МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЧЕРКАСЬКИЙ ДЕРЖАВНИЙ ТЕХНОЛОГІЧНИЙ УНІВЕРСИТЕТ
Факультет інформаційних технологій і систем
Кафедра програмного забезпечення автоматизованих систем

Курсовий проект
на тему «Порівняння практичної ефективності популярних Front-end фреймворків на прикладі створення веб-додатку типу To-Do»
з дисципліни «Науково дослідна робота»
Напрям підготовки 6.050103 програмна інженерія
Пояснювальна записка

Перевірив:
зав. кафедрою ПЗАС
Первунінський С. М.
_____
оцінка
“___” _____ ____ р.

Виконав:
Студент групи МПЗ-1904
Гаврилюк В. Є.
Залікова книжка №19113
“___” _____ ____

Черкаси 2020

## ЗМІСТ

## СПИСОК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАЧЕНЬ

HTML - Hypertext Markup Language

CSS - Cascading Style Sheets

MVC - Model view controller

Url - Uniform Resource Locator

SDK - Software Development Kit

IDE - Integrated Development Environment

REST - Representational State Transfer

API - Application Programming Interface

CLI - Command Line Interface

DLL - Dynamic-link library

SPA - Single Page Application

# ВСТУП

На сьогодні комп'ютерні технології зробили значний прорив у своєму розвитку. Комп'ютери стають дедалі потужнішими і меншими, все сильніше входячи в наше повсякденне життя. Розповсюдження персональних комп'ютерів, а також їх компактних аналогів, таких як смартфони, планшети чи розумні годинники, призвело до постійного використання великої кількості програм. Але велика розповсюдженість подібної техніки є одночасно і її мінусом, адже всі вони працюють на різних операційних системах та мають різні вимоги до власного програмного забезпечення. Веб-додатки покликані вирішити цю проблему розрізненості в створенні програм під різні платформи. Так як доступ до мережі інтернет сьогодні може здійснюватися практично з будь-якого персонального комп'ютера чи компактного носимого пристрою, адже програми браузери є під кожну операційну систему, то й доступ до веб-додатків можливий із будь-якого подібного пристрою. Для створення користувацьких інтерфейсів цих додатків найчастіше використовують три фронтенд фреймворки: Angular, React та Vue. Також їх часто використовують і для створення інтерфейсів звичайних веб-сайтів, які також є невід’ємною частиною нашого життя.

Під час своєї роботи мені доводиться часто стикатися із цими трьома найпопулярнішими фреймворками. Всі вони створені досягати однієї і тієї ж мети - полегшення створення користувацьких інтерфейсів для веб-сайтів та веб-додатків -, але різними шляхами та програмними прийомами. Тому мені стало цікаво дослідити їх переваги на практиці, а не в теорії.

Метою курсового проекту є дослідження практичних переваг від використання одного із популярних фронтенд фреймворків. Основною вимогою є перевірка на прикладі створення веб-додатку.

Актуальність полягає в тому, що фронтенд фреймворки допомагають пришвидшити та дещо спростити розробку користувацьких інтерфейсів та дозволити програмісту сконцентрувати свою увагу на більш низькорівневій логіці. Хоч фреймворки і потребують спеціаліста із специфічними знаннями, але один такий робітник, може заміти кілька звичайних спеціалістів, які б розробляли додаток без використання фреймворку.

Об’єктом дослідження курсового проекту є стандартна архітектура користувацького інтерфейсу веб-додатку типу To-Do, а також механізми та програмні прийоми, за допомогою яких цей інтерфейс реалізують фреймворки.

Предметом дослідження даного курсового проекту є веб-додаток, для якого реалізується один і той самий користувацький інтерфейс, але за допомогою трьох різних фронтенд фреймворків, з метою визначення найбільшої практичної вигоди від використання кожного з них.

# 1. КОРОТКІ ВІДОМОСТІ ПРО ФРЕЙМВОРКИ

Розглянемо слово "фреймворк", яке є дійсно новим неологізмом і не так давно з'явилося в нашій мові. Слово почали використовуватися приблизно в першій половині XXI століття. Якщо розглядати переклад слова з англійського – це "конструкція" або "структура".

Суть фреймворка полягає якраз в перекладі слова. Це програмне середовище спеціального призначення, своєрідний каркас, який використовується для того, щоб істотно полегшити процес об'єднання певних компонентів при створенні програм. Це основа, яка дозволяє додавати компоненти в залежності від потреб. База, на якій можна сформувати програму будь-якого призначення досить швидко і без особливих труднощів.

Якщо порівнювати динамічну бібліотеку (DLL), яка відрізняється вельми обмеженим функціоналом, і фреймворк, що вважається основою програм - можна виділити суттєву перевагу фреймворків. Саме фреймворк є сполучною ланкою, яка об'єднує всі використовувані програмні компоненти. Також всередині фреймворка часто є необхідні тематичні бібліотеки.

Класифікація фреймворків:

- фреймворки додатків;
- фреймворки програмних моделей;
- фреймворки концептуальних моделей.

Всі розглянуті в даній курсовій роботі фреймворки призначені для створення SPA. Додаток, на якому проводилося дослідження також являється односторінковим веб-додатком.

Single Page Application (SPA чи односторінковий застосунок) реалізує зручні для користувача сервіси, що наповнені інтерактивом. Найпростішим прикладом є Gmail.

SPA — веб-застосунок, розташований на одній фізичній HTML сторінці. Така сторінка одноразово завантажує усі необхідні ресурси (JavaScript, CSS, images тощо.) і більше не перезавантажується. Переходи за посиланнями не призводять до реального перезавантаження сторінки, а її вміст змінюється «на

льоту», тобто динамічно. За необхідністю виконується запит на сервер для отримання даних, і після їх отримання формується контент «нової» сторінки. Кожна окрема віртуальна сторінка прописується у маршрутизаторі (router). У нашому застосунку існує два маршрути: головна та чат.

## 1.1. Фреймворк Angular

Angular (зазвичай так називають фреймворк Angular 2 або Angular 2+, тобто вищі версії) — написаний на TypeScript front-end фреймворк з відкритим кодом, який розробляється під керівництвом Angular Team у компанії Google, а також спільнотою приватних розробників та корпорацій. Angular — це AngularJS, який переосмислили та який був повністю переписаний тією ж командою розробників.

Angular представляє фреймворк для створення клієнтських додатків. В основному, Angular призначений для створення комплексних enterprise-застосунків, а саме односторінкових веб застосунків (SPA). Всередині фреймворку реалізовано:

- модульність;
- анімації;
- маршрутизація;
- робота з бекендом;
- зберігання/обробка/відображення даних;
- робота з формами та шаблонами тощо.

В цьому плані Angular є спадкоємцем іншого фреймворка AngularJS. У той же час Angular це не нова версія AngularJS, а принципово новий фреймворк.

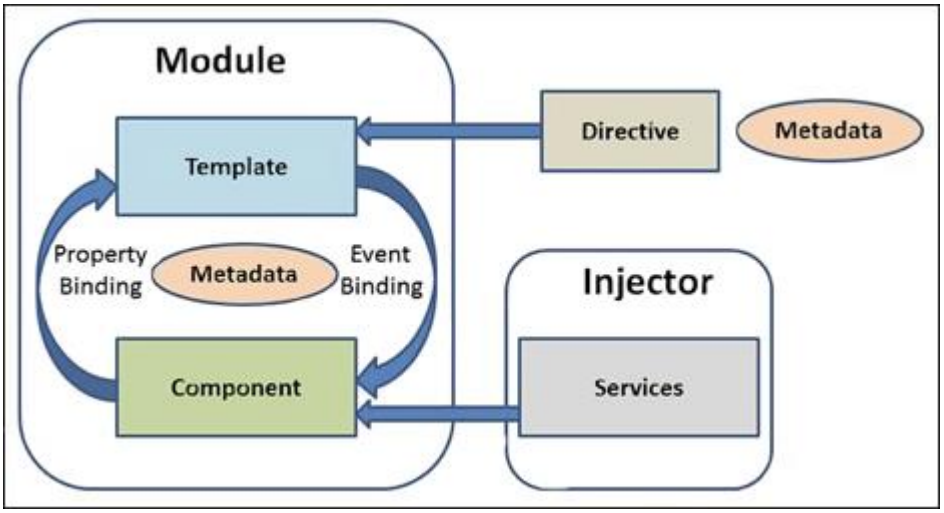


*Рисунок 1.1 – Загальна архітектура веб-додатку на Angular.*

Перш за все, варто зазначити, що Angular застосунки пишуться на TypeScript, а не на чистому JavaScript. Версія синтаксису для JavaScript не отримала широкого розповсюдження, тому на даний момент у документації усі синтаксичні конструкції описані з використанням синтаксису TypeScript.

Коротко оглянемо основні пункти. Архітектура Angular складається з:

- module;
- component;
- template;
- service;
- router;
- pipe;
- directives.

Модулі (Module) — структурні одиниці застосунку, які інкапсулюють певну логіку. В Angular це структури, які зберігають певні компоненти, директиви та сервіси, об’єднані певною логікою. Прикладом може слугувати профіль користувача, модуль для написання листа, огляд списку листів тощо.

Компоненти (Component) — typescript клас, який зберігає дані та логіку відображення цих даних у шаблоні (представленні). Шаблон тісно пов’язаний з компонентом. Дані з компонента можна з легкістю відображати у шаблоні,

використовуючи спеціальний синтаксис. Компонент також може «знімати» дані з шаблона та отримувати їх безпосередньо у скрипті.

Шаблон (Template) — фрагмент html-коду з додаванням спеціального синтаксису. Він дозволяє впроваджувати в шаблон дані з компонента без використання innerHTML та подібних методів. Шаблон прописується у компоненті та є частиною його конфігурації.

Сервіс (Service) в Angular являє собою typescript класи, які виконують задачі, пов'язані з отриманням, зберіганням та обробкою даних. Наприклад, логування, перетворення даних для подальшої передачі у компонент, звернення до backend та ін. На відміну від компонентів та директив сервіси не працюють з представленнями (шаблонами) напряму.

Задачі сервісів:

- Надання даних застосунку. Сервіс сам може зберігати дані у пам'яті або, з метою отримання даних, звертатися до якогось джерела даних, наприклад, до сервера;
- Сервіс може організувати канал взаємодії між окремими компонентами застосунку;
- Сервіс може інкапсулювати бізнес-логіку, різноманітні обчислювальні задачі, задачі з логування, які краще виносити поза компоненти. Таким чином, код компонентів буде зосереджений, безпосередньо, на роботі з представленням. До того ж, можемо розв'язати проблему повторення коду, якщо нам знадобиться виконати одну й ту саму задачу у різних компонентах і класах.

Роутер (Router)— маршрутизатор, який призначений для переходу між екранами з метою відображення різного контенту.

Іншими словами, коли в адресному рядку браузера у вас змінюється фрагмент URL, маршрутизатор відстежує ці зміни та завантажує ту або іншу частину застосунку.

Директиви та Пайпи — більш специфічні конструкції, які простіше продемонструвати у коді, ніж описати словами.

## 1.2. Фреймворк React

React (старі назви: React.js, ReactJS) — відкрита JavaScript бібліотека для створення інтерфейсів користувача, яка покликана вирішувати проблеми часткового оновлення вмісту веб-сторінки, з якими стикаються в розробці односторінкових застосунків. Розробляється Facebook, Instagram і спільнотою індивідуальних розробників.

React дозволяє розробникам створювати великі веб-застосунки, які використовують дані, котрі змінюються з часом, без перезавантаження сторінки. Його мета полягає в тому, щоб бути швидким, простим, масштабованим. React обробляє тільки користувацький інтерфейс у застосунках. Це відповідає видові у шаблоні модель-вид-контролер (MVC), і може бути використане у поєднанні з іншими JavaScript бібліотеками або в великих фреймворках MVC, таких як AngularJS. Він також може бути використаний з React на основі надбудов, щоб піклуватися про частини без користувацького інтерфейсу побудови веб-застосунків. Як бібліотеку інтерфейсу користувача React найчастіше використовують разом з іншими бібліотеками, такими як Redux.

React має власні особливості, якими він не схожий на інші фреймворки.

Одностороння передача даних. Властивості передаються в рендерер компоненту, як властивості HTML тегу. Компонент не може напряму змінювати властивості, що йому передані, але може їх змінювати через callback функції. Такий механізм називають «властивості донизу, події нагору».

React підтримує віртуальний DOM, а не покладається виключно на DOM браузера. Це дозволяє бібліотеці визначити, які частини DOM змінилися, порівняно зі збереженою версією віртуального DOM, і таким чином визначити, як найефективніше оновити DOM браузера. Таким чином програміст працює зі сторінкою, вважаючи що вона оновлюється вся, але бібліотека самостійно вирішує які компоненти сторінки треба оновити.

Компоненти React зазвичай написані на JSX. Код написаний на JSX компілюється у виклики методів бібліотеки React. Розробники можуть так само писати на чистому JavaScript. JSX нагадує іншу мову, яку створили у компанії

Фейсбук для розширення PHP, XHP. Код, написаний у JSX, потребує перетворення за допомогою такого інструменту, як Babel, для того, щоб його могли зрозуміти веб-браузери. Ця обробка, як правило, виконується під час процесу збірки, перш ніж програма буде запущена.

React використовують не лише для рендерингу HTML в браузері. Наприклад, Facebook має динамічні графіки які рендеряться в теги <canvas>, Netflix та PayPal використовують ізоморфне завантаження для рендерингу ідентичного HTML на сервері та клієнті.

Методи життєвого циклу – це різні методи, які вбудовуються за допомогою ReactJS. Вони дозволяють розробнику обробляти дані в різних точках життєвого циклу програми React. Наприклад:

- shouldComponentUpdate – це метод життєвого циклу, який каже Javascript оновити компонент, використовуючи логічні змінні.
- componentWillMount – це метод життєвого циклу, який каже Javascript налаштувати певні дані перед монтуванням компонентів (вставлення у віртуальний DOM).
- componentDidMount – це метод життєвого циклу, подібний до компонента WillMount, за винятком того, що він працює після методу render, і може використовуватися для додавання JSON-даних, а також для визначення властивостей та станів.
- render є найважливішим методом життєвого циклу, необхідним у будь-якому компоненті. Метод render – це те, що з'єднується з JSX і відображати власний JSX.

## 1.3. Фреймворк Vue

Vue.js — JavaScript-фреймворк що використовує шаблон MVVM для створення інтерфейсів користувача на основі моделей даних, через реактивне зв'язування даних. Vue – це прогресивний фреймворк для створення користувацьких інтерфейсів. На відміну від фреймворків-монолітів, Vue створений придатним для поступового впровадження. Його ядро в першу чергу вирішує завдання рівня уявлення (view), що спрощує інтеграцію з іншими

бібліотеками та існуючими проектами. З іншого боку, Vue повністю підходить і для створення складних односторінкових додатків (SPA), якщо використовувати його спільно з сучасними інструментами та додатковими бібліотеками.

Vue має власні особливості, якими він не схожий на інші фреймворки.

Vue використовує синтаксис шаблонів на основі HTML, що дозволяє декларативно зв'язувати рендеринг DOM з основними екземплярами даних в Vue. Всі Vue шаблони валідні HTML, і можуть бути розпарсені браузерами та HTML парсерами. Всередині Vue компілює шаблони в рендерингові функції віртуального DOM. В поєднанні з реактивною системою, Vue здатний розумно обчислити кількість компонентів для ре-рендингу та застосувати мінімальну кількість маніпуляцій з DOM, коли стан застосунку зміниться.

В Vue ви можете використовувати синтаксис шаблонів або напряму писати рендерингові функції використовуючи JSX. Для того, щоб це зробити просто замініть шаблон на рендерингову функцію. Рендерингова функція відкриває можливості для потужних патернів базованих на компонентах - для прикладу, нова транзитна система тепер повністю базована на компонентах, що використовує рендерингові функції всередині.

Одна із найвиразніших особливостей Vue — це ненав'язлива реактивна система. Моделі це просто плоскі JavaScript об'єкти. Це робить керування станами дуже простим та інтуїтивним. Vue надає оптимізований ре-рендеринг з коробки без потреби робити що-небудь додатково. Кожен компонент слідкує за своїми реактивними залежностями під час рендерингу, тому система знає точно коли має відбуватись ре-рендеринг і які компоненти потрібно ре-рендерити.

- Vue надає різноманітні шляхи для застосування ефектів переходу, коли елемент додають, оновлюють або видаляють з DOM. Наприклад:
- автоматичне застосування класів для CSS переходів та анімацій
- інтегрування сторонніх бібліотек для CSS анімацій, таких як Animate.css
- використовувати JavaScript для прямих маніпуляцій з DOM під час переходів
- інтегрування сторонніх JavaScript бібліотек анімацій, таких як Velocity.js

Vue сам по собі не включає роутингу, та є vue-router пакет, який вирішує це питання. Він підтримує зв'язування вкладенних шляхів з вкладеними компонентами і пропонує деталізований контроль над переходами. Vue дозволяє створення додатків за допомогою компонетів. Якщо додати vue-router до цього, все що потрібно зробити це зв'язати ваші компоненти з роутами і дозвольте vue-router вирішувати де їх рендерити.

## 2. ОГЛЯД СТВОРЕННЯ НОВОГО ПРОЕКТУ

Всі три додатки розроблені з використанням стандартних CLI (ng generate для Angular, create-react-app для React і vue-cli для Vue). CLI - це, скорочення, яке розшифровується як Command Line Interface, тобто – інтерфейс командного рядка.

Тепер пропоную поглянути на зовнішній вигляд додатків, про які йде тут мова.

| | **Angular** | **React** | **Vue** |
|---|---|---|---|
| **Приблизний розмір стартового пакету додатку, кілобайти** | 500 | 100 | 80 |
| **Поточна версія** | 10.1.0 | 16.13.1 | 2.6.11 |

*Таблиця 2.1 – Порівняння розмірів пакетів новостворених додатків*

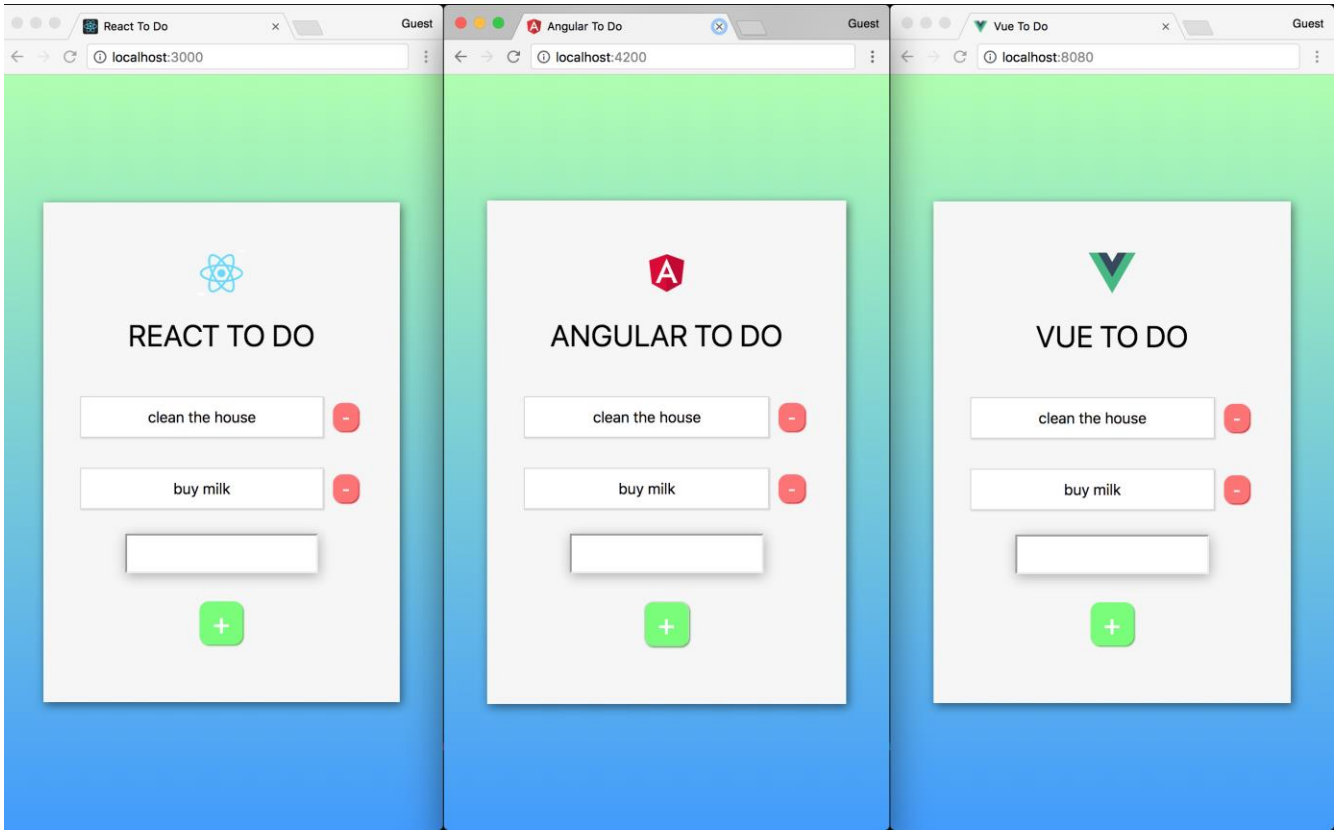


*Рисунок 2.1 – Вигляд додатків після написання (зліва на право): React, Angular та Vue*

У всіх додатках використовується абсолютно однаковий CSS-код, єдина різниця полягає в тому, де саме розміщені відповідні файли. З огляду на це, давайте поглянемо на структуру проектів.


*Рисунок 2.1 – Структура проектів (зліва на право): React, Angular та Vue*

У порівнянні з Vue і React, Angular має масу файлів. Одна з основних причин цього полягає в тому, що в той час як Vue поміщає все для одного

компонента в один файл, а React розбиває CSS в свій власний файл, Angular поміщає CSS в один файл, HTML в інший, а код компонента в третій. Хоча ми можемо помістити все це в один файл, але рекомендується зберігати їх окремо.

Ще одна причина такої кількості файлів полягає в тому, що Angular використовує TypeScript. TypeScript - це «надбудова JavaScript, яка надає необов'язкову статичну типізацію, класи і інтерфейси».

TypeScript насправді просто додає купу речей поверх JavaScript. Якщо ми напишемо рядок JavaScript в нашому коді TypeScript, він буде працювати в звичайному режимі. TypeScript дозволяє вам легко створювати класи і забезпечувати їх використання.

## 3. МУТАЦІЯ ДАНИХ В ДОДАТКУ

Зміна даних ще називається «мутацією даних». Мова йде про зміни, що вносяться в дані, які зберігає наш додаток. Так, якщо нам потрібно змінити ім'я якогось людини з «Джон» на «Марк», то мова йде про «мутацію даних». Саме в підході до зміни даних і знаходиться ключова відмінність між Angular, React і Vue. А саме, Vue створює об'єкт data, в якому знаходяться дані, і вміст якого можна вільно змінювати. React же створює об'єкт state, в якому зберігається стан додатку, і при роботі з яким для зміни даних потрібні деякі додаткові зусилля. Однак в React все влаштовано саме так не без причини, нижче ми про це поговоримо, а для початку розглянемо вищезазначені об'єкти.

Ось як виглядає об'єкт data, який використовується в Vue.

```
data() {
      return {
          list: [
              {
                todo: 'clean the house'
              },
              {
                todo: 'buy milk'
              }
          ],
      }
  },
```

Ось як виглядає об'єкт state, який використовується в React:

```
constructor(props) {
        super(props);
        this.state = {
            list: [
                {
                    'todo': 'clean the house'
                },
                {
                    'todo': 'buy milk'
                }
            ],
        };
    };
```

Як бачите, в обох випадках ми описуємо одні й ті ж дані, вони просто по-різному оформлені. В результаті можна сказати, що передача початкових даних компонентів в Vue і React виглядає дуже і дуже схоже. Але, як уже було сказано, підходи до зміни існуючих даних в цих фреймворками розрізняються.

В Angular же все, що нам потрібно зробити для відображення деяких даних, це дві речі:

- мати ці дані в якості поля в класі нашої компоненти, в даному випадку ToDo;
- вказати на нього у шаблоні за допомогою фігурних дужок {{ToDo.Item}}.

І це все, що нам потрібно зробити! Коли ми десь змінимо ToDo.Item, Angular оновить відображення даних за нас. React не забезпечує цих хуків, оскільки виклик setState все одно повторно відрендерить компонент.

## 4. СТВОРЕННЯ НОВИХ ЕЛЕМЕНТІВ ToDo

Для створення нових елементів ToDo в додатку на фреймворку React нам треба скористатися наступним кодом:

```
createNewToDoItem = () => {
    this.setState( ({ list, todo }) => ({
      list: [
          ...list,
        {
          todo
        }
      ],
      todo: ''
    })
  );
};
```

Тут у поля, яке призначено для введення даних (input), є атрибут value. Цей атрибут оновлюється автоматично завдяки використанню пари взаємопов'язаних функцій, які формують те, що називається двосторонньої прив'язкою даних. Цей різновид двостороннього зв'язку ми створюємо завдяки наявності додаткового прослуховувача подій onChange, прикріпленого до поля input. Погляньмо на код цього поля для того, щоб вам було зрозуміліше те, що тут відбувається.

```
<input type="text" value={this.state.todo} onChange={this.handleInput}/>
```

Функція handleInput викликається при зміні значення поля input. Це призводить до оновлення елемента ToDo, який знаходиться всередині об'єкта state, шляхом установки його в те значення, яке є в полі input. Ось як виглядає функція handleInput.

```
handleInput = e => {
  this.setState({
    todo: e.target.value
  });
};
```

Тепер, коли користувач натискає на сторінці додатка кнопку «+» для додавання в список нового запису, функція createNewToDoItem викликає метод this.setState і передає йому функцію. Ця функція приймає два параметри. Перший – це весь масив list з об'єкта state, а другий – це елемент todo, який оновлюється функцією handleInput. Потім функція повертає новий об'єкт, який містить колишній масив list, і додає новий елемент todo в кінець цього масиву. Робота зі списком організована з використанням оператора spread. І нарешті, в ToDo записується порожній рядок, що автоматично оновлює значення value в поля input.

Для додавання нового елемента в список справ в Vue використовується наступна конструкція.

```
createNewToDoItem() {
```

```
    this.list.push(
        {
            'todo': this.todo
        }
    );
    this.todo = '';
}
```

У Vue в поля введення є директива v-model. Вона дозволяє організувати двосторонню прив'язку даних. Погляньмо на код цього поля і поговоримо про те, що тут відбувається.

```
<input type="text" v-model="todo"/>
```

Директива v-model прив'язує поле до ключа, який є в об'єкті даних, який називається toDoItem. Коли сторінка завантажується, в toDoItem записани порожній рядок і виглядає це як todo: ‘’.

Якщо тут вже є якісь дані, щось на зразок todo: 'add some text here', то в поле введення потрапить такий же текст, тобто – 'add some text here'. У будь-якому випадку, якщо повернутися до прикладу з нового рядка, текст, який ми введемо в поле, потрапить, за рахунок прив'язки даних, у властивість todo. Це і є двостороння прив'язка даних, тобто, введення нових даних в поле призводить до запису цих даних в об'єкт data, а оновлення даних в об'єкті призводить до появи цих даних в полі.

Тепер згадаємо функцію createNewToDoItem(), про яку ми говорили вище. Як можна побачити, ми кладемо вміст todo в масив list, а потім записуємо в todo порожній рядок.

В Angular створення нового елементу відбуваєтсья дещо простіше, адже фреймворк багато роботи виконує сам без нашого втручання. Коли ми приступаємо до створення нового елементу, наш компонент списку - це лише список завдань. ngFor може взяти наш список і відобразити кожен елемент, як і foreach. Саме так це виглядає в коді класу нашої компоненти.

```
export class ToDoComponent implements OnInit {
     constructor() {}
     ToDos: ToDo[] = [new ToDo("clean the house"), new ToDo("buy milk")];
```

```
      Input: string = "";
      ngOnInit() {}
      deleteItem(i: number) {
            this.ToDos.splice(i, 1);
      }
      addItem() {
            this.ToDos = this.ToDos.concat(new ToDo(this.Input));
      }
}
```

А ось так це виглядає на шаблоні нашої компоненти.

```
<div *ngFor="let ToDo of ToDos; index as i" >
  <ToDoItem [ToDo]="ToDo" (Deleted)="deleteItem(i)"></ToDoItem>
</div>
<input type="text" class="form-control" id="name"
       required [(ngModel)]="Input" name="name" #name="ngModel">
```

Велика різниця тут полягає в тому, що Angular буде виконувати оновлення без нашої участі. Коли ми отримуємо якісь введені дані, все, що нам потрібно зробити, це передати рядок у новий об’єкт ToDo та додати його до нашого списку, а про решту подбає Angular.

```
this.ToDos = this.ToDos.concat(new ToDo(this.Input));
```

## 5. ВИДАЛЕННЯ ЕЛЕМЕНТІВ ІЗ СПИСКУ

В React операція видалення елементів із списку виконується так.

```
deleteItem = indexToDelete => {
    this.setState(({ list }) => ({
      list: list.filter((toDo, index) => index !== indexToDelete)
    }));
};
```

У той час як функція deleteItem знаходиться в файлі ToDo.js, звернутися до неї без проблем можна і з ToDoItem.js, спочатку передавши цю функцію як властивість в <ToDoItem />. Ось як це виглядає:

```
<ToDoItem deleteItem={this.deleteItem.bind(this, key)}/>
```

Тут ми спочатку передаємо функцію, що робить її доступною дочірнім компонентам. Крім того, ми здійснюємо прив'язку this і передачу параметра key. Цей параметр використовується функцією для того, щоб відрізнити елемент ToDoItem, який потрібно видалити, від інших елементів. Потім, всередині компонента ToDoItem, ми робимо наступне.

```
<div className="ToDoItem-Delete" onClick={this.props.deleteItem}>-</div>
```

Все, що потрібно зробити для того, щоб звернутися до функції, що знаходиться в батьківському компоненті – це скористатися конструкцією this.props.deleteItem.

Операція видалення елементів із списку у Vue виконується так.

```
onDeleteItem(todo){
  this.list = this.list.filter(item => item !== todo);
}
```

У Vue потрібно дещо інший підхід до видалення елементів, ніж той, яким ми користувалися в React. А саме, тут треба виконати три дії. По-перше, ось що треба зробити в елементі, для якого потрібно буде викликати функцію його видалення.

```
<div class="ToDoItem-Delete" @click="deleteItem(todo)">-</div>
```

Потім потрібно створити функцію emit у вигляді методу в дочірньому компоненті (в даному випадку - в ToDoItem.vue), яка виглядає наступним чином.

```
deleteItem(todo) {
    this.$emit('delete', todo)
}
```

Далі, ви можете помітити, що ми, додаючи ToDoItem.vue всередині ToDo.vue, звертаємося до функції.

```
<ToDoItem v-for="todo in list"
          :todo="todo"
          @delete="onDeleteItem" // <-- this :)
          :key="todo.id" />
```

Це – те, що називається призначеним для користувацьким прослуховувачем подій. Він реагує на виклики emit з рядком delete. Якщо він фіксує подібну подію, він викликає функцію onDeleteItem. Вона знаходиться всередині ToDo.vue, а не в ToDoItem.vue. Ця функція, як уже сказано вище, просто фільтрує масив todo, що знаходиться в об'єкті data для того, щоб видалити з нього елемент, на який натиснули.

Крім того, варто відзначити, що в прикладі з використанням Vue можна було б просто написати код, що відноситься до функції emit, всередині прослуховувача @click. Виглядати це може так.

```
<div class="ToDoItem-Delete" @click="$emit('delete', todo)">-</div>
```

Це зменшило б кількість кроків, необхідних для видалення елемента списку, з трьох до двох. Як саме вчинити – питання вподобань розробника.

Видалення елементів списку в Angular є дещо простішим з погляду роботи програміста. Нам не потрібно нічого передавати від нашого компонента ToDoItem, нам просто потрібно лише ініціювати подію. Typescript повинен знати, якого типу належать наші вихідні дані, але оскільки нам все одно, ми просто прописуємо тип any, тому наш Output виглядає так:

```
@Output() Deleted: EventEmitter<any> = new EventEmitter();
```

Нам все ще потрібно ініціювати цю подію в ToDoItem. У нас є чудово відформатований div, який виглядає як кнопка форм і нам просто потрібно підключити цю функцію до події кліку.

```
<div class="ToDoItem-Delete" (click)="deleteItem()">-</div>
```

Потім deleteItem викликає .emit () на Deleted:

```
deleteItem() {
  this.Deleted.emit();
 }
```

А далі Angular передає цю подію до ToDoComponent. Ще в нашому ToDoComponent ми пов’язали функцію deleteItem() із нашим компонентом ToDoItem. Ми також передаємо йому індекс поточного елемента, тому ми знаємо, який саме з них потрібно видалити.

```
<div *ngFor="let ToDo of ToDos; index as i" >
    <ToDoItem [ToDo]="ToDo" (Deleted)="deleteItem(i)"></ToDoItem>
</div>
```

Тоді все, що нам потрібно зробити, це видалити ToDo за цим індексом, і Angular оновить вивід елементів.

```
deleteItem(i: number) {
  this.ToDos.splice(i, 1);
}
```

Якщо підвести короткі підсумки цього розділу, то можна сказати, що в React доступ до функцій, описаних в батьківських компонентах, організовується через this.props (з огляду на те, що props передається дочірнім компонентів, що є стандартним прийомом, який можна зустріти буквально всюди). В Vue ж дочірні компоненти повинні викликати події за допомогою функції emit, а ці події вже обробляються батьківським компонентом. Аналогічна логіка із Vue є і в Angular – нам також потрібно із дочірніх компонентів передавати саму подію, за допомогою emit в батьківський компонент, де й буде відбуватися вся наступна обробка даних. Одна в Angular програмісту потрібно менше задумуватися над зміною виведення даних на екран, так як фреймворк сам оновить вивід автоматично.

# 6. ПЕРЕДАЧА ДАНИХ ДОЧІРНІМ КОМПОНЕНТАМ

В React властивості передаються дочірньому компоненту при його створенні. Наприклад, так:

```
<ToDoItem key={key} item={todo} />
```

Тут можна бачити дві властивості, переданих компоненту ToDoItem. З цього моменту до них можна звертатися в дочірньому компоненті через this.props.

Наприклад, для того, щоб звернутися до властивості item.todo, досить використовувати конструкцію this.props.item.

У Vue властивості дочірнім компонентів також передаються при їх створенні.

```
<ToDoItem v-for="todo in list"
            :todo="todo"
            :key="todo.id"
            @delete="onDeleteItem" />
```

Після цього вони передаються в масив props дочірнього компонента, наприклад, за допомогою конструкції props: ['todo']. Звертатися до цих властивостей в дочірніх компонентах можна по імені, в нашому випадку це ім'я 'todo'.

В Angular передача даних в дочірні компоненти відбувається за дещо іншою схемою, так як розділення на компоненти в цьому фреймворку працює інакше. Оскільки компонент ToDoItem стосується лише відображення одного завдання, то компонент ToDoComponent повинен мати можливість надати компоненту ToDoItem дані для відображення. Ми також повинні мати змогу повідомити ToDoComponent, що користувач натиснув кнопку «Видалити». Angular досягає цього за допомогою " Inputs" та " Outputs".

Inputs дуже прості, вони можуть мати будь-який тип, який ми хочемо, і будуть обов'язковими для передачі. У цьому компоненті ми просто отримуємо

елемент списку як Input. Ми повідомляємо Angular, що конкретне поле є вхідним, ставлячи перед ним декоратор @Input().

```
@Input() ToDo: ToDo = new ToDo("");
```

Потім наш батьківський компонент просто повинен передати компоненту ToDoItem такий елемент списку:

```
<ToDoItem [ToDo]=”ToDo”></ToDoItem>
```

А далі ми просто використовуємо прив’язку для відображення нашого елементу списку.

## 7. ПЕРЕДАЧА ДАНИХ БАТЬКІВСЬКИМ КОМПОНЕНТАМ

В React спочатку дочірньому компоненту передається функція, як властивість, там, де викликається дочірній компонент. Потім виконується планування виклику цієї функції, наприклад, шляхом додавання її в якості обробника onClick, чи її виклик, шляхом звернення до this.props.whateverTheFunctionIsCalled. Це призводить до виклику функції, що знаходиться в батьківському компоненті. Саме цей процес описаний в розділі про видалення елементів зі списку.

При використанні Vue, в дочірньому компоненті досить написати функцію, яка передає дані батьківської функції. У батьківському компоненті пишеться функція, яка прослуховує події передачі значення. При виникненні подібної події така функція викликається. Як і у випадку з React, опис цього процесу можна знайти в розділі про видалення елементів зі списку.

Angular передає дані із дочірнього елементу в батьківський за допомогою логіки "Outputs". Outputs – це не просто деякі дані, це також подія, яка викликає все, що спостерігає за цим Output. Angular Output можуть бути будь-якого типу (наприклад, string), але Output повинні бути типу EventEmitter. EventEmitters видають значення, яке може бути будь-якого типу, який ми хочемо. Отже, якщо

нам потрібно вивести рядок, ми просто створили б поле класу із декоратором, як описано нижче:

```
@Output() Example: EventEmitter<string> = new EventEmitter();
```

А батьківський компонент може отримати подію таким чином:

```
<ExampleComponent (Example)="exampleHandler($event)"></ExampleComponent>
```

Таким чином, в React та Vue передача даних із дочірньої копоненти в батьківську здійснюється за допомогою передачі функцій напряму. В Angular же ми передаємо дані через спеціальну функцію із декоратором, на яку ми підписуємося в батьківському компоненті та очікуємо на дані, що прийдуть нам через цю функцію.

# ВИСНОВОК

Під час дослідження було розглянуто реалізацію веб-додатку типу ToDo за допомогою трьох найпопулярніших фронтенд фреймворків. Розглянуто архітектуру проекту на кожному із фреймворків. Досліджено реалізацію цієї архітектури засобами різних фреймворків, а також порівняно відмінності в їх реалізації та оцінено ефективність кожної із них.

Генерація проектів, а також її окремих початкових компонентів, виконувалася за допомогою засобів CLI кожного із фреймворків. Такий підхід забезпечує правильне створення нових компонентів та їх коректне підключення в корінь проекту. Для новачків цей спосіб допомагає уникнути багатьох проблем ще на етапі його створення, а для професіоналів економить час та сили, позаяк виконує за них рутинні дії.

Під час проведення дослідження було виділено переваги кожного із фреймворків. Наприклад, React та Vue мають простішу структуру проекту, хоча у Vue вона виявилася найпростішою. В обох фреймворках менше різного роду звернень до компонентів самого фреймворку та коду, що потрібен лише для правильної його роботи, але не для, власне, виконання логіки самого завдання. Хоча це одночасно є і мінусом цих фреймворків, адже спеціалістові потрібно багато речей контролювати самому, як то оновлення виведення даних чи заміна даних в інших місцях додатку після їх модифікації. На фоні цього Angular виглядає зручнішим, так як позбавляє цих рутинних дій програміста, виконуючи все сам в автоматичному режимі. Хоча його мінусами можна вважати високу складність архітектури та структури проекту і велику кількість коду, який потрібен лише для правильного функціонування самого фреймворку, а не для виконання логіки поставленого завдання. Але висока складність архітектури для професіоналів є одночасно і плюсом, так як це дозволяє дуже детально налаштувати проект під поточні потреби проекту.

Результатом дослідження являється визначена практична перевага перевірених фреймворків для різного роду проектів. Так Vue більше підходить для новачків у фронтенді та для невеликих проектів із мінімальною бізнес-логікою. React підходить тим програмістам, що можуть швидко освоювати

багато маленьких деталей по ходу розробки. Цей фреймворк найкраще підходить для середньої величини проектів, а також для тих, які потребують переходу по частинам від старої платформи на нову, так як його можна використовувати не пошкоджуючи роботу вже наявного коду. Angular же більше підходить для великих ентерпрайз проектів з високою складністю та потребою в захисті даних. Ним можуть користуватися лише спеціалісти із попередньою підготовкою та під наглядом професіоналів певний час на старті. Додатки на цьому фреймворку відрізняються наявністю великої кількості бізнес-логіки, а також найчастіше модифікованої під певні особливості проекту, а не стандартної, архітектури.

## СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ


1. medium.com [Електронний ресурс] : [Інтернет-портал]. – Електронні дані.– Режим доступу: https://medium.com/javascript-in-plain-english/i-created-the-exact-same-app-in-react-and-vue-here-are-the-differences-e9a1ae8077fd – Порівняння деяких аспектів роботи фреймворків React та Vue
2. medium.com [Електронний ресурс] : [Інтернет-портал]. – Електронні дані.– Режим доступу: https://medium.com/javascript-in-plain-english/i-created-the-exact-same-app-in-react-and-vue-part-2-angular-39b1aa289878 – Опис деяких аспектів роботи фреймворку Angular
3. reactjs.org [Електронний ресурс] : [Інтернет-портал]. – Електронні дані.– Режим доступу: https://uk.reactjs.org/docs/getting-started.html – Документація фреймворку React
4. angular.io [Електронний ресурс] : [Інтернет-портал]. – Електронні дані.– Режим доступу: https://angular.io/docs – Документація фреймворку Angular
5. vuejs.org [Електронний ресурс] : [Інтернет-портал]. – Електронні дані.– Режим доступу: https://vuejs.org/v2/guide/ – Документація фреймворку Vue
6. habr.com [Електронний ресурс] : [Інтернет-портал]. – Електронні дані.– Режим доступу: https://habr.com/ru/company/ruvds/blog/343022/ – Огляд основ роботи фреймворку React
7. wiki.cuspu.edu.ua [Електронний ресурс] : [Інтернет-портал]. – Електронні дані.– Режим доступу: https://wiki.cuspu.edu.ua/index.php/Angular – Коротка історія фреймфорку Angular
8. fructcode.com [Електронний ресурс] : [Інтернет-портал]. – Електронні дані.– Режим доступу: https://fructcode.com/ru/blog/features-of-popular-frameworks-html-css-php-and-python-frameworks/ – Огляд загальної архітектури фреймворків для веб-сайтів та веб-додатків