ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ АВТОМАТИЗОВАНИХ СИСТЕМ

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

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

Виконав: Кухарчук Артем Сергійович
здобувач вищої освіти
група ІСДМ-61

Керівник: Аліна ТУШИЧ
науковий ступінь, вчене звання доктор філософії, доцент

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

Київ 2023

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення автоматизованих систем

Ступінь вищої освіти Магістр

Спеціальність Інформаційні системи та технології

Освітньо-професійна програма Інформаційні системи та технології

ЗАТВЕРДЖУЮ
Завідувач кафедрою ІПЗАС

_____ Каміла СТОРЧАК
«_____» _____ 2023 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Кухарчук Артем Сергійович
(прізвище, ім'я, по батькові здобувача)

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

керівник кваліфікаційної роботи Аліна ТУШИЧ доктор філософії, доцент,
(Ім'я, ПРІЗВИЩЕ науковий ступінь, вчене звання)
затверджені наказом Державного університету інформаційно-комунікаційних технологій від «19» 10.2023р. №145

2. Строк подання кваліфікаційної роботи «29» грудня 2023р.

3. Вихідні дані до кваліфікаційної роботи: науково-технічна література, технічна документація центрів обробки даних, вимоги до систем моніторингу ЦОД.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
Дослідження принципів розробки на Angular та React
Аналіз та порівняння розробки на Angular та React

Розробка макету та створення веб-додатків на React та Angular

5. Перелік графічного матеріалу: *презентація*
    1. Використання Angular та React для створення веб-додатків
    2. Технічні аспекти розробки веб-додатків на React та Angular
    3. Прототипування користувацького інтерфейсу веб-додатку
    4. Розробка веб-додатків на React та Angular
    5. Порівняння React та Angular

6. Дата видачі завдання «19» жовтня 2023 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз наявної науково-технічної літератури	19.10-05.11.23	
2	Вивчення теоретичних основ застосування React та Angular у веб-розробці	05.11-12.11.23	
3	Дослідження технічних аспектів застосування React та Angular у веб-розробці	13.11-18.11.23	
4	Аналіз розробки на React та Angular	19.11-23.11.23	
5	Розробка додатків на React та Angular	24.11-03.12.23	
6	Прототипування мобільного додатку курсів англійської мови	04.12-10.12.23	
7	Оформлення роботи: вступ, висновки, реферат	11.12-20.12.23	
8	Розробка демонстраційних матеріалів	21.12-29.12.23	

Здобувач вищої освіти _____ Артем КУХАРЧУК
(підпис) *(Ім'я, ПРІЗВИЩЕ)*

Керівник
кваліфікаційної роботи _____ Аліна ТУШИЧ
(підпис) *(Ім'я, ПРІЗВИЩЕ)*

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня магістра: 89 стор.,40 Рисунок, 30 джерел.

Мета цієї науково-дослідницької роботи полягає в систематичному та об'єктивному аналізі, порівнянні та оцінці методів розробки веб-додатків, використовуючи два відомі фреймворки - Angular і React.

Об'єктом дослідження є процес створення веб-додатків.

Предметом дослідження є порівняння та оцінка ефективності використання фреймворків Angular та React під час розробки веб-додатків.

Короткий зміст дослідження - у даній роботі проводиться аналіз способів створення веб-додатків на основі фреймворків Angular та React. Вона охоплює теоретичне вивчення основних принципів кожного з фреймворків, дослідження моделі компонентів та стратегій управління станом програми. Також вона включає порівняльний аналіз, виконаний через створення реальних додатків на обох платформах, з оцінкою їх продуктивності, можливості масштабування та зручності використання. Результатом є об'єктивні висновки про переваги кожного фреймворку та рекомендації для вибору найбільш відповідного залежно від конкретних потреб у розробці веб-додатків.

КЛЮЧОВІ СЛОВА: ВЕБ - ДОДАТКИ, ANGULAR, REACT, ПОРІВНЯННЯ.

ABSTRACT

The textual part of the qualification work for obtaining an educational master's degree: 89 pages, 40 figures, 30 sources.

The purpose of this research work is to systematically and objectively analyze, compare and evaluate methods for developing web applications using two well-known frameworks - Angular and React.

The object of research is the process of creating web applications.

The subject of the study is to compare and evaluate the effectiveness of using Angular and React frameworks in the development of web applications.

Summary of the study - this work analyzes the ways of creating web applications based on the Angular and React frameworks. It covers the theoretical study of the basic principles of each of the frameworks, the study of the model of components and strategies for managing the state of the program. It also includes a comparative analysis performed through the creation of real applications on both platforms, with an assessment of their performance, scalability and usability. The result is objective conclusions about the benefits of each framework and recommendations for choosing the most suitable one depending on the specific needs for developing web applications.

KEYWORDS: WEB - APPLICATIONS, ANGULAR, REACT, COMPARISON.

ЗМІСТ

ВСТУП

Актуальність теми: Тема стала актуальною внаслідок стрімкого розвитку галузі веб-розробки та постійних оновлень технологій. Angular та React уважаються ключовими фреймворками, що широко застосовуються у цій сфері. Порівняльний аналіз їх методів дозволяє виявити переваги й обмеження кожного, що допомагає розробникам вибрати найбільш підходящий інструмент для своїх проектів. Це особливо важливо в умовах стрімкого росту веб-індустрії та потреби у швидких, ефективних та масштабованих веб-додатках. Дослідження відображає сучасні тенденції у галузі веб-технологій та сприяє вдосконаленню процесу розробки.

Мета дослідження полягає в систематичному та об'єктивному аналізі, порівнянні та оцінці методів розробки веб-додатків, використовуючи два відомі фреймворки - Angular і React.

Завдання дослідження включають:

- Теоретичний аналіз;
- Дослідження моделей компонентів;
- Вивчення стратегій управління станом;
- Практичний порівняльний аналіз через розробку додатків;
- Оцінка результатів.

Об'єктом дослідження є процес створення веб-додатків.

Предметом дослідження є порівняння та оцінка ефективності використання фреймворків Angular та React під час розробки веб-додатків.

Методи дослідження:

- Аналіз літератури та документів
- Експериментальні дослідження
- Анкетування та опитування
- Аналіз результатів тестування

- Порівняльний аналіз коду

Апробація результатів магістерської роботи: Кухарчук А. С. «Дослідження та порівняння методів розробки веб-додатків на Angular та React». Тези доповіді на Всеукраїнській Науково-технічній конференції «Сучасний стан та перспективи розвитку IoT». – Київ, 18 квітня 2023 р.

Публікації: Кухарчук А. С. «Дослідження та порівняння методів розробки веб-додатків на Angular та React». Стаття у загальногалузевому науково-виробничому журналі «Зв'язок», м.Київ - №1, 2024. – С.185-201.

Науковою новизною є застосування новітніх технологій для поліпшення процесу створення веб-додатків з використанням фреймворків React та Angular. Вона проводить докладний порівняльний аналіз функціональних можливостей обох фреймворків, зокрема їхню продуктивність, ефективність та швидкість розробки. Крім того, досліджується потенціал цих фреймворків у великих проектах. Робота висвітлює різноманітні методи створення компонентів та їх використання для розробки веб-додатків і проводить аналіз практичних сценаріїв застосування обох фреймворків у реальних умовах. Отримані результати можуть служити основою для вибору найбільш підходящого інструментарію при створенні веб-додатків, враховуючи їхні специфічні вимоги та потреби.

1 ТЕОРЕТИЧНІ ОСНОВИ ВЕБ-ФРЕЙМВОРКІВ ТА БІБЛІОТЕК: ANGULAR І REACT

1.1 Огляд основних принципів та функціоналу Angular

Angular - це відкритий фреймворк, сприяння якому надається компанією Google. Його архітектура ґрунтується на модель-відображення-контролер (MVC), що спрощує процес розробки та тестування. Розглянемо цю архітектуру більш детальніше:

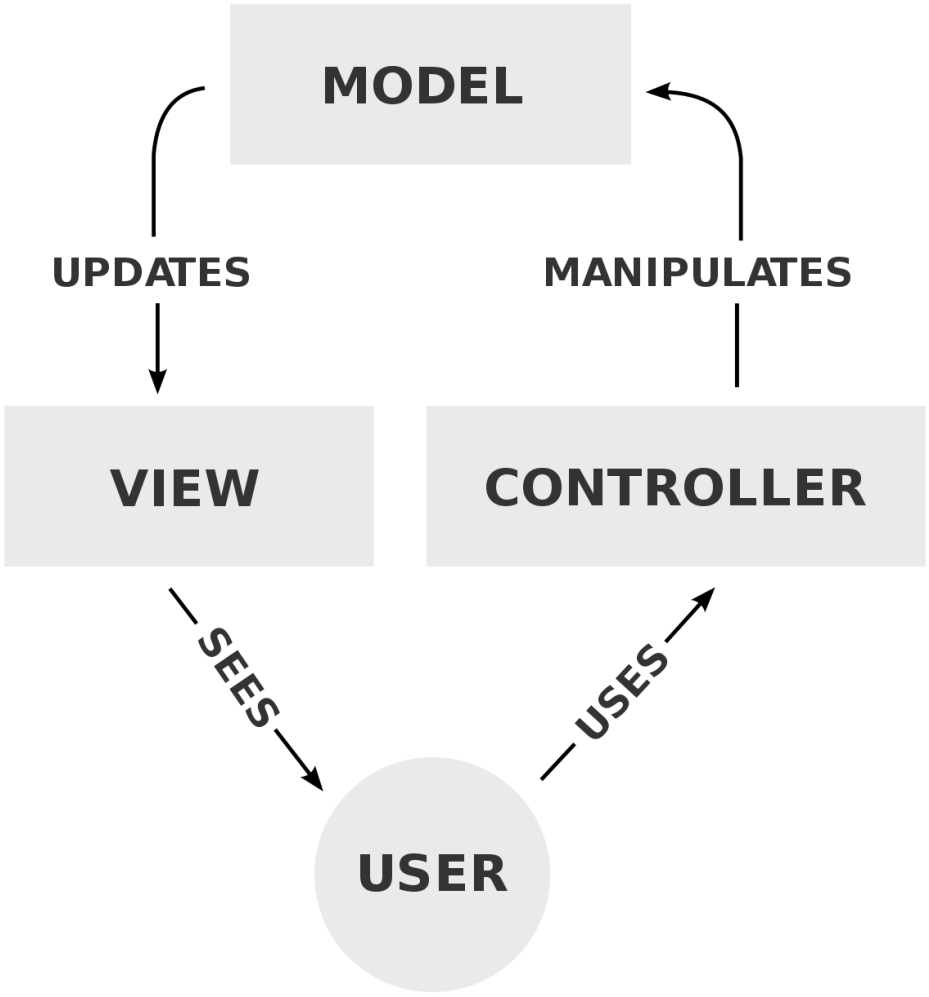


Рисунок 1.1 MVC модель[1]

- Модель в архітектурі MVC представляє собою компонент, який зосереджений на управлінні даними та бізнес-логікою додатку. Це може бути клас або об'єкт, який визначає структуру та поведінку даних. В основному, модель забезпечує методи для читання та запису даних, а також операції, які можна виконати з цими даними, такі як валідація, обчислення та зберігання.

Модель не має прямого зв'язку з інтерфейсом користувача, що робить її незалежною від того, як дані відображаються або вводяться користувачем. Це сприяє відокремленню логіки обробки даних від представлення, забезпечуючи більшу гнучкість та легкість підтримки коду.

Важливо визначити, що модель в MVC може включати не лише дані, але і правила їхньої обробки. Вона відповідає за забезпечення коректності та цілісності даних, що визначається бізнес-логікою застосунку. Таким чином, модель стає центральним елементом, який керує даними та відображає основну логіку додатку в архітектурі MVC.

- У паттерні архітектури MVC, відображення виступає як інтерфейс, через який користувач має можливість взаємодіяти з додатком та отримує інформацію. Однією з ключових функцій відображення є відображення даних, які представлені в моделі, у зручній та зрозумілій формі, такій як текст, графіка чи елементи інтерфейсу.

Важливою роботою відображення є також збір введених користувачем даних та їх подальша передача контролеру для оброблення. Хоча воно може взаємодіяти з контролером через методи чи події, важливо зауважити, що пряма взаємодія з моделлю не передбачена. Це розділення логіки представлення від логіки оброблення даних робить код більш гнучким та легким для утримання.

Крім того, відображення може відслідковувати події, що генеруються моделлю або контролером, і реагувати на них. Це забезпечує реактивний підхід та дозволяє динамічно оновлювати інтерфейс у відповідь на зміни в додатку.

Отже, відображення в MVC функціонує як місце взаємодії з користувачем та відображенням інформації, зберігаючи при цьому прозору відокремленість від моделі та взаємодіючи з контролером для оброблення користувацьких взаємодій.

- У контексті архітектурного паттерну MVC, контролер грає важливу роль у взаємодії між користувачем, моделлю та відображенням. Цей компонент відповідає за обробку введених користувачем даних та керування наступними кроками в системі.

Контролер приймає введення від користувача, що може бути подією або іншою формою взаємодії, і визначає, які конкретні дії повинні бути виконані відповідно до цього введення. Наприклад, він може співпрацювати з моделлю для отримання чи збереження даних, а також запускати оновлення відображення для відображення змін.

Роль контролера також полягає в реалізації логіки застосунку, що допомагає відокремити бізнес-логіку від інтерфейсу користувача. Це сприяє забезпеченню чистоти та розширюваності коду.

Отже, у паттерні архітектури MVC контролер виконує важливу функцію — він не тільки обробляє введення користувача, а й керує взаємодією та координацією між моделлю та відображенням, що робить його центральним елементом для успішної роботи всієї системи.

Ця структура дозволяє зручно масштабувати веб-додатки на Angular, роблячи їх оптимальними для різноманітних проектів.

Рисунок 1.2 Angular [2]

Розробник Angular відіграє ключову роль у формуванні сучасних та функціональних веб-додатків, діючи як головний архітектор та будівельник фронтенду проєкту. Його завдання включає розробку компонентів - самостійних модулів, які об'єднують HTML-шаблони, CSS-стилі та TypeScript-код, формуючи користувацький інтерфейс програми. Основна майстерність полягає у розбитті складних завдань на менші компоненти для полегшення супроводу проєкту та підвищення перевикористання коду.

Крім того, важливим аспектом роботи є використання сервісів. Розробник використовує їх для надання загальної функціональності, доступної для використання в різних компонентах додатка. Це допомагає відокремлювати логіку від компонентів і забезпечує чистоту коду. Вміння створювати, інтегрувати сервіси та використовувати ін'єкцію залежностей є ключовим для ефективної роботи з ними.

Розробник Angular повинен мати навички реагування на користувацькі взаємодії та управління HTTP-запитами. Це означає вміння

обробляти введення користувачів, виконувати асинхронні операції і створювати динамічні додатки, які реагують на зміни.

Навички роботи з маршрутизацією є також важливими: визначення шляхів, налаштування навігації та керування станами додатка.

Для успішної роботи розробника Angular потрібне глибоке розуміння JavaScript і TypeScript, а також базові знання HTML і CSS. Досвід у асинхронному програмуванні і розуміння принципів роботи HTTP-запитів також є важливими. Володіння інструментами командного рядка і Angular CLI спрощує автоматизацію розробки.

Також потрібні навички налагодження і профілювання коду для швидкого виявлення та виправлення помилок. Написання чистого і оптимізованого коду, дотримання принципів SOLID (Рисунок 1.2) і компонентної архітектури, використання стандартів кодування та кращих практик розробки допомагають створювати масштабовані та легко підтримувані додатки.

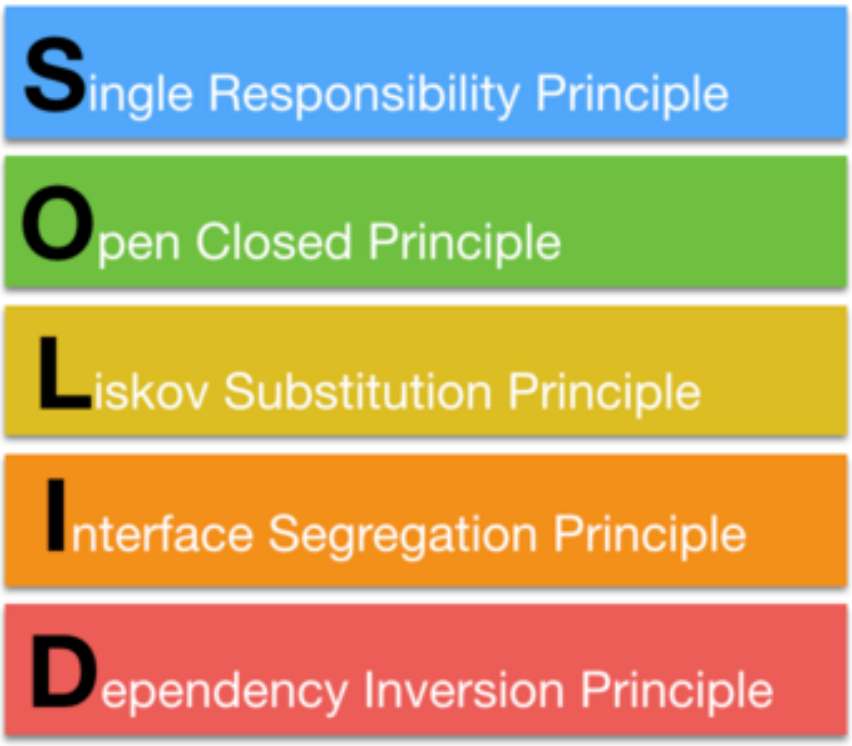


Рисунок 1.3 Принципи SOLID [3]

- Принцип єдиної відповідальності (Single Responsibility Principle):
  Кожен клас повинен відповідати лише за одну обов'язковість або завдання, тобто мати лише одну причину для зміни. Кожен клас має виконувати лише конкретну функцію чи завдання.
- Принцип відкритості/закритості (Open/Closed Principle):
  Програмні сутності, такі як класи, модулі та функції, повинні бути відкритими для розширення новою функціональністю, але закритими для змін. Це означає, що змінювати існуючий код не повинно бути потрібно для внесення нового функціоналу.
- Принцип підстановки Ліскова (Liskov Substitution Principle):
  Об'єкти базового класу повинні бути замінюваними об'єктами його похідних класів без порушення коректності програми. Це означає, що поведінка об'єкта може бути заміщена поведінкою його підкласів без негативних наслідків.
- Принцип інтерфейсу (Interface Segregation Principle):
  Клієнти не повинні залежати від методів інтерфейсів, які вони не використовують. Тобто, класи повинні залежати лише від тих інтерфейсів, які вони реально використовують, і не повинні бути прив'язані до зайвих методів.
- Принцип залежностей (Dependency Inversion Principle):
  Модулі верхнього рівня не повинні залежати від модулів нижнього рівня. Обидва рівні повинні залежати від абстракцій, а не від конкретних деталей реалізації. Крім того, самі абстракції не повинні залежати від деталей, а навпаки - деталі повинні залежати від абстракцій.

Angular - це потужний інструмент для створення односторінкових додатків (SPA), які працюють без перезавантаження сторінки.

Основні переваги та можливості Angular можна описати так:

1. Компонентна архітектура: його архітектура ґрунтується на концепції компонентів - це невеликі, автономні блоки коду, які можна використовувати знову та легко управляти ними. Це сприяє розділенню додатка на зручні частини, спрощує розробку та підтримку.
2. Двостороннє зв'язування даних: Angular дозволяє зв'язувати дані між різними частинами додатка - компонентами та шаблонами. Це дозволяє реалізувати динамічні зміни даних у користувацькому інтерфейсі при зміні даних у внутрішній моделі додатка. Такий підхід робить додаток більш гнучким та зручним у використанні.
3. Сервіси та ін'єкція залежностей сприяють зручнішому використанню окремих частин програми. Сервіси дозволяють створювати загальні функції, а ін'єкція залежностей дозволяє компонентам отримувати доступ до цих функцій без необхідності їх ручного створення.
4. Маршрутизація в Angular дає можливість зв'язувати різні частини програми з різними URL-адресами, що дозволяє переміщатися між різними сторінками без перезавантаження всієї програми.
5. Цикл життя компонентів - кожна частина програми має свої етапи життя, під час яких можна виконувати певні дії.
6. Angular підтримує два типи компіляції - JIT (Just-In-Time) та AOT (Ahead-Of-Time), що впливає на продуктивність програми та розмір її коду.

Angular побудовано на базі TypeScript (Рисунок 1.3), який розширює можливості JavaScript і гарантує сумісність з будь-яким коректним кодом JavaScript. TypeScript пропонує більш компактний синтаксис порівняно зі звичайним JavaScript і, завдяки типізації, допомагає виявляти помилки ще на етапі написання коду. Хоча вивчення TypeScript потребує деякого часу, це інвестиція в ефективність роботи в подальшому.

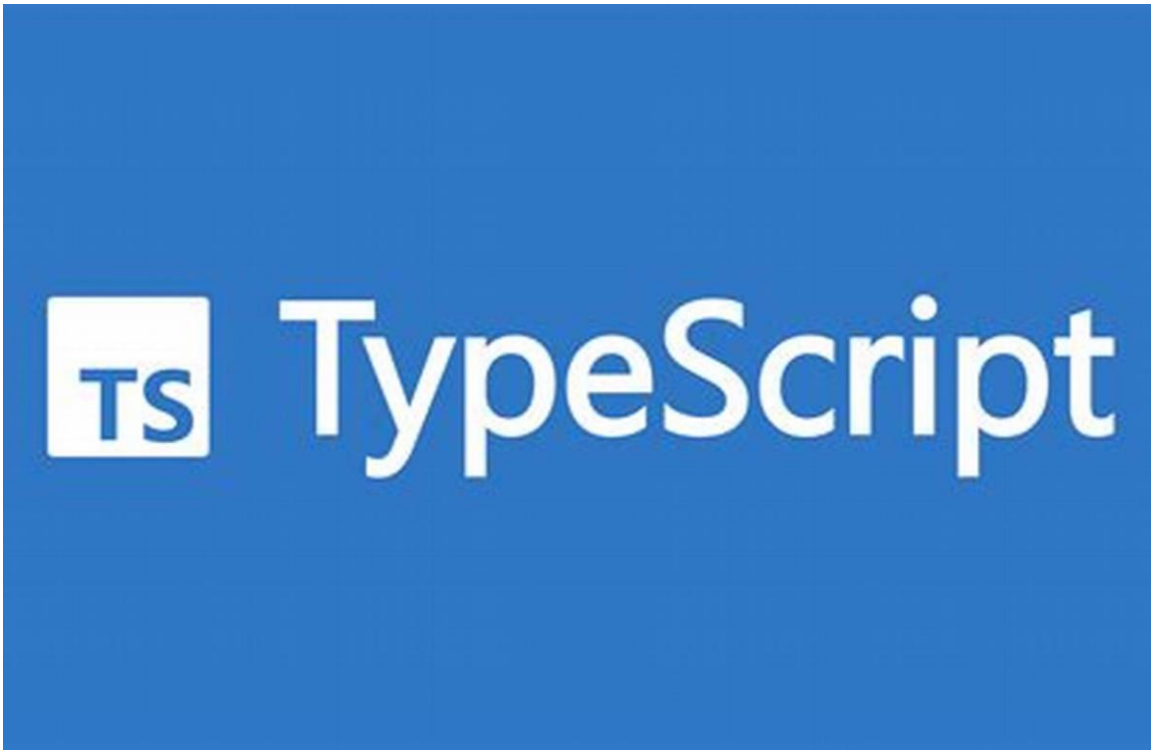


Рисунок 1.4 TypeScript [4]

У свою чергу, у Angular компоненти виступають основними будівельними блоками. Кожен компонент містить TypeScript клас, що використовує декоратор @Component(), шаблон HTML та стилі.

Клас — це місце, де реалізується логіка, необхідна для компонента. Це може бути реалізація функцій, обробники подій, властивості та посилання на сервіси. Зазвичай класи розташовуються у файлах з іменами у форматі feature.component.ts, наприклад, footer.component.ts, login.component.ts або user-info.component.ts. Компоненти створюються за допомогою декоратора @Component(), який містить метадані для Angular, щоб вказати, де знаходяться HTML та CSS для цього компонента.

Елементи можуть успадковувати загальні стилі з файлу styles.css, розташованого в основній папці програми, і використовувати або перевизначати їх за власними потребами. Можна встановлювати конкретні стилі для елемента безпосередньо в декораторі @Component() або вказувати шлях до файлу CSS.

На жаль, занурення у світ Angular - це складна і обширна справа, яка має свою власну філософію. Це може становити виклик для новачків, що вимагає часу і звикання.

Однак для успішного освоєння цього фреймворка необхідно володіти певними знаннями і вміннями в декількох ключових областях:

- Написання застосунків на Angular найефективніше здійснюється на TypeScript, що вимагає хорошого розуміння цієї мови програмування. Використання сучасного JS (ES6) також можливе, але менш поширене.
- Освоєння TypeScript, як розширення JS, також є важливим аспектом.
- Для прискорення процесу розробки важливо опанувати Angular CLI.
- Встановлення Angular та інших компонентів зазвичай виконується через npm, вимагаючи відповідних знань.
- Знання налаштування таск-раннерів, таких як Gulp або Grunt, може бути корисним для виконання різноманітних операцій перед розгортанням додатка в продакшн.
- Наразі широко використовуються інструменти для мініфікації, як-от UglifyJS, і пакувальники, наприклад, Webpack.

Рисунок 1.5 Webpack [5]

- Важливо мати навички налагодження коду в процесі розроблення програми з використанням інструментів налагодження, наприклад, Augury.
- Тестування Angular-застосунків, які часто є складними, вимагає знання інструментів, таких як Jasmine (фреймворк для тестування) і Protractor (для наскрізного тестування).

Це показує, що для створення клієнтських веб-застосунків необхідно опанувати безліч інструментів і методик. Однак не варто впадати у відчай: існує безліч онлайн-ресурсів для вивчення всіх цих інструментів і методів. Хоча це потребує часу, ви отримаєте цінний досвід і зможете легко створювати складні додатки.

Також важливо враховувати, що іноді використання Angular може бути зайвим. Для невеликих або середніх проєктів без складного користувацького інтерфейсу та взаємодії може бути доцільніше вдатися до звичайного JavaScript. Оцінка вимог, функціональності нового застосунку та дедлайнів є ключовою перед ухваленням рішення про використання JS-фреймворку.

1.2 Огляд основних принципів та функціоналу React

React JS () – це JavaScript-бібліотека, розроблена Facebook, яка служить для створення користувацьких інтерфейсів. Цей інструмент добре відомий серед розробників своєю високою популярністю та здатністю створювати високоефективні та масштабовані програми. У React JS ключове значення мають компоненти – це окремі блоки коду, які відповідають за відображення певної частини користувацького інтерфейсу.

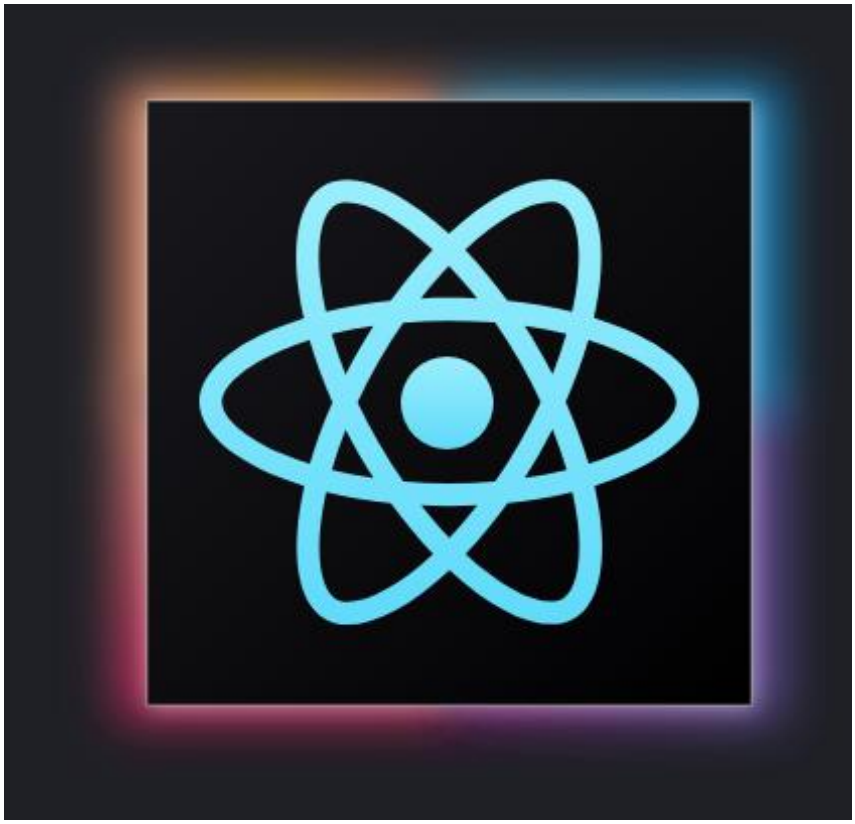


Рисунок 1.6 React JS [6]

React – це засіб для створення веб-додатків, який акцентує на користувацькому інтерфейсі. Головна його мета полягає в створенні інтерактивних, динамічних і відгукних інтерфейсів для користувачів. Ця технологія дозволяє розробляти програми, які швидко відображаються та мають плавні переходи між сторінками, створюючи при цьому багатофункціональні та взаємодіючі додатки.

Крім того, React має свої переваги:

1. Використання віртуального DOM та ефективних алгоритмів оновлення дозволяє уникнути зайвих змін у реальному DOM, що покращує продуктивність додатків.

2. Архітектура React, заснована на компонентах, сприяє розділенню інтерфейсу на незалежні частини. Це спрощує розробку, тестування та

підтримку коду, оскільки компоненти можна використовувати повторно та змінювати з легкістю.

3. React підтримує односторонній потік даних, що робить управління станом додатків простішим та передбачуваним. Це полегшує відлагодження та тестування застосунків.

4. Велика та активна спільнота розробників, що використовують React, забезпечує безліч ресурсів, бібліотек та інструментів для розробки. Крім цього, існує багато сторонніх бібліотек, які підтримують React і допомагають розширити його можливості.

Також ще однією перевагою, є можливість використання розширення JSX (Рисунок 1.3)

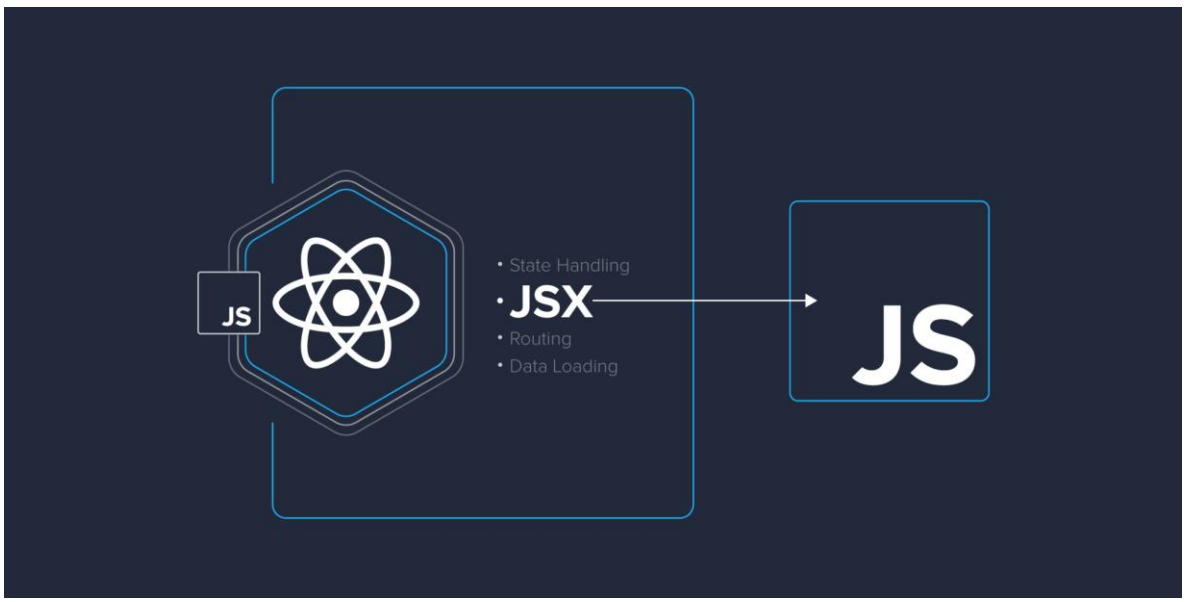


Рисунок 1.7 Розширення JSX [7]

JSX є розширенням JavaScript, яке дозволяє вставляти структуру інтерфейсу користувача безпосередньо в код за допомогою синтаксису, схожого на HTML. Використання JSX в React надає значну гнучкість та виразність при розробці інтерфейсів.

Однією з ключових переваг JSX є його декларативний підхід, що дозволяє описувати бажаний стан інтерфейсу, не замислюючись над конкретними операціями для його оновлення. Цей синтаксис робить код більш зрозумілим і лаконічним, полегшуючи розуміння для розробників.

Ще однією важливою характеристикою JSX є його тісна інтеграція з JavaScript. Це означає, що розробники можуть використовувати всі можливості JavaScript безпосередньо в коді JSX, роблячи його потужним інструментом для впровадження складних функціональностей та динамічних властивостей.

Завдяки JSX, створення компонентів та взаємодія з ними стає інтуїтивно зрозумілим і зручним для розробників, що робить його важливою складовою під час розробки веб-інтерфейсів з використанням React.

React - чудовий інструмент для створення проектів будь-якого розміру. Ця технологія дає можливість зручно розширювати та використовувати компоненти, легко поєднується з іншими бібліотеками та фреймворками. Крім цього, React підтримує серверний рендеринг, що сприяє прискоренню завантаження сторінок та оптимізації їх для пошукових систем.

У React є всі потрібні можливості для ефективної розробки потужних та швидких інтерфейсів. Вона спрощує роботу з компонентами, управлінням станом додатків та має значну підтримку від спільноти розробників. Ця бібліотека ідеально підходить для колективної розробки завдяки своєму уніфікованому інтерфейсу та шаблону робочого процесу.

React - потужний інструмент для створення проектів будь-якого розміру. Ця технологія дозволяє легко розширювати та використовувати компоненти, безпроблемно поєднується з іншими бібліотеками та

фреймворками. Крім того, React підтримує серверний рендеринг, що сприяє швидкому завантаженню сторінок та їх оптимізації для пошукових систем.

У React є всі необхідні можливості для ефективної розробки потужних та швидких інтерфейсів. Вона спрощує роботу з компонентами, управлінням станом додатків та має значну підтримку від спільноти розробників. Ця бібліотека ідеально підходить для колективної розробки завдяки своєму уніфікованому інтерфейсу та шаблону робочого процесу.

Розробники використовують різні інструменти для створення, тестування, відлагодження та оптимізації React-додатків. Вони обирають засоби залежно від конкретних вимог проекту та особистих уподобань. Наприклад:

1. React Developer Tools: це розширення для браузера, що надає розробникам інструменти для інспектування, відлагодження та профілювання React-додатків. Воно дозволяє переглядати ієрархію компонентів, аналізувати їх стан та властивості, а також перевіряти швидкість оновлення компонентів.

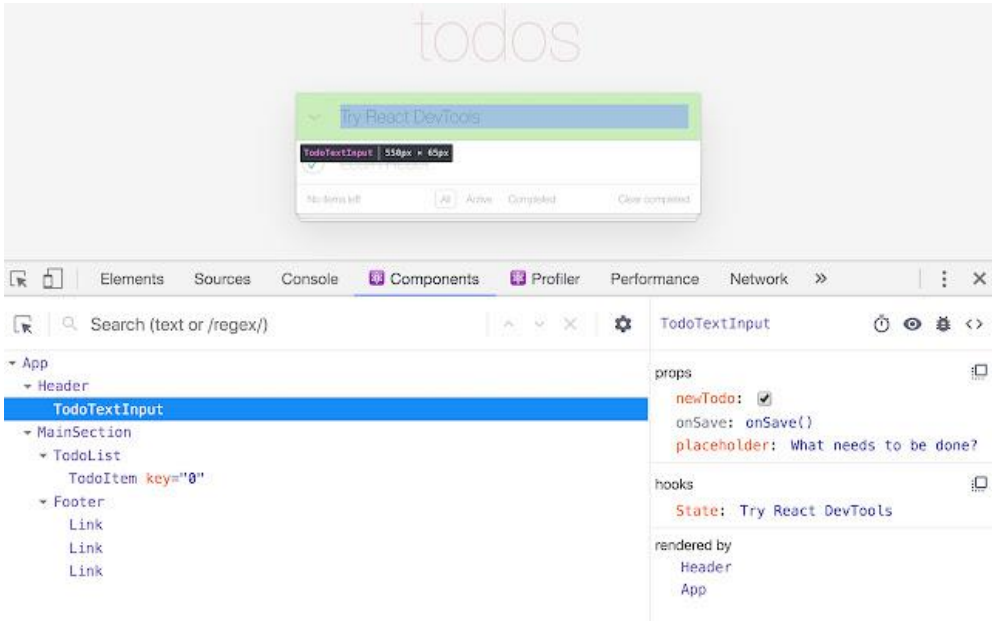


Рисунок 1.8 React Developer Tools [8]

2. Редактори коду, такі як Visual Studio Code, Sublime Text або Atom, мають розширення, плагіни та корисні функції для роботи з React.

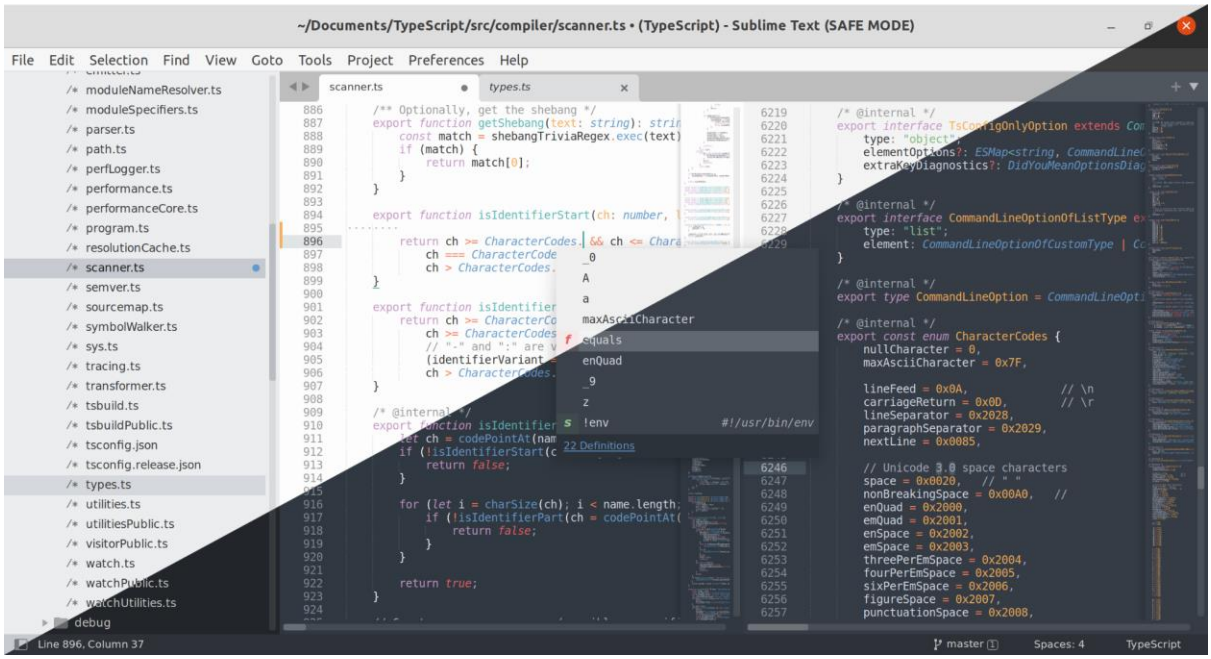


Рисунок 1.9 Sublime Text [9]

3. Create React App: цей інструмент допомагає швидко створювати новий проект, має зручний шаблон та автоматично налаштовує середовище розробки.

4. Бандлери модулів, такі як Webpack або Parcel, використовуються для збирання та пакування React-коду та його залежностей в JavaScript-файли. Це дозволяє ефективно керувати залежностями, зменшує розмір файлів та покращує завантаження додатку.

5. Babel дозволяє використовувати нові функції та синтаксис JavaScript, які ще не підтримуються всіма браузерами.

6. Для тестування React-додатків розробники використовують різні фреймворки, такі як Jest, React Testing Library або Enzyme. Вони дозволяють писати тести для перевірки функціональності компонентів.

7. Git та SVN використовуються для керування кодом, роботи з репозиторіями та спільної розробки.

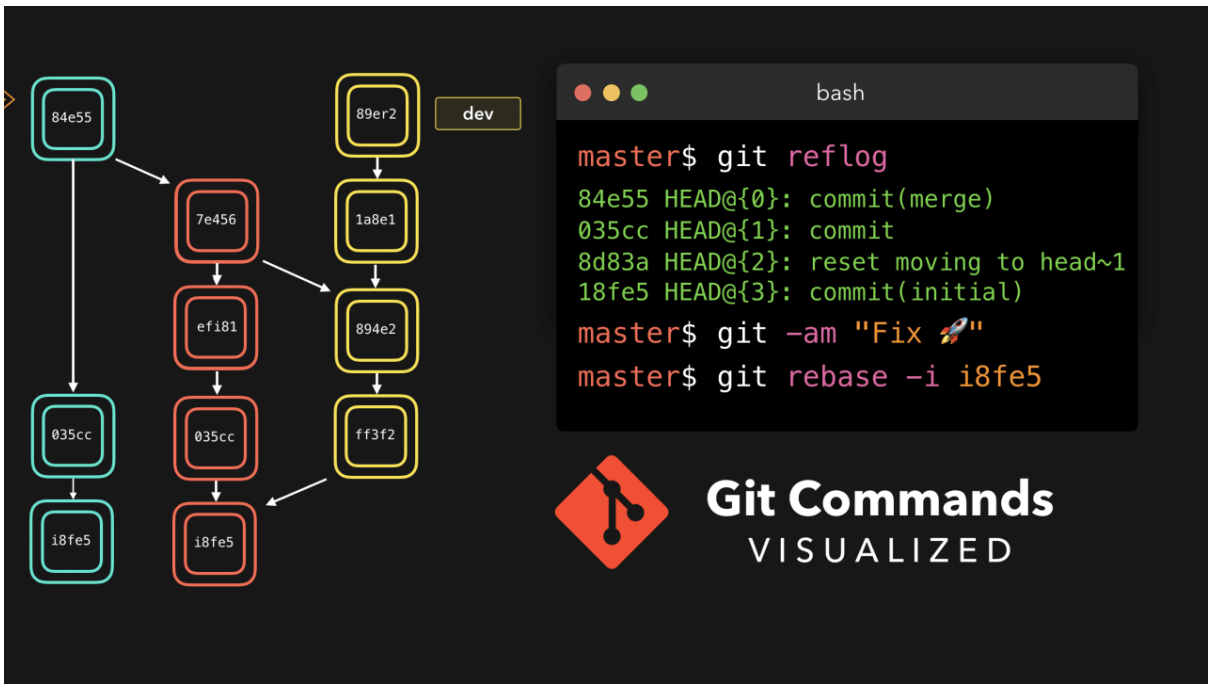


Рисунок 1.10 Git [10]

Розробники вибирають інструменти, враховуючи потреби проекту. Оновлення можна вивчати через уроки React JS на YouTube, вебінари колег або просунуті курси з React.

1.3 Порівняння концепцій та підходів до розробки у Angular і React

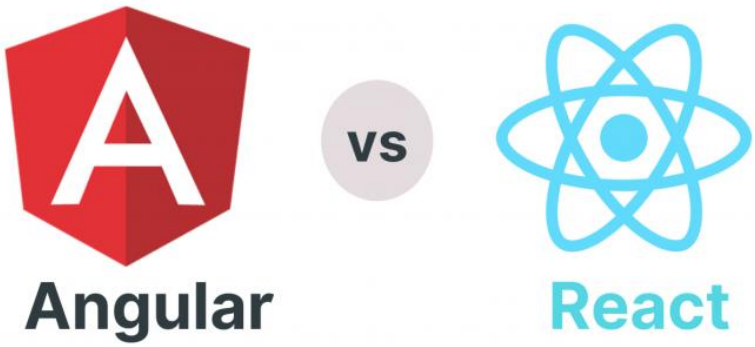


Рисунок 1.11 Порівняння Angular та React [11]

Angular - це фреймворк, розроблений групою в Google, спрямований на спрощення створення складних веб-додатків та організацію розробки за найкращими стандартами.

Унікальність і сила Angular базуються на двонаправленому зв'язуванні даних. Цей механізм дозволяє автоматично синхронізувати дані між інтерфейсом користувача і кодом, забезпечуючи більшу реактивність та зручність. При взаємодії з інтерфейсом Angular автоматично оновлює дані в коді, і, навпаки, зміни в коді миттєво відображаються в інтерфейсі. Ця швидка взаємодія сприяє більш ефективній розробці та дозволяє створювати інтерактивні додатки.

Компонентний підхід Angular є ще однією його сильною стороною. Розробники створюють користувацькі елементи (компоненти) та збирають їх у складні інтерфейси. Це сприяє простоті перевикористання компонентів у різних частинах додатка, поліпшує зрозумілість коду та його підтримку.

Angular пропонує широкий спектр вбудованих інструментів та функцій для розробки, таких як маршрутизація, управління станом і HTTP-запити. Це робить його універсальним фреймворком для різних завдань і дозволяє розробникам зосередитися на функціональності, уникаючи зайвого написання рутинного коду.

React - це бібліотека, яка легка та гнучка, розроблена групою від Facebook. Основна мета React - спростити створення користувацьких інтерфейсів та надати простий старт у веб-розробці.

Одна з унікальних особливостей React - це JSX (JavaScript XML), який дозволяє писати код, що нагадує звичайний HTML, але в той же час є справжнім JavaScript всередині. Цей підхід робить розробку компонентів

легкою для сприйняття і зручною. Кожен компонент - це невеликий блок коду, що об'єднує логіку та представлення інтерфейсу. Цей компонентний підхід полегшує читання та підтримку коду.

Однією з особливостей React є віртуальна DOM (Document Object Model). Коли дані змінюються, React оновлює лише змінені частини віртуальної DOM, а потім ефективно оновлює реальний DOM (тобто те, що бачить користувач). Цей підхід робить фреймворк дуже швидким і реагуючим, що має велике значення для сучасних веб-додатків.

Ангуляр - це повноцінний фреймворк, який пропонує розширений набір готових інструментів і функціоналу для створення веб-додатків. Він включає вбудований механізм для управління станом додатку, налаштування маршрутів та інших ключових можливостей.

У порівнянні з Ангуляр, Реакт - це бібліотека, спрямована на створення користувацьких інтерфейсів. Він не має такого широкого спектру вбудованих інструментів, але акцентується на створенні інтерфейсів та компонентів.

2 АНАЛІЗ МЕТОДІВ РОЗРОБКИ ВЕБ-ДОДАТКІВ У ФРЕЙМВОРКАХ ANGULAR І REACT

2.1 Компонентна модель у Angular та React

Веб-компоненти - це інноваційна технологія у веб-розробці, яка змінює підхід до створення елементів користувацького інтерфейсу для подальшого повторного використання. Цей стандарт дозволяє розробникам створювати власні HTML-елементи, обладнані власною логікою та стилями, та використовувати їх так само просто, як звичайні HTML-компоненти.

Веб-компоненти складаються з трьох ключових технологічних складових:

1. Custom Elements: ця функція дозволяє створювати користувацькі HTML-елементи з унікальними іменами та властивостями, які можуть мати спеціальні можливості.

2. Shadow DOM: цей механізм ізолює стилі та структуру DOM веб-компонента від основного DOM документа, уникнувши конфліктів стилів та імен елементів.

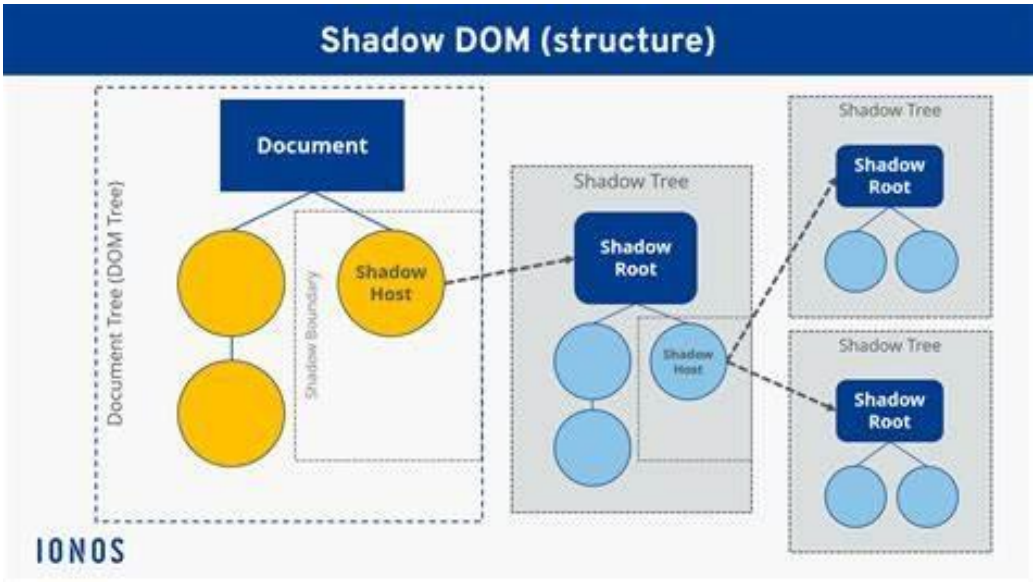


Рисунок 2.1 структура Shadow DOM [9]

3. HTML Templates: HTML-шаблони визначають структуру веб-компонента та його відображення на сторінці, що дозволяє створювати повторювані елементи у користувацькому інтерфейсі.

Переваги веб-компонентів у веб-розробці:

1. Можливість повторного використання: веб-компоненти створюють елементи, які можна застосовувати знову в різних проектах або на окремих сторінках вашого веб-сайту. Це спрощує розробку та підтримку.

2. Модульність: створюючи веб-компоненти з власною логікою та інтерфейсом, ви можете легко працювати з окремими частинами сайту та утримувати їх незалежно.

3. Мінімізація конфліктів: shadow DOM дозволяє уникнути зіткнень стилів та ідентифікаторів між вашими веб-компонентами та основними сторінками, забезпечуючи більшу чистоту коду та краще управління.

4. Зручний дизайн: Використання HTML Templates спрощує створення шаблонів для ваших компонентів та їхнє використання у різних випадках, що полегшує роботу з дизайном.

5. Сумісність з усіма браузерами: Веб-компоненти підтримуються всіма сучасними браузерами, що робить їх доступними для широкого кола користувачів і забезпечує універсальність використання.

Компоненти представляють собою ключову складову **Angular**, відповідальну за формування частини користувацького інтерфейсу в додатку. Оскільки Angular спеціалізується на створенні додатків з активним інтерактивним інтерфейсом, саме це пояснює величезне значення компонентів.

Веб-браузери надають загальні інструменти для створення користувацького інтерфейсу, такі як HTML, CSS і фрагменти JavaScript. Ці засоби дозволяють створювати веб-сайти, проте розробникам програмного забезпечення потрібні більш конкретні "абстракції", що є ключовими для бізнесу чи проекту.

Наприклад, у HTML є базові абстракції, такі як заголовок, посилання, зображення тощо. Однак ці елементи занадто узагальнені, щоб повністю задовольнити потреби бізнесу чи конкретні вимоги додатка. Вони служать лише основними блоками, які самі по собі не відображають реальні потреби у більш конкретних формах функціональності.

Фреймворки, такі як Angular, дозволяють групувати функціональні частини та дані, присвоюючи їм імена, щоб вони стали складовою словника додатку. Наприклад, в HTML немає "Входу", але за допомогою Angular можна створити LogInComponent (Рисунок 2.2), який об'єднує усе, що стосується входу в систему. Його можна розробляти та тестувати окремо від інших частин додатку. Після завершення розробки його можна додати для взаємодії з користувачами.

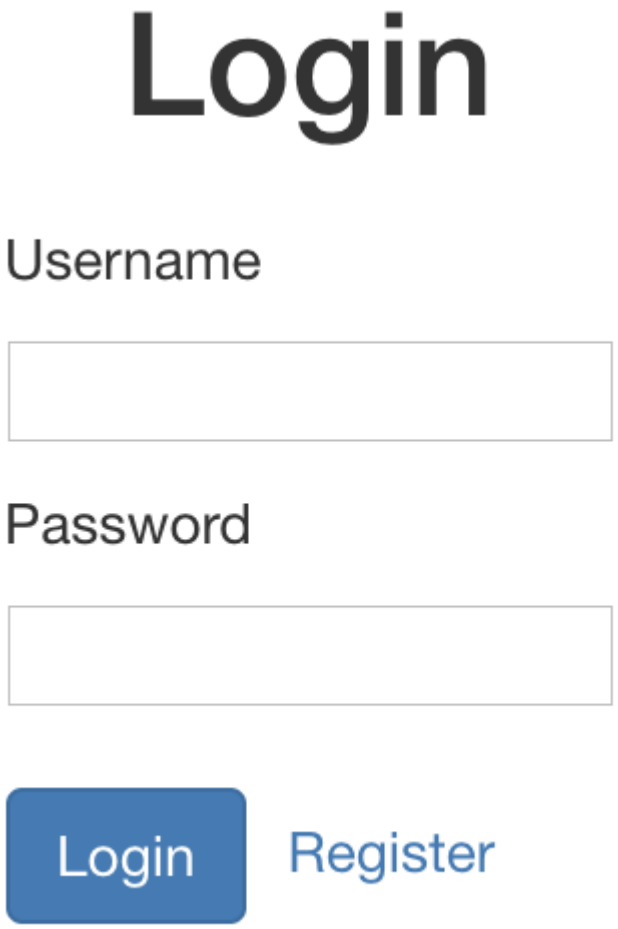


Рисунок 2.2 Компонент входу [10]

Для розробників Angular додаток представляє собою ієрархію компонентів, кожен з власними завданнями. Деякі компоненти забезпечують конкретні функції (наприклад, LogInComponent), тоді як інші можуть бути використані у різних сценаріях (наприклад, компонент для відображення деталей товарів у каталозі).

Більшість Angular-додатків починаються з одного ключового компонента, який називається AppComponent. Під час створення цього компонента ми створюємо TypeScript клас з метаданими, які відзначені декоратором @Component ({...}), а також шаблон, схожий на HTML, але розширений за допомогою виразів прив'язки.

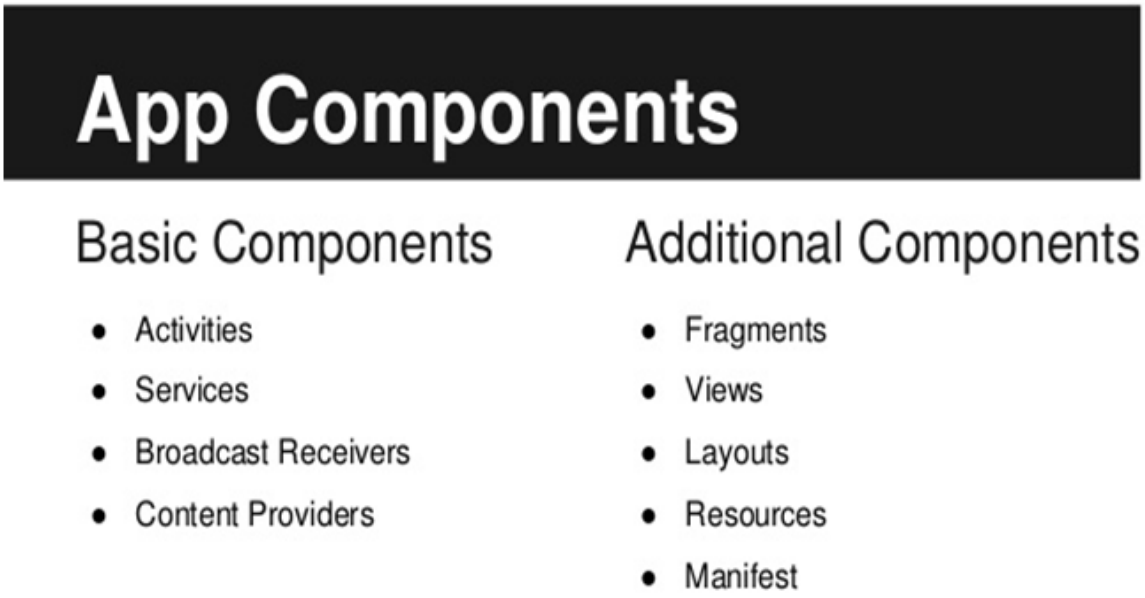


Рисунок 2.3 Компоненти веб-додатку [11]

Це відрізняє його від звичайного HTML. Angular використовує розширений набір елементів та атрибутів через кутові шаблони, щоб перетворити декларативне програмування HTML на імперативне. Цей підхід дозволяє використовувати умовні оператори, цикли та інші конструкції, що робить Angular потужним інструментом для розробки.

React, створений компанією Facebook для розробки веб-інтерфейсів у JavaScript, базується на компонентній архітектурі. Головна його ідея - формувати складні веб-додатки з малих, переиспользуваних модульних компонентів. Кожен елемент React це окремий блок інтерфейсу, який самостійно керує своїм відображенням та станом.

Ця архітектура дозволяє краще організовувати код, спрощує управління програмами та забезпечує їх підтримку. Якщо правильно структурувати компоненти, інтерфейс можна розбити на менші модульні частини, що полегшує їх розробку, тестування та відлагодження. Цей підхід

до створення веб-додатків швидко став популярним серед розробників і вважається ключовим у світі front-end розробки.

З використанням компонентної архітектури в React отримано кілька важливих переваг:

1. Краща структурування та модульність коду: архітектура React дозволяє розбити інтерфейс програми на менші компоненти, які можна використовувати повторно. Це призводить до більшого порядку та зрозумілості коду, спрощуючи його обслуговування та поліпшуючи організацію проекту.

2. Можливість повторного використання компонентів: компоненти можна використовувати в різних частинах програми або навіть у кількох додатках, що усуває необхідність повторення коду та сприяє впровадженню принципу DRY (Don't Repeat Yourself).

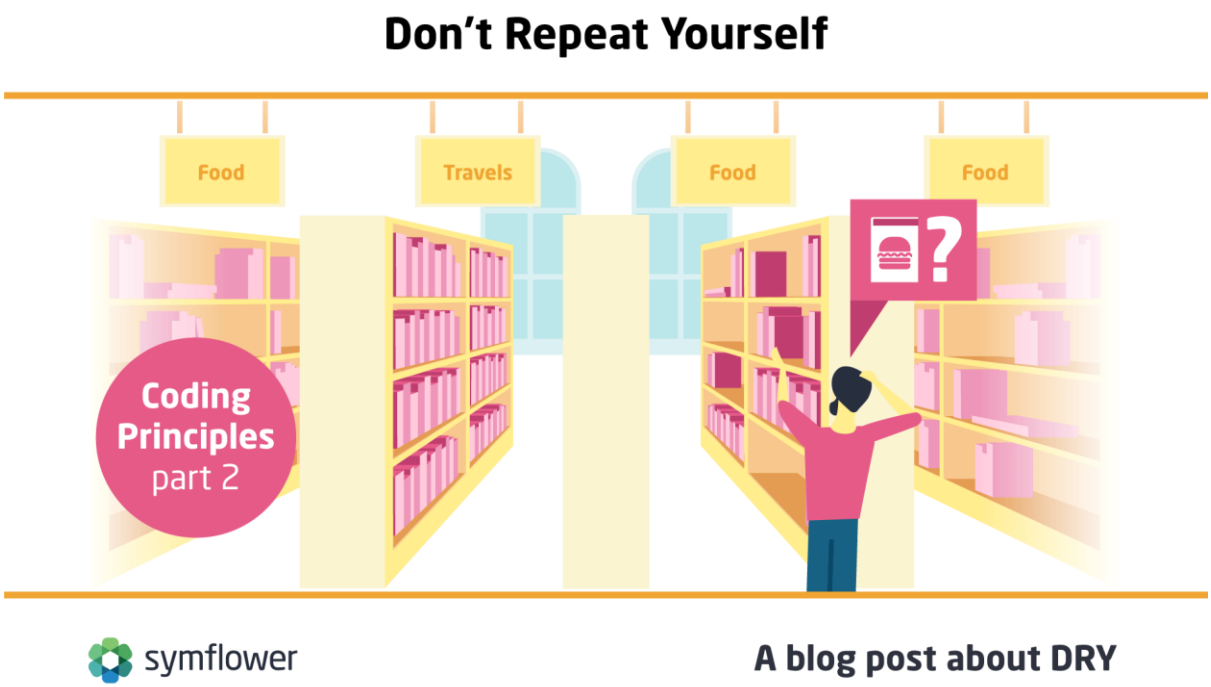


Рисунок 2.4 Принцип DRY [12]

3. Легкість тестування та обслуговування: завдяки тому, що компоненти в React зазвичай є невеликими та фокусуються на конкретних завданнях, написання та підтримка тестів для окремих функцій стає простішим завдяки їх відокремленості. Оновлення одного компонента не впливає на інші, що сприяє стабільності роботи програми в цілому.

4. Чітка розділеність обов'язків: кожен компонент у React-додатку відповідає за свій власний рендеринг та управління станом. Це забезпечує чітку структуру завдань і дозволяє розробникам фокусуватися на конкретних частинах інтерфейсу користувача, зменшуючи загальну складність.

Розуміння та володіння етапами життєвого циклу компонентів у React є ключовими для створення додатків, які працюють ефективно та можуть масштабуватись. Ці етапи охоплюють всі аспекти від створення (додавання до DOM) до знищення (видалення з DOM) компонентів. Методи життєвого циклу у React є функціями, що дозволяють розробникам контролювати поведінку компонента й викликаються на різних етапах його життя.

Етапи життєвого циклу компонента React описуються так:

1. Конструктор (Constructor): Підготовка компонента до встановлення, ініціалізація початкового стану та встановлення зв'язків з обробниками подій.

2. componentDidMount: Викликається після встановлення компонента у DOM. Часто використовується для отримання даних або налаштування підписок розробниками.

3. componentDidUpdate: Активується після оновлення компонента для виконання побічних ефектів або додаткового рендерингу, враховуючи зміни у деталях або стані компонента.

4. componentWillUnmount: Викликається перед демонтажем та знищенням компонента. Це оптимальний момент для очищення ресурсів, таких як таймери або підписки.

Керування станом має важливе значення, оскільки життєвий цикл компонента у React є його основою. Кожен компонент може мати власний внутрішній стан, представлений об'єктом JavaScript, який можна оновлювати за допомогою setState. React автоматично перерендерює компонент при зміні стану для відображення оновлених даних.

При праці з архітектурою компонентів у React важливо розрізняти між "контейнерами" та "презентаційними" компонентами. Ці два типи виконують різні функції, і розуміння їх призначення є ключовим для підтримки чистої та оптимізованої структури додатків.

Контейнерні компоненти, відомі також як "розумні" компоненти, відповідають за управління логікою та станом програми. Вони є головними точками взаємодії з зовнішніми ресурсами, наприклад, API, та використовуються, коли компонент потребує загального стану програми. Контейнерні компоненти можуть також з'єднуватися з Redux-сховищем та використовувати диспетчерську діяльність.

Основні характеристики контейнерних компонентів включають:

- Керування станом програми та логікою даних
- Підключення до Redux-сховища (Рисунок 2.3)
- Реалізація методів життєвого циклу, таких як componentDidMount і componentDidUpdate
- Передача даних і функцій дочірнім компонентам через пропси.

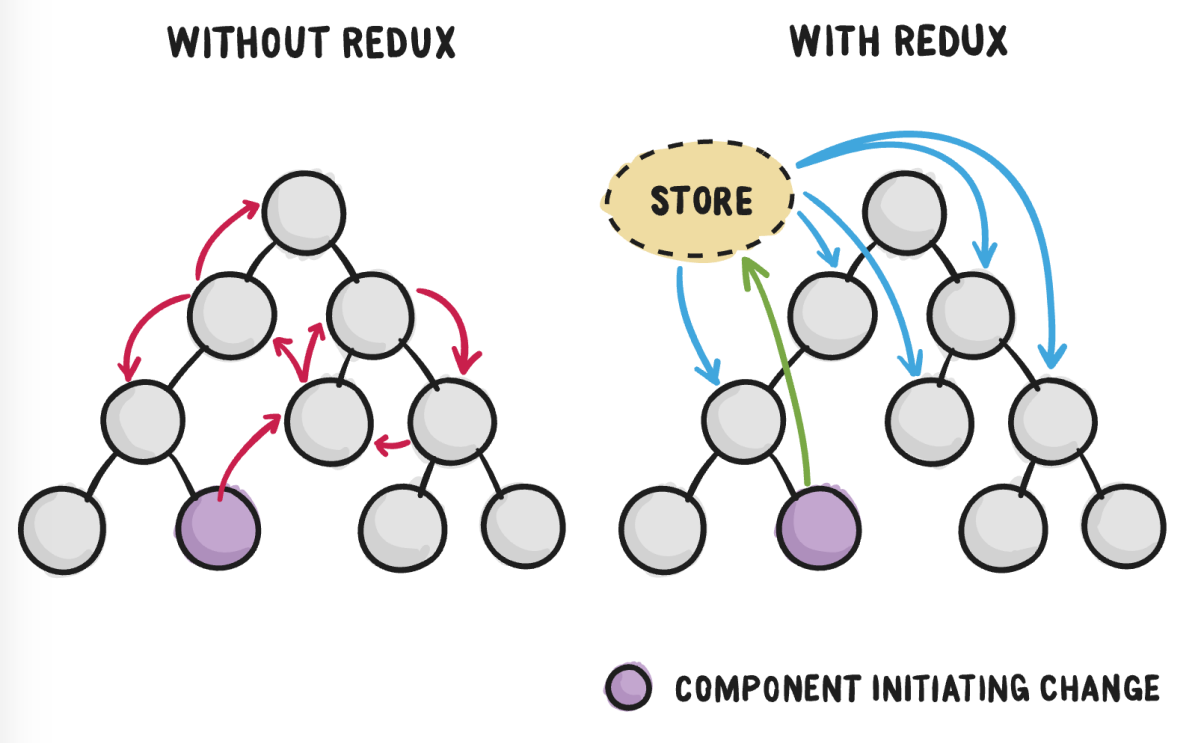


Рисунок 2.5 Redux-сховища [13]

Компоненти, що відтворюють інформацію, часто відомі як презентаційні, спрямовані на показ елементів інтерфейсу користувача та отримання даних через пропси від батьківських компонентів. Їх основне завдання полягає у візуалізації даних і не прямо пов'язане зі станом програми, зовнішніми API або Redux-сховищами.

Головні характеристики таких компонентів:

1. Зібрання та показ даних.
2. Показ інтерфейсу без управління станом.
3. Зазвичай реалізовані як функціональні компоненти.
4. Легко перевикористовувати в межах програми та інших проектів.

Успішний React-додаток досягає балансу між "розумними" і "презентаційними" компонентами. Це сприяє правильному розподілу

відповідальностей, спрощує обслуговування та допомагає зрозуміти роль кожного компонента в додатку.

Архітектура компонентів у React відкриває нові горизонти в сучасній веб-розробці, надаючи ефективний та структурований підхід до створення користувацьких інтерфейсів. Цей підхід ґрунтується на модульних компонентах, які можна застосовувати у різних частинах проекту. Глибоке вивчення шаблонів та принципів, описаних у цій статті, допоможе вам ефективно використовувати архітектуру React для розробки масштабованих, підтримуваних і високопродуктивних додатків, які витримають випробування часу.

2.2 Реактивний підхід у Angular та використання хуків у React

Розглянемо основні ідеї та методи створення програм з використанням **реактивного програмування**. Ця практика, що з'явилася недавно, зосереджується на обробці потоків даних та відповіді на їх зміни. Фактично, це спосіб написання програмного коду, який реагує на певні події всередині програми. Для руху в цьому напрямі важливо чітко відокремлювати дані на динамічні та статичні, і наша модель повинна автоматично обробляти зміни в динамічних даних через потоки. Спочатку реактивне програмування розглядалося як простий метод створення інтерфейсів користувача, анімацій та інших процесів, які змінюються з часом.

Його можна втілити різними способами, такими як:

- імперативне програмування;
- об'єктно-орієнтоване програмування;

- функціональне програмування.

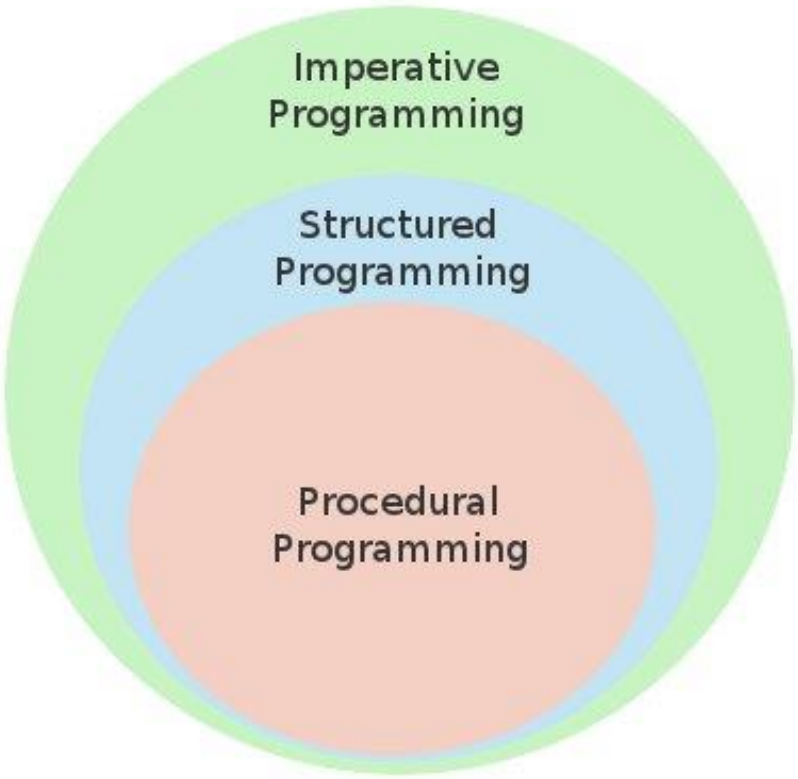


Рисунок 2.6 Якими способами здійснюється реактивне програмування [14]

Проте найбільш природнім фундаментом для реактивного програмування та роботи з реактивними структурами даних є підхід, заснований на функціональному програмуванні.

RxJS (Рисунок 2.3) - це одна з найпопулярніших бібліотек, що застосовує цю концепцію. Вона є ключовою складовою фреймворку Angular.

Рисунок 2.7 RxJS [15]

Реактивне програмування - це підхід до написання програм, де увага приділяється асинхронним потокам даних та опису реакцій на їх зміну. В контексті цього підходу, RxJS дозволяє трансформувати, компонувати та витягувати дані з цих потоків як у веб-браузері, так і на серверному рівні. Це стоїть у контрасті з імперативним підходом, де керування змінами вимагає явних дій та інструкцій.

Якщо застосовувати реактивний підхід, будь-який зазначений або складений тип даних, наприклад список, можна розглядати як послідовний потік і представляти його у спосіб як на рисунку 2.3.

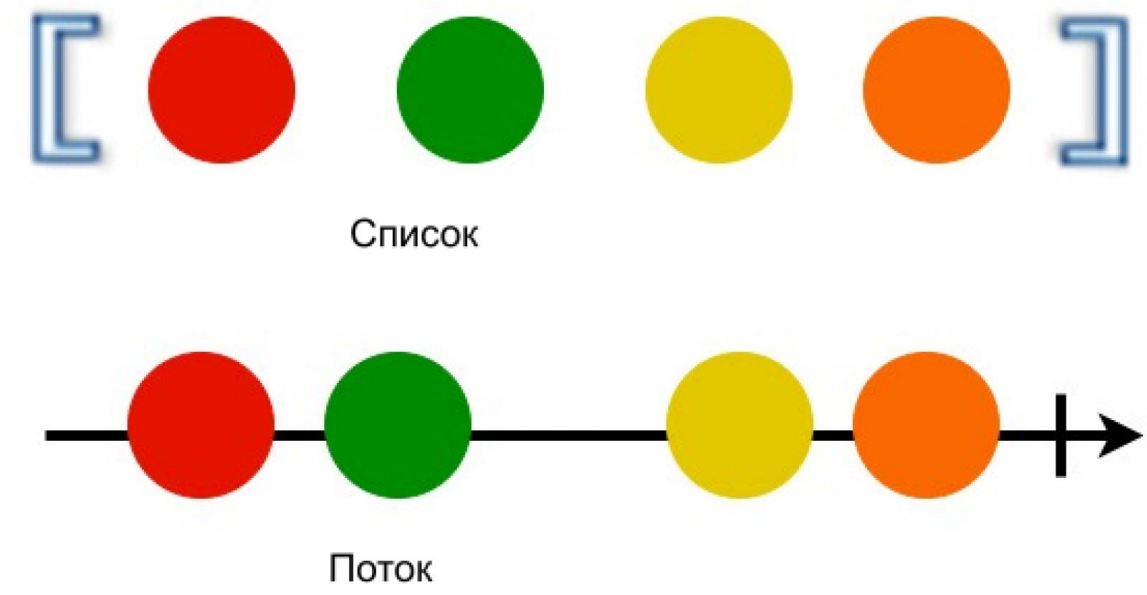


Рисунок 2.8 Представлення потоку даних [16]

На тимчасовій шкалі, кожен потік починається зі старту, послідовно генерує значення та завершується за допомогою вертикальної лінії. Цей потік відтворює послідовність подій, що відбуваються упорядковано у часі, таких як кліки на кнопках, запити на сервер, дані через сокет-з'єднання або анімації.

Наша здатність сприймати ці події дозволяє нам реагувати на них. Використовуючи звичайні функції, ми можемо комбінувати, змінювати та фільтрувати ці потоки.

Події у потоці можуть приймати три форми:

1. Значення,
2. Помилки,
3. Завершення.

Наша мета - перехоплювати ці події та описувати реакції на ці три типи асинхронних (або синхронних) подій. У цьому контексті ми будемо працювати з кількома програмними сутностями, перша з яких - це спостережуваність.

Спостережуваність, відома як Observable, є функцією з рядом визначених особливостей.

Ця функція приймає об'єкт-спостерігач (observer) як параметр, який має три методи:

- Next: для отримання наступного значення з потоку;
- Error: для повідомлення про помилку;
- Complete: для сигналу про завершення потоку.

Спостережуваність сприяє обміну повідомленнями між тими, хто створює дані, та тими, хто їх використовує. Існують дві стратегії взаємодії між виробником і споживачем даних.

У першій стратегії (pull) споживач визначає час і місце реакції на зміну або отримання даних. У другій стратегії (push) це робиться виробником.

У прикладі pull-стратегії, функція викликається з місця програми, де потрібні дані, але сама функція не має інформації про те, хто та коли користуватиметься її результатами.

У випадку push-стратегії функція-виробник обробляється обробниками (споживачами), які не мають інформації про те, коли та де дані будуть згенеровані.

RxJS використовує другу стратегію (push).

Спостережуваність залишається у пасивному стані до моменту підписки на неї через функцію subscribe, що відрізняє її від промісів, які відразу повертають значення.

Метод subscribe() повертає об'єкт підписки (subscription), включаючи функцію unsubscribe(), яка припиняє спостереження та призводить до припинення генерації подій, що робить його унікальним порівняно з промісами.

Observable є схожим на функцію без аргументів, але вміє отримувати та повертати багато значень, що відрізняє його від звичайних функцій.

RxJS бібліотека пропонує набір інструментів, які спрощують створення спостережень з різних джерел системи, таких як події, проміси, таймери і т.д. Давайте розглянемо кілька прикладів створення Observable-об'єктів із різних джерел.

У останньому важливому оновленні React з'явилися **хуки** – новий метод використання API бібліотеки. Це дозволяє уникнути створення класових компонентів і разом з тим зберігати всі їхні корисні можливості, такі як стан і обробка побічних ефектів. Використання хуків робить код програми більш чистим і послідовним, що стає фундаментом підходу "хуки в першу чергу" у React-розробці.

Хуки у React - це функції, які відкривають доступ до можливостей React, таких як стан та інші, без необхідності використання класових компонентів. Їх впровадили у React 16.8, і з того часу вони стали потужним інструментом для роботи зі станом, контекстом, ефектами та іншими аспектами React-додатків у функціональних компонентах.

Основні хуки в React включають:

1. useState: Цей хук дозволяє працювати зі станом у функціональних компонентах. Ви можете створювати та оновлювати значення стану в компоненті за допомогою цього хука.

2. useEffect: Він дає можливість виконувати певні дії після кожного рендерингу компонента. Це корисно для роботи з підпискою на дані, маніпулювання DOM або виконання певних дій після завантаження компонента.

3. useContext: Цей хук дозволяє отримати доступ до контексту React у функціональному компоненті. Він спрощує передачу даних через дерево компонентів, уникнувши передачі пропсів кожному рівню компонентів.

4. useReducer: Надає можливість організувати стан у функціональних компонентах за допомогою підходу зі зменшенням. Це корисно, коли стан складний та має багато варіантів оновлення.

5. useCallback та useMemo: Ці хуки сприяють оптимізації продуктивності компонентів шляхом уникнення повторних розрахунків функцій або значень під час рендерінгу.

Ці хуки роблять функціональні компоненти у React більш потужними, надаючи їм доступ до багатьох можливостей, які раніше були доступні тільки для класових компонентів. Вони спрощують управління станом, ефектами та контекстом у вашому React-додатку.

Так, `useEffect` у React дійсно є потужним інструментом для керування побічними ефектами у функціональних компонентах. Він відповідає за виконання побічних операцій, які виходять за межі звичайного оновлення користувацького інтерфейсу. Це можуть бути запити до сервера, підписки на дані, маніпуляції DOM у браузері тощо.

Завдяки `useEffect` ви можете описати реакції на зміни стану або пропсів, виконати певні дії після рендерингу компонента та очистити ресурси, коли компонент більше не потрібен. Цей хук замінює функціонал `componentDidMount`, `componentDidUpdate` і `componentWillUnmount` у класових компонентах React.

Важливою перевагою `useEffect` є можливість чіткого управління побічними ефектами в функціональних компонентах, що забезпечує логічну та структуровану взаємодію з зовнішнім середовищем та іншими частинами вашого додатку. Це дозволяє уникнути непотрібного зв'язку між компонентами та забезпечує більшу гнучкість у роботі з побічними ефектами.

2.3 Інструменти тестування та їх застосування у кожному з фреймворків

Тестування має на меті перевірити, чи працює програма правильно. Якщо всі важливі аспекти функціоналу програми будуть протестовані, це дозволить швидко виявити будь-які неполадки чи проблеми, які можуть виникнути.

Автоматизоване тестування додатків на React часто використовує Jest разом із @testing-library/react або Testing Library. Однак, існують альтернативи, такі як Mocha, Jasmine та AVA. Щодо Testing Library, вона вважається альтернативою Enzyme, яку досі використовують багато розробників.

Рисунок 2.9 Jest [17]

Jest (Рисунок 2.3) - це набір інструментів для тестування JavaScript, спрямований на виконання тестів на JavaScript і TypeScript, і чудово інтегрується з React.

Цей фреймворк розроблений з фокусом на простоту і має потужний та елегантний API для створення самостійних тестів, порівняння знімків, фіксації результатів, оцінки покриття коду тестами та інших тестових завдань.

Підхід Testing Library орієнтований на тестування додатків з точки зору користувача. Це призводить до написання інтеграційних тестів, де кілька компонентів системи взаємодіють між собою.

Наприклад, якщо говорити про кнопку, за допомогою Testing Library ви не перевіряєте, чи була викликана функція onClick під час кліку на кнопку. Ви зосереджуєтеся на тестуванні конкретного зовнішнього ефекту, який виникає внаслідок кліку на цю кнопку.

React Testing Library - це інструмент, спеціально створений для тестування JavaScript, зорієнтований на перевірку React-компонентів. Він дозволяє моделювати взаємодію користувача з окремими компонентами та перевіряти їх виведення, щоб гарантувати правильну реакцію інтерфейсу користувача. Цей інструмент спрощує процес тестування, надаючи можливість взаємодії з компонентами умовно та візуально, що допомагає виявити помилки та забезпечити стабільність програми. Його головна мета - перевірити, чи реагують компоненти на взаємодію з користувачем коректно і чи вони працюють відповідно до очікувань.

Mocha (Рисунок 2.3), як фреймворк для тестування JavaScript, спрямований на Node.js додатки, пропонує широкий спектр функцій. Він охоплює підтримку браузера, можливість асинхронного тестування, генерацію звітів щодо покриття коду тестами і гнучкість у використанні різних бібліотек для перевірки. Розробнику дається можливість налаштувати тести з урахуванням власних потреб, визначаючи інструменти та підходи для тестування їхнього коду. Можливість увімкнення та

вимкнення більшості допоміжних бібліотек та інструментів під час тестування також наявна.

Рисунок 2.10 Mocha [18]

Подібно до Jest та інших фреймворків, Mocha може поєднуватися з Enzyme для роботи з React, а також з Chai та іншими бібліотеками. Багато розробників віддають перевагу використанню саме цього фреймворку для тестування своїх React-додатків, особливо коли вони потребують специфічних налаштувань та розширеного набору інструментів. Однак використання Mocha може мати свої недоліки, наприклад, деякі завдання, такі як створення знімків, вимагають встановлення додаткових інструментів та додаткової конфігурації.

Chai є бібліотекою тверджень та очікувань для вузла та браузера, спрямованою на BDD/TDD. Ця бібліотека ідеально взаємодіє з будь-яким тестовим фреймворком у JavaScript. Вона часто використовується разом з Mocha та Enzyme, а також має можливість інтеграції з Jest та Enzyme. Основні інтерфейси Chai, такі як expect, should і assert, допомагають встановлювати очікування в тестах.

Карма не є просто тестовим фреймворком або бібліотекою тверджень. Це інструмент для тестування, який можна поєднати з Jasmine, Mocha та іншими. Він створює HTTP-сервер і генерує HTML-файл для виклику тестів. Карма дозволяє виконувати JavaScript-код у кількох реальних браузерах, просто запускаючи цей HTML-файл.

Ця утиліта була розроблена для спрощення процесу отримання повернення з тестів, щоб зв'язок між написанням коду та перевіркою вашого тестування був простішим, не потребуючи глибокого розбирання конфігурацій. Локально можна запускати тести та перевіряти їх у різних реальних браузерах на пристроях, таких як телефони, планшети або комп'ютери, все це в межах вашої інтегрованої середи розробки (IDE).

Jasmine визначається як "JavaScript Test Framework для браузерів і Node.js". Це інструмент для тестування BDD в JavaScript, який не обмежується браузерами, DOM або конкретним JavaScript фреймворком. Зазвичай його використовують у в контексті фреймворків, таких як Angular, через його командні можливості.

Жасмин ідеально підходить для тестування React-додатків, включаючи Babel і Enzyme. Навіть існує спеціальна бібліотека утиліт, яка спрощує цей процес тестування.

Занурення у процес тестування React може стати складним завданням, особливо при використанні таких тестових фреймворків, як Jest, без підтримки Enzyme від AirbnbEng. Enzyme не є фреймворком для тестування, але скоріше інструментом для тестування React, який спрощує процес перевірки виведення компонентів шляхом абстрагування їх рендерингу.

Enzyme дозволяє рендерити компоненти, знаходити елементи та взаємодіяти з ними. Можна спробувати використовувати Enzyme разом з Jest або спробувати інтеграцію його з Karma, Mocha та іншими.

Cypress - це JavaScript фреймворк для наскрізного тестування, який полегшує налаштування, написання, запуск та налагодження тестів у браузері. Він поставляється з власною панеллю управління, яка може контролювати стан наших тестів. Також, оскільки Cypress працює в реальному браузері, паралельно можна використовувати інструменти розробника самого браузера.

Можна «подорожувати в часі» за допомогою знімків, автоматично перезапускати зміни тестів, а також займатися налагодженням без зайвих складнощів. З вбудованим розпаралелюванням і балансуванням навантаження, тестування налагодження в CI також набагато простіше. Однак не можливо використовувати Cypress для роботи з двома браузерами одночасно, і це може бути незручністю.

Хоча Cypress не настільки популярний, як Puppeteer, він може бути корисним для наскрізного тестування React-додатків. І ця бібліотека навіть створювалася для полегшення процесу.

Cypress - це фреймворк на JavaScript, призначений для проведення енд-ту-енд тестування, що спрощує встановлення, написання, запуск та налагодження тестів у вашому веб-браузері. Цей інструмент обладнаний власною панеллю керування, що дозволяє контролювати хід вашого тестування. Оскільки Cypress працює в реальному браузері, ви можете одночасно використовувати інструменти розробника браузера.

Його функціонал включає можливість знімків часу, автоматичне перезапуску змін у тестах та просте налагодження, що створює зручний інтерфейс для користувача. Завдяки можливостям розпаралелювання та

розподілу навантаження, які вбудовані у нього, Cypress спрощує тестування в умовах неперервної інтеграції. Проте варто зазначити, що Cypress не дозволяє працювати одночасно з двома браузерами, що може стати певним обмеженням.

Навіть якщо популярність Cypress не настільки висока, цей інструмент виявляється дуже корисним для енд-ту-енд тестування React-додатків. Він спроектований для спрощення процесу тестування та може бути ефективним варіантом для цієї мети.

Тестування Angular-додатків також може бути різноманітним, перерахуємо такі фреймворки та бібліотеки:

- Jasmine
- TestBed, який входить до складу Angular Testing Library, пропонує зручний інтерфейс для налаштування тестового середовища в Angular додатках. Цей клас дозволяє створювати та налаштовувати оточення для тестування компонентів, сервісів та інших складових Angular додатків.

Основні методи класу TestBed включають:

- configureTestingModule(): Налаштовує модуль Angular для тестування, приймаючи конфігурацію модуля для використання під час тестування компонентів.
- createComponent(): Створює екземпляр компонента для тестування, приймаючи компонент, який потрібно протестувати, та повертаючи об'єкт з екземпляром компонента та його DOM елементом для подальшого тестування.
- get(): Дозволяє отримувати доступ до служб та інших залежностей, зареєстрованих у тестовому модулі, для використання у ваших тестах.

- compileComponents(): Використовується для компіляції шаблонів компонентів перед тестуванням. Цей метод забезпечує попередню компіляцію шаблонів для використання у тестах.
- overrideComponent(): Дозволяє змінювати параметри компонента під час тестування для спрямування на конкретну поведінку, яка цікавить вас.
- overrideProvider(): Цей метод надає можливість перевизначати служби або залежності, зареєстровані в тестовому модулі, надаючи змінені або макетовані версії служб для спеціального тестування.
- inject(): Використовується для коректної ін'єкції залежностей у тестах та отримання доступу до залежностей, зареєстрованих у тестовому модулі.

- ComponentFixture

ComponentFixture у Angular - це клас, який відкриває можливість отримати доступ до екземпляру компонента для його подальшого тестування. Цей клас дозволяє вам взаємодіяти з компонентом під час тестів та виконувати різноманітні перевірки його стану та функціональності.

- ComponentFixture.debugElement

`ComponentFixture.debugElement` - це функціонал, який входить до складу Angular Testing Library. Він дозволяє отримувати доступ до DOM-елементів, що пов'язані з компонентом, використовуючи `ComponentFixture`. Це дає можливість виконувати різноманітні перевірки та отримувати доступ до властивостей елементів під час тестування компонентів Angular.

- Клас By

У фреймворку Angular, клас `By` входить до інструментів для тестування та спрямований на вибір елементів DOM під час створення тестів для Angular.

`By` має різноманітні методи для пошуку елементів на сторінці за різними стратегіями. Основні методи в класі `By` включають:

○ `By.css`: Використовує CSS селектор для пошуку елементів. Наприклад, `By.css('.class-name')` знайде всі елементи з класом `.class-name`.

○ `By.id`: Знаходить елемент за його `id`. Наприклад, `By.id('element-id')` знайде елемент з вказаним `id`.

○ `By.tagName`: Знаходить всі елементи з певним тегом. Наприклад, `By.tagName('input')` знайде всі елементи `<input>` на сторінці.

○ `By.xpath`: Використовує XPath для знаходження елементів. Це більш гнучкий метод, але менш рекомендований через складність у використанні.

- Unit-тести: тестування сервісів

Unit-тести - це спосіб тестування програмного забезпечення, який націлений на перевірку окремих частин коду (зазвичай функцій, методів або класів) на їхню правильність роботи. У світі Angular, unit-тести використовуються для оцінки функціональності окремих модулів коду у додатку.

Щоб переконатися у коректності роботи Angular-сервісів, застосовують unit-тести. Сервіси в Angular відповідають за виконання певних завдань, обробку даних, взаємодію з сервером та інші операції. Під час написання unit-тестів для Angular-сервісів перевіряється, чи виконуються очікувані дії кожного методу або функції цих сервісів.

Для unit-тестування Angular-сервісів використовуються різноманітні інструменти, такі як Jasmine або Karma, для створення та виконання тестів. Основна мета полягає у перевірці правильності роботи кожного методу сервісу за різних умов виклику, включаючи граничні ситуації, помилки та нормальний режим роботи.

Наприклад, якщо у вас є Angular-сервіс, який робить запити до сервера для отримання даних, unit-тести можуть перевірити, чи відбувається коректна взаємодія з сервером, чи вірно оброблюються отримані дані та як сервіс реагує на різні сценарії, такі як успішні чи неуспішні відповіді під час з'єднання з сервером.

Узагальнюючи, unit-тести для Angular-сервісів спрощують перевірку правильності роботи окремих складових функціональності вашого додатку та забезпечують впевненість у їхній надійності та працездатності.

- Метод whenStable

Метод `whenStable` у фреймворку Angular служить для контролю ситуації, коли всі асинхронні дії в додатку завершились і стан додатку став стійким. Це гарантує, що всі зміни, запити до сервера чи інші асинхронні події завершили своє виконання перед проведенням конкретних перевірок у тестах.

Зазвичай, метод `whenStable` використовується разом з іншими тестовими методами, такими як `compileComponents`, `fixture.detectChanges` і `flushMicrotasks`, щоб переконатися, що всі асинхронні завдання завершились перед тим, як проводити перевірки у вашому тесті.

- Unit-тести: тестування httpClient

При тестуванні `httpClient` в Angular, unit-тести дають змогу перевіряти його функціональність та взаємодію з іншими частинами програми, уникнувши реальних HTTP-запитів.
Основні принципи unit-тестування `httpClient` в Angular включають:

1. Мокування HTTP-запитів: тести використовують мокування для імітації роботи `httpClient`, замінюючи реальні HTTP-запити функціями, які повертають заздалегідь визначені результати.
2. Перевірка методів та відповідей: тести перевіряють різні методи `httpClient`, такі як GET, POST, PUT, DELETE тощо, щоб переконатися, що вони виконуються з правильними параметрами та повертають очікувані результати.
3. Обробка помилок: тестування включає сценарії обробки помилок, коли запит до сервера завершується невдачею, щоб переконатися, що програма коректно реагує на помилки та їхнє відновлення.
4. Параметризація та конфігурація: unit-тести можуть перевіряти виклики `httpClient` з різними параметрами та у різних конфігураціях, щоб забезпечити його коректну роботу в різних умовах.
5. Ізоляція компонентів: тести створюються для ізольованого тестування `httpClient` без залежностей від зовнішніх ресурсів або середовища виконання.

Ці аспекти дозволяють перевірити правильність роботи `httpClient` в Angular та забезпечити його коректну функціональність у межах окремих модулів чи компонентів програми.

- Unit-тести: тестування pipe

При тестуванні пайпів в Angular, unit-тести використовуються для перевірки поведінки конкретного пайпу, щоб підтвердити його правильність та відповідність умовам обробки даних. Пайпи в Angular використовуються для зміни формату виводу у шаблонах компонентів.
При написанні unit-тестів для пайпів в Angular, важливо враховувати такі аспекти:

1. Встановлення очікуваного результату: тестувальник визначає очікуваний результат для певного пайпу, враховуючи його логіку обробки даних.
2. Формулювання специфікацій для тестування: наприклад, перевірка коректності роботи пайпу для форматування дати або часу.
3. Створення сценаріїв для тестування: це включає створення тестових даних, застосування пайпу до цих даних та порівняння отриманого результату з очікуваним.
4. Виконання тестів: запуск тестів, які включають виклик пайпу з тестовими даними та перевірку отриманого результату на відповідність очікуваному.

Для тестування пайпів в Angular можна використовувати функціональність фреймворка для створення тестових компонентів або інструменти, які надає сам Angular для створення імітованих середовищ та перевірки роботи пайпів. Важливо переконатися, що тести охоплюють різноманітні сценарії використання пайпу для гарантії його коректної роботи в будь-яких умовах використання.

3 ПОРІВНЯЛЬНИЙ АНАЛІЗ РОЗРОБКИ ВЕБ-ДОДАТКІВ НА ANGULAR ТА REACT

3.1 Створення дизайну веб-додатку Easy English для порівняння Angular та React

Для порівняння Angular та React було створено дизайн веб-додатку за допомогою застосунку Figma. Розглянемо даний застосунок детальніше.

Figma - це онлайн-редактор, призначений для створення макетів, прототипів і дизайну користувальницького інтерфейсу (UI) і користувальницького досвіду (UX). Це потужний інструмент, який дає змогу дизайнерам працювати в режимі реального часу, спільно редагувати та ділитися своїми проєктами в хмарі. [19]

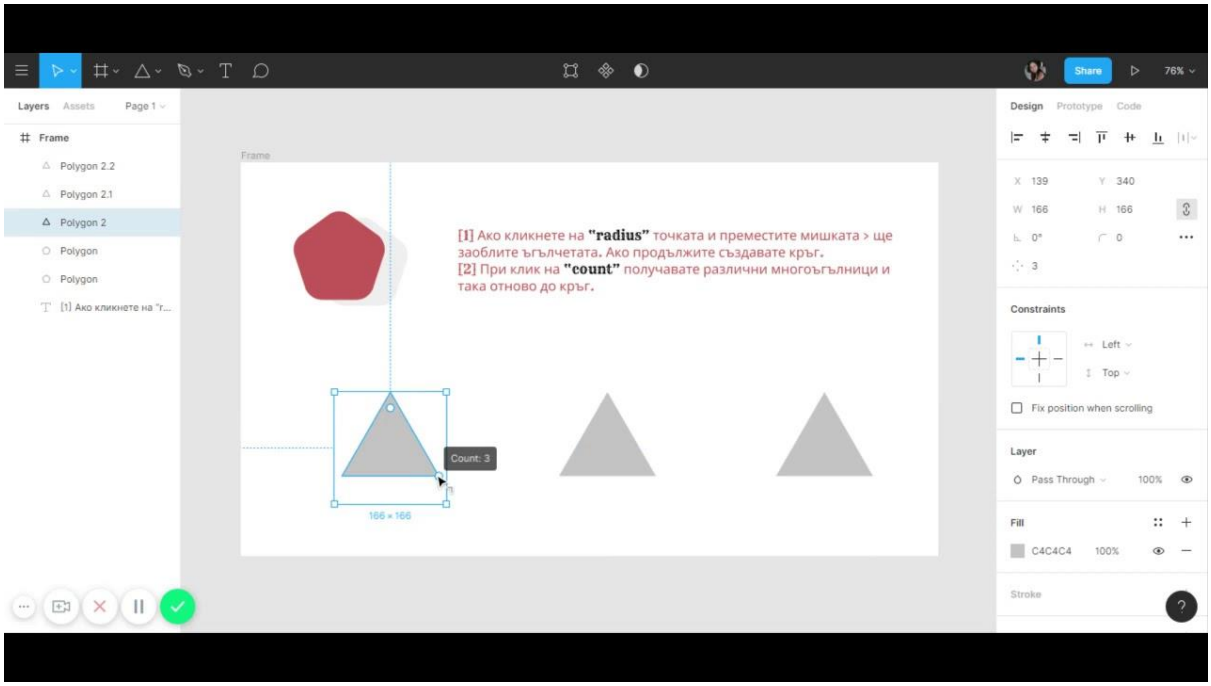


Рисунок 3.1 Інтерфейс Figma [20]

Figma має багато переваг для користувачів, включаючи можливість спільної роботи в реальному часі, доступність та легкість використання. Ця платформа дозволяє створювати компоненти, що спрощує процес дизайну й дозволяє повторно використовувати елементи інтерфейсу. Крім цього, Figma дає можливість створювати прототипи та готувати презентації своїх проєктів.

Ця платформа не обмежується лише векторною графікою; вона також підтримує використання растрових зображень. Це особливо цінується дизайнерами, які хочуть створювати інтерактивні прототипи з реалістичними зображеннями.

Figma стала невід'ємним інструментом для безлічі професіоналів у сфері графічного дизайну та для команд, які займаються розробкою веб-інтерфейсів та мобільних додатків. Ця платформа набула популярності завдяки своїм унікальним можливостям:

1. Реальночасова співпраця: Figma дозволяє користувачам працювати над проектами одночасно у реальному часі. Це особливо корисно для розподілених географічно або віддалених команд, оскільки всі зміни відображаються миттєво, сприяючи ефективній співпраці.

2. Доступність: Figma працює у веб-браузері й не вимагає встановлення спеціального програмного забезпечення. Це робить його доступним з будь-якого пристрою та операційної системи.

3. Макети та прототипи: Figma дозволяє створювати макети і прототипи в одному інструменті. Це спрощує процес розробки та тестування користувацького досвіду, що є ключовим етапом у графічному дизайні.

4. Векторна та растрова графіка: Підтримка растрової графіки дозволяє працювати з різними типами зображень, роблячи Figma універсальним інструментом для дизайнерів.

5. Створення бібліотек та компонентів: Figma дозволяє створювати бібліотеки та компоненти для повторного використання у різних проектах. Це підвищує продуктивність та сприяє формуванню єдиної стилістики для всіх проєктів компанії чи бренда.

Отже, для порівняння було обрано саме веб-додаток онлайн курсів англійської мови. Було створено дизайн у нейтральних кольорах, які б привертали увагу. На рисунку 3.2 можна побачити інтерфейс головної сторінки, на якому розміщено меню додатку, рядок пошуку, загальна інформація про курси.

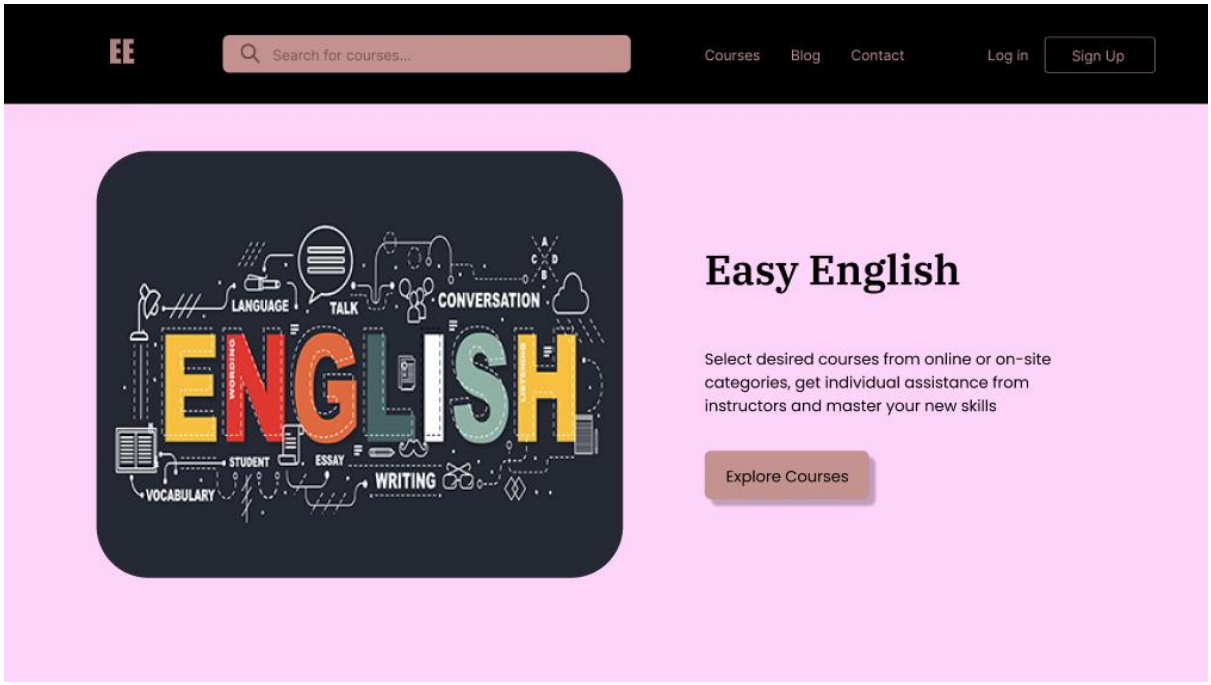


Рисунок 3.2 Головна сторінка веб-додатку Easy English

На рисунку 3.3 можна побачити сторінку Найпопулярніших курсів, є сторінка онлайн - курсів, курсів на сайті та курси з підприємництва англійською мовою. Можна побачити рейтинг курсів, кількість студентів, що вступили та пройшли курси та їх оцінки.

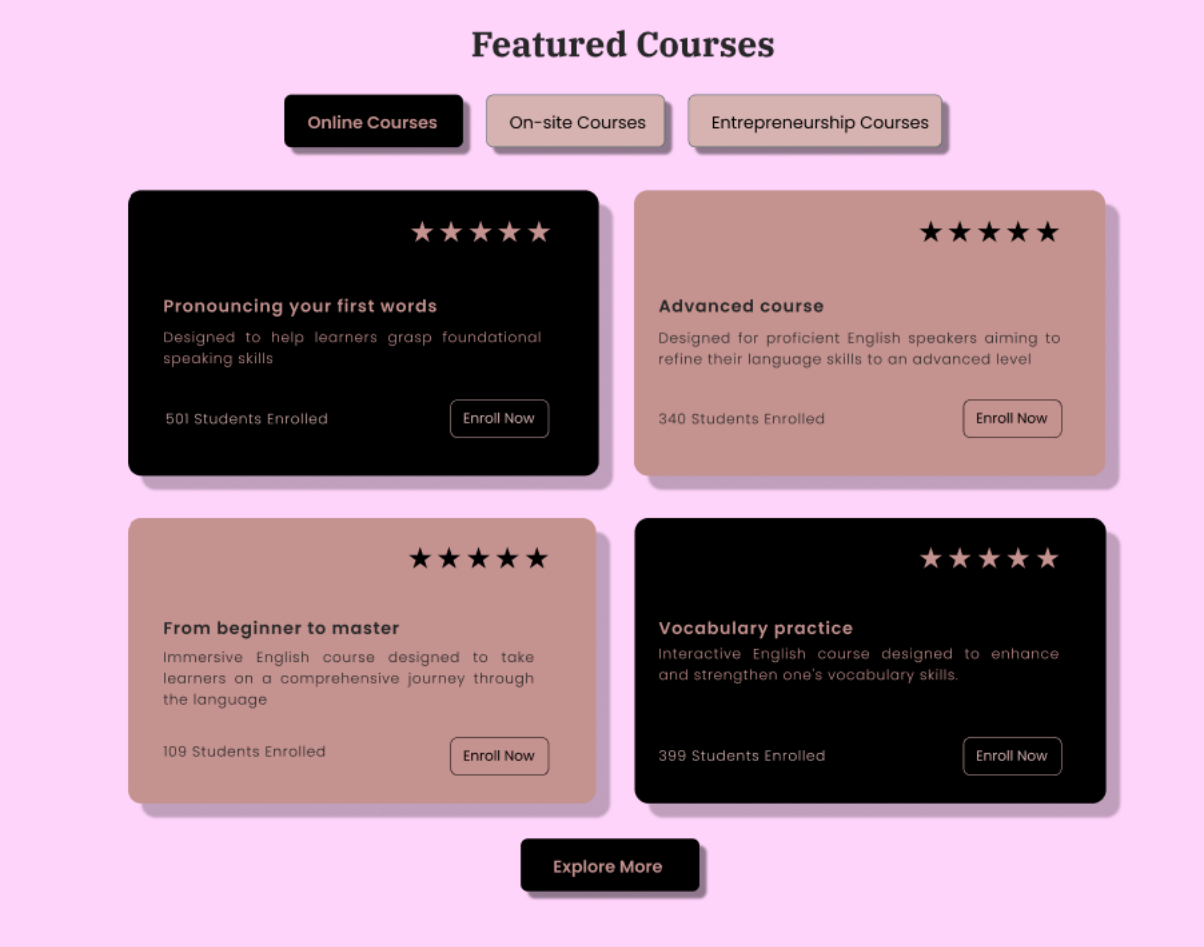


Рисунок 3.3 Сторінка найпопулярніших курсів

На рисунку 3.4 можна побачити сторінку відгуків.

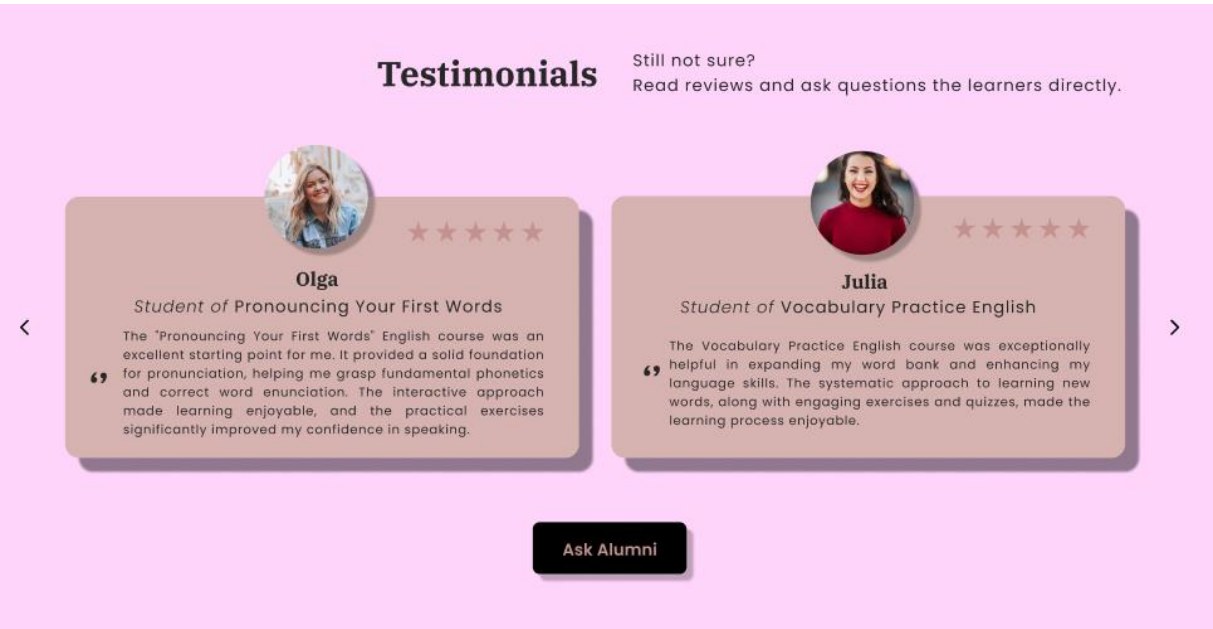


Рисунок 3.4 Сторінка відгуків з курсів

На рисунку 3.5 можна побачити сторінку з усіма доступними курсами, є меню курсів, можна встановити персональні фільтри за тривалістю, типом, постачальником курсів та ціною.

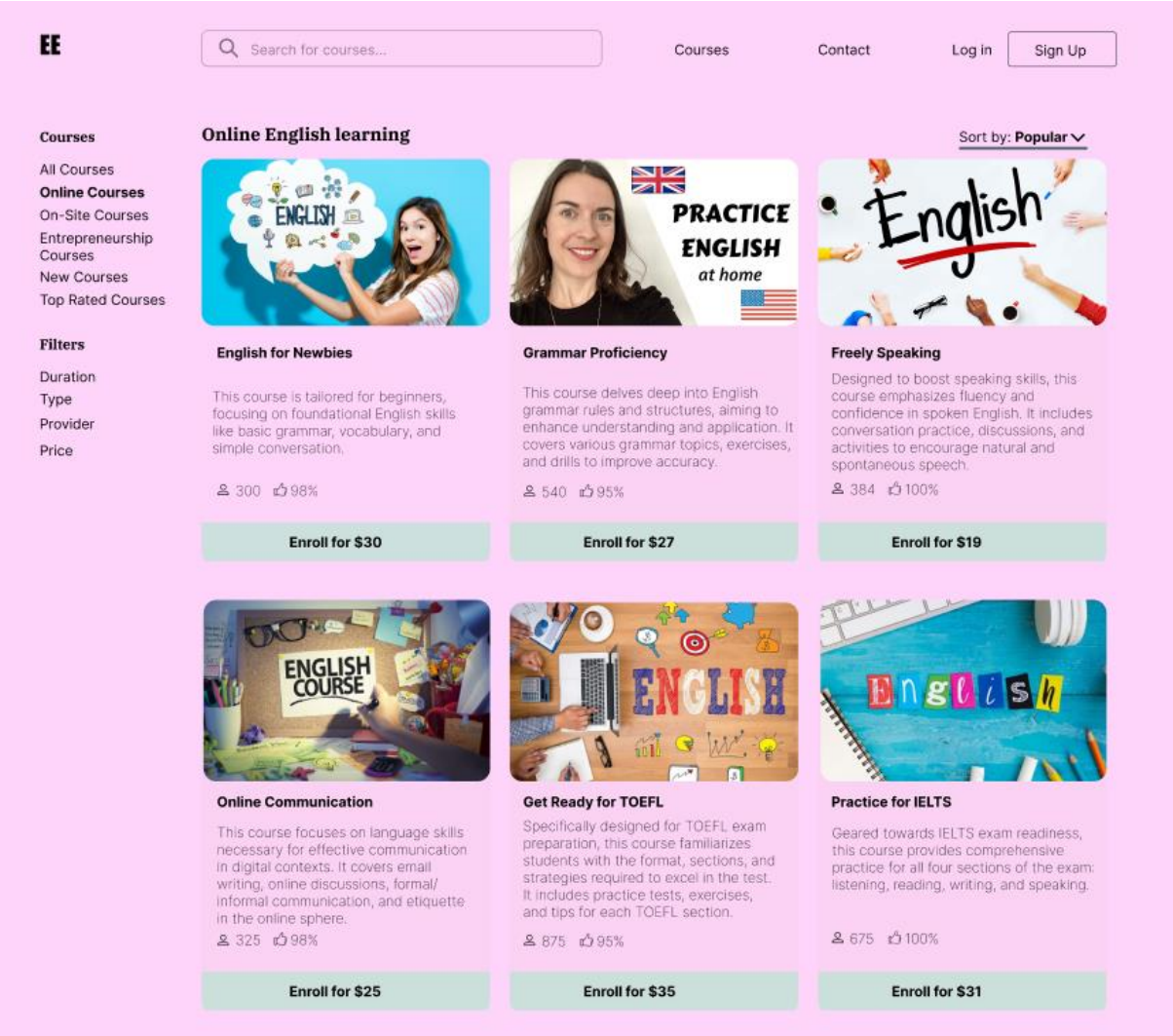


Рисунок 3.5 Сторінка усіх курсів

3.2 Порівняння середовищ розробки Angular та React для створення веб-додатку Easy English

Спершу розглянемо створення додатку за допомогою фреймворку **React.**

Був створений застосунок Easy English на базі фреймворку React, використовуючи його потужні можливості для розробки живих інтерфейсів. Було створено компоненти для показу списку курсів та подробиць кожного курсу. Був впроваджений механізм авторизації користувача через створення відповідної форми для введення імені та пароля. Створено компоненти для взаємодії з базою даних і обробки інформації про курси, модулі та уроки. Забезпечено зручний та привабливий інтерфейс за допомогою CSS-стилізації та використання фреймворків для візуальної привабливості.

```jsx
import React from 'react';

const CoursesList = ({ courses }) => {
  return (
    <div>
      <h2>Список курсів</h2>
      <ul>
        {courses.map(course => (
          <li key={course.id}>
            <h3>{course.title}</h3>
            <p>{course.description}</p>
            <button>Почати курс</button>
          </li>
        ))}
      </ul>
    </div>
  );
};
```

Рисунок 3.6 Код `CoursesList`

Цей код на рисунку 3.6 є частиною функціонального компонента у React, який називається `CoursesList`.

Він призначений для показу списку курсів на веб-сторінці. Давайте розглянемо його ближче:

1. Рядок `import React from 'react';` імпортує бібліотеку React, яка дозволяє використовувати JSX та компоненти React.

2. `const CoursesList = ({ courses }) => { ... }` це оголошення саме функціонального компонента `CoursesList`, який очікує на вхід масив `courses`. Цей аргумент - це дані про курси, які потрібно відобразити.

3. Частину `return ( ... )` складає HTML-розмітка, що відображає список курсів:

- `<div>` - контейнер, що містить весь перелік курсів.
- `<h2>Список курсів</h2>` - заголовок, що вказує назву списку курсів.
- `<ul>` - маркований список, де кожен курс представлений як елемент списку.
- `{courses.map(course => ( ... ))}` - використання методу `map()`, щоб перебрати кожен курс у масиві `courses` та створити для нього елемент списку `<li>`. Кожен елемент `<li>` містить:
- `<h3>{course.title}</h3>` - назва курсу, що відображається як заголовок третього рівня.
- `<p>{course.description}</p>` - опис курсу, що відображається як абзац.
- `<button>Почати курс</button>` - кнопка для старту курсу.

4. `};` - ця частина завершує тіло функції компонента.

5. `export default CoursesList;` - це експорт компонента `CoursesList`, щоб його можна було використовувати в інших частинах коду.

Отже, цей компонент відображає список курсів, що передаються через властивість `courses`, створюючи для кожного курсу заголовок, опис та кнопку для його початку.

```
export default CoursesList;
const courses = [
  {
    id: 1,
    title: 'English for Newbies',
    description: "This course is tailored for beginners,
  }
];
```

Рисунок 3.7 Створення масиву об'єктів курсів

Цей код на рисунку 3.7 містить інформацію про один певний курс, спрямований на тих, хто тільки починає вивчати англійську мову під назвою 'English for Newbies'. Він визначає масив `courses`, де містяться дані про доступні курси у додатку.

У цьому масиві знаходиться лише один елемент, що представляє об'єкт з трьома важливими полями:

- `id`: унікальний ідентифікатор цього курсу у системі.
- `title`: назва курсу - 'English for Newbies'.
- `description`: опис курсу, що вказує на його спрямованість на новачків та акцент на основних аспектах англійської мови, таких як базова граматика, словниковий запас та прості навички у мовленні.

```jsx
import React from 'react';

const CourseDetails = ({ course }) => {
  return (
    <div>
      <h2>{course.title}</h2>
      <p>{course.description}</p>
      <h3>Модулі</h3>
      <ul>
        {course.modules.map(module => (
          <li key={module.id}>
            <h4>{module.title}</h4>
            <p>{module.description}</p>
            <button>Почати модуль</button>
          </li>
        ))}
      </ul>
    </div>
  );
};

export default CourseDetails;
```

Рисунок 3.8 Компонент для відображення деталей курсу

Код на рисунку 3.8 написаний на React призначений для показу деталей певного курсу. Компонент `CourseDetails` відображає інформацію про курс та його складові. Цей компонент приймає дані про курс у вигляді об'єкту `course`.

У компоненті:

- Елемент `<h2>{course.title}</h2>` показує назву курсу.

- `<p>{course.description}</p>` відтворює опис курсу.
- Блок `<ul>...</ul>` формує список модулів курсу.
- `{course.modules.map(module => ... )}` використовує метод `map` для перебору кожного модуля курсу та створення елементу `<li>` для кожного з них.
- `<button>Почати модуль</button>` містить кнопку, яка, ймовірно, дозволяє користувачеві розпочати вивчення певного модулю.

Цей компонент розроблено для використання у веб-додатку з онлайн-курсами, де користувач може оглядати та взаємодіяти з деталями курсу, зокрема, починати вивчення окремих модулів.

```
import React, { useState } from 'react';
const Login = ({ onLogin }) => {
  const [username, setUsername] = useState('');
  const [password, setPassword] = useState('');
  const handleLogin = () => {
    onLogin(username);
  };
  return (
    <div>
      <h2>Вхід</h2>
      <input
        type="text"
        placeholder="Ім'я користувача"
        value={username}
        onChange={e => setUsername(e.target.value)}
      />
      <input
        type="password"
        placeholder="Пароль"
        value={password}
        onChange={e => setPassword(e.target.value)}
      />
      <button onClick={handleLogin}>Увійти</button>
    </div>
  );
};
export default Login;
```

Рисунок 3.9 Компонент для авторизації користувача (Login)

Цей фрагмент коду на рисунку 3.9 розглядає створення компонента авторизації користувача у React.

Ось розширений аналіз кожного етапу:

1. `import React, { useState } from 'react';` - завантаження бібліотеки React та використання функції `useState` для створення стану у компоненті.

2. `const Login = ({ onLogin }) => {` - створення функціонального компонента `Login`, який отримує параметр `onLogin` у якості функції. Ця функція викликається під час спроби входу користувача.

3. `const [username, setUsername] = useState('');` і `const [password, setPassword] = useState('');` - створення станів `username` та `password`, які зберігають ім'я та пароль користувача відповідно. Початкове значення для обох полів - пустий рядок.

4. `const handleLogin = () => { onLogin(username); };` - функція `handleLogin`, що викликає функцію `onLogin` з поточним ім'ям користувача (`username`) як аргументом.

5. JSX-код, що рендерить форму входу:

- `<h2>Вхід</h2>` - заголовок форми.
- `<input type="text" placeholder="Ім'я користувача" value={username} onChange={e => setUsername(e.target.value)} />` - поле для введення імені користувача. Значення поля пов'язане зі станом `username`, а зміна значення відбувається при введенні тексту.
- `<input type="password" placeholder="Пароль" value={password} onChange={e => setPassword(e.target.value)} />` - поле для введення паролю користувача. Значення поля пов'язане зі станом `password`, а зміна значення відбувається при введенні тексту.

- `<button onClick={handleLogin}>Увійти</button>` - кнопка, яка викликає функцію `handleLogin` при натисканні.

6. `export default Login;` - експорт компонента `Login` для подальшого використання його в інших частинах програми.

Цей код створює просту форму входу, що дозволяє користувачу ввести ім'я та пароль, зберігаючи їх у внутрішньому стані компонента. При натисканні кнопки "Увійти" виконується функція, яка передає ім'я користувача до зовнішньої функції `onLogin`, що використовується для обробки входу користувача.

```
import React from 'react';

const LessonTasks = ({ tasks }) => {
  return (
    <div>
      <h3>Завдання уроку</h3>
      <ol>
        {tasks.map(task => (
          <li key={task.id}>
            <p>{task.description}</p>
            <input type="text" placeholder="Введіть відповідь" />
            <button>Перевірити</button>
          </li>
        ))}
      </ol>
    </div>
  );
};

export default LessonTasks;
```

Рисунок 3.10 Відображення конкретного уроку у веб-додатку

Код на рисунку 3.10 реалізує React-компонент, який відображає деталі конкретного уроку у веб-додатку з онлайн-курсами.

Ось розшифровка кожного рядка:

1. `import React from 'react';` - цей рядок імпортує бібліотеку React, що дає можливість використовувати React у цьому файлі.

2. `const LessonDetails = ({ lesson }) => {` - тут створюється функціональний компонент `LessonDetails`, який отримує об'єкт `lesson` як вхідний параметр.

3. JSX-код для відображення деталей уроку:

- `<div>` - контейнер для відображення деталей уроку.
- `<h2>{lesson.title}</h2>` - показ назви уроку, яка передається через параметр `lesson`.
- `<p>{lesson.description}</p>` - показ опису уроку, який також передається через параметр `lesson`.
- `<video controls>` - тег для відтворення відео. Атрибут `controls` додає елементи управління відео (наприклад, кнопки "Play", "Pause" тощо).
- `<source src={lesson.videoURL} type="video/mp4" />` - встановлення відео для відтворення. Адреса відео (URL) передається через властивість `src` з параметру `lesson`.
- `Your browser does not support the video tag.` - повідомлення, яке відображається, якщо браузер не підтримує тег `<video>`.
- `<button>Завдання уроку</button>` - кнопка, яка може бути використана для переходу до завдань уроку. У цьому випадку, кнопка лише імітує наявність функціоналу.

4. `};` - закриття функції компонента.

5. `export default LessonDetails;` - експорт компонента `LessonDetails` для подальшого використання його в інших частинах програми.

Цей компонент створює блок інформації про певний урок, включаючи його назву, опис та відео. Його можна використовувати для відображення деталей уроків у веб-додатку.

Рисунок 3.11 Компонент для відображення списку уроків у модулі

Код на рисунку 3.11 є React-компонентом, що відображає список уроків у модулі.

Ось розгорнутий аналіз кожного рядка наведеного вище коду:

1. `import React from 'react';` - імпорт бібліотеки React, необхідної для створення React-компонентів.

2. `const LessonsList = ({ lessons }) => {` - оголошення функціонального компонента `LessonsList`. Ця функція приймає параметр `lessons` - масив об'єктів, які представляють уроки.

3. `return (` - початок повернення JSX-коду, який буде відображений у браузері.

4. `<div>` - відкриття блоку елементів у JSX.

5. `<h3>Список уроків</h3>` - заголовок списку уроків.

6. `<ul>` - ненумерований список.

7. `{lessons.map(lesson => (` - використання методу `map` для ітерації кожного об'єкту масиву `lessons` та створення елементів списку.

8. `<li key={lesson.id}>` - кожен урок відображається як елемент списку `<li>`. Параметр `key` допомагає React ідентифікувати кожен елемент у списку.

9. `<p>{lesson.title}</p>` - відображення назви уроку як абзацу.

10. `<button>Розпочати урок</button>` - кнопка для початку уроку, яка поки лише відображається, без прив'язки до конкретної функціональності.

11. `</li>)}` - закриття тегу для кожного елементу списку уроків.

12. `</ul>` - закриття тегу ненумерованого списку.

13. `</div>` - закриття блоку елементів.

14. `);` - закриття рядка з поверненням.

15. `};` - закриття оголошення функції компонента.

16. `export default LessonsList;` - експорт компонента `LessonsList` для подальшого використання його в інших частинах програми.

Отже, цей компонент створює ненумерований список уроків на основі даних, що передаються через параметр `lessons`. Кожен урок відображається як окремий елемент списку, який містить назву уроку та кнопку "Розпочати урок".

Розглянемо створення тих самих елементів за допомогою **Angular.**

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

@Component({
  selector: 'app-courses-list',
  template: `
    <div>
      <h3>List of Courses</h3>
      <ul>
        <li *ngFor="let course of courses">
          {{ course }}
        </li>
      </ul>
    </div>
  `,
})
export class CoursesListComponent {
  courses: string[] = ['English for Newbies'];
}
```

Рисунок 3.12 Створення масиву об'єктів курсів та компоненту з курсами

Цей фрагмент коду на рисунку 3.12 описує компонент Angular під назвою `CoursesListComponent`, який відповідає за відображення списку курсів у вашому веб-додатку.

Подивимося на кожен рядок докладніше:

1. `import { Component } from '@angular/core';` - цей рядок імпортує декоратор `Component` з пакету `@angular/core`, що дає можливість створювати Angular компоненти.

2. `@Component({ ... })` - цей рядок містить декоратор, який додає метадані до класу `CoursesListComponent`, визначаючи його шаблон та інші налаштування.

3. `selector: 'app-courses-list'` - вказує селектор компонента, який визначає, як використовувати його у шаблоні (наприклад, `<app-courses-list></app-courses-list>`).

4. `template: `...` - тут міститься шаблон компонента, що визначає його вигляд. У цьому випадку, тут є заголовок `<h3>` та список `<ul>`, який створюється за допомогою директиви `*ngFor`.

5. `<li *ngFor="let course of courses">{{ course }}</li>` - це використання директиви `*ngFor`, яка проходить по кожному елементу масиву `courses` та відображає його як окремий елемент списку `<li>`. `{{ course }}` відображає назву курсу всередині кожного елементу списку.

6. `CoursesListComponent` - це клас компонента, який містить логіку для відображення списку курсів.

7. `courses: string[] = ['English for Newbies'];` - оголошення масиву `courses`, який містить рядки з назвами курсів, у тому числі курс "English for Newbies" та інші приклади.

Отже, цей компонент призначений для відображення списку курсів у Angular-додатку.

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

@Component({
  selector: 'app-course-details',
  template: `
    <div *ngIf="course">
      <h2>{{ course.title }}</h2>
      <p>{{ course.description }}</p>
    </div>
  `,
})
export class CourseDetails {
  @Input() course: any;
}
```

Рисунок 3.13 Компонент для відображення деталей курсу

На рисунку 3.13 зображений компонент Angular `CourseDetails` спрямований на відображення деталей курсу.

Ось розгорнутий розбір його складових:

1. `import { Component, Input } from '@angular/core';`: імпорт необхідних модулів з Angular для створення компонента та роботи з введеними даними (`Input`).

2. `@Component({ ... })`: декоратор `@Component`, що описує компонент Angular. У ньому вказано селектор (`selector`) та шаблон (`template`) для цього компонента.

3. `selector: 'app-course-details'`: селектор, за допомогою якого можна використовувати цей компонент у шаблонах Angular. У даному випадку, компонент можна викликати як `<app-course-details></app-course-details>`.

4. `template: `...``: шаблон компонента, що містить HTML-структуру для відображення деталей курсу. Використовуються директиви Angular, такі як `*ngIf`, для умовного відображення елементів, залежно від наявності даних.

5. `<div *ngIf="course"> ... </div>`: цей блок коду відображається лише у разі наявності об'єкту `course` (інформації про курс). Це дозволяє уникнути відображення порожнього блоку, якщо дані відсутні.

6. `{{ course.title }}` та `{{ course.description }}`: використання двох фігурних дужок `{{ }}` для відображення властивостей `title` та `description` об'єкту `course` в відповідних HTML-елементах. Ці дані передаються до компонента через директиву `@Input()`.

7. `@Input() course: any;`: вказівка на те, що властивість `course` є вхідним параметром для цього компонента. Вона може приймати будь-який тип даних (`any`). Це дозволяє зовнішнім компонентам передавати дані про курс для відображення деталей.

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

@Component({
  selector: 'app-login',
  template: `
    <div>
      <h2>Login</h2>
      <form (ngSubmit)="login()">
        <input type="text" placeholder="Username" [(ngModel)]="username" name="username" required>
        <input type="password" placeholder="Password" [(ngModel)]="password" name="password" required>
        <button type="submit">Login</button>
      </form>
    </div>
  `,
})
export class LoginComponent {
  username: string = '';
  password: string = '';

  login() {
    console.log('Username:', this.username);
    console.log('Password:', this.password);
  }
}
```

Рисунок 3.14 Компонент для авторизації користувача (Login)

Фрагмент коду на рисунку 3.14 визначає компонент `LoginComponent` у фреймворку Angular.

Ось розширений опис його функціоналу:

1. Шаблон компонента:

- Відображає заголовок "Login" `<h2>Login</h2>`.
- Містить форму, яка викликає метод `login()` при поданні.
- Має два поля введення: одне для імені користувача `<input type="text">` та інше для паролю `<input type="password">`.
- Поля взаємодіють з властивістю `ngModel`, які пов'язані з `username` та `password` відповідно.
- Містить кнопку "Login", яка викликає метод `login()` при натисканні.

2. Клас компонента:

- Називається `LoginComponent`.
- Включає дві властивості: `username` і `password`, що зберігають введені користувачем дані.
- Містить метод `login()`, який виводить введені дані користувача (ім'я та пароль) у консоль за допомогою `console.log`.

Отже, цей Angular-компонент відображає форму для входу користувача, зчитує та зберігає введені дані. При натисканні кнопки "Login" він виводить ці дані у консоль. У застосунку метод `login()` використовувався б для передачі цих даних на сервер для автентифікації користувача.

```
import { Component, Input } from '@angular/core';
import { Lesson } from './lesson.model';

@Component({
  selector: 'app-lesson-details',
  template: `
    <div *ngIf="lesson">
      <h2>{{ lesson.title }}</h2>
      <p>{{ lesson.description }}</p>
      <!-- Other details of the lesson -->
    </div>
  `,
})
export class LessonDetailsComponent {
  @Input() lesson: Lesson | undefined;
}
```

Рисунок 3.15 Відображення конкретного уроку у веб-додатку

Цей Angular код, на рисунку 3.15 компонент відповідає за показ інформації про певний урок у веб-додатку.

Ось розглянемо кожен елемент:

1. `import { Component, Input } from '@angular/core';` - імпорт декораторів `Component` та `Input` з Angular, які дозволяють створювати компоненти та отримувати вхідні дані.

2. `import { Lesson } from './lesson.model';` - імпорт моделі `Lesson` з файлу `lesson.model`, де, ймовірно, описана структура даних уроків.

3. `@Component({ ... })` - використання декоратора `@Component`, що містить метадані компонента, такі як його селектор (`selector`) та шаблон (`template`).

4. `selector: 'app-lesson-details'` - селектор компонента, який визначає його ім'я для використання у шаблонах.

5. `template: `...` - шаблон компонента, де міститься розмітка для відображення інформації про певний урок. Використовує директиву `*ngIf` для перевірки наявності об'єкта `lesson` перед його відображенням.

6. `{{ lesson.title }}` та `{{ lesson.description }}` - використання вбудованого виразу Angular для відображення назви та опису уроку. Ці значення беруться з об'єкта `lesson`, який передається як вхідний параметр.

7. `@Input() lesson: Lesson | undefined;` - оголошення властивості `lesson` як вхідного параметра через декоратор `@Input()`. Цей параметр очікує об'єкт типу `Lesson` або `undefined` і використовується для передачі даних у компонент.

Цей компонент відображає назву та опис певного уроку, отриманого через вхідний параметр `lesson`, і був використаний для відображення інформації про урок у додатку на Angular.

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

@Component({
  selector: 'app-lessons-list',
  template: `
    <div>
      <h3>Список уроків</h3>
      <ul>
        <li *ngFor="let lesson of lessons" [key]="lesson.id">
          <p>{{ lesson.title }}</p>
          <button (click)="startLesson(lesson.id)">Розпочати урок</button>
        </li>
      </ul>
    </div>
  `,
})
export class LessonsListComponent {
  @Input() lessons: any[] = [];

  startLesson(lessonId: number) {
  }
}
```

Рисунок 3.16 Компонент для відображення списку уроків у модулі

Роздивимось детальніше елементи коду на рисунку 3.16:

1. `import { Component, Input } from '@angular/core';` - імпорт необхідних модулів та класів з бібліотеки Angular для подальшого використання.

2. `@Component({...})` - анотація, що визначає `LessonsListComponent` як Angular-компонент. У ній міститься інформація про селектор (`selector`) та шаблон (`template`), які визначають його відображення.

3. `selector: 'app-lessons-list'` - селектор компонента, за допомогою якого він може бути включений у шаблони інших компонентів.

4. `template: `...` - шаблон компонента, що містить HTML-код для відображення списку уроків. Використовується директива `*ngFor`, щоб створити елементи списку для кожного уроку з масиву `lessons`.

5. `<li *ngFor="let lesson of lessons" [key]="lesson.id">` - ця конструкція використовує директиву `*ngFor`, щоб створити елементи списку для кожного об'єкту `lesson` з масиву `lessons`. `[key]` вказує ключ для кожного елемента списку (зазвичай використовується `trackBy` для списків у Angular).

6. `<p>{{ lesson.title }}</p>` - відображення назви кожного уроку у вигляді абзацу.

7. `<button (click)="startLesson(lesson.id)">Розпочати урок</button>` - кнопка "Розпочати урок", що викликає метод `startLesson()` при кліку. Проте, цей метод поки порожній (відсутній код у тілі методу).

8. `@Input() lessons: any[] = [];` - оголошення вхідного параметра `lessons`, який компонент отримує з зовнішнього середовища.

9. `startLesson(lessonId: number) { }` - метод `startLesson`, що може бути викликаний при кліку на кнопку "Розпочати урок". Він приймає `lessonId`

як параметр, але наразі є порожнім. Логіка для цього методу ще не реалізована.

`LessonsListComponent` відображає список уроків, проте для повноцінної функціональності кнопки "Розпочати урок" потрібно додати логіку у метод `startLesson()`.

3.3 Оцінка продуктивності, масштабованості та швидкодії розроблених додатків

Оцінювання проводилося за певними пунктами:

1. Швидкодія в програмуванні означає ефективність виконання програми або її окремих частин, а також реакцію системи на події, які відбуваються в процесі роботи. У веб-розробці це охоплює кілька ключових аспектів:

- Час відгуку: Це інтервал, коли система реагує на запити користувача. Швидка відповідь на запити свідчить про ефективну роботу системи.
- Швидкість виконання коду: Це час, потрібний програмі для виконання певних завдань чи операцій. Швидкий код забезпечує підвищену продуктивність програми.
- Оновлення візуальних елементів: Це оцінка часу, необхідного для відображення або оновлення вмісту на сторінці. Сюди входить завантаження сторінки, оновлення даних, реакція на дії користувача та інші фактори.

У фронтенд-фреймворках, наприклад, React чи Angular, швидкодія включає реакцію на дії користувача, швидкість відображення компонентів

на сторінці, а також оптимізацію роботи з DOM та віртуальним DOM, що безпосередньо впливає на загальну продуктивність додатку.

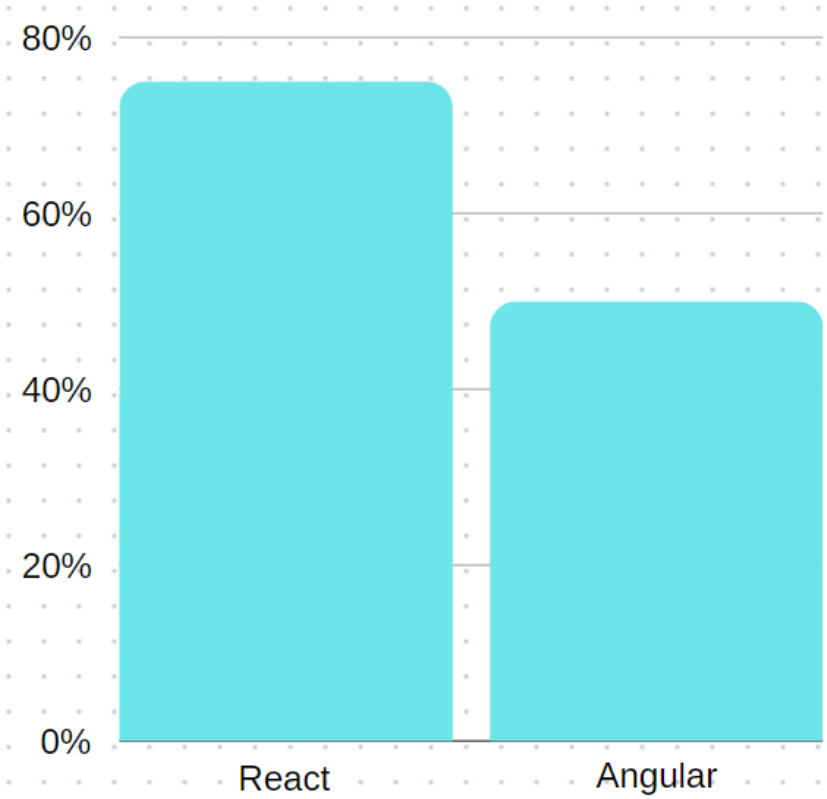


Рисунок 3.17 Швидкодія

На рисунку 3.17 можна побачити, що торкаючись теми швидкодії React є більш швидким ніж Angular.У більшості випадків, завдяки віртуальному DOM, React може бути швидшим у відображенні змін та операцій оновлення візуальних елементів.

2. У розробці програмного забезпечення, масштабованість означає здатність системи чи програми ефективно працювати при збільшенні обсягів даних, трафіку або користувачів, при цьому зберігаючи високу продуктивність та швидкодію.

У веб-розробці, масштабованість показує, наскільки веб-додаток чи система може обробляти зростаючі обсяги запитів, даних та користувачів, залишаючись стабільним та продуктивним. Це означає, що навіть при

значному збільшенні ресурсів додаток може працювати ефективно, не втрачаючи швидкодії та функціональності.

Оцінюючи масштабованість, розглядають такі аспекти:

- Продуктивність: Здатність додатку обробляти більше запитів чи користувачів без втрати швидкодії.
- Стійкість: Чи залишається додаток стабільним при збільшенні обсягів даних чи користувачів, не викликаючи помилок через перевантаження сервера.
- Масштабування: Легкість збільшення обсягів системи під ростуче навантаження, включаючи горизонтальне та вертикальне масштабування.
- Ефективне використання ресурсів: Оптимальне використання ресурсів (пам'ять, обчислювальна потужність) системи при збільшенні обсягів даних чи користувачів.
- Реактивність: Швидкість реакції системи на зміни у навантаженні чи обсягах даних.
- Горизонтальне та вертикальне масштабування: Можливість додавання ресурсів або розподілу навантаження на багато серверів (горизонтальне масштабування) та покращення характеристик окремих серверів (вертикальне масштабування).

Масштабованість є важливою у розробці програм, бо вона гарантує стабільну та ефективну роботу системи при зростанні навантаження та обсягів даних.

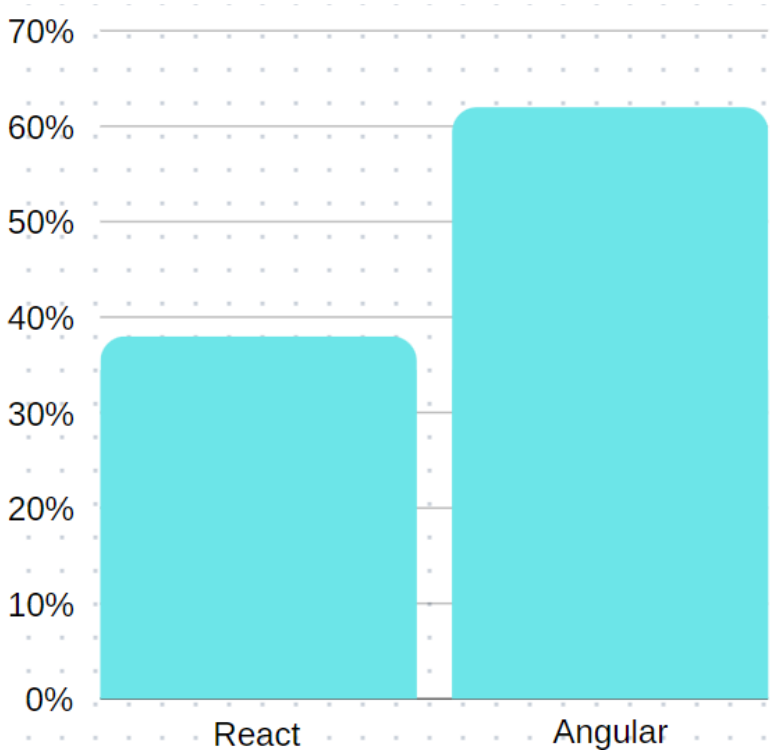


Рисунок 3.18 Масштабованість

На рисунку 3.18 можна простежити масштабованість фреймворків і зробити висновок, що з приводу масштабованості вони дуже різні.

React зазвичай добре підходить для невеликих і середніх проектів, але може потребувати більшої уваги до управління станом та організації коду великих додатків.

Angular навпаки забезпечує вбудовані інструменти для організації та підтримки великих проектів, проте може бути перебільшеною для менших за обсягом або менш складних проектів.

3. У розробці програмного забезпечення, витривалість означає, як добре система або додаток можуть працювати продовжуючи надійно та без перерв навіть під час помилок, відмов або несподіваних умов. Важливі складові витривалості включають:

- Стійкість до помилок: Система повинна вміти виявляти, вирішувати та відновлюватися після помилок, не порушуючи цілісність даних.

- Відновлення після відмов: Механізми, які дозволяють системі продовжувати працювати після проблем або відновлюватися після перебоїв у роботі.
- Надійність роботи: Забезпечення сталого доступу та правильної роботи системи, навіть при великому навантаженні або неочікуваних обставинах.
- Запобігання втратам даних: Механізми резервного копіювання та відновлення даних, що дозволяють уникнути втрат важливої інформації в разі відмови системи.
- Тестування та відлагодження: Здатність швидко виявляти, тестувати та усувати помилки, щоб забезпечити стабільну та надійну роботу системи.

4. Управління ресурсами:

- Це концепція ефективного використання доступних ресурсів у системі, таких як час CPU, пам'ять, мережеві ресурси. Мета - оптимізувати їх розподіл та використання для максимальної ефективності роботи системи. Наприклад, це може включати оптимізацію алгоритмів, кешування даних та використання асинхронних запитів для зменшення навантаження на сервери.

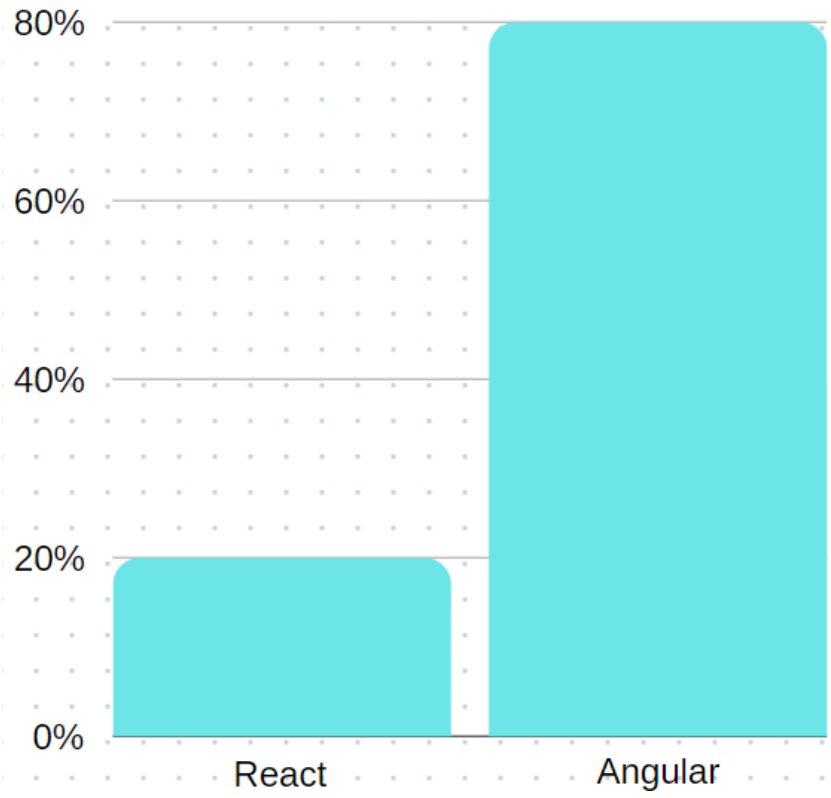


Рисунок 3.19

На рисунку 3.19 можна побачити, що у використанні та управлінні ресурсами Angular є набагато більш зручнішим.

React - вимагає більш ручне управління ресурсами, зокрема станом додатку та пам'яттю.

Angular - має вбудовані інструменти для керування ресурсами, що робить його менш витратним у відношенні до пам'яті та управління станом.

5. Реагування:

- Це здатність системи швидко реагувати на зміни вхідних даних або умов навантаження. У контексті масштабування це означає, наскільки швидко система може пристосовуватися до змін, збільшуючи або зменшуючи обчислювальні ресурси для забезпечення стабільності та продуктивності.

ВИСНОВКИ

Розробляючи веб-додаток на за допомогою фреймворків React та Angular було особисто проаналізовано декілька факторів розробки.

1. Швидкодія:

- React: Завдяки оптимізації віртуального DOM, React може бути швидшим у відображенні візуальних елементів. Інтелігентна обробка змін дозволяє уникнути зайвих перерахунків та оновлень.
- Angular: Має вбудовані механізми для виявлення змін, проте це може впливати на швидкодію через більшу кількість операцій перерахунку.

2. Зрозумілість:

- React: Використання JSX дозволяє комбінувати HTML та JavaScript, що спрощує сприйняття для розробників, знайомих з цими мовами.
- Angular: Розділення логіки та представлення у шаблонах полегшує розуміння коду, особливо для тих, хто має досвід з HTML та TypeScript.

3. Продуктивність розробки:

- React: Зазвичай вимагає менше коду для вирішення окремих завдань, але потребує вибору додаткових бібліотек для різних функціональних можливостей.
- Angular: Надає стандартні рішення для багатьох аспектів, що спрощує розробку та дозволяє розпочати роботу без необхідності вибору додаткових бібліотек.

4. Підтримка:

- React: Має велику та активну спільноту, що сприяє швидкому вирішенню проблем та доступу до різних рішень.

- Angular: Забезпечує широкий спектр інструментів та документації завдяки підтримці від компанії Google.

5. Продуктивність при великих проектах:

- React: Зручний для невеликих і середніх проектів, але потребує уваги до організації коду та управління станом в великих проектах.
- Angular: Має вбудовані інструменти для структуризації великих проектів, що сприяє організації коду.

6. Переносимість:

- React: Дозволяє переносити компоненти на платформи React Native для розробки мобільних додатків.
- Angular: Менш гнучкий у плані перенесення компонентів на інші платформи, порівняно з React.

Обидва фреймворки мають свої сильні та слабкі сторони, а вибір зазвичай залежить від конкретних потреб проекту та від уподобань розробників у команді.

ПЕРЕЛІК ПОСИЛАНЬ

1) Contributors to Wikimedia projects. Model–view–controller - Wikipedia. *Wikipedia, the free encyclopedia*. [Електронний ресурс] - Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Model–view–controller.

2) SEO Guide to Angular: Everything You Need to Know. *Search Engine Journal*. [Електронний ресурс] - Режим доступу до ресурсу: https://www.searchenginejournal.com/angular-seo-guide/303849/.

3) 关于OOP的SOLID原则. *wodewone's blog*. [Електронний ресурс] - Режим доступу до ресурсу: https://wodewone.github.io/2017/04/05/关于OOP的SOLID原则/.

4) TypeScript 编译错误问题的解决方法. *知乎专栏*. [Електронний ресурс] - Режим доступу до ресурсу: https://zhuanlan.zhihu.com/p/568341210.

5) Understanding Webpack (Configuring your own project with webpack). *Seiji's Portfolio*. [Електронний ресурс] - Режим доступу до ресурсу: https://www.seijivillafranca.com/post/understanding-webpack-configuring-your-own-project-with-webpack.

6) Ambriz J. C. A. Emulating React and JSX in Vanilla JS | Toptal®. *Toptal Engineering Blog*. [Електронний ресурс] - Режим доступу до ресурсу: https://www.toptal.com/javascript/emulating-react-jsx-in-vanilla-javascript.

7) Overview of React.js. *Patterns.dev*. [Електронний ресурс] - Режим доступу до ресурсу: https://www.patterns.dev/react/.

8) React Developer Tools. *Chrome Web Store*. [Електронний ресурс] - Режим доступу до ресурсу: https://chromewebstore.google.com/detail/react-developer-tools/fmkadmapgofadopljbjfkapdkoienihi.

9) Sublime Text 4. *Sublime Text - Text Editing, Done Right*. [Електронний ресурс] - Режим доступу до ресурсу: https://www.sublimetext.com/blog/articles/sublime-text-4.

10) Hallie L. CS Visualized: Useful Git Commands. *DEV Community*.[Електронний ресурс] - Режим доступу до ресурсу: https://dev.to/lydiahallie/cs-visualized-useful-git-commands-37p1.

11) Angular vs React: What to choose for web development in 2022?. *EnProwess Blog*. [Електронний ресурс] - Режим доступу до ресурсу: https://www.enprowess.com/blogs/angular-vs-react/.

12) IONOS editorial team. Shadow DOM. *IONOS Digital Guide*. [Електронний ресурс] - Режим доступу до ресурсу: https://www.ionos.co.uk/digitalguide/websites/web-development/shadow-dom/.

13) GitHub - chinmaymahajan/Registration-and-Login-using-MERN-stack: Simple Registration and Login component with MERN stack. *GitHub*. [Електронний ресурс] - Режим доступу до ресурсу: https://github.com/chinmaymahajan/Registration-and-Login-using-MERN-stack

14) Android Application Components - Techplayon. *Techplayon*. [Електронний ресурс] - Режим доступу до ресурсу: https://www.techplayon.com/applications-component/.

15) Programming principle "DRY": Don't Repeat Yourself. *Symflower for IntelliJ IDEA - Smart Unit Test Generator for Java*. [Електронний ресурс] - Режим доступу до ресурсу: https://symflower.com/en/company/blog/2022/programming-principle-dry/.

16) redux-store - Fullstack Station. *Fullstack Station*. [Електронний ресурс] - Режим доступу до ресурсу: https://fullstackstation.com/huong-dan-su-dung-redux-hieu-qua-trong-ung-dung-react/redux-store/.

17) Яка різниця між імперативним, процедурним та структурованим програмуванням?. *QA Stack*. [Електронний ресурс] - Режим доступу до ресурсу: https://qastack.com.ua/software/117092/whats-the-difference-between-imperative-procedural-and-structured-programming.

18) Custom RxJS Operators: Enhancing Your Reactive Programming Arsenal. *Medium*. [Електронний ресурс] - Режим доступу до ресурсу: https://levelup.gitconnected.com/custom-rxjs-operators-enhancing-your-reactive-programming-arsenal-565591e3bdf2.

19) Реактивное программирование с RxJS и Angular. *IT-компанія повного циклу розробки програмного забезпечення WEZOM - Київ, Україна*. [Електронний ресурс] - Режим доступу до ресурсу: https://wezom.com.ua/blog/reaktivnoe-programmirovanie-s-rxjs-i-angular.

20) Getting Started · Jest. *Jest · Delightful JavaScript Testing*. [Електронний ресурс] - Режим доступу до ресурсу: https://jestjs.io/docs/getting-started.

21) Contributors to Wikimedia projects. Mocha (фреймворк) – Вікіпедия. *Википедия – свободная энциклопедия*. [Електронний ресурс] - Режим доступу до ресурсу: https://ru.wikipedia.org/wiki/Mocha_(фреймворк).

22) Що таке Фігма та навіщо вона потрібна?. *Lemon School*. [Електронний ресурс] - Режим доступу до ресурсу: https://lemon.school/blog/chto-takoe-figma-i-zachem-ona-nuzhna.

23) Panda's Design. Как да създадем многоъгълници във Figma, 2019. *YouTube*. [Електронний ресурс] - Режим доступу до ресурсу: https://www.youtube.com/watch?v=EseJNgyJG9M.

24) Angular vs React: A Detailed Side-by-Side Comparison. *Kinsta®*. [Електронний ресурс] - Режим доступу до ресурсу: https://kinsta.com/blog/angular-vs-react/.

25) Comparative analysis of Angular and React development frameworks. *Research Gate*. [Електронний ресурс] - Режим доступу до ресурсу: https://www.researchgate.net/publication/374345000_Comparative_analysis_of_Angular_and_React_development_frameworks.

26) React vs Angular: A Comparative Analysis - Appventurez. *Appventurez*. [Електронний ресурс] - Режим доступу до ресурсу:https://www.appventurez.com/blog/react-vs-angular-comparative-analysis/.

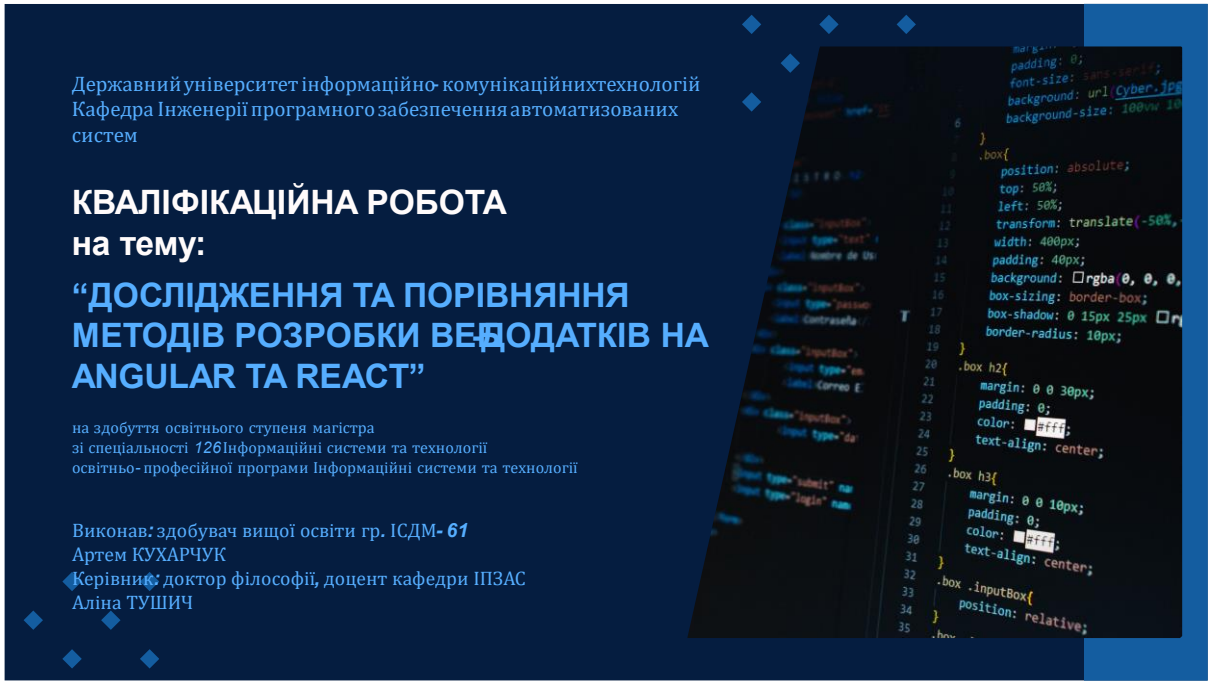
27) React vs Angular in 2023: A Comprehensive Comparison. *Medium*. [Електронний ресурс] - Режим доступу до ресурсу: https://medium.com/@reshailawan/react-vs-angular-in-2023-a-comprehensive-comparison-4821703cd073.

28) Romanyuk O. Angular vs React: Which One to Choose for Your App. *freeCodeCamp.org*. [Електронний ресурс] - Режим доступу до ресурсу: https://www.freecodecamp.org/news/angular-vs-react-what-to-choose-for-your-app-2/.

29) Sols M. Angular vs. React–Which One To Choose For Your Web App?. *AI-enhanced digital product development · Monterail*. [Електронний ресурс] - Режим доступу до ресурсу: https://www.monterail.com/blog/angular-vs-react.

30) Comparative analysis of Angular and React development frameworks. *Semantic Scholars*. [Електронний ресурс] - Режим доступу до ресурсу:https://www.semanticscholar.org/paper/Comparative-analysis-of-Angular-and-React-Skrzypiec-Plechawska-Wójcik/e3d788ee547193210360f3c06a53a956d5acfb08.

ПРЕЗЕНТАЦІЯ




React JS
REACT JS
Компонентна архітектура
В основі React лежить компонентна архітектура, що дозволяє побудову ефективних та динамічних веб-додатків, розбитих на невеликі, самостійні компоненти.
Ефективне управління та продуктивність
React дозволяє ефективно управляти станом додатку та забезпечує швидке оновлення інтерфейсу, концентруючись лише на необхідних змінах, що забезпечує високу продуктивність та реакцію на події.
Динамічність
React надає зручний інструментарій для розробки динамічних веб-додатків, сприяючи швидкій розробці, підтримці і розширенню веб-інтерфейсів з використанням компонентів та ефективного управління станом.


ANGULAR
ANGULAR
Заснований на TypeScript
Angular використовує TypeScript, що забезпечує статичну типізацію та інші переваги для розробки більших та більш складних проектів, забезпечуючи надійний код та підвищуючи продуктивність розробників.
Розширений інструментарій
Angular пропонує розширений інструментарій, включаючи компонентну архітектуру для розбиття додатку на невеликі та повторно використовувані компоненти, систему залежностей та інші функції, що полегшують розробку та підтримку великих проектів.
Декларативна структура
Angular пропонує декларативну структуру коду, що полегшує розуміння та підтримку проектів. Крім того, вбудована система інструментів для тестування спрощує створення та виконання тестів, що робить розробку надійною та забезпечує високу якість програмного забезпечення.

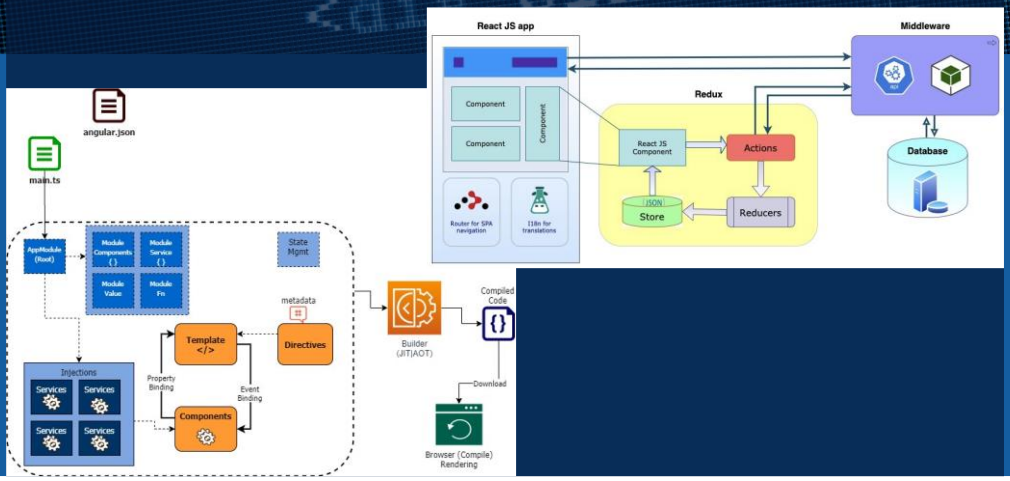
ПОРІВНЯЛЬНИЙ АНАЛІЗ

Характеристика	React JS	Angular
Тип	Бібліотека JavaScript	Фреймворк
Мова програмування	JavaScript (з можливістю TypeScript)	TypeScript
Архітектура	Компонентна	Компонентна
Спосіб оновлення інтерфейсу	Віртуальний DOM	Реактивність та Зона
Управління станом	Flux, Redux	Вбудований в Angular
Тестування	Розширені можливості тестування через Jest та інші	Вбудована система для тестування
Спільнота та Підтримка	Велика спільнота, активний розвиток	Велика спільнота, підтримка від Google
Продуктивність	Швидка реакція на зміни через віртуальний DOM	Ефективне управління пам'яттю та оптимізації
Використання	Широке використання в індустрії	Широке використання в великих корпоративних проектах
Масштабованість	Підходить для різних масштабів проектів	Добре підходить для великих та складних проектів
Навчальна Крива	Низька	Помірна, особливо для початківців

КОМПОНЕНТНИЙ ПІДХІД

Характеристика	React JS	Angular
Синтаксис компонентів	JSX – дозволяє вбудовувати HTML у JavaScript	HTML з підтримкою директив та властивостей Angular
Структура компонентів	Декларативна, описується через функціональні та класові компоненти	Ініціалізується через класи та шаблони компонентів
Пропси (властивості)	Передаються в компоненти та використовуються для передачі даних	Використовуються для передачі даних в компоненти та зовнішніх директив
Стан компонентів	Внутрішній стан для управління динамікою компонента	Зберігається в класових властивостях та оновлюється через двостороннє зв'язування даних
Директиви/ Атрибути	Мало використовує директив, більш оснований на властивостях та методах компонентів	Використовує директиви для маніпуляції DOM та властивостей компонентів
Цикли життя компонентів	Має життєвий цикл для управління відомими подіями життєвого циклу компонента	Власний цикл життєвого циклу компонента, який може бути складним для вивчення
Інжектори та Сервіси	Використовує сервіси та контексти для обміну даними між компонентами	Використовує інжектор та Angular сервіси для обміну даними та бізнес-логікою
Шаблонізація	Використовує JSX для шаблонізації та відображення компонентів	Використовує HTML та шаблони для створення вигляду компонентів
Двостороннє зв'язування	Здійснюється через стан та пропси для контролю введення та виведення	Використовує [(ngModel)] для зручного двостороннього зв'язування
Розширюваність	Мало вбудованих інструментів, але може бути розширений за допомогою бібліотек та плагінів	Має велику кількість вбудованих функцій та можливостей для розширення за допомогою модулів та бібліотек
Компіляція	Не потребує попередньої компіляції, використовується прямо в браузері	Вимагає попередньої компіляції, що може зменшити швидкодію під час розробки
Виробництво (Production)	Має можливість оптимізації для виробництва, але не вбудовано	Вбудовані засоби оптимізації та зменшення розміру для виробництва

КОМПОНЕНТНИЙ ПІДХІД
REACT JS / ANGULAR



РЕАКТИВНИЙ ПІДХІД

Характеристика	React JS	Angular
Реактивність та Зона	Використовує реактивний підхід через бібліотеку RxJS	Широко використовує концепцію Зон для виявлення та відстеження змін в додатку
Обробка Подій	Здійснює обробку подій за допомогою власної системи та бібліотеки React Hooks	Використовує події та спеціальні об'єкти подій для взаємодії з компонентами
Реактивні Оператори	Може використовувати реактивні оператори для роботи з потоками даних	Широкий спектр вбудованих реактивних операторів для роботи з асинхронними подіями та даними
Асинхронність та Запити	Використовує проміси та async/await для асинхронних операцій	Має вбудовану підтримку для роботи з асинхронними операціями, включаючи HTTP запити
Реактивне Програмування	Реактивність реалізована через використання RxJS або інших бібліотек	Вбудована підтримка реактивного програмування через RxJS та інші інструменти
Спостереження за Змінами	Використовується патерн слідкування за змінами для виявлення та реакції на зміни	Використовується система Зон для виявлення та сповіщення про зміни в контексті додатка
Поведінка на Вхідні Зміни	Вимагає явного визначення реакцій на вхідні зміни в компонентах	Використовує автоматичну виявлення та реакцію на зміни в контексті Зони
Валідація та Обробка Помилок	Залежить від розробників для реалізації валідації та обробки помилок	Має вбудовану систему валідації та обробки помилок в контексті Зони
Масштабованість та Робота з Потоками	Забезпечує гнучкість у роботі з потоками та масштабованість	Широкі можливості для роботи з потоками та підтримка великих та складних додатків

REACT JS ПІДХОДИТЬ ДЛЯ

01 Проектів з великою кількістю відокремлених компонентів, де функціональність може бути розподілена між ними.

02 Розробки легких та швидких веб-додатків, орієнтованих на високий рівень продуктивності.

03 Ситуацій, де потрібна гнучкість та відкритість до додаткових бібліотек і плагінів для розширення функціоналу.

04 Проектів, де важлива швидкість розробки та велика спільнота для підтримки та розвитку.

- **Facebook**
- **Instagram**
- **Airbnb**
- **Netflix**

ANGULAR ПІДХОДИТЬ ДЛЯ

01 Великих та складних підприємницьких додатків, які вимагають повного фреймворку для стандартизації та керування кодом

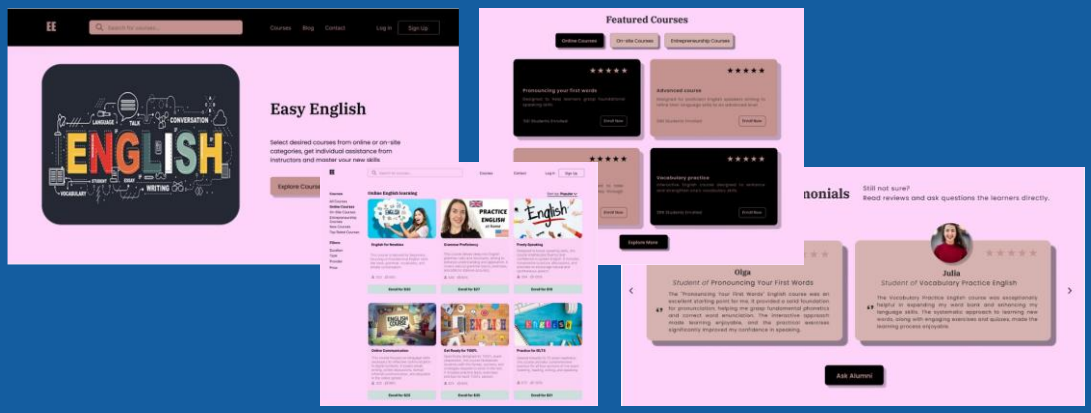
02 Проектів, де безпека є пріоритетом, і вбудовані інструменти для обробки аутентифікації та авторизації грають важливу роль

03 Розробки великих та масштабованих додатків, які потребують високого ступеня модульності та ієрархії компонентів

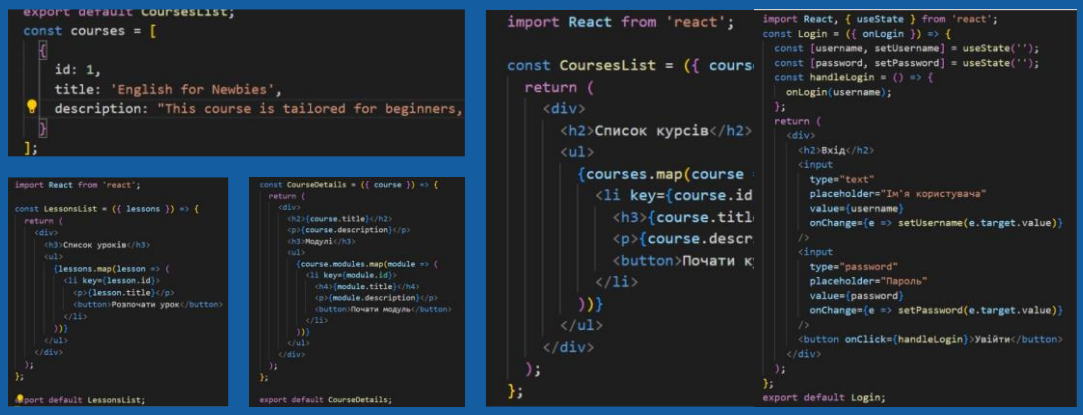
04 Проектів, де важливий однорідний стиль коду та жорстка архітектура для управління складністю

- **Google**
- **Bank of America**
- **LinkedIn**
- **IBM**

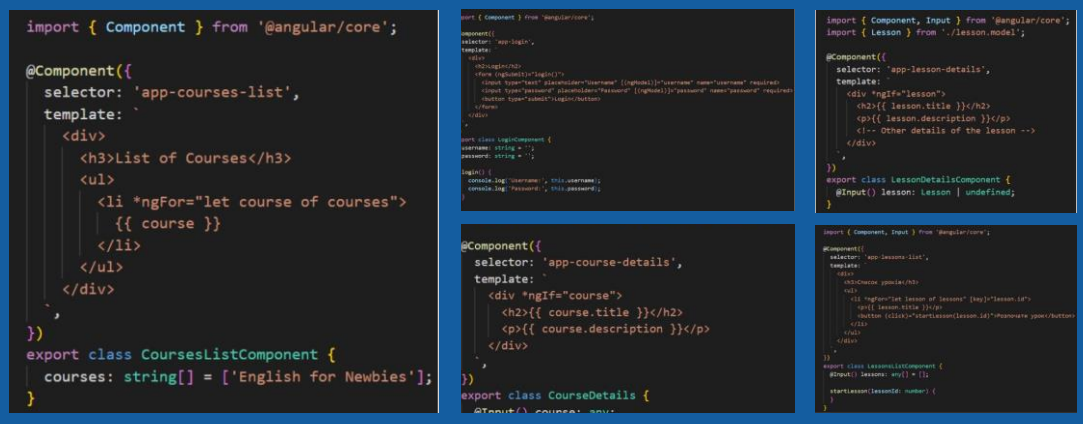
ПРАКТИЧНЕ ПОРІВНЯННЯ



ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ **REACT JS**

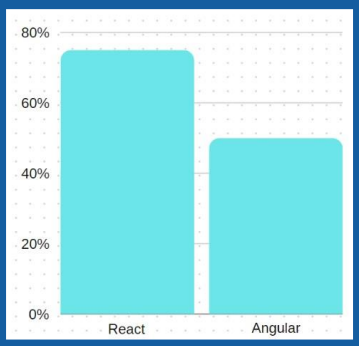


ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ **ANGULAR**

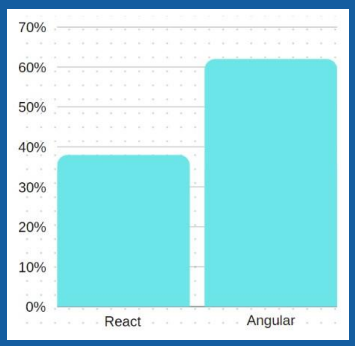


РЕЗУЛЬТАТИ

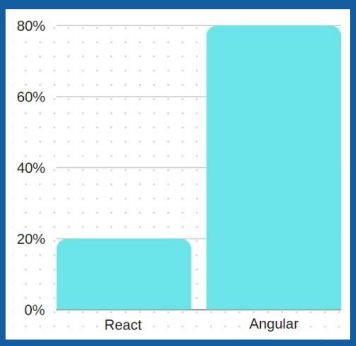
ШВИДКОДІЯ

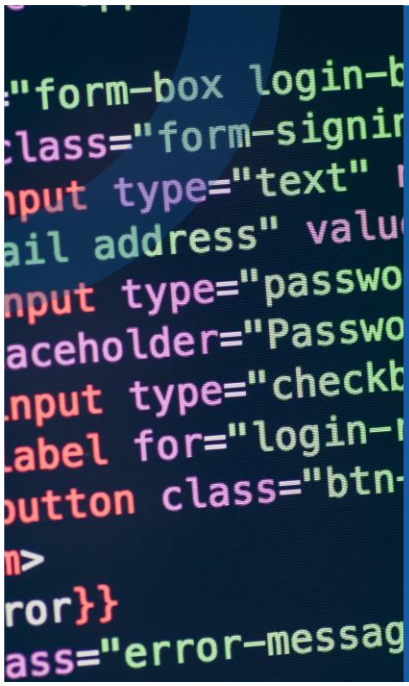


МАСШТАБОВАНІСТЬ



ПРОДУКТИВНІСТЬ





ВИСНОВКИ

1.

Швидкодія:

- React виграє завдяки оптимізації віртуального DOM та інтелігентній обробці змін, уникаючи зайвих перерахунків.
- Angular, хоча має вбудовані механізми виявлення змін, може стикатися з впливом на швидкодію через більшу кількість операцій перерахунку.

2.

Зрозумілість:

- React використовує JSX, що полегшує комбінування HTML та JavaScript та сприяє сприйняттю для розробників.
- Angular полегшує розуміння коду через розділення логіки та представлення у шаблонах, особливо для тих, хто має досвід з HTML та TypeScript.

3.

Продуктивність розробки:

- React, зазвичай, вимагає менше коду для окремих завдань, але потребує вибору додаткових бібліотек.
- Angular надає стандартні рішення для багатьох аспектів, що спрощує розробку та дозволяє розпочати роботу без вибору додаткових бібліотек.

АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

Кухарчук А. С. «Дослідження та порівняння методів розробки веб-додатків на Angular та React». Тези доповіді на Всеукраїнській Науково-технічній конференції «Сучасний стан та перспективи розвитку IoT». – Київ, 18 квітня 2023 р.

Кухарчук А. С. «Дослідження та порівняння методів розробки веб-додатків на Angular та React». Стаття у загальногалузевому науково-виробничому журналі «Зв'язок», м.Київ - №1, 2024. – С.185-201.

Дякую за увагу !