонента.

Angular компоненти також можуть бути комбіновані і створювати більші структури додатку.

2.11.2.2 Модулі (NgModules)

Angular застосовує концепцію модулів, що дозволяє розбивати проект на функціональні частини. Модулі можуть містити компоненти, сервіси, директиви, пайпи та інші елементи. Модулі забезпечують організовану структуру проекту і дають змогу керувати залежностями між частинами додатку.

- CoreModule: Містить компоненти, сервіси і модулі, що використовуються в додатку на всіх його етапах.
- SharedModule: Містить спільні компоненти, директиви та пайпи, які можуть бути повторно використані в різних частинах додатку.
- FeatureModule: Модулі, що відповідають за конкретні функціональні області.

2.11.2.3 Dependency Injection

Angular активно використовує Dependency Injection (DI), що дозволяє ін'єктувати сервіси, компоненти та інші залежності в інші частини додатку без необхідності створювати їх вручну. Це забезпечує високу гнучкість і спрощує тестування.

2.11.2.4 Управління станом

Angular зазвичай використовує NgRx для управління глобальним станом, що базується на концепціях Redux. NgRx використовує потоки даних (Observables) для роботи з асинхронними операціями і забезпечує централізоване управління станом.

2.11.3. Вибір архітектури для проекту

При виборі архітектури для проекту потрібно враховувати кілька факторів, таких як:

- Масштаб проекту: Якщо проект невеликий і не потребує складного управління станом, React з використанням Context API або TanStack Query може бути ідеальним вибором.
- Підхід до структури проекту: Якщо проект вимагає чіткої організації, таких як поділ на модулі, сервіси, і компоненти, Angular з його ін'єкцією залежностей і модулями може бути більш підходящим.
- Підтримка масштабованості: Для великих додатків з численними компонентами і складною логікою краще використовувати Angular, оскільки він надає вбудовані інструменти для масштабування і організації великого проекту.

2.12 Загальна академічність

Загальна академічність у контексті розробки веб-додатків включає в себе аналіз того, як фреймворки та бібліотеки, такі як React та Angular, враховують сучасні парадигми програмування та архітектурні патерни. Цей розділ порівнює, яким чином ці фреймворки підтримують важливі концепції програмування та забезпечують правильну архітектуру для побудови масштабованих, підтримуваних і ефективних веб-додатків.

2.12.1 Парадигми програмування в React та Angular

React підтримує функціональне програмування (FP), зокрема через компонентний підхід, який дозволяє створювати чисті та передбачувані функції. Компоненти в React є функціями, які приймають дані як аргументи і повертають JSX, що дозволяє легко здійснювати тестування та повторне використання компонентів. Реактивний підхід також інтегрується з такими інструментами, як useState, useEffect і useReducer, що дозволяє застосовувати концепції FP для керування станом і побічними ефектами.

Angular, з іншого боку, більше орієнтований на об'єктно-орієнтоване програмування (ООП). Компоненти в Angular є класами, що використовують декоратори для визначення поведінки. Це дозволяє використовувати успадкування, інтерфейси та ін'єкцію залежностей, що є типовими концепціями ООП. Angular також активно підтримує патерни, такі як Dependency Injection (DI), що дозволяє покращити тестування та знижує зв'язність компонентів.

2.12.2 Патерни проектування

Приклади архітектурних патернів, підтримуваних у React:

- Компонентно-орієнтований підхід. Кожен компонент є ізольованим модулем, який відповідає за певну частину інтерфейсу. Це дозволяє легко тестувати, змінювати та підтримувати код, застосовуючи принципи модульності та повторного використання.
- Реактивне програмування через використання хуків, таких як useState, useEffect, та useReducer, що дозволяє ефективно управляти станом та асинхронними операціями.
- Контейнерно-презентаційний патерн. Компоненти можуть бути поділені на "контейнери", які відповідають за управління станом, і "презентаційні" компоненти, які зосереджуються лише на відображенні.

Приклади архітектурних патернів, підтримуваних в Angular:

- Модульний підхід. Angular активно використовує модулі для структурування додатків. Модулі дозволяють об'єднувати компоненти, сервіси, директиви та інші елементи в одну логічну одиницю, що спрощує розробку та підтримку великих додатків.
- MVC (Model-View-Controller). Хоча Angular не є строго MVC-фреймворком, він підтримує подібну архітектуру, де контролери відповідають за логіку, моделі — за дані, а вигляд (view) визначається компонентами. Це дозволяє розділяти відповідальність і знижує зв'язність.

– Dependency Injection (DI) — цей патерн є основою архітектури Angular. DI дозволяє компонуванню залежностей без створення тісних зв'язків між модулями та класами, покращуючи тестованість і масштабованість.

2.12.3 Розширюваність і підтримка

І React, і Angular пропонують потужні механізми для розширення функціональності за допомогою патернів, таких як HOC (Higher-Order Components) у React і Декоратори в Angular. Це дозволяє створювати більш гнучкі та налаштовувані компоненти, що можуть бути адаптовані до різних сценаріїв використання.

– У React застосовуються HOC, що дозволяє модифікувати або розширювати компоненти, не змінюючи їх самі. Це також допомагає в повторному використанні логіки.
– У Angular використовуються декоратори (наприклад, @Component, @Injectable), які дозволяють додавати додаткову функціональність класам або методам, змінюючи їх поведінку.

2.12.4 Загальна академічність архітектур

У контексті розробки веб-додатків важливо також зазначити, що обидва фреймворки підтримують найкращі практики програмування та архітектурного проектування, зокрема:

– Розділення відповідальності. У React це реалізується через компонентно-орієнтований підхід, а в Angular — через модульну організацію коду.
– Масштабованість і тестованість. Оскільки обидва фреймворки дозволяють ізолювати бізнес-логіку, компоненти та модулі, це полегшує тестування та масштабування великих додатків.

Враховуючи ці принципи React та Angular надають потужні інструменти для створення надійних і ефективних веб-додатків, що відповідають вимогам сучасних стандартів програмування та архітектурних патернів.

3 ПОРІВНЯННЯ REACT ТА ANGULAR ЗА БІЗНЕС КРИТЕРІЯМИ

3.1 Простота використання

React і Angular є двома найпопулярнішими фреймворками для розробки веб-додатків, але з різним підходом до організації розробки. React фокусується на компонентному підході, що забезпечує високу гнучкість у проектуванні додатків. Його API є мінімалістичним і гнучким, що дозволяє використовувати різні підходи для роботи з даними, управлінням станом і компонентами. Це дає розробникам свободу вибору, але водночас вимагає більше часу на прийняття рішень про архітектуру.

Angular, на відміну від React, є більш комплексним фреймворком, який надає все необхідне для розробки веб-додатків "з коробки", включаючи маршрутизацію, управління станом, інтерсептори, валідацію форм і багато інших функцій. Це може значно спростити процес розробки, оскільки більшість рішень вже готові і інтегровані в екосистему. Однак, Angular може здатися важким і менш гнучким через свою "важку" архітектуру та велику кількість правил і стандартів.

3.2 Поріг входу (навчання)

React має менший поріг входу. Нові розробники можуть швидко почати працювати з ним, оскільки React концентрується на UI-компонентах і є досить простим для розуміння в плані концепцій. Однак, як тільки проект стає більш масштабним, починають виникати складнощі у виборі інструментів для управління

станом, маршрутизації та інших важливих аспектів розробки. Для цього потрібно мати більше досвіду або використовувати додаткові бібліотеки.

Angular вимагає більш глибокого розуміння. Вивчення Angular включає не тільки основи JavaScript, а й знання TypeScript, інтерфейсів, сервісів, модулів і двостороннього зв'язку даних. Хоча Angular має багато вбудованих функцій, його навчання може бути складним для початківців, особливо якщо вони не знайомі з більш строгими патернами програмування або типами даних.

3.3 Швидкість розробки

React дозволяє розробникам швидко розпочати створення додатків, але при цьому він потребує додаткових бібліотек і інструментів для управління складними задачами. Це означає, що за певних обставин розробка може займати більше часу, оскільки розробники повинні самостійно інтегрувати потрібні функції.

Angular пропонує більше вбудованих рішень, що дозволяє зекономити час на інтеграції різних функцій. Це робить швидкість розробки в Angular вищою, оскільки вам не потрібно витрачати час на підключення сторонніх бібліотек. Однак, початковий час на налаштування і навчання може бути більшим, оскільки фреймворк потребує більш глибокого розуміння.

3.4 Популярність на ринку

React є більш популярним у порівнянні з Angular, і його частка на ринку продовжує зростати. Це пояснюється його гнучкістю, широким вибором бібліотек

і відносною простотою, а також підтримкою багатьох великих компаній, таких як Facebook (що створив React), Instagram і WhatsApp.

Angular є менш популярним за React, але продовжує використовуватися великими компаніями, зокрема Google, Microsoft і іншими, які створили великі додатки на Angular. Angular часто вибирають для складних корпоративних додатків, де потрібна чітка структура і більша кількість вбудованих функцій.

3.5 Доступність розробників на ринку

React має велику кількість розробників, і це робить його вибір більш вигідним з точки зору залучення спеціалістів. Оскільки React є популярним і має велике співтовариство, знайти досвідчених розробників React на ринку праці буде легше, ніж для Angular.

Angular також має достатньо фахівців, але їхня кількість на ринку дещо менша через складніший поріг входу і менш широку популярність у порівнянні з React. Тим не менше, великі компанії, що використовують Angular, часто шукають спеціалістів саме в цій області.

3.6 Кваліфікація спеціалістів

React дозволяє розробникам мати різний рівень кваліфікації. Хоча початковий рівень може бути досить низьким, для вирішення складних завдань (наприклад, для великих, масштабованих додатків) потрібен досвід і знання додаткових інструментів (наприклад, Redux, TypeScript, або системи тестування).

Angular вимагає від розробників більш високого рівня кваліфікації, оскільки фреймворк має більше вбудованих концепцій і патернів. Розробники Angular повинні добре розуміти TypeScript, знати концепції компонентів, сервісів і залежностей, а також працювати з більш складними архітектурними рішеннями.

3.7 Вартість спеціалістів на ринку

React фахівці зазвичай мають середню вартість на ринку праці, оскільки їх велика кількість і попит на них досить високий. Це робить зарплати конкурентними і доступними для більшості компаній.

Angular розробники можуть бути дорожчими, оскільки їх менше, і для того, щоб працювати з Angular, потрібні більш кваліфіковані спеціалісти. Це може означати, що компаніям буде складніше знайти недорогих фахівців для роботи з Angular.

ВИСНОВКИ

У роботі проведено глибоке дослідження двох популярних інструментів для розробки веб-додатків — React та Angular. Аналіз показав, що обидва фреймворки мають свої переваги, недоліки та сферу застосування залежно від типу проекту.

React, який є бібліотекою для створення інтерфейсів користувача, демонструє вищий рівень популярності серед розробників. Згідно з даними опитування State of Javascript 2023: Клієнтські фреймворки, 64.7% респондентів, які працювали з React, позитивно оцінюють цей інструмент, тоді як лише 19.7% мають негативний досвід. Це свідчить про його високу адаптивність та зручність для використання в широкому спектрі проектів, особливо тих, які потребують компонентного підходу та гнучкої архітектури. З іншого боку, 6.35% респондентів, які мають лише теоретичні знання про React, оцінили його позитивно, що підкреслює доступність навчання, хоча 8.92% теоретичних оцінок залишаються негативними.

Angular, як повноцінний фреймворк, показує більш суперечливі результати. Лише 19.63% респондентів позитивно оцінюють Angular після роботи з ним, тоді як 26.21% висловлюють негативний досвід. Значна частина розробників, які знайомі з Angular теоретично, висловили негативну оцінку (40.97%), що може бути пов’язано зі складністю вивчення та використання фреймворку порівняно з React. Проте Angular залишається ефективним вибором для масштабних корпоративних проектів завдяки своїй суворій структурі та інтегрованим інструментам, що зменшує потребу у зовнішніх залежностях.

Порівняння React та Angular з точки зору бізнесу дає зрозуміти, що вибір між цими фреймворками залежить від ряду факторів. React виграє у простоті, швидкості розробки та доступності розробників. Він має більшу популярність на

ринку, що забезпечує легший доступ до спеціалістів. Водночас, Angular є більш комплексним і вимагає від розробників більш високої кваліфікації, але може бути кращим вибором для великих і складних проектів, де потрібна чітка архітектура та більше вбудованих функцій. Обираючи між цими технологіями, компанії повинні враховувати вимоги до проекту, доступність розробників і бюджет.

З огляду на отримані дані, технічним лідерам, архітекторам програмного забезпечення та керівникам проектів варто обирати технологію залежно від вимог проекту, масштабів продукту, структури команди та доступних термінів розробки. React може бути кращим вибором для швидкого запуску стартапів, створення мінімально життєздатних продуктів (MVP) або комерційних продуктів, які потребують високої гнучкості та простоти інтеграції. Angular, натомість, більше підходить для довгострокових корпоративних рішень із суворими вимогами до архітектури, оскільки надає широкий набір інструментів «з коробки».

Для правильного вибору технології важливо враховувати потенційний життєвий цикл продукту, очікувані обсяги підтримки та оновлень, а також наявність розробників із відповідною експертизою. Використання React може знизити початкові витрати на розробку, але вимагатиме більшої уваги до вибору та інтеграції сторонніх бібліотек. Angular забезпечує стабільність і узгодженість процесів розробки, що є цінним для великих команд і тривалих проектів. Таким чином, усвідомлений вибір технології сприятиме створенню конкурентоспроможного комерційного продукту, що відповідатиме поточним потребам ринку.

ПЕРЕЛІК ПОСИЛАНЬ

1. [Електронний ресурс] Офіційний сайт React. https://reactjs.org.
2. [Електронний ресурс] Офіційний сайт Angular. https://angular.io.
3. [Електронний ресурс] Angular vs React: An In-depth Comparison https://www.simform.com/blog/angular-vs-react/
4. [Електронний ресурс] React vs Angular: What Should You Choose? https://maybe.works/blogs/react-vs-angular.
5. [Електронний ресурс] Angular vs React: A Detailed Comparison. https://www.cleveroad.com/blog/angular-vs-react/.
6. [Електронний ресурс] Which to choose for your app: Angular vs React. https://medium.com/@Quickway_Infosystems/which-to-choose-for-your-app-angular-vs-react-bd640b552990.
7. [Електронний ресурс] Angular vs React: A Detailed Comparison. https://hygraph.com/blog/angular-vs-react.
8. [Електронний ресурс] Angular vs React: The Ultimate Comparison Guide. https://geekflare.com/angular-vs-react/.
9. [Електронний ресурс] Офіційна документація React Router. https://reactrouter.com/home.
10. [Електронний ресурс] Офіційний сайт Redux. https://redux.js.org/.
11. [Електронний ресурс] Офіційний сайт Redux Toolkit. https://redux-toolkit.js.org/.
12. [Електронний ресурс] Офіційна документація RxJS. https://rxjs.dev/.
13. [Електронний ресурс] Офіційний сайт Tanstack Query. https://tanstack.com/query/latest.
14. [Електронний ресурс] Why React Query? https://ui.dev/why-react-query.
15. [Електронний ресурс] Офіційна документація Axios. https://axios-http.com/docs/intro.
16. [Електронний ресурс] Офіційний сайт React Hook Form. https://www.react-hook-form.com/get-started/.
17. [Електронний ресурс] State of Javascript 2023: Клієнтські фреймворки https://2023.stateofjs.com/ua-UA/libraries/front-end-frameworks/

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНФОРМАЦІЙНИХ СИСТЕМ ТА ТЕХНОЛОГІЙ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Дослідження та порівняння методів розробки веб-додатків на Angular та React»

Виконав:

здобувач вищої освіти — Сергій СТАДНИК
група ІСДМ-63

Керівник: — Каміла СТОРЧАК
науковий ступінь, вчене звання — *д.т.н., професор*

Київ 2025

Мета дослідження: Аналіз та порівняння методів розробки веб-додатків за допомогою Angular та React з урахуванням технічних і бізнес-критеріїв для обґрунтованого вибору технології для різних типів проектів.

Об'єкт дослідження: Сучасні фреймворки для розробки веб-додатків.

Предмет дослідження: Особливості, переваги та недоліки Angular і React у контексті веб-розробки.

Завдання: Дослідження та порівняння методів розробки веб-додатків на Angular та React

Компоненти в React

```jsx
import { useState, useEffect } from 'react';
import styles from './Posts.module.scss'

const Posts = (props) => {
    const [posts, setPosts] = useState([]);

    useEffect(() => {
        props.getPosts().then((newPosts) => {
            setPosts(newPosts);
        }).catch((error) => {
            console.error('Error fetching posts:', error);
        });
    }, []);

    return (
        <div className={styles.container}>
            <div>Posts list</div>
            {posts.map(post => (
                <div key={post.id}>
                    <h2 className={styles.postTitle}>{post.title}</h2>
                    <p>{post.text}</p>
                </div>
            ))}
        </div>
    )
}

export default Posts;
```

Компоненти в Angular

posts.component.ts

```typescript
import { Component, OnInit } from '@angular/core';
import { PostsService } from './posts.service';

@Component({
  selector: 'app-posts',
  templateUrl: './posts.component.html',
  styleUrls: ['./posts.component.scss']
})
export class PostsComponent implements OnInit {
  posts: Array<{ id: number; title: string; text: string }> = [];
  constructor(private postsService: PostsService) {}

  ngOnInit(): void {
    this.postsService.getPosts().subscribe({
      next: (newPosts) => {
        this.posts = newPosts;
      },
      error: (err) => {
        console.error('Error fetching posts:', err);
      }
    });
  }
}
```

posts.component.html

```html
<div class="container">
    <div>Posts list</div>
    <div *ngFor="let post of posts" class="post">
      <h2 class="post-title">{{ post.title }}</h2>
      <p>{{ post.text }}</p>
    </div>
</div>
```

МАРШРУТИЗАЦІЯ У REACT

```jsx
import { BrowserRouter as Router, Routes, Route, Navigate } from 'react-router-dom';
import checkAuthStatus from 'src/utils/auth';
import Home from 'src/pages/Home';
import Login from 'src/pages/Login';
import Dashboard from 'src/pages/Dashboard';
import Settings from 'src/pages/Settings';

const ProtectedRoute = ({ children }) => {
  return checkAuthStatus() ? children : <Navigate to="/login" replace />;
};

const App = () => {
  return (
    <Router>
      <Routes>
        <Route path="/" element={<Home />} />
        <Route path="/login" element={<Login />} />

        <Route
          path="/dashboard/*"
          element={
            <ProtectedRoute>
              <Routes>
                <Route path="" element={<Dashboard />} />
                <Route path="settings" element={<Settings />} />
              </Routes>
            </ProtectedRoute>
          }
        />
      </Routes>
    </Router>
  );
};

export default App;
```

МАРШРУТИЗАЦІЯ У ANGULAR

app.routes.ts

```ts
import { Routes } from '@angular/router';
import { HomeComponent } from './pages/home/home.component';
import { LoginComponent } from './pages/login/login.component';
import { DashboardComponent } from './pages/dashboard/dashboard.component';
import { SettingsComponent } from './pages/settings/settings.component';
import { canActivateAuth } from './auth/access.guard';

export const routes: Routes = [
  { path: '', component: HomeComponent },
  { path: 'login', component: LoginComponent },
  {
    path: 'dashboard',
    canActivate: [canActivateAuth], // Захист маршруту через Guard
    children: [
      { path: '', component: DashboardComponent },
      { path: 'settings', component: SettingsComponent },
    ],
  },
];
```

access.guard.ts

```ts
import { inject } from "@angular/core"
import { AuthService } from "./auth.service"
import { Router } from "@angular/router"

export const canActivateAuth = () => {
    const isLoggedIn = inject(AuthService).isAuth

    if (isLoggedIn) {
        return true
    }

    return inject(Router).createUrlTree(['/login'])
}
```

УПРАВЛІННЯ ЗАПИТАМИ ТА СТАНОМ

Асинхронні запити

JS Fetch API

AXIOS

TanStack Query

Вбудований HttpClient сервіс

Управління станом

(state management)

Вбудовані Context API,
useState, useReducer

RxJS

Redux

Mobx

NGRX

Reactive State for Angular

Akita

ДОДАТКОВІ ТЕХНІЧНІ АСПЕКТИ

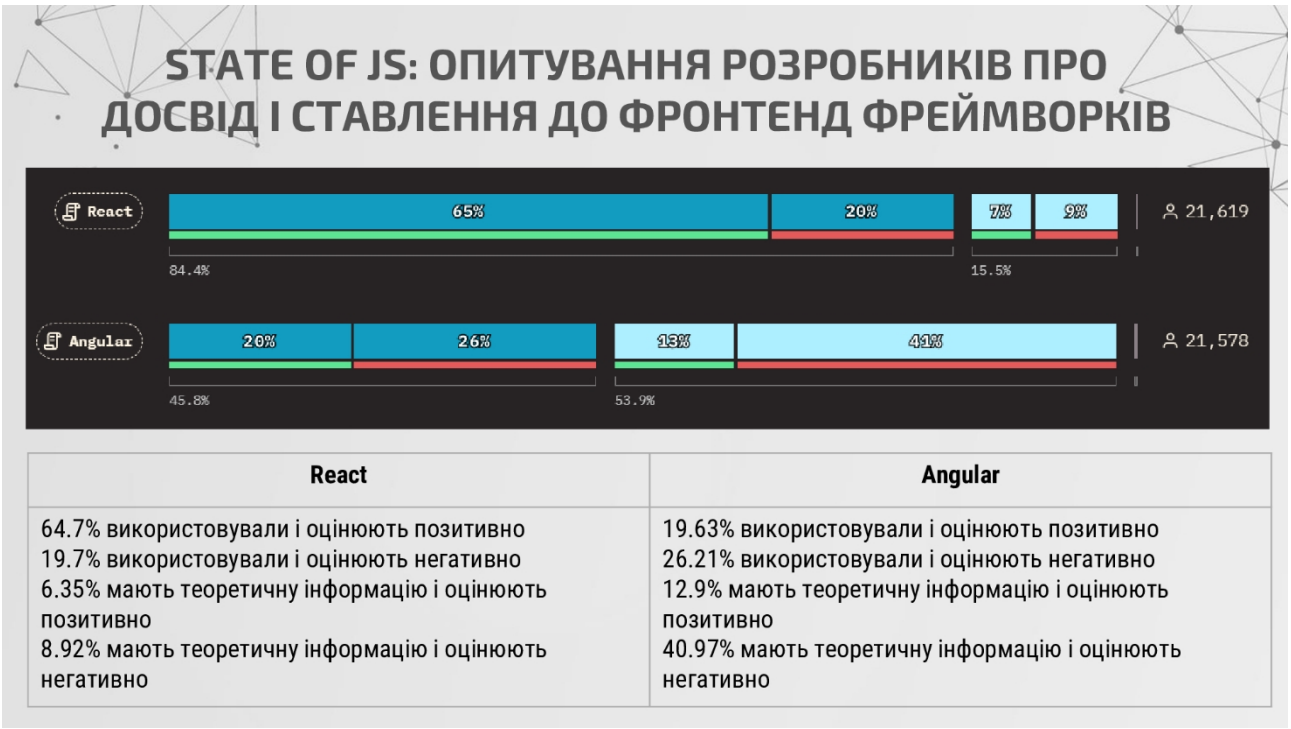
Аспект	React	Angular
Мова	JavaScript. TypeScript опціонально	TypeScript
Розширюваність і підтримка	HOC (Higher-Order Components)	Використання декораторів @Component, @Injectable
Продуктивність	Virtual DOM, useCallback, useMemo, React.memo	Change Detection, компілятор Ahead-of-Time (AOT)
Підтримка SSR (Server-Side Rendering)	Використання бібліотеки Next.js для зручної реалізації SSR	Вбудована підтримка SSR
Тестування	Використання Jest, Testing Library, Cypress	Вбудовані інструменти для юніт-тестів і end-to-end тестування
Мобільна розробка	Використання React Native для створення кросплатформних мобільних додатків	Використання Ionic Framework або NativeScript для створення кросплатформних мобільних додатків із доступом до нативних функцій

ЗАГАЛЬНЕ ПОРІВНЯННЯ

Критерій	React	Angular
Розробка та підтримка	Facebook	Google
Архітектура	Заснована на компонентах — гнучка та модульна, ідеальна для створення багаторазових UI компонентів.	Суворо структурований фреймворк, який підтримує архітектуру MVVM (Model-View-ViewModel).
Компонування	Дозволяє використовувати зовнішні бібліотеки (axios, Redux, React Router, React Hook Form).	Має комплексний підхід із вбудованими інструментами (HtpClient, RxJS, Router, Forms).
Гнучкість	Висока гнучкість завдяки мінімальній кількості вбудованих обмежень.	Має сувору структуру, що забезпечує стабільність і передбачуваність, але зменшує гнучкість у налаштуванні.
Крива навчання	Легше і швидше вивчити, особливо для тих, хто має знання JavaScript, HTML та CSS.	Крутіша крива навчання обумовлена більшою складністю фреймворка, обов'язковим використанням TypeScript та необхідністю освоєння додаткових концепцій

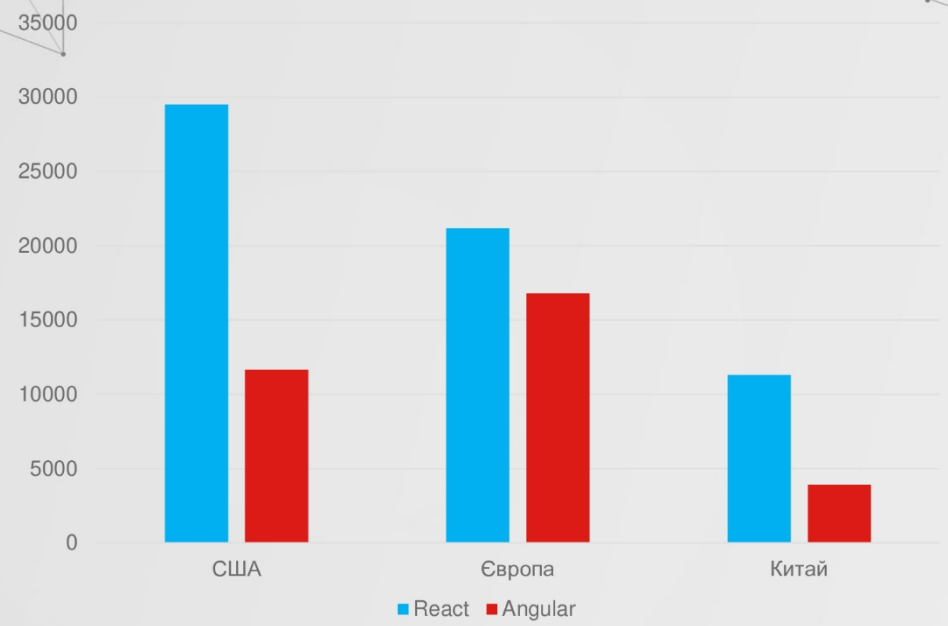
ПРОСТОТА ВИКОРИСТАННЯ, ПОРІГ ВХОДУ, ШВИДКІСТЬ РОЗРОБКИ

Аспект	React	Angular
Простота використання	Легка для розуміння, завдяки компонентному підходу та JSX.	Більш складний синтаксис через використання директив, служб та модулів.
Поріг входу	Низький: можна швидко почати навіть з базовими знаннями JavaScript, HTML та CSS. Велика кількість ресурсів для новачків (документація, спільнота, курси).	Високий: потрібно вивчати TypeScript, RxJS та специфіку Angular архітектури. Вичерпна документація, але потребує більше часу для освоєння.
Швидкість розробки	- Швидка розробка завдяки простоті компонентів і модульності. - Простий старт, адже можна додавати лише потрібні бібліотеки поступово. - Ідеальний для швидкого прототипування та MVP проектів.	- Повільніша розробка через складну структуру додатків і вимоги до архітектури. - Потребує більше часу через обов'язкове використання директив, служб і модулів. - Стартовий час довший, але зручний для масштабних корпоративних проектів.



React	Angular
64.7% використовували і оцінюють позитивно 19.7% використовували і оцінюють негативно 6.35% мають теоретичну інформацію і оцінюють позитивно 8.92% мають теоретичну інформацію і оцінюють негативно	19.63% використовували і оцінюють позитивно 26.21% використовували і оцінюють негативно 12.9% мають теоретичну інформацію і оцінюють позитивно 40.97% мають теоретичну інформацію і оцінюють негативно

КІЛЬКІСТЬ ВАКАНСІЙ НА LINKEDIN



ВИСНОВКИ

React — популярна і гнучка бібліотека, що дозволяє швидко адаптуватися до змін. Завдяки високій популярності та великій кількості доступних інструментів, він підходить для розробки як невеликих стартапів, MVP так і великих масштабних рішень.

Angular забезпечує велику кількість функцій «з коробки», що робить його більш підходящим для довготривалих корпоративних проектів з суворими вимогами до архітектури і стабільності коду. Він підходить для складних систем, де важливі стандарти і чітка структура.

Вибір між технологіями залежить від вимог проекту: масштабу, складності, архітектурних вимог, доступності розробників та бюджету.

ДЯКУЮ ЗА УВАГУ