Міністерство освіти і науки України
Національний технічний університет
«Дніпровська політехніка»

Інститут електроенергетики
(інститут)

Факультет інформаційних технологій
(факультет)

Кафедра Програмного забезпечення комп'ютерних систем
(повна назва)

ПОЯСНЮВАЛЬНА ЗАПИСКА
кваліфікаційної роботи ступеня

магістра
(назва освітньо-кваліфікаційного рівня)

студента *Степанова Олександра Борисовича*
(ПІБ)

академічної групи *121М-19-1*
(шифр)

спеціальності *121 Інженерія програмного забезпечення*
(код і назва спеціальності)

на тему: *Методи, алгоритми та програмне забезпечення для дослідження ефективності фреймворків згідно з потребами проекту*

_____ *О.Б. Степанов*

Керівники	**Прізвище, ініціали**	**Оцінка за шкалою**		**Підпис**
		рейтинговою	**інституційною**	
розділів кваліфікаційної роботи				
спеціальний	Доц. Сироткіна О.І.			
економічний	Доц. Касьяненко Л.В.			

Рецензент				

Нормоконтролер	Доц. Сироткіна О.І.			

Дніпро
2020

Міністерство освіти і науки України
Національний технічний університет
«Дніпровська політехніка»

ЗАТВЕРДЖЕНО:
Завідувач кафедри
Програмного забезпечення комп'ютерних систем
(повна назва)

_____ І.М. Удовик
(підпис) (прізвище, ініціали)

« » _____ 20 20 Року

ЗАВДАННЯ
на виконання кваліфікаційної роботи магістра

спеціальності *121 Інженерія програмного забезпечення*
(код і назва спеціальності)

студенту *121М-19-1* *Степанову Олександру Борисовичу*
(група) (прізвище та ініціали)

Тема кваліфікаційної роботи *Методи, алгоритми та програмне забезпечення для дослідження ефективності фреймворків згідно з потребами проекту*

1 ПІДСТАВИ ДЛЯ ПРОВЕДЕННЯ РОБОТИ

Наказ ректора НТУ «Дніпровська політехніка» від 22.10.2020 р. № 888-с

2 МЕТА ТА ВИХІДНІ ДАНІ ДЛЯ ПРОВЕДЕННЯ РОБІТ

Об'єкт досліджень – процес вибору фреймворків в залежності від критеріїв проекту.

Предмет досліджень – методи створення та оцінки показників ефективності веб додатків з використанням різних технологій.

Мета роботи – пришвидшення процесу розробки веб додатків за рахунок створення оптимальної стратегії вибору веб технологій.

3 ОЧІКУВАНІ НАУКОВІ РЕЗУЛЬТАТИ

Наукова новизна полягає в тому, що отримали подальший розвиток методи ефективності використання фреймворків виходячи з потреб проекту.

Практична цінність роботи полягає в тому, що результати дослідження можуть бути застосовані як індивідуальними розробниками, так і великими компаніями для вибору найбільш відповідної технологій для реалізації проекту на етапі його планування.

4 ЕТАПИ ВИКОНАННЯ РОБІТ

Найменування етапів робіт	**Строки виконання робіт (початок – кінець**
Аналіз теми та постановка задачі	11.09.2020-30.09.2020
Дослідження характеристик, принципів та підходів до використання front-end фреймворків. Виділення потреб проектів для подальшого аналізу.	01.10.2020-30.10.2020
Зіставлення статистичних даних та показників ефективності технологій з потребами проекту.	02.11.2020-11.12.2020

Завдання видав _____ *Доц. Сироткіна О.І.*
(підпис) (прізвище, ініціали)

Завдання прийняв до виконання _____ *Степанов О.Б.*
(підпис) (прізвище, ініціали)

Дата видачі завдання: *07.09.2020 р.*

Термін подання кваліфікаційної роботи до ЕК *18.12.2020*

РЕФЕРАТ

Пояснювальна записка: 78 с., 18 рис., 3 дод., 49 джерел.

Об'єкт дослідження: процес вибору фреймворків в залежності від критеріїв проекту.

Предмет дослідження: методи створення та оцінки показників ефективності веб додатків з використанням різних технологій.

Мета магістерської роботи: пришвидшення процесу розробки веб додатків за рахунок створення оптимальної стратегії вибору веб технологій.

Методи дослідження. Для виконання поставлених завдань було використане прикладне програмне забезпечення та бібліотеки з дослідження показників ефективності, а також статистичні дані опитувань серед розробників.

Наукова новизна полягає в тому, що отримали подальший розвиток методи ефективності використання фреймворків виходячи з потреб проекту.

Практична цінність роботи полягає в тому, що результати дослідження можуть бути застосовані як індивідуальними розробниками, так і великими компаніями для вибору найбільш відповідної технологій для реалізації проекту на етапі його планування.

У розділі «Економіка» проведено розрахунки трудомісткості розробки програмного забезпечення, витрат на створення ПО і тривалості його розробки.

Список ключових слів: front-end, back-end, framework, компонент, інтерфейс, бібліотека, стан, життєвий цикл, Virtual DOM, React, Angular, Vue, JavaScript, MVC, WEB.

ABSTRACT

Explanatory note: 78 p., 18 fig., 3 applications, 49 sources.

Object of research: the process of selecting frameworks based on project's criteria.

Subject of research: methods of creation and evaluation performance indicators of web applications by using different technologies.

Purpose of Master's thesis: speed up the web application development process by creating the optimal strategy of web technology selection.

Research methods. To achieve the goals, appropriate software, libraries for analyzing performance and also survey statistics across the developers were used.

Originality of research consists of the future development of the frameworks effective use methods based on the project's needs.

Practical value of the results consists of experimental data that could be used not only by individual developers, but also big companies for selecting the most suitable technology for project implementation at the planning stage.

In the Economics section we calculated the complexity of software development and the costs of software development, the duration of the actual development are calculated.

Keywords: frontend, backend, framework, component, interface, library, state, lifecycle, Virtual DOM, React, Angular, Vue, JavaScript, MVC, WEB.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

UI – User Interface

ПЗ – Програмне забезпечення

ЖЦ – Життєвий цикл

SPA – Single Page Application

PWA – Progressive Web Application

CRUD – Create Read Update Delete

MVC – Model View Controller

ВСТУП

Актуальність теми дослідження. Сучасні бізнес-процеси, зосереджені на компонентно-орієнтованому підході, який прийшов з мобільної розробки і успішно увійшов у веб за допомогою таких компаній, як Facebook і Google.

Вони створили реактивні фреймворки, які повністю змінили підхід до звичної веб-розробки, зробивши її якісною і такою, що масштабується, що дозволяє створювати додатки будь-якої складності за допомогою веб технологій.

Так як Facebook і Google розробляють повністю динамічні додатки, де більше 90% компонентів – реактивні, тобто вони мають свій стан і працюють у фоновому режимі, чекаючи оновлення даних, щоб змінити інтерфейс, тому використання реактивних фреймворків повністю виправдовує себе.

Але більшість веб-сайтів не такі. Кількість динамічних частин в додатку становить 20-30% і використання реактивних фреймворків уповільнює роботу сайту, особливо на мобільних пристроях.

На сьогоднішній день проблема продуктивності реактивних фреймворків стоїть досить гостро, адже продуктивність впливає не тільки на відсоток відмов користувачів, але і на просування в пошукових системах, а рішення даної проблеми має бути комплексним, зберігаючи всі зручності компонентного підходу.

На даний час вже існують реактивні фреймворки, де продуктивність відображення контенту досить близька до швидкості роботи JavaScript. При цьому вони все одно мають проблему повільного першого завантаження, особливо на мобільних пристроях.

Мета дослідження полягає у пришвидшенні процесу розробки веб додатків за рахунок створення оптимальної стратегії вибору веб технологій.

Завдання дослідження. Для досягнення поставленої мети були виконані наступні завдання:

- відстежити еволюцію технологій розробки користувацького інтерфейсу;
- описати сутність поняття фреймворку;
- визначити принципи застосування фреймворків;
- розкрити можливості та області використання React;
- розкрити можливості та області використання Angular;
- розкрити можливості та області використання Vue;
- проаналізувати потреби проектів;
- дослідити ефективність фреймворків;
- навести визначення фреймворку відповідно до потреб проекту;
- виконати верифікацію результатів дослідження.

Об’єкт дослідження: процес вибору фреймворків в залежності від критеріїв проекту.

Предмет дослідження: методи створення та оцінки показників ефективності веб додатків з використанням різних технологій.

Методи дослідження. Для виконання поставлених завдань було використане прикладне програмне забезпечення та бібліотеки з дослідження показників ефективності, а також статистичні дані опитувань серед розробників.

Наукова новизна полягає в тому, що отримали подальший розвиток методи ефективності використання фреймворків виходячи з потреб проекту.

Практичне значення роботи полягає в тому, що результати дослідження можуть бути застосовані як індивідуальними розробниками, так і великими компаніями для вибору найбільш відповідної технологій для реалізації проекту на етапі його планування.

Структура і обсяг роботи. Робота складається з вступу, чотирьох розділів і висновків. Містить 78 сторінок, в тому числі 67 сторінок тексту основної частини з 18 рисунками, списку використаних джерел з 49 найменуваннями на 5 сторінках, 3 додатка на 6 сторінках.

РОЗДІЛ 1
ТЕОРЕТИЧНІ ОСНОВИ FRONT-END ФРЕЙМВОРКІВ

1.1. Еволюція технологій розробки користувацького інтерфейсу

Як відомо, процес проникнення інформаційних технологій практично в усі сфери людської діяльності продовжує розвиватися. Комп'ютери та інформаційні системи стають все більш дружніми і зрозумілими навіть для людини, яка не розуміє в галузі інформатики. Це стало можливим тому, що користувачі і їх програми взаємодіють з обчислювальною технікою за допомогою спеціального програмного забезпечення – операційних систем. Операційна система надає інтерфейси і для додатків, що виконуються, і для користувачів.

Інтерфейс в широкому сенсі слова – це спосіб взаємодії різних об'єктів між собою.

Інтерфейс (UI - англ. User interface) – різновид інтерфейсів, в якому одна сторона представлена людиною (користувачем), інша – машиною або пристроєм. Являє собою сукупність засобів і методів, за допомогою яких користувач взаємодіє з різними пристроями і апаратурою.

На сьогоднішній день можна виділити такі різновиди інтрфейсу:

- текстовий інтерфейс (інтерфейс командного рядка);
- графічний інтерфейс (GUI) або WIMP;
- SILK-інтерфейс.

А тепер дамо короткий опис цих видів інтерфейсу:

Текстовий призначений для користувача інтерфейс – різновид інтерфейсу користувача, яка використовує набір букв і цифр. Також, його різновид – інтерфейс командного рядка – має окремі переваги перед графічним інтерфейсом і використовується до цих пір в деяких областях.

WIMP – інтерфейс (Window, Image, Menu, Pointer). характерною особливістю цього виду інтерфейсу є те, що діалог з користувачем ведеться не

за допомогою команд, а за допомогою графічних образів – меню, вікон, інших елементів.

Хоча і в цьому інтерфейсі подаються команди машині, але це робиться "побічно", через графічні образи.

SILK – інтерфейс (Speech, Image, Language, Knowledge). В рамках цього інтерфейсу йде звичайна "розмова" людини і комп'ютера. При цьому комп'ютер знаходить для себе команди, аналізуючи людську мову. Результат виконання команд він також перетворює в зрозумілу людині форму. Цей вид інтерфейсу найбільш вимогливий до апаратних ресурсів комп’ютера і поки має багато недоліків.

Сучасні напрямки розвитку:

Біометрична технологія:

Ця технологія виникла в кінці 90-х років XX століття і на цей момент ще розробляється. Для управління комп'ютером використовується вираз обличчя людини, напрямок його погляду, розмір зіниці і інші ознаки. Для ідентифікації користувача використовується малюнок райдужної оболонки його очей, відбитки пальців і інша унікальна інформація. Зображення зчитуються з цифрової відеокамери, а потім за допомогою спеціальних програм розпізнавання образів з цього зображення виділяються команди. Ця технологія, очевидно, займе своє місце в програмних продуктах і додатках, де важливо точно ідентифікувати користувача комп'ютера. Уже застосовується в деяких смартфонах для того, щоб ідентифікувати користувача або, наприклад визначити, що він заснув. Однак масове поширення технологія ще не отримала.

Нейрокомп'ютерний інтерфейс

Всі маніпулятори мають один спільний недолік – невелику, порівняно зі швидкістю думки, швидкість передачі інформації. Якщо припустити, що давати команди комп'ютера можна за допомогою думки, то необхідність у використанні яких би то ні було маніпуляторів відпадає зовсім. І якою б фантастичною ні здавалася ця ідея, сьогодні вже є реальні передумови того, що зовсім скоро людина зможе подумки віддавати накази комп'ютера.

Нейрокомп'ютерних інтерфейс (званий також прямий нейронний інтерфейс або мозковий інтерфейс, в англомовній літературі brain-computer interface, BCI) - фізичний інтерфейс прийому або передачі сигналів між живими нейронами біологічного організму з одного боку, і електронним пристроєм з іншого боку. В односпрямованих інтерфейсах, пристрої можуть або приймати сигнали від мозку, або послати йому сигнали (наприклад, імітуючи сітківку ока при відновленні зору електронним імплантантом). Двонаправлені інтерфейси дозволяють мозку і зовнішнім пристроям обмінюватися інформацією в обох напрямках.

Всі існуючі технології цього різновиду інтерфейсу можна розбити на два напрямки – безпосередню взаємодію з нейронами з імплантацією в тіло спеціальних пристроїв і зняття зовнішніх сигналів (в основному, імпульсів мозкової активності) за допомогою зовнішніх датчиків.

1.2. Фреймворк: сутність поняття

Фреймворк (англ, framework – структура, каркас) – сукупність рішень по архітектурі, структурі і способам об'єднання компонентів системи, які можуть бути застосовані для деякої безлічі однотипних завдань.

В області програмування під фреймворком розуміють безліч класів і способів їх взаємодії [1].

У фреймворків є дві основні функції: робота на серверній стороні (back-end) і робота на клієнтській стороні (front-end).

Front-end фреймворки пов'язані із зовнішньою частиною програми. Простими словами, вони відповідають за зовнішній вигляд програми. Back-end відповідає за внутрішній устрій додатку. Розглянемо обидва типи детальніше.

Серверні фреймворки. Правила та архітектура таких фреймворків не дає можливості створити веб-додаток з багатим інтерфейсом. Вони обмежені в своїй функціональності, проте все одно є можливість створювати прості сторінки і різні форми. Також вони можуть формувати вихідні дані і

відповідати за безпеку в разі атак. Все це виразно може спростити процес розробки. Серверні фреймворки в основному відповідають за окремі, але критично важливі частини програми, без яких вона не може нормально працювати. Ось кілька найпопулярніших фреймворків і мови, з якими вони працюють:

Django – Python;

Zend – PHP;

Express.js – JavaScript;

Ruby on Rails – Ruby.

Клієнтські фреймворки. На відміну від серверних, клієнтські фреймворки ніяк не пов'язані з логікою програми. Цей тип фреймворків працює в браузері. З їх допомогою можна поліпшити і впровадити нові призначені для користувача інтерфейси. Front-end фреймворки дозволяють створювати різні анімації і односторінкові додатки. Всі клієнтські фреймворки відрізняються по функціональності і використанню. Ось деякі з них:

Backbone + Marionette;

Angular;

Ember.js;

Vue.js.

Всі ці фреймворки використовують JavaScript.

Багатофункціональні фреймворки. Meteor відомий як full-stack веб-фреймворк. Це означає, що він задовольняє майже всі потреби як з боку клієнта, так і з боку сервера, що робить Meteor надзвичайно популярним. Не потрібно витрачати час на те, щоб налагодити взаємодію між двома фреймворками через REST API – є можливість просто вибрати Meteor і прискорити процес розробки. Але це не головна особливість цього фреймворка. Обидві сторони – серверна і клієнтська – працюють на одній мові, тому є можливість створювати і використовувати для них один і той же код. Наступна особливість – «режим реального часу» – коли щось змінюється в одному інтерфейсі, зміни відбуваються і в інших. Як приклад можна взяти документ

або таблицю із загальним доступом. Коли програміст додає коментарі або якось змінює вміст, інші користувачі теж це бачать.

Розглянемо найбільш актуальні фреймворки.

Express

Зараз JavaScript-фреймворк Express один з найбільш поширених інструментів веб-розробки. Його використовують великі компанії Accenture, IBM і Uber, а також інші фреймворки, наприклад, Kraken, Sails і Loopback.

Express позиціонується як мінімалістичний, швидкий і дуже гнучкий фреймворк. Він надає всі необхідні можливості, при цьому активно використовуючи всі переваги і потужність Node.js. Підтримує REST API.

Можливо, найбільший недолік Express, особливо для початківців - занадто велика гнучкість. Одну і ту ж річ можна зробити по-різному.

Django

Ще один відомий серед IT-лідерів (Google, YouTube, Instagram) фреймворк для веб-розробки, на цей раз на Python. Django має Model-View-Template структуру і слідує кращим принципам проектування: DRY (Don’t Repeat Yourself, укр. - не повторюй) і Угода по конфігурації.

У Django присутня аутентифікація, обмін повідомленнями, маршрутизація, робота з базою даних, адмінських частина сайту.

Особливий пріоритет віддається безпеці. Фреймворк реалізує багато важливих принципів захисту самостійно, наприклад, запобігає виконанню коду на рівні шаблонів. Крім того, є ряд методів та інструментів, які можуть застосовуватися на розсуд розробника.

Rails

Популярний Ruby-фреймворк з класичною структурою Model-View-Controller. Rails успішно працює в Airbnb, GitHub, Hulu і Shopify.

Інструмент лояльний до новачків і має невисокий початковий поріг входження. Недолік – складний процес розгортання і запуску на продакшені.

Щоб зробити роботу з фреймворком швидше і ефективніше, створено безліч корисних “гемів” (англ. gems - пакети і бібліотеки), які можна підключити до додатку.

Laravel

MVC-фреймворк для самої поширеної мови інтернету – PHP. Laravel відносно молода платформа, але вже значно використовувана.

Багато можливостей, наприклад, підтримка API, доступні з самого фреймфорку. Крім того є багато корисних пакетів з додатковою функціональністю.

Основна проблема Laravel – недостатня продуктивність в порівнянні з Django або Express. Для важких проектів це може стати істотним мінусом.

Spring

Для повноти картини списку backend-фреймворків для веб-розробки не вистачає тільки Java. Spring – "стратегічно важливий фреймворк", професійний, досить гнучкий і дуже надійний. По суті, це колекція фреймворків у фреймворку, більшість з яких може працювати незалежно один від одного.

Angular

Спеціалізація Angular – повноцінний односторінковий (SPA).

Фреймворк досить "впертий", він строго нав'язує програмістові своє бачення програми. Для розробки використовується TypeScript, що швидше за гідність, ніж недолік. Мова JavaScript дуже гнучка, але ця гнучкість може бути причиною безлічі помилок.

Основні мінуси Angular – його розмір у порівнянні з іншими JS-фреймворками і вроджена SEO. Втім, останній недолік цілком можна виправити оптимізацією.

React

Не дуже правильно називати React фреймворком, це скоріше бібліотека компонентів для веб-розробки. Однак його значення таке велике, що історично жодне порівняння без нього не обходиться.

Саме React від Facebook ввів "моду" на компонентну архітектуру і віртуальний DOM.

Розробка ведеться на особливому діалекті JavaScript - JSX. Це суміш звичного JS з таким же звичним HTML. І в цілому це інтерфейс-орієнтований інструмент, що істотно спрощує роботу з веб-сторінкою в браузері.

React можна використовувати не тільки на клієнті, але і на стороні сервера.

Vue

Розпочавшись як проект одного розробника Google, Vue.js дуже швидко виріс в один з найпопулярніших JavaScript-фреймворків.

Це дуже гнучкий інструмент з прогресивною структурою, який легко інтегрувати в уже існуючі проекти. Компонентна архітектура і багата екосистема дозволяє розробляти складні додатки з мінімальними витратами.

Ember

У 2015 році Ember був названий кращим JavaScript-фреймворком. Чотири роки по тому він все ще популярний. Спільнота продовжує розширюватися, з'являються нові функції і релізи. Інструмент використовується в Google, Microsoft, Heroku і Netflix.

Серед стандартних методів в Ember доступна двостороння прив'язка даних, а також безліч корисних функцій і компонентів.

Основна мета фреймворку – максимізувати продуктивність розробника. Для цього він застосовує кращі практики програмування.

Backbone

Дуже легкий і стильний фреймворк з Model-View структурою, призначений для створення SPA.

Backbone має всього одну залежність – це бібліотека Underscore.

Фреймворк володіє багатою екосистемою, яка в поєднанні з Mustache і Marionette дозволяє створювати повноцінні клієнтські програми.

Фреймворки для веб-розробки багато в чому схожі, навіть якщо реалізовані на різних мовах програмування. Це не дивно, адже вони вирішують одні й ті ж завдання.

Проте, кожен з перерахованих фреймворків індивідуальний. У них різні підходи, методи і поведінка в розробці.

1.3. Принципи застосування фреймворків

За останнє десятиліття з'явилася величезна кількість фреймворків. Такі фреймворки як Spring і Ruby on Rails стали дуже популярними і їх вивчення сильно підвищує ймовірність працевлаштування. Але на кожен успішний фреймворк доводиться велика кількість фреймворків, які так і не набрали популярність.

Дуглас Шмідт описує фреймворк як набір програмних сутностей (таких як класи, об'єкти і компоненти), які допомагають будувати архітектуру для подібних додатків [2]. Згідно з цим визначенням, фреймворк - це каркас додатка, що містить відповідні компоненти. Очікується, що для створення програми розробник розширить і налаштує фреймворк шляхом додавання своєї логіки.

Головним показником корисності фреймворка є те, що він допомагає розробнику підвищити продуктивність і якість коду. Наприклад, сучасні фреймворки (такі як Play, django) часто надають інструменти для кодогенерації або готові каркаси типових додатків. Так само добре продуманий фреймворк задіє всі заходи безпеки, допомагаючи розробнику писати більш захищені додатки.

Основоположною характеристикою фреймворка є так-звана інверсія управління (рис. 1.1). Зазвичай фреймворк грає головну роль і викликає код самого додатка. Потік управління інвертується тут – фреймворк викликає додаток, а не навпаки. На схемі нижче видно взаємодію між фреймворком, додатком і бібліотекою.

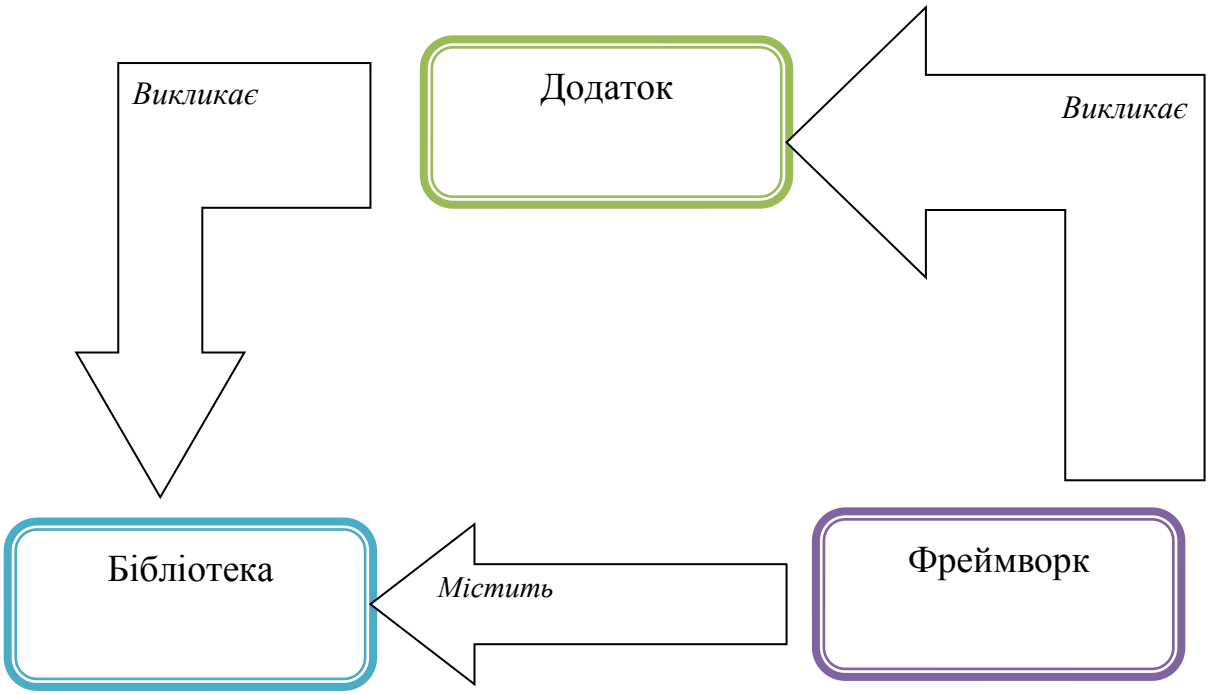

Рис. 1.1. Схема взаємодії між фреймворком, додатком і бібліотекою

Додатковою перевагою використання фреймворка у великих командах розробників є те, що він допомагає слідувати єдиним стандартам. У кожного фреймворка є загальноприйняті рішення типових задач, стиль написання коду і готові реалізації часто використовуваних інструментів. Це дозволяє зберегти єдиний стиль програмування у всіх частинах програми.

Звичайно, не всі розробники люблять використовувати фреймворки. Вони скаржаться на високий поріг входження, необхідність діяти в рамках обраного фреймворку, низьку продуктивність і т.д. Але сьогодні більшість із цих проблем не актуальні. Фреймворки цілком вдало справляються з ними, дотримуючись описаних принципів.

Успіх фреймворка залежить від безлічі факторів, включаючи людей, що стоять за ним, якості коду, документації, спільноти, підтримки, маркетингу та багато чого іншого. Одним з найбільш значущих чинників є філософія, якої дотримувалися творці фреймворка під час розробки.

Досить давно Тім Петерс сформулював двадцять принципів хорошого дизайну. Вони відомі як "The Zen of Python". Схожі ідеї властиві і багатьом іншим мовам. І багато із цих принципів застосовні в розробці фреймворків.

Серед них можна виділити найбільш важливі:

- простота;
- монолітність;
- послідовність;
- очевидність;
- угоди вище конфігурації.

Простота

Вивчення нового фреймворка – це нелегке завдання. Вивчаючи новий фреймворк, розробник інвестує в нього свій час і сили і він очікує, що ці інвестиції окупляться в його поточному та наступних проектах. У той час, як вивчаючи бібліотеку, розробник може знайомитися з її API в міру необхідності, фреймворк вимагає повного розуміння його принципів до того, як розробник приступить до його використання. Тому дуже важливо спроектувати фреймворк так, щоб він був зрозумілим, доступним і приємним в роботі.

Щоб досягти простоти необхідно звести до мінімуму кількість обмежень, які нав'язує фреймворк. Так само то правила, які є, повинні бути логічні і добре задокументовані. Чим більше правил диктує фреймворк, тим крутіше крива навчання і тим менше ймовірність того, що розробник вибере його. Коли правила логічні, розробник може швидше їх засвоїти. Так само без документації фреймворк буде марний, тому що ніхто не захоче займатися реверс інжинірингом щоб зрозуміти логіку його роботи.

Монолітність

Тут термін "монолітний" відноситься до фреймворків, побудованих у вигляді єдиної сильно пов'язаної кодової бази. Коли веб-фреймворки тільки почали набирати популярність, вони як правило були монолітними тому їх основною метою було забезпечити підтримку повного стека завдань для розробки веб-додатків. Згодом стало ясно, що монолітні системи мають багато

проблем. Наприклад, внесення змін в невелику частину фреймворка вимагало повторного тестування і випуску нової версії всього фреймворка. А це в свою чергу спричинило оновлення і повторне збирання всіх додатків, що використовують цей фреймворк. Те, що код в монолітному фреймворку сильно пов'язаний робить важким дотримання зворотної сумісності між різними версіями. Нарешті, з появою великої кількості невеликих вузькоспеціалізованих фреймворків (для логування, кешування, роботи в базою даних і т.д.) розробники стали менш охоче користуватися величезними монолітними фреймворками.

Сучасні фреймворки повинні мати слабо-пов'язану архітектуру. Full-stack фреймворки (наприклад Spring) перетворилися в набори слабо-пов'язаних компонентів, які можуть бути використані окремо або замінені сторонніми рішеннями. Спеціалізовані фреймворки взаємодіючі "за контрактом" роблять кінцевий додаток менш залежним від конкретного фремворка. Хорошим прикладом є популярні сьогодні фреймворку начебто Sinatra, Express.js і Martini. Ці фреймворки надають базу для навігації і обробки запитів до веб-додатку. Самі по собі ці фреймворки дуже маленькі, але відкрита архітектура дозволяє легко розширювати їх до нескінченності, використовуючи сторонні рішення.

Послідовність

Будучи послідовним фреймворк всюди використовує однакові підходи в дизайні архітектури, іменування, організації коду і т.д. Послідовний фреймворк знижує поріг входження тим, що розробник вивчивши аспекти однієї частини фреймворка, може на її прикладі швидко освоїтися в архітектурі, що залишилася. Послідовність допомагає розробнику знизити кількість помилок через неправильне використання компонентів фреймворка.

Наявність

Цей принцип передбачає написання такого коду, що є документацією сам для себе і уникає незрозумілостей. На те є дві причини. По-перше, явний код легше для розуміння і супроводу. Якщо код не вимагає пояснень, то

розробнику, який буде його супроводжувати (а це не обов'язково буде його автор), не доведеться витрачати багато часу, щоб зрозуміти, як цей код працює. По-друге, явний код менш схильний до помилок. Хоча явність вимагає написання більшого числа рядків коду, вона знижує ймовірність того, що розробнику доведеться переглядати вихідний код, щоб зрозуміти, як цей він працює.

Угоди вище конфігурації

Суть цього принципу полягає в тому, що фреймворк у всіх випадках використовує розумні значення за замовчуванням, при цьому дозволяючи змінити їх через конфігурацію. Мета цього – знизити кількість рішень, які необхідно приймати розробникові, тим самим роблячи роботу з фреймворком простіше.

Вперше принцип "угоди вище конфігурації" був застосований у фреймворку Ruby on Rails. Він надає бібліотеку ActiveRecord, яка зіставляє програмні класи і таблиці бази даних. За угодою назва таблиці - це назва класу в множині. Так, клас Account буде зіставлений з таблицею Accounts. Але якщо назва таблиці відрізняється, розробник може вказати це явно в конфігурації.

Багато MVC фреймворків дотримуються цього принципу для направлення запитів. Ця угода дозволяє розробнику не вказувати правила перенаправлення для методів. Проте, якщо розробнику потрібно використовувати особливі правила перенаправлення, він може це зробити, вказавши їх явно.

Принцип "угоди вище конфігурації" дозволяє розробнику писати менше коду. Але так само він призводить до збільшення правил, яким має слідувати розробник. Крім того, він конфліктує з принципом "явне краще неявного", який обговорювався вище. Деякі фреймворки, такі як Spring на початкових версіях використовували правила перенаправлення за замовчуванням, але в більш пізніх версіях було прийнято вимагати від розробника вказувати перенаправлення явно, використовуючи анотації.

Розробка успішного фреймворка – це знаходження балансу між простотою і функціональністю. Протягом всієї розробки необхідно приймати рішення, керуючись перерахованими вище принципами.

1.4. Висновки до першого розділу

У межах першого розділу здійснено розкриття теоретичних основ front-end фреймворків, розкрито поняття та сутність фреймворку, здійснено опис принципів застосування фреймворків.

Фреймворки для веб-розробки багато в чому схожі, навіть якщо реалізовані на різних мовах програмування. Це не дивно, адже вони вирішують одні й ті ж завдання.

Проте, кожен з перерахованих фреймворків індивідуальний. У них різні підходи, методи і поведінка в розробці.

РОЗДІЛ 2
СТАН РИНКУ FRONT-END ФРЕЙМВОРКІВ

2.1. Можливості та області використання React

React.js – один з кращих і найбільш поширених на сьогоднішній день способів створити односторінковий додаток. Це гнучкий і зручний фреймворк, що використовує компонентний підхід, тобто увесь додаток має бути розділений на компоненти, які зазвичай є представленням конкретної візуальної сутності. Перед початком розробки, додаток зазвичай розділяють на візуальні компоненті, які потім і будуть реалізовані за допомогою React (рис. 2.1).

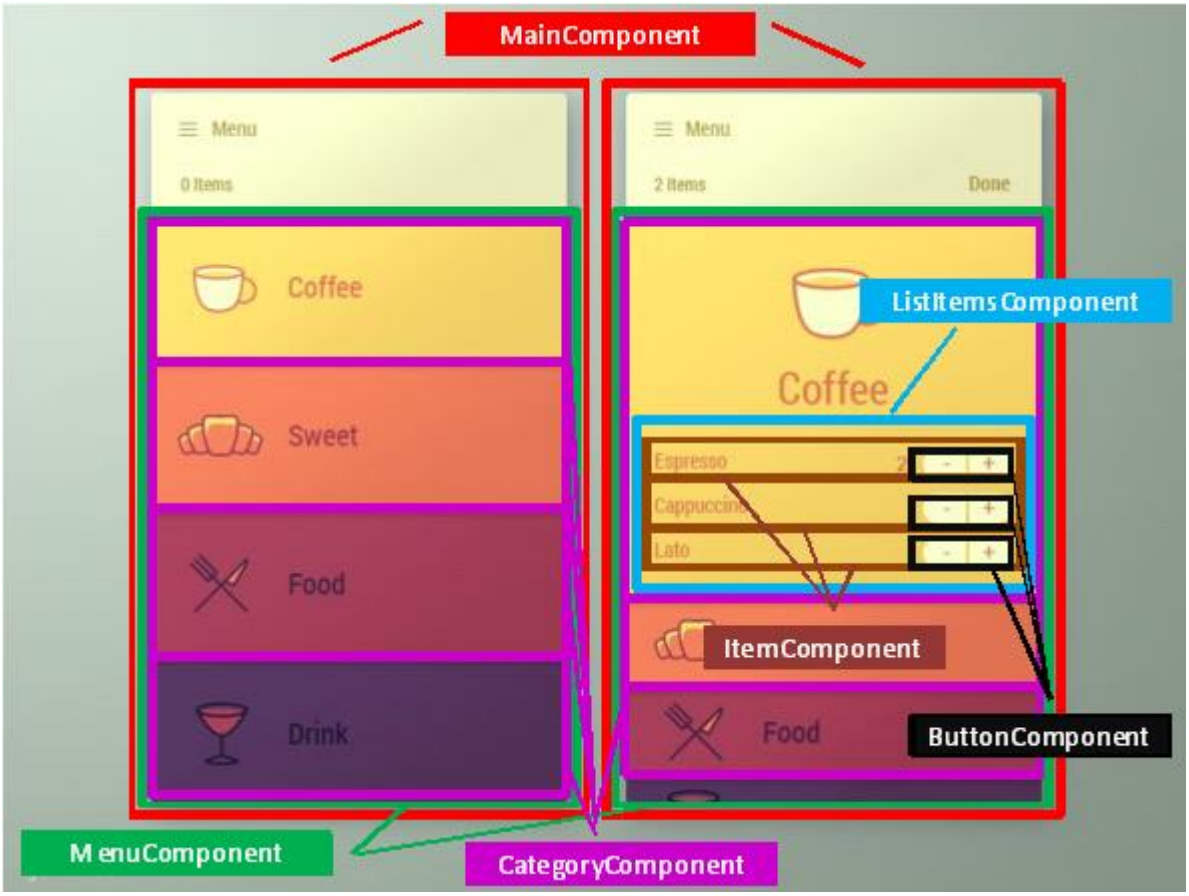

Рис. 2.1. Принцип розділення додатку на компоненти

React-розробка полягає в описі того, що потрібно вивести на сторінку (а не в складанні інструкцій для браузера, присвячених тому, як це робити). Це, крім іншого, означає значне скорочення обсягів шаблонного коду.

Компоненти в React можуть розширювати стандартний клас Component, що надає їм доступ до методів життєвого циклу компонента (рис. 2.2).

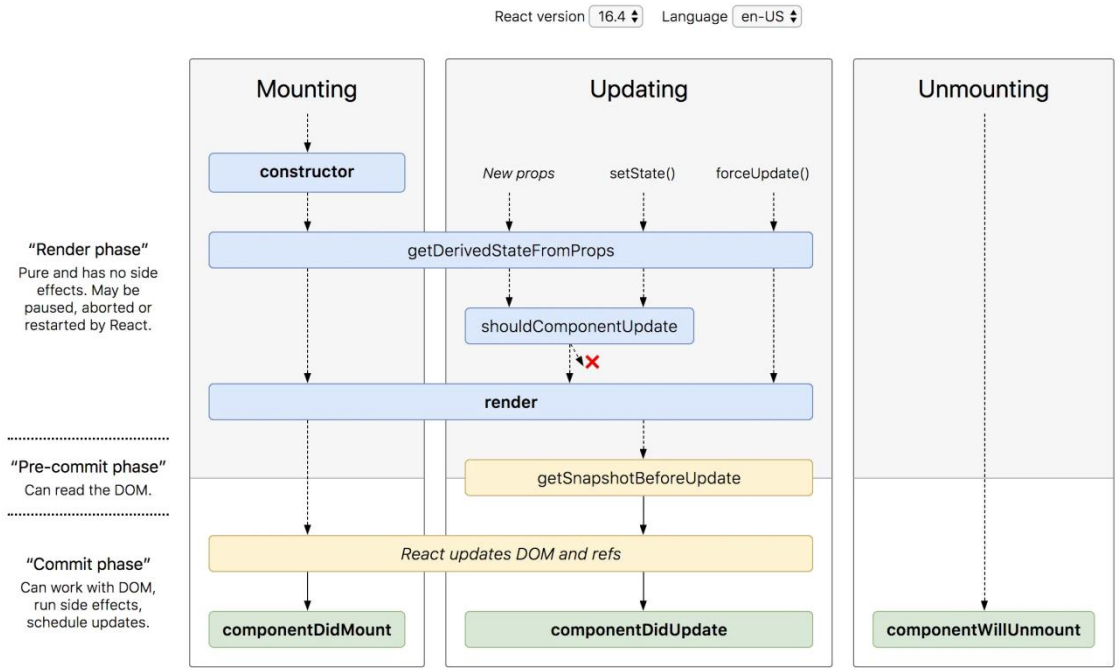

Рис. 2.2. Життєвий цикл React компоненту

Але для створення компоненту не обов’язково використовувати класи, це також можуть бути звичайні функції, що повертають необхідний вміст. Функції використовують коли компонент є досить простим, відповідає лише за відображення і не має змінюватись в часі.

Зараз React.js використовується в багатьох інших фреймворках та інструменти на зразок Next.js, GatsbyJs, Razzle, After.js і т. д.

Найчастіше у розробника на руках безліч компонентів, які доводиться обертати в div, так як render() дозволяє повернути лише один компонент. Таким чином ми додаємо в документ зайвий HTML-елемент.

Крива навчання – це важливий фактор, який потрібно враховувати при виборі будь-якого фреймворка. У цьому зв'язку треба зазначити, що в React є менше абстракцій, ніж у Angular. Якщо програміст знає JavaScript, то, ймовірно, зможе навчитися писати React-додатки буквально за день. Звичайно, для того, щоб навчитися робити це правильно, буде потрібен якийсь час, але приступити до роботи можна дуже і дуже швидко.

Якщо ж проаналізувати Angular, то виявиться, що якщо вирішено освоїти цей фреймворк, доведеться також вивчити нову мову (Angular використовує TypeScript), а також навчитися використовувати засоби командного рядка Angular і звикнути до роботи з директивами.

Деколи ми порушуємо семантику HTML, коли додаємо елементи в JSX, бо компонент має повертати лише один елемент, особливо при роботі зі списками (<ol>, <ul> і <dl>) і таблицями <table>. В таких випадках краще використовувати фрагменти React для групування декількох елементів, адже вони не створюють зайвого елементу:

```
import React, { Fragment } from 'react';

const ListItem = ({ item }) => {
  return (
    <Fragment>
      <dt>{item.name}</dt>
      <dd>{item.description}</dd>
    </Fragment>
  )
}

const Glossary = ({ items }) => {
  return (
    <dl>
      {items.map((item) => (
        <ListItem item={item} key={item.id} />
      ))}
    </dl>
  );
}
```

Завдяки createContext можна передавати дані через дерево компонентів, замість того, щоб вручну передавати властивості на кожному рівні. Тому якщо

є кілька компонентів, яким потрібні дані, краще використовувати контекст. Якщо є тільки один дочірній компонент, можна застосувати композицію. Приклад застосування контексту для передачи стилів у компоненти:

theme-context.js

```
export const themes = {
  white: {
    color: '#000000',
    background: '#ffffff',
  },
  black: {
    color: '#ffffff',
    background: '#222222',
  },
};

export const ThemeContext = React.createContext(
  themes.black // default value
);
```

themed-button.js

```
import { ThemeContext } from './theme-context';

const ThemedButton = ({ context, value, onClick }) => {
  return (
    <div>
      <button
        onClick={onClick}
        style={{ backgroundColor: context.background }}
      >
        {value}
      </button>
    </div>
  )
}

ThemedButton.contextType = ThemeContext;

export default ThemedButton;
```

У React версії 16 був представлений метод для відстеження кордонів помилок. За назвою зрозуміло, що це компоненти, які реєструють помилки у всіх дочірніх компонентах. Ідея дуже проста: створити компонент і використовувати його як основний кожен раз, коли знадобиться обробляти помилки. Якщо в якомусь з дочірніх компонентів виникне помилка, межі помилок будуть викликані для обробки. Межі помилок відловлюють помилки

відображення. Імперативні помилки на зразок тих, що виникають при обробці подій, повинні відловлюватися try/catch-блоком JavaScript.

Для логування інформації про помилку потрібно використовувати:

```
class YouCanRunButCantHide extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      errorFound: false
    };
  }

  componentDidCatch(error, info) {
    this.setState({
      errorFound: true
    });
    console.log('error: ', error);
    console.log('info: ', info);
  }

  render() {
    if (this.state.errorFound) {
      return <p>Error caught!</p>;
    }
    return this.props.children;
  }
}
```

Тепер його можна використовувати як звичайний компонент:

```
const ErrorSmugglerWrapper = () => (
  <YouCanRunButCantHide>
    <ErrorSmuggler />
  </YouCanRunButCantHide>
);


export default ErrorSmugglerWrapper;
```

Але якщо в самому компоненті ErrorBoundary виникне помилка - вона не буде відстежена.

Такий метод був наявний в React версії 15 під назвою unstable_handleError(). Цей метод більше не працює, і з бета-релізу 16 версії потрібно використовувати лише componentDidCatch().

Розробники React надали інструмент для автоматичної генерації початкового коду та підтримку його подальшої зміни під назвою Create React

App. При його використанні, вже повинен бути готовий конфігураційний файл webpack. Його завданням є створення єдиного вихідного файлу, який і буде містити весь додаток. Якщо використовуються сторонні бібліотеки і/або додаток збільшується в розмірах, то і вихідний файл буде ставати більше. Коли користувач зайде на сайт, браузер спочатку завантажить весь пакет і потім буде здійснено відображення вмісту сторінки. Це може сильно уповільнити сайт, тому набагато краще розділити код і таким чином розбити вихідний файл на частини. В результаті браузер буде завантажувати потрібні частини по мірі необхідності, що призведе до поліпшення часу завантаження сайту.

JavaScript є динамічною і слабо типізованою мовою, тому багато проблем виникають із-за неправильних типів. Для вирішення цієї проблеми можна використовувати різні інструменти для перевірки типів. Flow – відомий варіант. Він був розроблений в Facebook і часто використовується з React. Він дозволяє анотувати змінні, функції та компоненти React за допомогою спеціального синтаксису і дає можливість швидко відловлювати помилки.

Однією з найсильніших сторін React є той факт, що ця бібліотека не примушує розробника до використання класів. У Angular ж всі компоненти повинні бути реалізовані у вигляді класів. Це призводить до надмірного ускладнення коду, не даючи ніяких переваг.

Фреймворк React Native з'явився в 2015 році. Він призначений для розробки крос-платформних додатків з використанням бібліотеки React. Ці програми підтримують нативні можливості платформи, для яких їх створюють. У вихідної архітектури React Native були певні недоліки. Але, незважаючи на це, React Native отримав хорошу підтримку спільноти, його популярність поступово зростала, не в останню чергу – завдяки гучкої репутації React.

React Native – це приклад рішення, яке можна назвати «платформо-незалежним». В результаті виявляється, що головна мета фреймворку полягає в тому, щоб дозволити розробникам писати JavaScript-код, що використовує механізми React. Завдання React Native полягає в тому, щоб забезпечити

перетворення дерева React-компонентів у щось таке, що виявиться працездатним на різних мобільних платформах. Це означає наступне:

Коректне формування інтерфейсу.

Забезпечення доступу до нативних можливостей різних платформ.

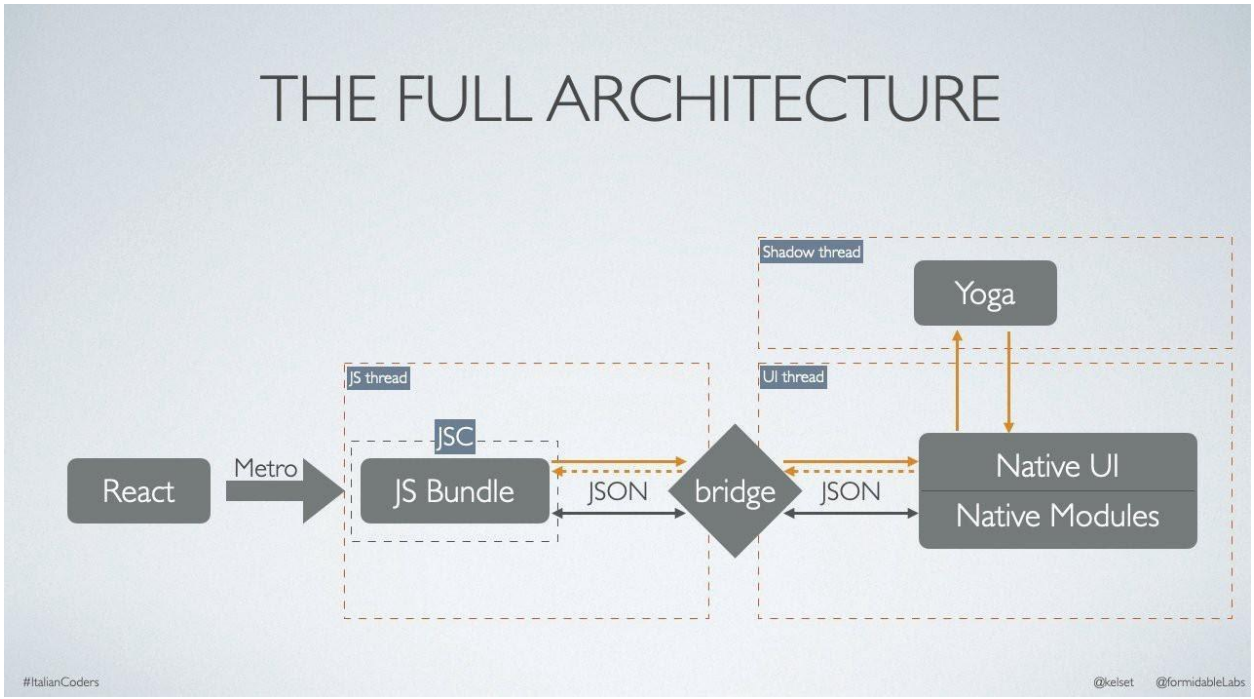

Рис.2.3. Стара архітектура React Native

У кожному React Native-додатку паралельно виконуються три наступних потоки:

- Потік JS. Це – потік, в якому здійснюється читання і компіляція JavaScript-коду. Тут же виконується основна частина логіки програми. Після цього Metro комбінує весь JS-код в єдиний файл, який відповідає за обробку JSX - і TS-конструкцій. Потім отриманий код відправляють виконавчому механізму JavaScriptCore, засобами якого цей код може бути запущений.
- Потік Native. Тут виробляється виконання нативного коду. Цей потік взаємодіє з JavaScript-потоком тоді, коли потрібно оновити інтерфейс або звернутися до стандартних функцій. Цей потік можна розділити на дві

частини. Перша, Native UI, відповідає за використання стандартних засобів формування інтерфейсу. Друга, Native Modules, надає доступ до особливих можливостей платформи, на якій працює додаток. Вона готова до роботи після запуску програми. Тобто, наприклад, якщо React Native використовує Bluetooth-модуль, він завжди буде активним, навіть у тому випадку, якщо він, насправді, не використовується.

- Потік Shadow. В ньому виконується перерахунок макету програми. Тут використовується Yoga, що є власною розробкою Facebook. У цьому потоці виконуються обчислення, пов'язані з формуванням інтерфейсу програми. Результати цих обчислень відправляють потоку, що відповідає за висновок інтерфейсу.

Для організації взаємодії між потоками JS і Native використовується модуль Bridge. Цей модуль написаний на C++, в його основі лежить асинхронна черга. Коли міст отримує дані від однієї із сторін, він серіалізує ці дані, перетворюючи в рядковій вигляд, і передає їх через чергу. Коли дані прибувають в пункт призначення, вони десеріалізуються.

В ході перепроектування архітектури React Native виконується поступова відмова від функцій мосту, на зміну яким приходить новий механізм, званий JavaScript Interface (JSI).

Застосування JSI відкриває можливості для деяких чудових поліпшень.

Перше таке поліпшення полягає в тому, що JS-бандл більше не покладається на JSC. Іншими словами, виконавчий механізм JSC тепер легко можна замінити на щось інше, що відрізняється більш високою продуктивністю, наприклад – на V8, що використовується в браузері Google Chrome.

Друге поліпшення – це те, що лежить в основі нової архітектури React Native. Воно полягає в тому, що, завдяки використанню JSI, в JavaScript можна зберігати посилання на C++-об'єкти (Host Objects) і викликати методи цих об'єктів. В результаті виявляється, що JavaScript-частина програми та нативні механізми будуть знати один про одного набагато більше, ніж раніше.

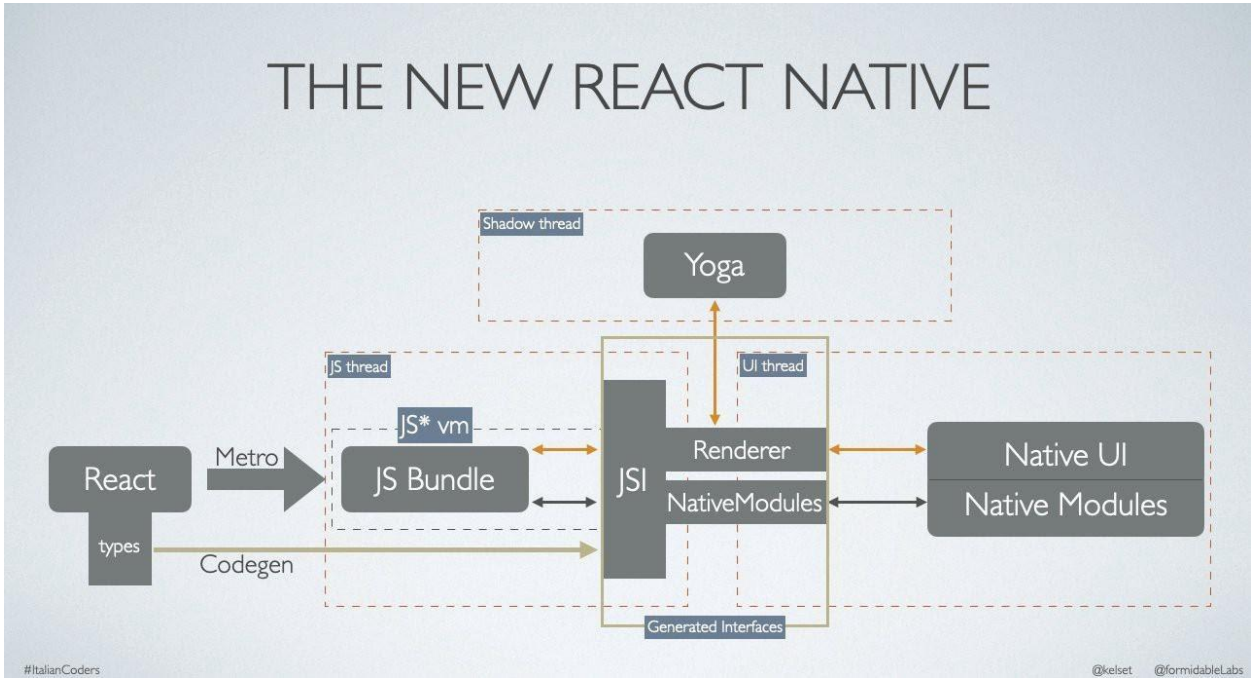

Рис. 2.4. Нова архітектура React Native

Іншими словами, JSI дозволяє забезпечити повну взаємодію між всіма потоками. Завдяки використанню концепції спільного володіння (shared ownership), JavaScript-код може запускати нативні методи безпосередньо з JS-потоку. При цьому немає необхідності в тому, щоб використовувати JSON для передачі даних. Це дозволяє позбутися від проблем, характерних для використання моста, пов'язаних з можливим переповненням черго і з асинхронною передачею даних.

2.2. Можливості та області використання Angular

В основі AngularJS полягає принцип, що декларативний стиль краще підходить для створення UI, а імперативний для написання бізнес-логіки.

При використанні данного фреймворку слід пам’ятати про наступне:

− Добре відокремлювати маніпуляцію DOM-му від логіки роботи програми. Це істотно покращує тестування коду.

- Добре вважати, що автоматизоване тестування програми настільки ж важливо, наскільки і написання самого додатка. Тестування дуже сильно залежить від того, як структуровано код.
- Добре відокремлювати розробку клієнтської частини від серверної. Це дозволяє вести розробку паралельно і покращує повторне використання на обох сторонах.
- Добре, коли фреймворк веде розробника по всьому циклу розробки програми: від проектування UI через написання бізнес-логіки до тестування.
- Добре, коли поширені завдання стають тривіальними, а складні – спрощуються.

AngularJS являє собою комплексний фреймворк. У стандартній поставці він надає наступні можливості:

- Все, що потрібно для створення CRUD-додатків: data binding, базові директиви для шаблонів, валідація форм, роутинг, deep linking, повторне використання компонентів, dependency injection, інструменти для взаємодії з серверними (RESTful) джерелами даних.
- Все, що потрібно для тестування: засоби для модульного тестування, end-to-end тестування, заміна реальних запитів тестовими даними.
- Шаблон типової програми, що включає в себе структуру каталогів і тестові скрипти.

AngularJS розробляється співробітниками Google і використовується, як мінімум, в одному сервісі Google – DoubleClick.

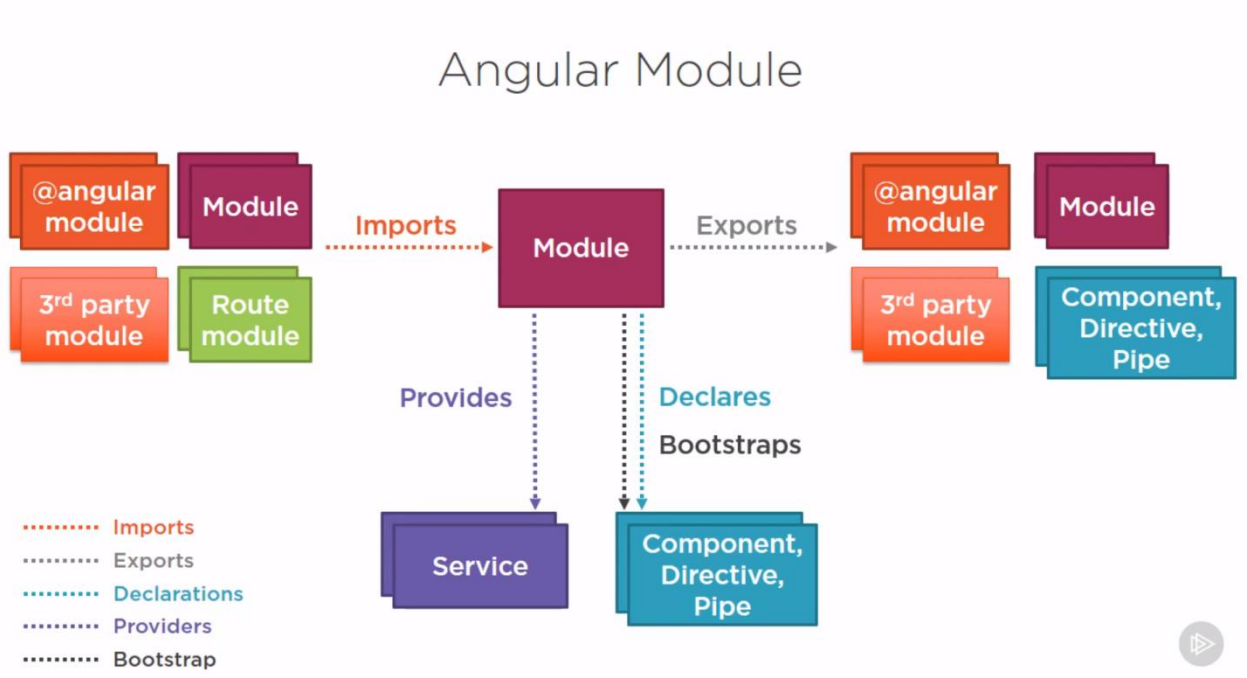

Рис. 2.5. Модульна структура Angular

Основні поняття AngularJS

Директиви

На директивах тримається практично вся декларативна частина AngularJS. Саме вони використовуються для збагачення синтаксису HTML. В процесі компіляції DOM директиви беруться з HTML і виконуються. Директиви можуть додати нову поведінку та/або модифікувати DOM. У стандартне постачання входить досить велика кількість директив для побудови веб додатків. Але ключовою особливістю є можливість розробки своїх директив, за рахунок чого HTML може бути перетворений в DSL.

Директиви іменуються за допомогою lowerCamelCase, наприклад, ngBind. При використанні директиву необхідно іменувати в нижньому регістрі з використанням в якості роздільника одного з спец символів: :, -, або _. За бажанням для отримання валідного коду можна використовувати префікси x- або data-. Приклади: ng:bind, ng-bind, ng_bind, x-ng-bind і data-ng-bind.

Директиви можуть використовуватися як елемент (<my-directive></my-directive>), атрибут (), у класі () або в коментарі (). Це залежить від того, як конкретна директива була розроблена.

Межі

Scope – це об'єкт, що має відношення до моделі в додатку. Він є контекстом для виконання виразів. Scope шикуються в ієрархічну структуру, схожу на DOM. При цьому вони успадковують властивості своїх батьківських scopes.

Scope є зв’язуючою ланкою між контролером і відображенням. В процесі виконання фази зв'язування шаблону директиви встановлюють спостереження ($watch) за виразами в рамках scope. $watch дає директивам можливість реагувати на зміни для відображення оновленого значення або яких-небудь інших дій. І контролери, і директиви мають посилання на scope, але не мають посилань один на одного. Так контролери ізолюються від директив і від DOM-а. За рахунок цього зростають можливості тестування програми.

Сервіси

Сервіси – існують в єдиному вигляді та виконують якусь конкретну задачу, яка є спільною для всіх або конкретного веб-додатку. Наприклад, $http сервіс, який є обгорткою над XMLHttpRequest. Кілька прикладів інших сервісів:

$compile — компіляція HTML-рядку або частини DOM-а в шаблон, зв'язування шаблону з конкретним scope;

$cookies — надає доступ на читання/запис до cookies.

$location — робота з адресним рядком

$resource — фабрика по створенню ресурсних об'єктів, призначених для взаємодії з серверними (RESTful) джерелами даних.

Для використання сервісу необхідно вказати його як залежність для контролера, іншого сервісу, директиви і т. п. AngularJS потурбується про все інше – створення, розв'язання залежностей і т. п.

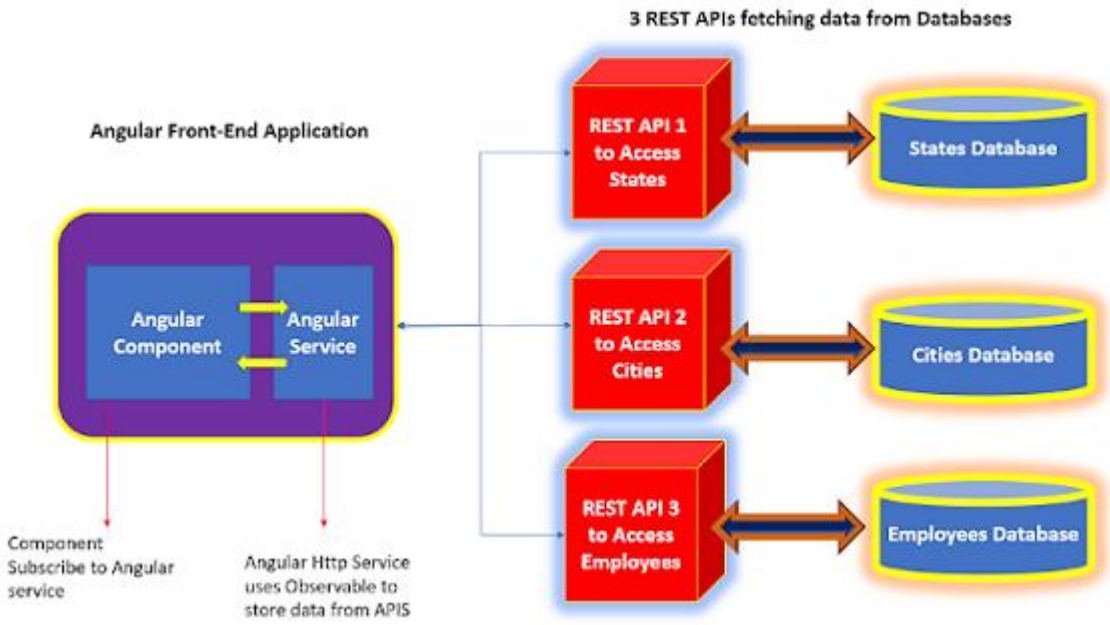

Рис. 2.6. Сценарій роботи з REST API в Angular

Фільтри

Фільтри призначені для форматування даних перед відображенням їх користувачеві, а також фільтрації елементів в колекціях.

Модулі

Додатки в AngularJS не мають основного виконуваного методу. Замість цього модуль виконує роль декларативного опису того, як додаток повинен бути завантажений. Завдяки цьому, наприклад, при написанні сценаріїв тестування можна довантажити додаткові модулі, які матимуть вищий пріоритет, ніж якісь налаштування, полегшуючи тим самим комплексне (end-to-end) тестування.

AngularJS Batarang

Це розширення для Chrome, яке полегшує налагодження AngularJS додатків. Дозволяє працювати з ієрархією scopes, дає можливість профілювання програми, візуалізує залежності між сервісами, відображає вміст scopes на сторінці елементів, дозволяє виводити і змінювати значення scope з консолі.

2.3. Можливості та області використання Vue

Творцем Vue.js є колишній співробітник Google і Meteor Dev Group - Еван Ю. Розробка фреймворку ведеться з 2013-го, а в лютому 2014-го відбувся перший публічний реліз. Vue широко використовується серед китайських компаній, наприклад: Alibaba, Baidu, Xiaomi, Sina Weibo і ін. Він входить в ядро Laravel і PageKit. Нещодавно вільна система керування репозиторіями GitLab теж перейшла на Vue.js.

В кінці вересня 2016-го вийшов реліз Vue.js 2.0, ще краще і з упором на продуктивність – тепер використовується віртуальний DOM, підтримується рендеринг на стороні сервера, можливість використовувати JSX і т. д. Хоча зараз він підтримується тільки співтовариством, він тримається гідно навіть на рівні продуктів таких гігантів, як Google і Facebook, і поступово наздоганяє їх по популярності.

Основними концепціями Vue є:

- конструктор;
- компоненти;
- директиви;
- переходи.

Будь-яка робота з Vue.js починається зі створення нової сутності Vue. В результаті ми маємо властивість “el” - що є елементом, за яким стежить Vue. У “template” знаходиться обраний елемент. В “data” зберігається поточний стан сутності, а метод “computed” надає нам обчислювані властивості.

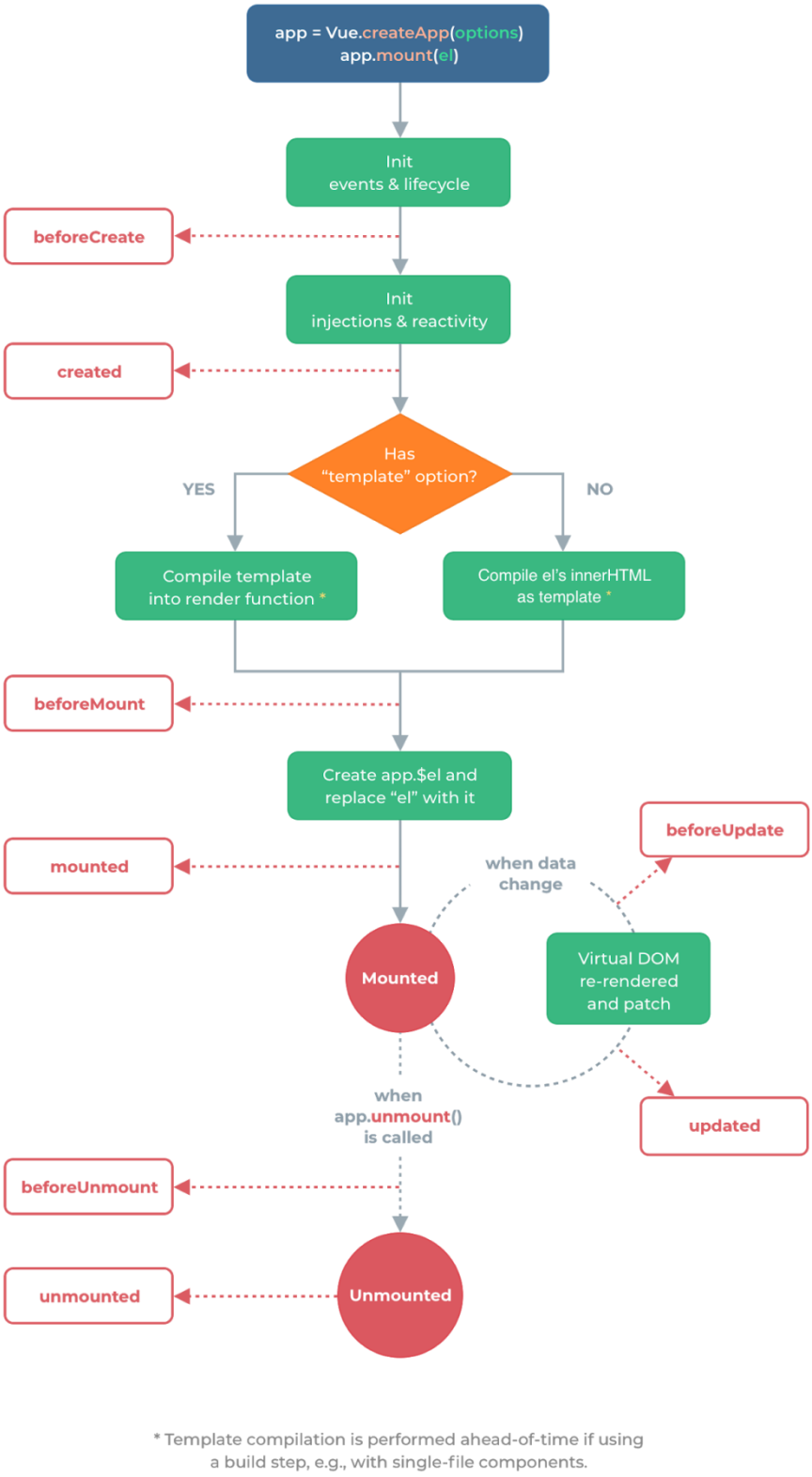

Рис. 2.7. Структура та життєвий цикл сутності Vue

Серед методів можна виділити наступні:

- beforeCreate – виконується одразу після ініціалізації;
- created – на цьому етапі ініціалізація завершена, що означає готовність усіх даних, обчислювальних властивостей, методів та функцій зворотного виклику для обробників подій. Проте сам компонент ще не відображений, та доступ до $el пока відсутній;
- beforeMount – викликається одразу перед відображенням, метод render() буде вперше викликаний сам тут;
- mounted – остання фаза відображення елементу.

При зміні стану:

- beforeUpdate – знову перебудовує віртуальний DOM і порівнює з реальним, застосовує зміни;
- updated – зміни прораховані;
- beforeUnmount – скасування усіх слухачів подій, внутрішніх компонентів і даних, проте на цій стадії увесь функціонал ще доступний;
- unmounted – сутність була повністю видалена.

Директиви

Директиви – спеціальні атрибути для додавання елементів html додаткової функціональності.

Розглянемо деякі вбудовані директиви (схожі з Angular):

- V-bind – динамічно зв'язується з одним або декількома атрибутами;
- V-cloak – приховує синтаксис шаблону, поки усі дані не будуть готові до відображення;
- v-if – умова для фонового елемента;
- V-else – позначає "else блок" для v-if;
- V-for – циклічно проходить масив об'єктів;
- V-model – пов'язує стан з input елементом;
- V-on – пов'язує слухача події з елементом;
- V-once – рендерить елемент тільки спочатку і більше не стежить за ним;

- V-pre – не компілює елемент і його дочірні елементи;
- V-show – перемикає видимість елемента, змінюючи властивість CSS display;
- V-text – оновлює textContent елемента.

Всі Vue-директиви мають префікс "v-". В директиву передається якесь значення стану, а в якості аргументів можуть бути атрибути html або події.

Компоненти

Компоненти допомагають розширити основні елементи html і впровадити пере виконуваний код. По суті, компоненти – повторно використовувані частини UI. На етапі проектування розбиваємо додаток на незалежні частини і отримуємо деревоподібну структуру компонентів.

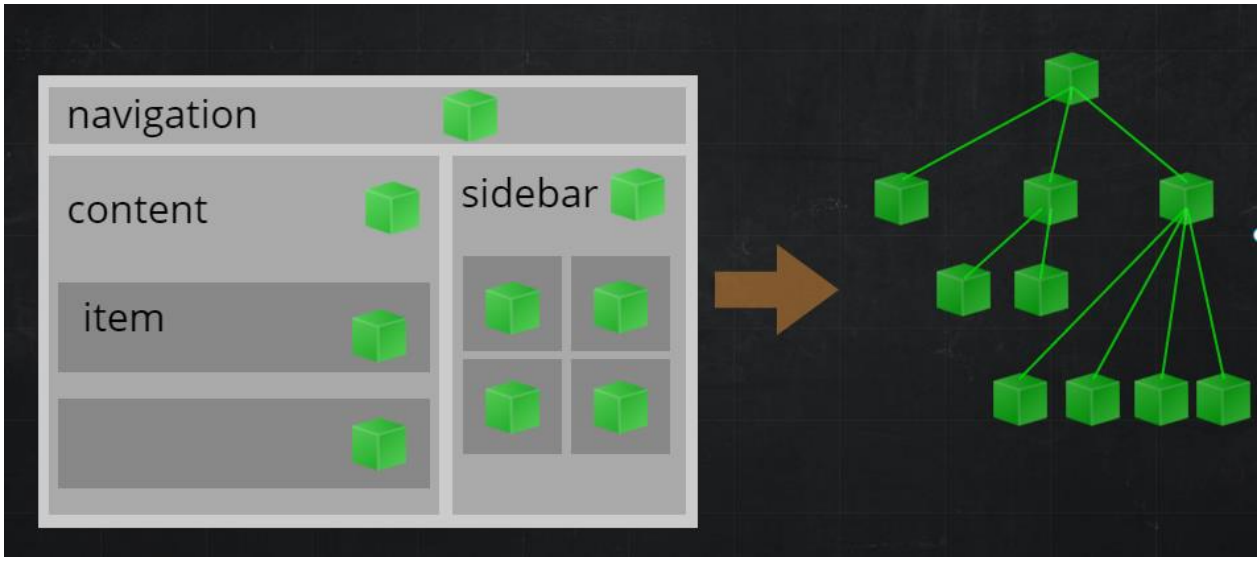

Рис. 2.8. Принцип формування структури проекту на Vue.js

В Vue.js немає особливих вимог до імен компонентів, але хороша практика – дотримуватися правил W3C щодо власних компонентів, тобто букви нижнього регістру і поділу через дефіс.

Новий компонент оголошується за допомогою Vue.component, і в перший аргумент передаємо ім'я нового тега.

Комунікація між vue-компонентами здійснюється за принципом "Props in, Events out". Тобто від батьківського елемента до дочірнього інформація

передається через властивості, а зворотній зв’язок утворюється через виклик подій (рис. 2.9).

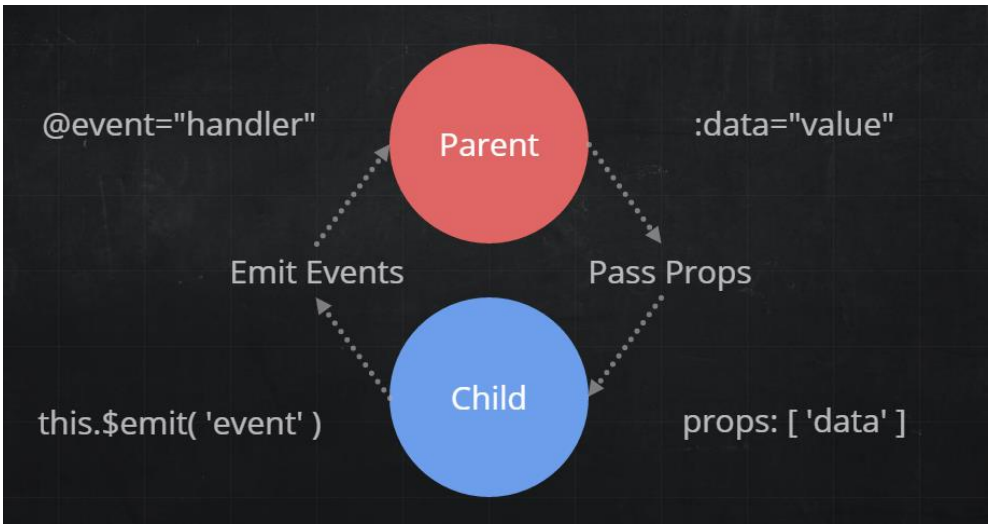

Рис.2.9. Зв’язок між вкладеними компонентами

Також у Vue.js можна використовувати так звані однофайлові компоненти - файли з розширенням .vue, що містять у собі стилі, шаблон і логіку. Реалізація стилів може бути виконана на будь-якому зручному препроцесорі (SASS, Stylus, PostCSS, Jade, та ін.) і мовою, що компілюється в JavaScript (CoffeeScript, TypeScript).

Переходи

Vue надає різні способи застосування анімаційних ефектів, коли елементи поновлені або вилучені з DOM. Вони включають в себе інструменти для:

- автоматичного застосування класів для CSS-переходів і анімації;
- інтеграції сторонніх бібліотек для CSS-анімації, таких як Animate.css;
- використання JavaScript для маніпуляції DOM;
- інтеграції сторонніх бібліотек JavaScript для анімацій, таких як Velocity.js.

Екосистема фреймворку

Маршрутизація

У Vue.js за маршрутизацію відповідає окремий пакет vue-router. Він підтримує вкладені маршрути до вкладених компонентів, пропонує спрощене API для навігаційних хуків, керована поведінка скролу і просунутий контроль переходів.

Ajax-запити

Для роботи з Ajax-запитів існує плагін vue-resource. Він надає можливості для створення веб-запитів та обробки відповідей за допомогою XMLHttpRequest або JSONP. Також особливістю плагіна є підтримка Promise API і URI шаблонів.

Управління станом

Vuex – патерн і бібліотека керування станом для додатків на Vue.js. Він надає централізований загальний стан для всіх компонентів в додатку і правила, що забезпечують передбачувану зміну стану.

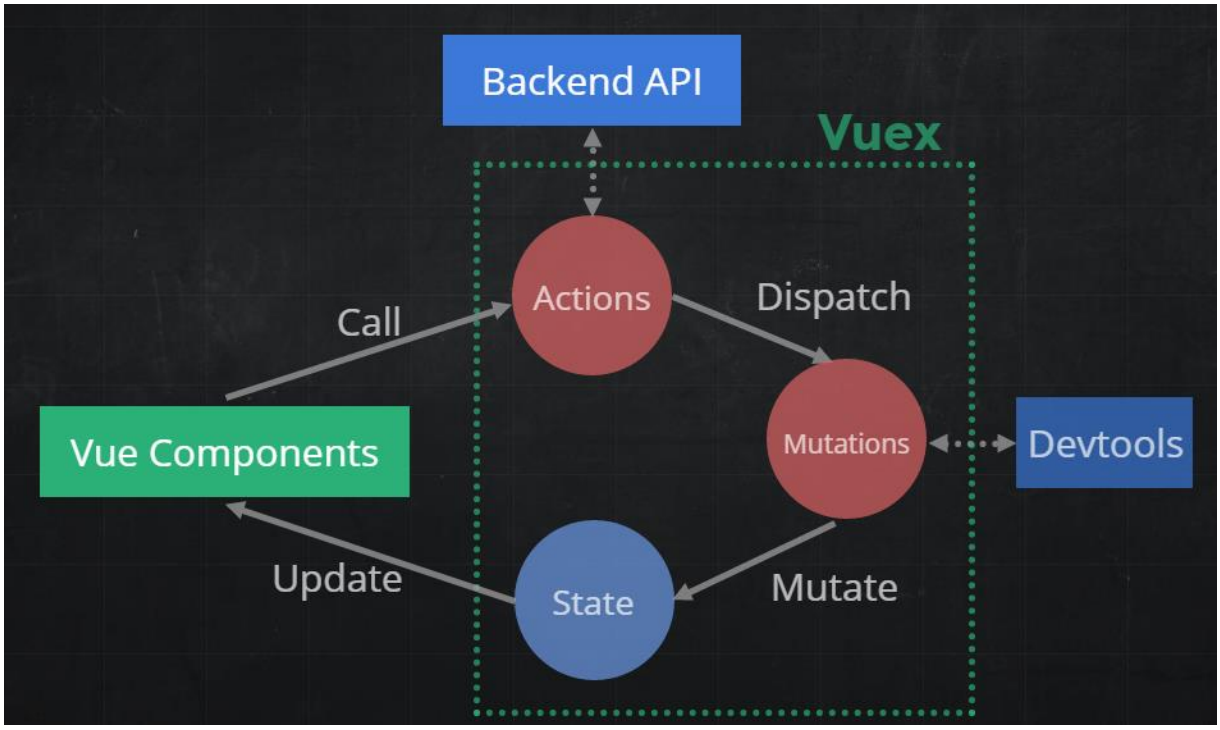

Рис. 2.10. Схема роботи додатку на Vue та Vuex

Складовими частинами додатку на Vue та Vuex (рис. 2.10) є:

– Стан (State), який є єдиним джерелом даних для компонентів.

– Vue-компоненти (Vue-components), які по суті є лише декларативним відображенням стану.

– Дії (Actions), які виловлюють подію, яка сталася, збирають дані із зовнішніх API і запускають потрібні мутації.

– Мутації (Mutations) – єдина частина, яка може змінювати стан і, отримавши дані від Actions, оновлює стан.

Система збирання та інструменти розробника

Для налагодження в браузері Google Chrome існує Vue-devtools, який дозволяє бачити компоненти, які є в додатку та їх поточний стан.

Також він чудово працює з Vuex і дозволяє виконувати так званий time-travel debugging: в браузері можемо побачити історію станів і перемикатися між ними.

У Vue.js завжди приділялася велика увага швидкодії. Прошарок, що відповідає за відображення, переписано на полегшену реалізацію віртуального DOM – Snabbdom.

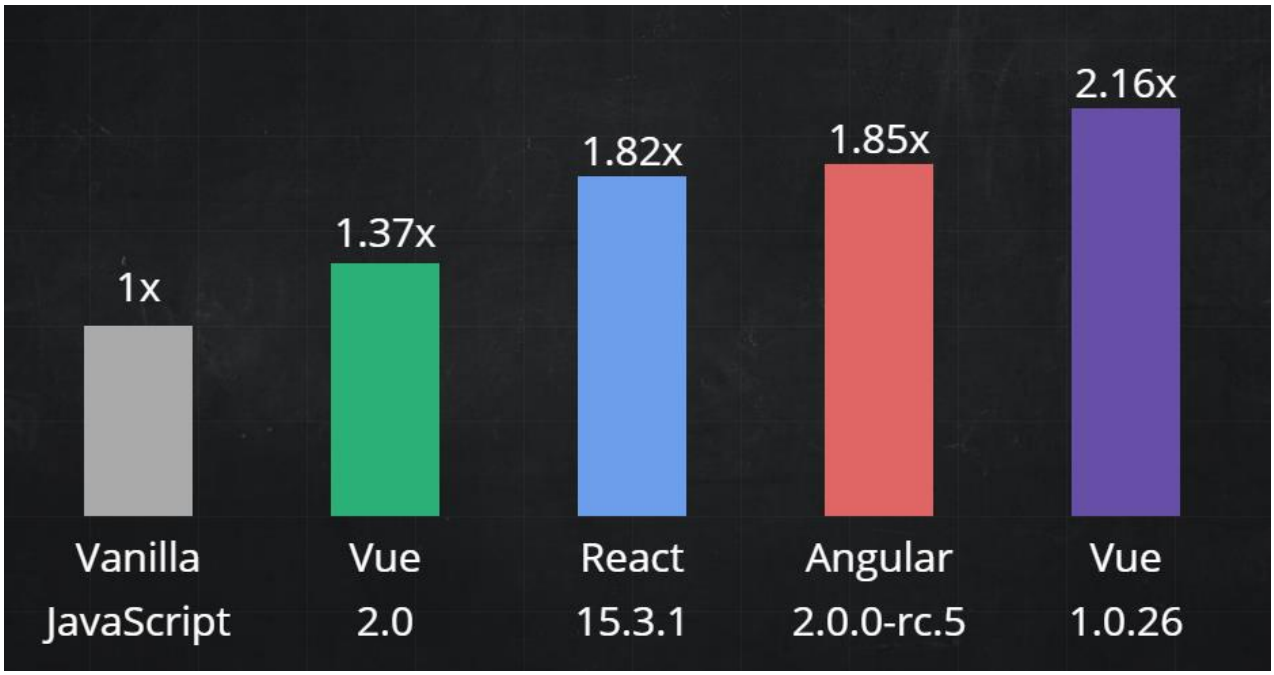

Рис. 2.11. Порівняння швидкодії фреймворків

Vue 2.0 підтримує відображення на стороні серверу. Для цього є модуль Vue-server-renderer, підтримує інші інструменти з екосистеми Vue (vue-router і vuex). Тепер набагато простіше створювати ізоморфні програми.

2.4. Переваги та недоліки фреймворків

Виділимо основні переваги і недоліки кожного фреймворка.

Плюси і мінуси Angular

Angular – це середа JavaScript MVVM, заснована в 2009 році, яка відмінно підходить для створення інтерактивних веб-додатків.

Переваги Angular:

1) Angular використовується разом з Typescript. Він має виняткову підтримку для цього.

2) Angular-language-service - забезпечує інтелектуальні можливості і автозаповнення шаблону HTML-компонента.

3) Детальна документація, що дозволяє розробнику отримати всю необхідну інформацію та зводить до мінімуму потребу застосування сторонніх ресурсів.

4) Одностороння прив'язка даних, яка забезпечує виняткову поведінку додатка, що зводить до мінімуму ризиків можливих помилок.

5) MVVM (Model-View-ViewModel), яка дозволяє розробникам працювати окремо над одним і тим же розділом програми, використовуючи один і той же набір даних.

6) Впровадження залежностей від компонентів, пов'язаних з модулями і модульність в цілому.

7) Структура і архітектура, спеціально створені для великої масштабованості проекту.

Недоліки Angular:

1) Різноманітність структур (Injectables, Components, Pipes, Modules і т. д.) може ускладнювати проект в порівнянні з React і Vue.js, що використовують тільки компоненти.

2) Відносно повільна продуктивність, враховуючи різні показники. З іншого боку, це можна легко вирішити, використовуючи так званий «ChangeDetectionStrategy», який допомагає вручну контролювати процес відображення компонентів.

Плюси і мінуси React

Переваги React:

1) Легко вивчити, завдяки простому дизайну, використання JSX (HTML-подібний синтаксис) для шаблонів і дуже докладної документації. Розробники мають змогу сфокусуватись на реалізації і менше турбуватись про код, специфічний для фреймворка.

2) Дуже висока швидкодія, завдяки реалізації React Virtual DOM і різним оптимізаціям відображення.

3) Відмінна підтримка генерації на стороні сервера, що робить його потужною платформою для контент-орієнтованих додатків.

4) Підтримка Progressive Web Application (PWA).

5) Прискорення процесу ініціалізації проекту завдяки Create React App.

6) Прив'язка даних є односторонньою, що означає менше небажаних побічних ефектів.

7) Redux, бібліотека для управління глобальним станом компонентів в React.

8) React реалізує концепції функціонального програмування, створюючи простий в тестуванні код, який може бути використаний повторно.

9) Додатки можуть бути створені за допомогою TypeScript або Facebook's Flow, що мають вбудовану підтримку JSX.

10) Перехід між версіями, як правило, дуже простий: Facebook надає «кодові модулі» для автоматизації більшої частини процесу.

11) Навички, отримані в React, можуть бути застосовані до розробки на React Native.

Недоліки React:

1) React не однозначний і залишає розробникам можливість вибирати спосіб реалізації самостійно. Це може бути вирішено сильним лідерством проекту і хорошими процесами.

2) Відсутній єдиний вбудований підхід до написання CSS в React, які поділяються на традиційні таблиці стилів (CSS Modules) і CSS-in-JS (бібліотеки для інкапсулювання стилів у компоненти, такі як Emotion і Styled Components).

3) React відходить від компонентів на основі класів, що може стати перешкодою для розробників, яким більш комфортно працювати з об'єктно-орієнтованим програмуванням (ООП).

4) Змішування шаблонів з логікою (JSX).

Плюси і мінуси Vue.js

Переваги Vue.js:

1) Посилений HTML. Це означає, що Vue.js має багато характеристик схожих з Angular, а це, завдяки використанню різних компонентів, допомагає оптимізації HTML- блоків.

2) Детальна документація. Vue.js має дуже детальну документацію, яка може прискорити процес навчання для розробників і заощадити багато часу на розробку програми, використовуючи тільки базові знання HTML і JavaScript.

3) Адаптивність. Може бути здійснений швидкий перехід від інших фреймворків до Vue.js через схожість з Angular і React з точки зору дизайну і архітектури.

4) Приголомшлива інтеграція. Vue.js можна використовувати як для створення односторінкових додатків, так і для більш складних веб-інтерфейсів додатків. Важливо, що невеликі інтерактивні елементи можна легко інтегрувати в існуючу інфраструктуру без негативних наслідків.

5) Масштабування. Vue.js може допомогти в розробці досить великих шаблонів багаторазового використання, які можуть бути зроблені майже за той же час, що і більш прості.

6) Крихітний розмір. Vue.js важить близько 20 КБ, зберігаючи при цьому свою швидкість і гнучкість, що дозволяє досягти набагато кращої продуктивності в порівнянні з іншими платформами.

Недоліки Vue.js:

1) Недолік ресурсів. Vue.js як і раніше займає досить невелику частку ринку в порівнянні з React або Angular, що означає, що обмін знаннями в цьому середовищі все ще перебуває на початковій стадії.

2) Ризик надмірної гнучкості. Іноді у Vue.js можуть виникнути проблеми при інтеграції у величезні проекти, і поки ще немає досвіду можливих рішень, але вони обов'язково з'являться найближчим часом.

2.5. Висновки до другого розділу

Не дивлячись на те що front-end фреймворки створені для вирішення одних і тих самих завдань, вони мають свої плюси і мінуси, а це означає, що при розробці продукту потрібно зробити правильний вибір для кожного окремого випадку.

Виділивши характерні риси для кожної технології можна зробити висновок що майже всі фреймворки є сенс застосовувати тільки коли проект досить великий, адже їх початкова конфігурація потребує певних витрат часу, а архітектури спрямовані на багаторазове використання окремих фрагментів.

Фреймворки з досить низьким порогом входу такі як React та Vue забезпечують більш швидкий розвиток проекту на старті, але можуть створити більше проблем на кінцевих стадіях, або у випадку незапланованого розширення, у той час як Angular навпаки потребує більше часу на початкових стадіях, але може дати більше гнучкості зі збільшенням проекту.

РОЗДІЛ 3
МЕТОДОЛОГІЯ ВИБОРУ ТЕХНОЛОГІЇ ДЛЯ ВИРІШЕННЯ ПРАКТИЧНИХ ЗАВДАНЬ

3.1. Виділення ключових критеріїв проекту

Зазвичай на сьогоднішній день вибір фреймворку розробником перед початком проекту залежить від його суб’єктивних вражень щодо певної технології: вона може йому подобатись, на ній можуть бути реалізовані відомі ресурси, або її може порадити інший розробник. Проте, якщо перед початком розробки не провести аналіз проекту, виділивши його характерні риси, поспішний вибір фреймворку може призвести до проблем на етапі розробки.

Виділення критеріїв проекту допомагає заздалегідь оцінити можливі ризики. Тому для початку потрібно визначити основні фактори ризику для проекту. Це можна зробити різними способами: аналіз минулих аналогічних проектів, підготовка всієї проектної команди до загальної наради, на якій кожен озвучує ризики, які він бачить в проекті, і використання стандартних списків ризиків, як правило, наявних в кожному проекті (специфічних для галузі в цілому або для компанії). Визначимо наступні показники:

- розмір і тип проекту;
- складність проекту;
- швидкість розробки;
- вартість спеціалістів;
- доступність спеціалістів;
- наявність готових рішень;
- наявність широкої підтримки спільноти;
- гнучкість рішень;
- вимоги до швидкодії;

Вибір технології виходячи з таких критеріїв вносить набагато більше об’єктивності і тим самим заощаджує час та гроші на розробку. Ці метрики

можна умовно розділити на дві групи: для першої можна визначити фреймворк спираючись на статистичні дані щодо його використання, проте для другої необхідно провести практичний аналіз вихідного коду фреймворку.

3.2. Зіставлення технологій на основі статистичних показників

Спочатку проведемо огляд якісних метрик проекту - тих які не потребують проведення розрахунків і можна спиратись на статистичні дані.

Перш за все потрібно виділяти розмір, тип і складність проекту. Тут можна виділити 3 категорії:

- Прості: візитки, односторінкові додатки, примітивні інтернет магазини - такі рішення зазвичай можуть виконуватись за домогою систем управління вмістом (Content Management System), або готовими тематичними інструментами;
- Середні: великі інтернет магазини, портали міського або регіонального значення, інтернет видання або журнали - такі проекти зазвичай виконуються за допомогою фреймворків;
- Великі: портали національного значення, міжнародні проекти, соціальні мережі, стрімінгові сервіси - зазвичай такі проекти розробляються з використанням лише стандартних засобів мов програмування, адже вони потребують найбільшої гнучкості та надійності.

Наступним досить важливим критерієм є швидкість розробки. Якщо проект є досить невеликим за обсягом, але потребує швидкої реалізації, найкращим вибором буде Vue. React займає середню позицію серед трьох, адже він найбільш збалансований з точки зору часу та обсягу розробки. Але оскільки він не пропонує великого різноманіття готових архітектурних принципів і рішень потрібно переконатись що у команді буде наявний хоча б один розробник з досить великим досвідом розробки і планування проектів та зможе власноруч налагодити процес. Angular в свою чергу підходить для проектів з

великим терміном реалізації, адже має набагато більше архітектурних нюансів, що притаманні великим проектам.

Доцільним також буде розглянути попит на ту чи іншу технологію в країні, а також середні показники заробітних виплат, адже це може мати значний вплив на бюджет проекту. Спираючись на дані пошукових запитів [35], можна зробити наступні висновки: в Україні переважає попит на розробників React, у той час як Vue та Angular займають другу позицію (рис. 3.1)

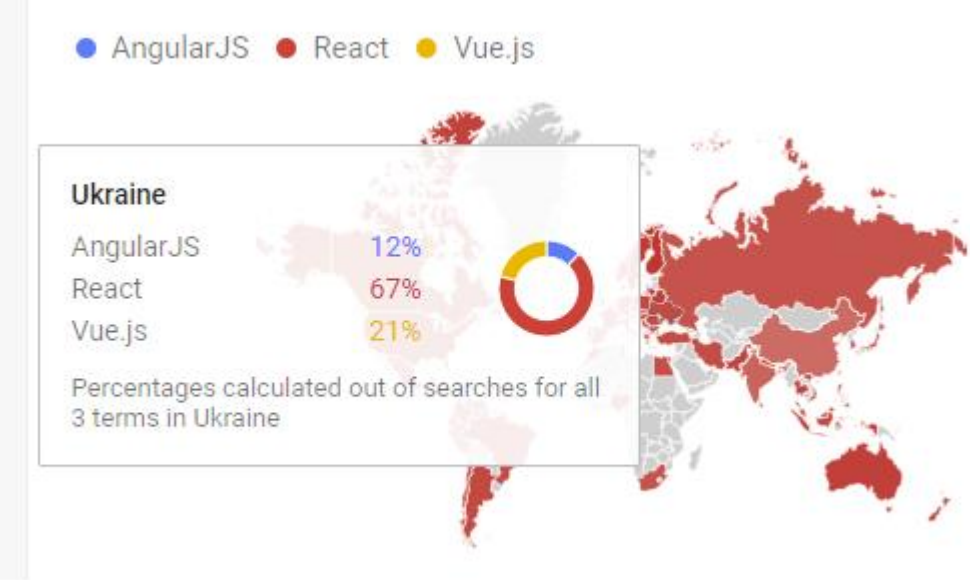

Рис 3.1. Порівняння попиту front-end фреймворків в Україні

Треба також розглянути тенденції у рамках світу, адже не рідко у якості замовників можуть виступати представники або компанії інших країн. Зробивши аналіз (рис. 3.2) можна побачити що React переважає у північно-західних країнах у той час як на півдні провідні позиції займає Vue. У Європі ж найбільшим попитом користується Angular. Отже, можна побачити що навіть географічний фактор має бути розглянутий при плануванні проекту.

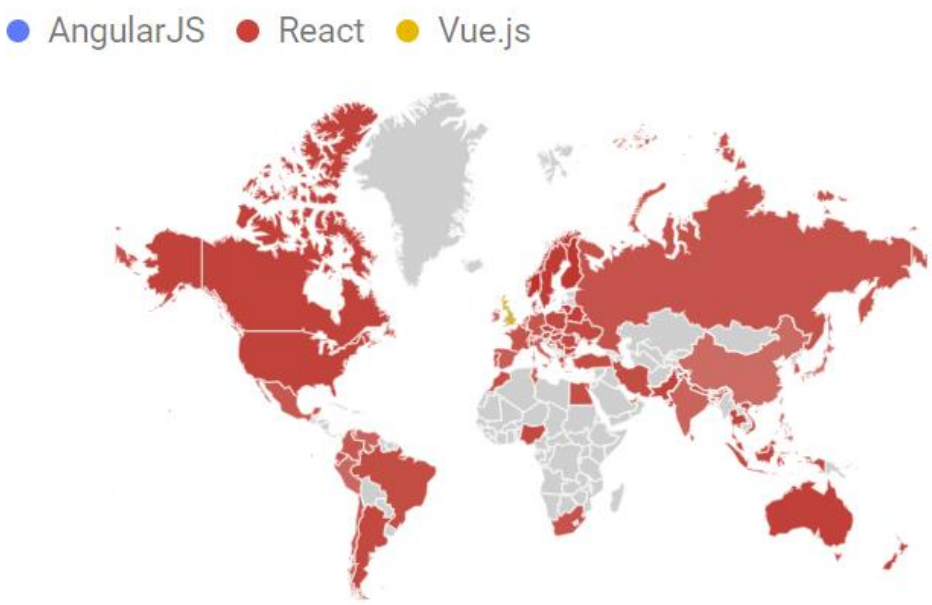

Рис 3.2. Порівняння попиту front-end фреймворків у світі

Велику роль у процесі розробки також відіграє наявність та підтримка технології спільнотою, адже у такому випадку швидкість пошуку рішень або готових шаблонів чи підходів буде досить високою, відповідно це підвищує продуктивність розробки. Тут треба звернути увагу на Vue - адже він вважається досить новим фреймворком і оскільки ще не набрав великого попиту, пошук інформації може уповільнити процес розробки. З іншої сторони React та Angular мають дуже велику підтримку спільноти в силу своєї розповсюдженості.

Можна також проаналізувати тенденцію зацікавленості розробників у обраних фреймворках, адже розробка проекту потребує певного часу і потрібно пересвідчитись що протягом цього часу актуальність обраної технології не почне знижуватись. Для цього використаємо дані ресурсу StateOfJS - щорічного міжнародного опитування веб-розробників [36].

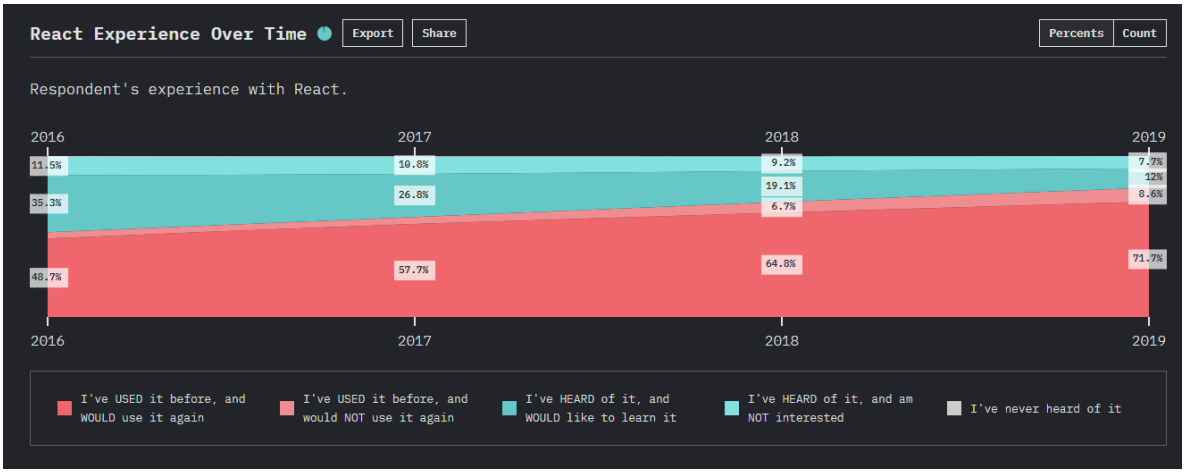

Рис. 3.3. Тенденція зацікавленості розробників у фреймворку React

Аналізуючи тенденцію зацікавленості у React (рис. 3.3) можна побачити що вона досить рівномірно зростає кожного року починаючи з 2016 і враховуючи активну підтримку і розвиток самого фреймворку можна досить впевнено вважати, що тенденція зросту не буде змінюватись у найближчому майбутньому.

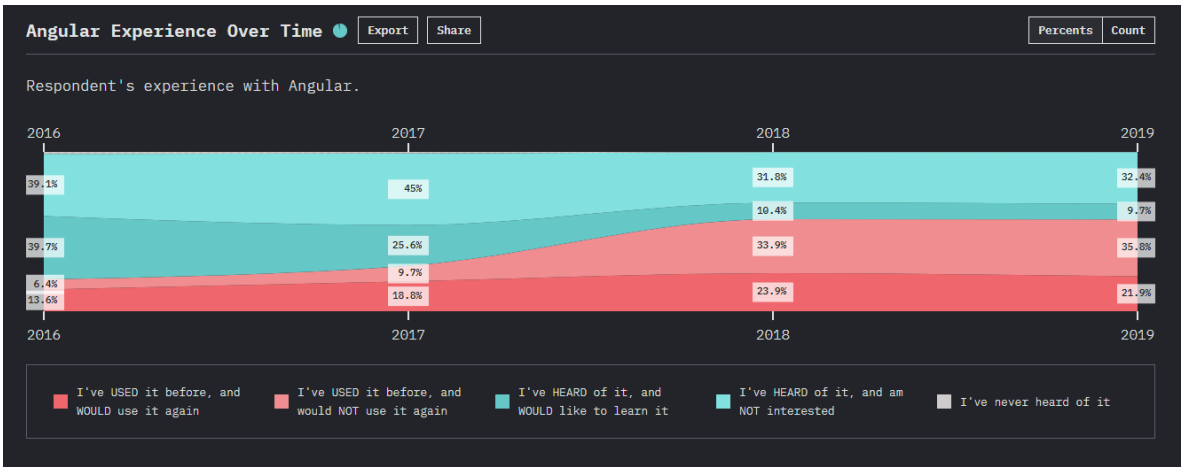

Рис. 3.4. Тенденція зацікавленості розробників у фреймворку Angular

Якщо ж подивиться на результати Angular (рис. 3.4) бачимо, що відсоток розробників що використовують та планують далі використовувати даний фреймворк майже не змінювався протягом часу. Також треба звернути увагу на стрімке збільшення кількості людей з негативним досвідом розробки починаючи з 2018 року. Досить великий відсоток займають розробники, що не

є зацікавленими у використанні цього фреймворку. Підсумовуючи можна сказати що загальна тенденція досвіду роботи з Angular не є дуже позитивною і це потрібно врахувати при плануванні проекту.

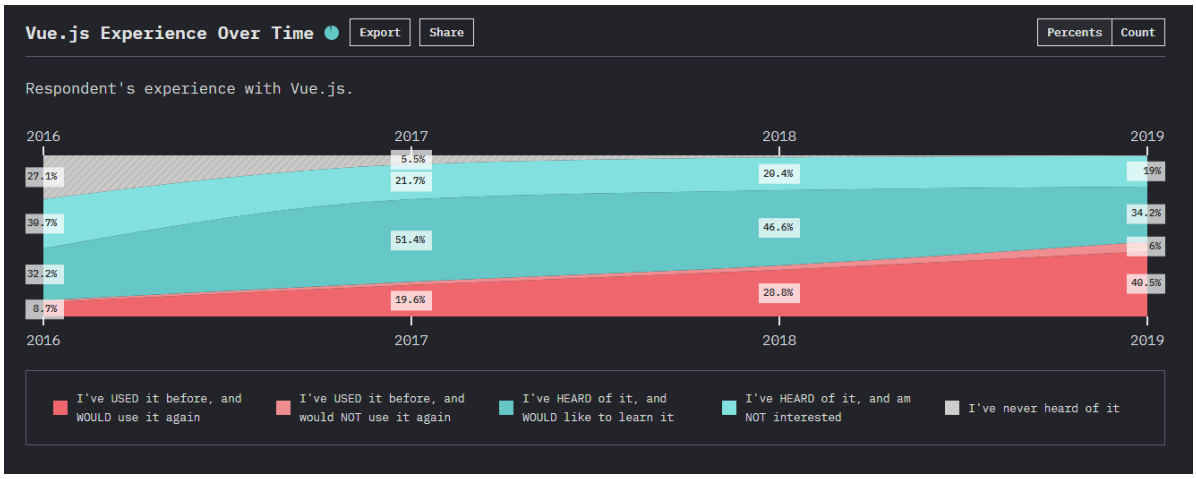

Рис. 3.5. Тенденція зацікавленості розробників у фреймворку Vue

Ситуація з Vue виглядає дуже позитивно (рис. 3.5) - зацікавленість та задоволення від його використання дуже стрімко зростає з кожним роком. Майже повна відсутність негативного досвіду дає інформацію про те що цей фреймворк є досить надійним. Враховуючи те, що даний фреймворк є досить новим, велика кількість опитуваних позитивно дивиться на можливість його застосування, що також є позитивним фактором.

Для кількісної оцінки якості програмного забезпечення потрібно використовувати чисельні заходи, які відображають певні властивості програмного продукту або його специфікації. Таким чином необхідно проаналізувати такі показники, як швидкість роботи, розмір, документованість.

3.3. Практичний аналіз вихідного коду фреймворків

В роботі проведено порівняльний аналіз продуктивності, якості і валідності коду трьох front-end фреймворків: React, Angular, Vue. Вибір

заснований на інформації, розташованої на офіційних github-репозиторіях даних продуктів (табл. 3.1).

Для тесту продуктивності використані мініфіковані версії вихідних кодів фреймворків, в той час як для оцінки якості і валідності коду використані версії для розробки.

Таблиця 3.1

Список порівняних front-end фреймворків

Назва	Кількість «зірок» на github.com, тис.	Посилання
React	161	https://github.com/facebook/react
Angular	59	https://github.com/angular/angular.js
Vue	177	https://github.com/vuejs/vue

Показники були обчислені за допомогою прикладного програмного забезпечення для статичного аналізу коду Scitools Understand 4.0, утиліт plato і complex-report. Результати підрахунку чисельних показників обраних фреймворків представлені в табл. 3.2.

Таблиця 3.2

Результати розрахунку показників коду фреймворків

Показник програмного забезпечення	Angular	Vue	React
1	2	3	4
Кількість рядків у коді	36595	11965	2333
Кількість виразів	14775	2655	875
Кількість рядків коментарів	8839	996	329

Продовження таблиці 3.2

1	2	3	4
Співвідношення кількості рядків коментарів та рядків коду	0,24	0,08	0,14
Цикломатична складність	3544	1173	415
Глибина вкладеності	10	9	9
Кількість попереджень	19	176	100
Середня важкість функцій згідно методу Холстеда	1323	5333	4356

Цикломатична складність — метрика програмного забезпечення, розроблена Томасом Мак Кабе, що використовується для оцінки складності програм. Обчислює кількість лінійно незалежних шляхів в алгоритмі роботи програми на основі її вихідних текстів. Зазвичай чим вище цей показник тим більший відсоток допущення помилки при виконанні і тим більша загальна складність додатку.

Середня важкість за методом Холстеда розраховується як:

$$D = \frac{\eta_1}{2} * \frac{N_2}{\eta_2} \tag{3.1}$$

Де η_1 - кількість різноманітних операторів, η_2 - кількість різноманітних операндів, N_2 - загальна кількість операндів.

З результатів значень метрик були отримані наступні висновки. Вихідний код Angular є найбільш документованим, має високу вкладеність блоків - 10 і саму високу цикломатичну складність (тобто кількість шляхів виконання програми) в своєму коді, що сильно ускладнює його підтримку. Але величина середньої важкості функцій згідно методу Холстеда є досить низькою в

порівнянні з іншими фреймворками. В той час Vue містить в своєму коді найменший відсоток рядків коментарів, що говорить про низьку документованість вихідного коду. Найнижчі показники цикломатичної складності і важкості Холстеда має React, що покращує його підтримку і налагодження проектів, заснованих на ньому.

React і Vue містять найбільшу кількість попереджень, що робить їх менш надійними ніж Angular проте критичні ситуації на практиці зустрічаються досить рідко.

Для проведення тестування продуктивності розглянутих фреймворків необхідно визначитися з предметом оцінки. Найбільш коректним буде використання веб-додатків, що мають абсолютно ідентичну функціональність, але реалізованих за допомогою різних фреймворків і мінімальною кількістю сторонніх бібліотек - лише тих, без яких практичне застосування фреймворків неможливо [4]. Для цього було реалізовано додаток для управління списком справ на трьох досліджуваних фреймворках.

Основним інструментом вимірювань є інструменти розробника браузера Chrome. Результати тестування продуктивності представлені в табл. 3.3.

Таблиця 3.3

Показники продуктивності фреймворків

Показник	Angular	React	Vue
Середній час завантаження веб-додатку, мс	640,88	475,1	266,49

З результатів проведення тестування продуктивності були отримані наступні висновки. Найкращі показники продуктивності демонструють Vue і React, в той час як Angular є найбільш повільним з розглянутих фреймворків. Дані результати безпосередньо корелюють з показниками метрик ПЗ, розрахованими раніше.

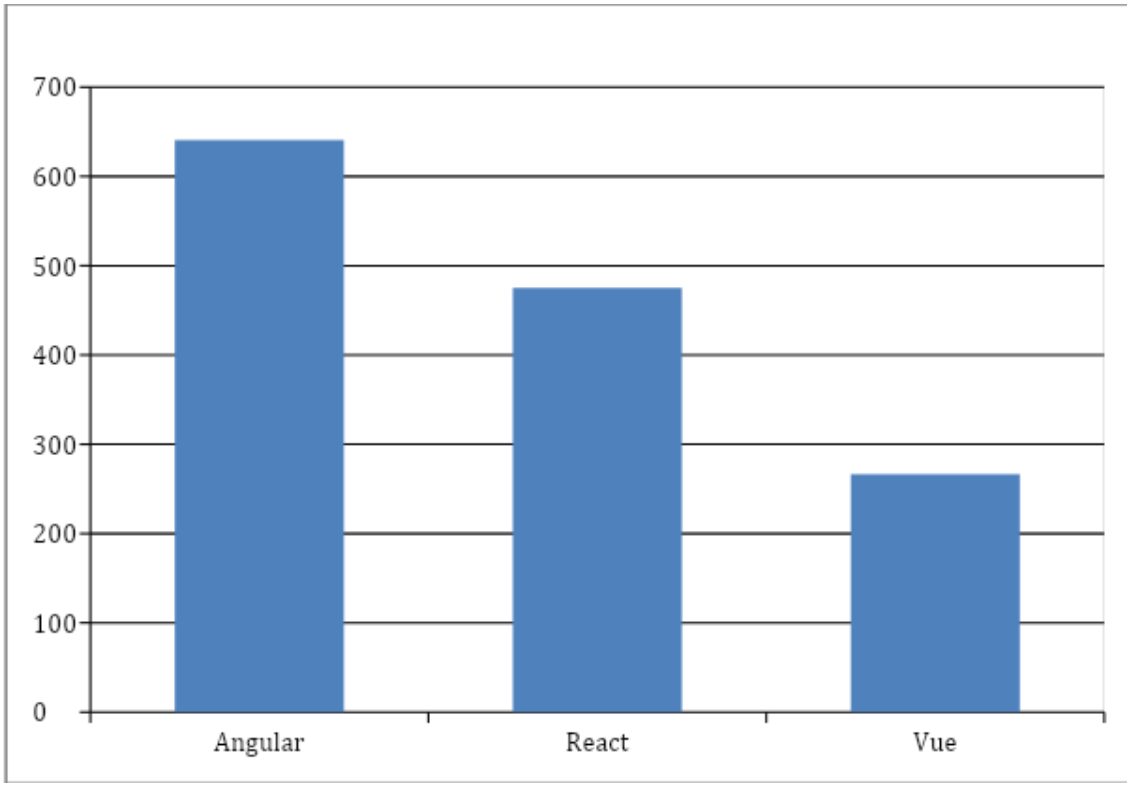

Рис. 3.6. Діаграма швидкості завантаження фреймворків

Для поєднання даних дослідження можна створити порівняльну таблицю (табл. 3.4), що буде містити всі досліджувані показники та їх відносну оцінку для кожного фреймворку за принципом від 1 до N, де 1 - найменш підходящий варіант, а N - найбільш підходящий варіант і дорівнює кількості досліджуваних об’єктів.

Таблиця 3.4

Порівняльна характеристика фреймворків

Показник	Angular	React	Vue
1	2	3	4
Величина проекту	3	2	1
Швидкість розробки	1	2	3
Попит на розробників	2	3	1

Продовження таблиці 3.4

1	2	3	4
Наявність спільноти	3	2	1
Тенденція розвитку	1	2	3
Середня важкість функцій згідно методу Холстеда	3	2	1
Цикломатична складність	1	3	2
Швидкість завантаження	1	2	3
Результат	15	18	15

Зібравши усі критерії можна побачити, що React є найбільш універсальним і лише небагато випереджає конкурентів. Проте немає такої технології, що змогла б явно перевершити конкурентів за усіма показниками. Із цього виходить, що для остаточного вибору найбільш підходящого фреймворка розробник повинен визначитися, який критерій буде мати більшу значимість в рамках конкретного проекту на підставі технічних вимог.

3.4. Висновки до третього розділу

У межах розділу було здійснено визначення найбільш важливих критеріїв проекту та зіставлено їх варіанти з досліджуваними фреймворками. Також було розглянуто статистичні дані опитувань серед розробників для визначення тенденцій розвитку технологій, а також географічний аспект тенденцій їх використання.

Також був проведений практичний аналіз вихідного коду фреймворків за числовими показниками якості та ефективності з використанням методу Холстеда та аналізом цикломатичної складності. Результатами дослідження є порівняльна таблиця характеристик фреймворків для найбільш важливих показників.

РОЗДІЛ 4
ЕКОНОМІЧНИЙ РОЗДІЛ

4.1. Визначення трудомісткості розробки програмного забезпечення

Початкові дані:

1) Годинна заробітна плата програміста, грн./год – 100.
2) Вартість машино-години ЕОМ, грн./год – 20.
3) Передбачуване число операторів – 1100.
4) Коефіцієнт складності програми – 1,5.
5) Коефіцієнт корекції програми в ході її розробки – 0,1.
6) Коефіцієнт збільшення витрат праці внаслідок неповного опису завдання – 1,2.
7) Коефіцієнт кваліфікації програміста, який визначається в залежності від стажу роботи за фахом – 1,2.
8) Витрати праці на підготовку і опис поставленого завдання – 50 год.

Трудомісткість розраховується за формулою:

$$t = t_{\text{и}} + t_{\text{а}} + t_{n} + t_{\text{отл}} + t_{\text{д}}$$

t_o – витрати праці на підготовку і опис поставленого завдання;

t_u – витрати праці на дослідження алгоритму вирішення задач;

t_a – витрати праці на розробку блок-схем алгоритму;

$t_п$ – витрати праці на програмування за готовою блок-схемою;

$t_{відл}$ – витрати праці на відладку програм на ПЕОМ;

$t_д$ – витрати праці на підготовку документації.

Сукупні витрати праці визначаються виходячи з умовного числа операторів у розроблюваному ПЗ.

Розрахунок очікуваних витрат праці:

$$Q = q \cdot C \cdot (1 + p), \text{людино} - \text{годин},$$

де q – передбачуване число операторів;

С – коефіцієнт складності програми;

р – коефіцієнт корекції програми в ході її розробки.

$$Q = 1100 \cdot 1{,}5 \cdot (1 + 0{,}1) = 1815 \text{ людино} - \text{годин}$$

Витрати праці на вивчення опису завдання визначаються з урахуванням уточнення опису і кваліфікації програміста за формулою:

$$tu = \frac{QB}{(75\ldots85)K}$$

де В – коефіцієнт збільшення витрат праці внаслідок неповного опису завдання;

К – Коефіцієнт кваліфікації програміста, який визначається в залежності від стажу роботи за фахом.

Витрати праці на розробку алгоритму рішення задачі:

$$tu = \frac{1815 * 1{,}2}{80 * 1{,}2} = 22{,}7 \text{ людино} - \text{годин}$$

Витрати на складання блок-схеми алгоритму:

$$t_a = \frac{Q}{(20\ldots25)\ K} \text{ людино} - \text{годин},$$

$$t_a = \frac{1815}{20 * 1{,}2} = 75{,}6 \text{ людино} - \text{годин},$$

Витрати на складання програми по готовій блок-схемі:

$$t_n = \frac{Q}{(20\ldots25)\ K} \text{ людино} - \text{годин},$$

$$t_n = \frac{1815}{23 * 1{,}2} = 78{,}2 \text{ людино} - \text{годин}$$

Витрати праці на налагодження програми на ЕОМ:

$$t_{\text{отл}} \frac{Q}{(4\ldots5)\ K} \text{ людино} - \text{годин},$$

$$t_{\text{отл}} \frac{1815}{4 * 1{,}2} = 378 \text{ людино} - \text{годин}$$

Витрати праці на підготовку документації:

$$t_{\text{д}} = t_{\text{др}} + t_{\text{до}}\ , \text{людино} - \text{годин},$$

де $t_{\text{др}}$ - трудомісткість підготовки матеріалів і рукопису.

$$t_{\text{др}} = \frac{Q}{(15\ldots20)\ K} \text{ людино} - \text{годин},$$

$$t_{\text{др}} = \frac{1815}{17 * 1{,}2} = 89 \text{ людино} - \text{годин}$$

$t_{\text{до}}$ – трудомісткість редагування, печатки й оформлення документації

$$t_{\text{до}} = 0{,}75 \cdot t_{\text{др}}, \text{людино} - \text{годин}$$

$$t_{\text{до}} = 0{,}75 \cdot 89 = 66{,}75 \text{ людино} - \text{годин}$$

$$t_д = 89 + 66.75 = 155,75 \text{ людино} - \text{годин}$$

Отримуємо трудомісткість розробки програмного забезпечення:

t = 1815 +22,7 + 75,8 +78,2 + 378 + 155,75 = 2520,35 людино – годин.

4.2. Розрахунок витрат на створення програмного забезпечення

Витрати на створення програмного забезпечення $К_{пз}$ включають витрати на заробітну плату розробників програми ($З_{зп}$) і витрати машинного часу, необхідного для налагодження програми на ЕОМ ($З_{мв}$).

$$К_{по} = З_{зп} + З_{мв}$$

$$З_{зп} = t\, C_{пр}$$

де t – загальна трудомісткість розробки ПЗ;

$С_{пр}$ – середня годинна заробітна плата програміста

$$З_{зп} = 2520,35 * 100 = 252035 \text{ грн.}$$

Витрати машинного часу, необхідного для налагодження програми на ЕОМ ($З_{мв}$) визначаються за формулою:

$$З_{мв} = t_{відл}\, C_{мч}$$

де $t_{відл}$ – трудомісткість налагодження програми на ЕОМ;

$С_{мч}$ – вартість машино-години ЕОМ.

$$З_{мв} = 378 * 20 = 7560 \text{грн.}$$

Таким чином витрати на створення програмного забезпечення, складуть:

Кпо = 756105 + 180900= 775005 грн.

$$К_{по} = 7560 + 252035 = 259595 \text{ грн.}$$

Очікувана тривалість розробки ПЗ:

$$T = \frac{t}{B_k \cdot F_p}$$

де B_k – число розробників;

F_p– місячний фонд робочого часу (при 40-ка годинному робочому тижні Fp = 176 годин).

$$T = \frac{2520{,}35}{1 \cdot 176} = 14 \text{ місяців}$$

4.3. Маркетингові дослідження

У наш час можна досить впевнено сказати, що такий показник як швидкість завантаження веб-сторінки дуже тісно пов’язаний з маркетинговим успіхом будь-якого продукту, адже він напряму впливає на конверсію (відношення загальної кількості користувачів до тієї кількості, що виконали бажану дію), кількість відмов та відвідуваність

Використання в проекті фреймворків дозволяє зменшити час завантаження додатку, що позитивно впливає на статистику його відвідування користувачами, а також зменшити кількість переходів між сторінками за рахунок динамічного відображення контенту та механізмам повільного фонового завантаження (lazy loading), що дозволяють завантажувати лише ту частину додатку, на якій безпосередньо знаходиться користувач.

4.4. Економічна ефективність

Оскільки в даній роботі розробка програмного забезпечення проводилась не з комерційних цілей, а виключно для проведення дослідження - розрахунок таких показників, як економічна ефективність, обсяг інвестицій, термін окупності та очікуємий прибуток не є можливим.

Проте можна відзначити, що використання фреймворків для реалізації проектів майже завжди несе позитивний економічний ефект, адже це дозволяє прискорити процес розробки, раніше завершити продукт і як результат зменшити термін окупності. Гнучка архітектура фреймворків також дозволяє досить легко розширювати продукт, зменшуючи фінансові та часові витрати.

ВИСНОВКИ

У процесі дослідження здійснено аналіз і наведено порівняльну характеристику front-end фреймворків. На основі вищевикладеного варто зробити наступний висновок:

Фреймворк – сукупність рішень по архітектурі, структурі і способам об'єднання компонентів системи, які можуть бути застосовані для безлічі однотипних завдань.

В області програмування під фреймворком розуміють безліч класів і способів їх взаємодії. Front-end фреймворки пов'язані із зовнішньою частиною програми. Простими словами, вони відповідають за зовнішній вигляд програми. Back-end відповідає за внутрішній устрій додатку. Розглянемо обидва типи детальніше. Основоположною характеристикою фреймворка є так-звана інверсія управління. Зазвичай фреймворк грає головну роль і викликає код самого додатка. Потік управління інвертується тут – фреймворк викликає додаток, а не навпаки. Головним показником корисності фреймворка є те, що він допомагає розробнику підвищити продуктивність і якість коду. Наприклад, сучасні фреймворки часто надають інструменти для кодогенерації або готові каркаси типових додатків. Так само добре продуманий фреймворк задіє всі заходи безпеки, допомагаючи розробнику писати більш захищені додатки.

Додатковою перевагою використання фреймворка у великих командах розробників є те, що він допомагає слідувати єдиним стандартам. У кожного фреймворка є загальноприйняті рішення типових задач, стиль написання коду і готові реалізації часто використовуваних інструментів. Це дозволяє зберегти єдиний стиль програмування у всіх частинах програми.

Фреймворки для веб-розробки багато в чому схожі, навіть якщо реалізовані на різних мовах програмування. Це не дивно, адже вони вирішують одні й ті ж завдання. Проте, кожен з перерахованих фреймворків індивідуальний. У них різні підходи, методи і поведінка в розробці.

React.js – один з кращих і найбільш поширених способів створити односторінковий додаток. Це гнучкий і зручний фреймворк, за допомогою якого можна або додати компонент на існуючий сайт, або створити новий сайт з нуля. React-розробка полягає в описі того, що потрібно вивести на сторінку (а не в складанні інструкцій для браузера, присвячених тому, як це робити). Це, крім іншого, означає значне скорочення обсягів шаблонного коду.

AngularJS створений для тих розробників, які вважають, що декларативний стиль краще підходить для створення UI, а імперативний для написання бізнес-логіки.

Виділивши характерні риси для кожної технології можна зробити висновок що майже всі фреймворки є сенс застосовувати тільки коли проект досить великий, адже їх початкова конфігурація потребує певних витрат часу, а архітектури спрямовані на багаторазове використання окремих фрагментів. Для формальної оцінки якості програмного забезпечення (ПЗ) використовуються метрики – чисельні заходи, які відображають певні властивості програмного продукту або його специфікації.

В роботі проведено порівняльний аналіз продуктивності, якості і валідності коду трьох найбільш поширених front-end фреймворків: React, Angular, Vue. Вибір заснований на інформації, розташованої на офіційних github-репозиторіях даних продуктів.

Для тесту продуктивності використані мініфіковані версії вихідних кодів фреймворків, в той час як для оцінки якості і валідності коду використані версії для розробки.

З результатів значень метрик були отримані наступні висновки. Вихідний код Angular є найбільш документованим, має високу вкладеність блоків - 10 і саму високу цикломатичну складність (тобто кількість шляхів виконання програми) в своєму коді, що сильно ускладнює його підтримку. Але величина середньої важкості функцій згідно методу Холстеда є досить низькою в порівнянні з іншими фреймворками. В той час Vue містить в своєму коді найменший відсоток рядків коментарів, що говорить про низьку

документованість вихідного коду. Найнижчі показники цикломатичної складності і важкості Холстеда має React, що покращує його підтримку і налагодження проектів, заснованих на ньому. React і Vue містять найбільшу кількість попереджень, що робить їх менш надійними ніж Angular проте критичні ситуації на практиці зустрічаються досить рідко.

Для проведення тестування продуктивності розглянутих фреймворків необхідно визначитися з предметом оцінки. Найбільш коректним буде використання веб-додатків, що мають абсолютно ідентичну функціональність, але реалізованих за допомогою різних фреймворків і мінімальною кількістю сторонніх бібліотек - лише тих, без яких практичне застосування фреймворків неможливо. З результатів проведення тестування продуктивності були отримані наступні висновки. Найкращі показники продуктивності демонструють Vue і React, в той час як Angular є найбільш повільним з розглянутих фреймворків. Дані результати безпосередньо корелюють з показниками метрик ПЗ, розрахованими раніше.

Використання в проекті фреймворків дозволяє зменшити час завантаження додатку, що позитивно впливає на статистику його відвідування користувачами, а також зменшити кількість переходів між сторінками за рахунок динамічного відображення контенту та механізмам повільного фонового завантаження (lazy loading), що дозволяють завантажувати лише ту частину додатку, на якій безпосередньо знаходиться користувач.

Можна також відзначити, що реалізація проектів з використанням фреймворків майже завжди несе позитивний економічний ефект, адже це дозволяє прискорити процес розробки, раніше завершити продукт і як результат зменшити термін окупності. Гнучка архітектура фреймворків також дозволяє досить легко розширювати продукт, зменшуючи фінансові та часові витрати.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Rakic G., Budimac Z. Problems in Systematic Application of Software Metrics and Possible Solution [Электронный ресурс]. – Режим доступа: https://arxiv.org/ftp/arxiv/papers/1311/1311.3852.pdf, своб.

2. Gizas A., Christodoulou S., Papatheodorou T. Comparative Evaluation of JavaScript Frameworks // Proceedings of the 21st Annual Conference on World Wide Web Companion. – 2012. – P. 513–514.

3. Barkmann H., Lincke R., Löwe W. Quantitative Evaluation of Software Quality Metrics in OpenSource Projects // Proceedings of International Conference on Advanced Information Networking and Applications. – 2009. – P. 1067–1072.

4. Graziotin D., Abrahamsson P. Making Sense out of a Jungle of JavaScript Frameworks, Towards a Practitioner-friendly Comparative Analysis // Lecture Notes in Computer Science. – 2014. – P. 334– 337.

5. Бодров М.Ю. Веб-приложения как эволюционное развитие Web // Сб. научных трудов по материалам III Международной научно-практической конференции «Теоретические и прикладные аспекты современной науки». – 2014. – № 1. – С. 18–21.

6. Digital, Social and Mobile in 2015 report indicates [Електронний ресурс] / URL: http://wearesocial.com/sg/special-reports/digital-socialmobile-2015.

7. Веб-фреймворки и с чем их едят [Електронний ресурс] / URL: http://iwsm.ru/blog/show/veb-freymvorki-i-s-chem-ih-edyat.

8. Фреймворки в веб-разработке [Електронний ресурс] / URL: https://web-creator.ru/articles/about_frameworks.

9. Результаты тестирования шести ведущих фреймворков на производительность [Електронний ресурс] / URL: http://www.alrond.com/ru/2007/jan/25/rezultaty-testirovanija-6-frameworks/.

10. Адаптивные CSS-фреймворки, сетки, классы видимости [Електронний ресурс] / URL: http://klondike-studio.ru/blog/responsive-cssframework/.

11. Обзор CSS-фреймворков [Електронний ресурс] / URL: http://iantonov.me/page/obzor-css-frejmvorkov.

12. Використання PHP фреймворків в розробці сайту [Електронний ресурс] / URL: http://ukrbukva.net/page,5,39718-Ispol-zovanie-PHPfreiymvorkov-v-razrabotke-saiyta.html.

13. Сравнение каркасов веб-приложений [Електронний ресурс] / URL: https://ru.wikipedia.org/wiki/Сравнение_каркасов_вебприложений.

14. Обзоры Web-фреймворков [Електронний ресурс] / URL: https://praktikatech.wordpress.com/category/обзоры-web-фреймворков/

15. Каркас веб-приложений [Електронний ресурс] / URL: https://ru.wikipedia.org/wiki/Каркас_веб-приложений.

16. Что такое фреймворк? [Електронний ресурс] / URL: http://www.dbhelp.ru/what-is-framework/page/.

17. Десять причин избегать тяжеловесных фреймворков, а также лишних зависимостей в проекте [Електронний ресурс] / URL: http://eax.me/avoid-frameworks/.

18. Ruby on Rails [Електронний ресурс] / URL: http://www.rubyonrails.ru/.

19. What is a Web Framework? [Електронний ресурс] / URL: https://jeffknupp.com/blog/2014/03/03/what-is-a-web-framework/.

20. Популярные frontend и backend фреймворки [Електронний ресурс] / URL: http://web-diz.com.ua/poleznosti/populyarnye-frontendi-backend-freymvorki/.

21. О фреймворках [Електронний ресурс] / URL: http://web-elive.com/stati/raznoe/o-frejmvorkax/.

22. Веллинг Л. Разработка веб-приложений с помощью PHP и MySQL / Л. Веллинг, Л. Томсон. – Москва: Вильямс, 2010. – 848 с.

23. Шлосснейгл Д. Профессиональное программирование на PHP / Джордж Шлосснейгл. – Москва: Вильямс, 2006. – 624 с.

24. Кузнецов М. В. PHP 5 на примерах / М. В. Кузнецов, И. В. Симдянов, С. 103 В. Голышев. – Санкт-Петербург: БХВ-Петербург., 2005. – 576 с.

25. Кухарчик А. С. PHP: обучение на примерах / А.С.Кухарчик. – Минск: Новое знание, 2004. – 240 с

26. Аткинсон Л. PHP 5. Библиотека профессионала / Л. Аткинсон, З. Сураски. – Москва: Вильямс, 2006. – 944 с.

27. Мазуркевич А. М. PHP: Настольная книга программиста / А. М. Мазуркевич, Д. С. Еловой. – Минск: Новое знание, 2004. – 480 с.

28. Суэринг С. PHP и MySQL. Библия программиста / С. Суэринг, Д. Парк, Т. Конверс. – Москва: Вильямс, 2010. – 912 с.

29. Дронов В. А. PHP 5/6, MySQL 5/6 и Dreamweaver CS4. Разработка интерактивных Web-сайтов / Владимир Александрович Дронов. – Санкт Петербург: БХВ-Петербург., 2009. – 544 с.

30. Angular: материал из Википедии – свободной энциклопедии: [Електронний ресурс] / URL: https://uk.wikipedia.org/wiki/Angular

31. React: материал из Википедии – свободной энциклопедии: [Електронний ресурс] / URL: https://uk.wikipedia.org/wiki/React

32. Vue.js: материал из Википедии – свободной энциклопедии: [Електронний ресурс] / URL: https://uk.wikipedia.org/wiki/Vue.js

33. Ревзин В. А. Комплексная автоматизация проектных организаций: цели, условия, результаты [Електронний ресурс] / URL: https://issuu.com/cadmaster/ docs/cadmaster-2005.4-29. (дата звернення 14.11.2020).

34. Google Trends [Електронний ресурс] URL: https://trends.google.com/trends/?geo=US. (дата звернення 14.11.2020).

35. State Of JS - Yearly JavaScript Survey [Електронний ресурс] / URL: https://2019.stateofjs.com/front-end-frameworks/. (дата звернення 14.11.2020).

36. Веллинг Л., Томсон Л. Разработка веб приложений с помощью PHP и MySQL. Москва: Вильямс, 2010. 848 с. 78 Вісник ХНАДУ, вип. 87, 2019

37. Шлосснейгл Д. Профессиональное программирование на PHP. Москва: Вильямс. 2006. 624 с.

38. Кухарчик А. С., Еловой Д. С. PHP: обучение на примерах. Минск: Новое знание, 2004. 240 с.

39. Мазуркевич А. М. PHP: Настольная книга программиста. Минск: Новое знание, 2004. 480 с.

40. Фреймворки в веб-разработке [Електронний ресурс] / URL: https://web-creator.ru/articles/about_frameworks. (дата звернення 09.12.2020).

41. Використання PHP фреймворків в розробці сайту [Електронний ресурс] / URL: http://ukrbukva.net/page,5,39718- Ispol-zovanie-PHP-freiymvorkov-v-razrabotkesaiyta.html. (дата звернення 14.11.2020).

42. Гардейчик С. М., Шербаф А. И. Программная платформа LARAVEL для создания вебориентированных приложений и сервисов Вести БДПУ. Серия 3. 2017. № 3. С. 82–90.

43. Гуренко В. В., Бородин А. Ф., Назарков Д. А. Сравнительный анализ фреймворков для вебразработки. Технологии инженерных и информационных систем. 2017. № 2. С. 3–14.

44. Lockhart J. Modern PHP. New Features and Good Practices. O'Reilly Media, 2015, 268 с.

45. Егорова И.Н., Михно Е.В., Куковской А.А. Основные тенденции разработки сайтов учебных заведений. Восточно-Европейский журнал передовых технологий. 2008. Т. 2. № 2 (32). С. 9–12.

46. Performance benchmark of popular PHPframeworks [Електронний ресурс] / URL: https://systemsarchitect.net/2013/04/23/ performance-benchmark-of-popular-phpframeworks/ (дата звернення 14.11.2020)

47. Лучшие PHP-фреймворки для использования в 2019 году [Електронний ресурс] URL: https://techrocks.ru/2019/07/23/best-phpframeworks-2019/ (дата звернення 14.11.2020).

48. Пономарев А. С. Нечеткие множества в задачах автоматизированного управления и принятия решений: учеб. пособие. Х: НТУ ХПИ, 2005. 232 с.

49. Раскин, Л. Г., Серая О. В. Нечеткая математика. Основы теории. Приложения. Х.: Парус, 2008. – 352 с.

ДОДАТОК А

ЛІСТИНГ ПРОГРАМИ

Реалізація додатку за допомогою фреймворку React

```
import React, { useState } from 'react';

const initialTasks = [
  { content: 'Pass standards compliance check', isDone: false },
  { content: 'Defend master degree diploma', isDone: false },
];

const ToDoItem = ({ todo = {}, onToggle }) => {
  return (
    <li>
      <input type="checkbox" checked={todo.isDone} onChange={onToggle} />
      <span>{todo.content}</span>
    </li>
  );
};

const ToDoList = ({ todos = [], onToggle }) => {
  return todos.map((todo, index) => (
    <ToDoItem
      key={index}
      todo={todo}
      onToggle={() => onToggle(index)}
    />
  ));
};

const App = () => {
  const [todos, setTodos] = useState(initialTasks);
  const [newTodoContent, setNewTodoContent] = useState('');
  const [todosCount, setTodosCount] = useState(initialTasks.length);

  const onToggle = (index) => {
    const updatedTodos = [...todos];

    updatedTodos[index].isDone = !updatedTodos[index].isDone;
    setTodos(updatedTodos);
  };

  const onChange = ({ target = { value: '' } }) => {
    setNewTodoContent(target.value);
  };

  const onAdd = () => {
    setTodos([
      ...todos,
      { content: newTodoContent, isDone: false }
    ]);
    setTodosCount(prevCount => prevCount + 1);
    setNewTodoContent('');
  };


  const onClearCompleted = () => {
```

```
    const filteredTodos = todos.filter(todo => !todo.isDone);

    setTodos(filteredTodos);
    setTodosCount(filteredTodos.length);
  };

  return (
    <div>
      <h2>You've got <span className="emphasis">{todosCount}</span> things to
do</h2>
      <ul>
        <ToDoList todos={todos} onToggle={onToggle} />
      </ul>
      <input
        placeholder="I need to..."
        type="text"
        value={newTodoContent}
        onChange={onChange}
      />
      <button onClick={onAdd}><h2>Add</h2></button>
      <p>
        <button onClick={onClearCompleted}>Clear completed</button>
      </p>
    </div>
  );
};

export default App;


import React from 'react';
import ReactDOM from 'react-dom';
import App from './App';

ReactDOM.render(<App />, document.getElementById("root"));
```

Реалізація додатку за допомогою фреймворку Angular

```
const initialTasks = [
  { content: 'Pass standards compliance check', isDone: false },
  { content: 'Defend master degree diploma', isDone: false },
];

const TodoController = ($scope) => {
  $scope.todos = initialTasks;

  $scope.getTotalCount = () => $scope.todos.length;

  $scope.onAdd = () => {
    $scope.todos.push({ content: $scope.newTodoContent, isDone: false });
    $scope.newTodoContent = '';
  };




  $scope.onClearCompleted = () => {
    $scope.todos = $scope.todos.filter(todo => !todo.isDone);
  };
}
```

```
angular
  .module('todoApp', [])
  .controller('TodoController', TodoController);


<!doctype html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport"
        content="width=device-width, user-scalable=no, initial-scale=1.0,
maximum-scale=1.0, minimum-scale=1.0">
  <meta http-equiv="X-UA-Compatible" content="ie=edge">
  <script src="src/angular.development.js"></script>
  <script src="src/main.js"></script>
  <title>AngularJS App</title>
</head>
<body>
  <div ng-app="todoApp" ng-controller="TodoController">
    <h2>You've got <span class="emphasis">{{getTotalCount()}}</span> things to
do</h2>
    <ul>
      <li ng-repeat="todo in todos">
        <input type="checkbox" ng-model="todo.isDone"/>
        <span>{{todo.content}}</span>
      </li>
    </ul>
    <input
      placeholder="I need to..."
      type="text"
      ng-model="newTodoContent"
      ng-model-instant
    />
    <button ng-click="onAdd()">Add</button>
    <p>
      <button ng-click="onClearCompleted()">Clear completed</button>
    </p>
  </div>
</body>
</html>
```

Реалізація додатку за допомогою фреймворку Vue

```
new Vue({
  el: "#app",
  data: {
    newTodoContent: '',
    todos: [
      {
        content: 'Pass standards compliance check',
        isDone: false
      },
      {
        content: 'Defend master degree diploma',
        isDone: false
      }
    ],
  },
  methods: {
    onAdd: function() {
```

```
      this.todos.push({
        content: this.newTodoContent,
        isDone: false,
      });
      this.newTodoContent = '';
    },
    onToggle: function(todo) {
      todo.isDone = !todo.isDone;
    },
    onClearCompleted: function() {
      this.todos = this.todos.filter(todo => !todo.isDone);
    },
  },
  computed: {
    totalCount: function() { return this.todos ? this.todos.length : 0 },
  }
});
```

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport"
        content="width=device-width, user-scalable=no, initial-scale=1.0,
maximum-scale=1.0, minimum-scale=1.0">
  <meta http-equiv="X-UA-Compatible" content="ie=edge">
  <script src="src/vue.development.js"></script>
  <title>Vue App</title>
</head>
<body>
  <div id="app">
    <h2>You've got <span class="emphasis" v-if="totalCount">{{ totalCount
}}</span> things to do</h2>
    <ul>
      <li class="list-item" v-for="todo in todos">
        <div @click="onToggle(todo)">
          <input type="checkbox" />
          <span>{{ todo.content }}</span>
        </div>
      </li>
    </ul>
    <input
        v-model="newTodoContent"
        placeholder="I need to..."
        type="text"
    />
    <button @click="onAdd">Add</button>
    <p>
      <button @click="onClearCompleted">Clear completed</button>
    </p>
  </div>
  <script src="src/main.js"></script>
</body>
</html>
```

ДОДАТОК Б

ВІДГУК
керівника економічного розділу
на кваліфікаційну роботу магістра
на тему:
«Методи, алгоритми та програмне забезпечення для дослідження ефективності фреймворків згідно з потребами проекту»
студента групи 121м-19-1 Степанова Олександра Борисовича

Керівник економічного розділу
доцент каф. ПЕП та ПУ, к.е.н. **Л. В. Касьяненко**

ДОДАТОК В

ПЕРЕЛІК ДОКУМЕНТІВ НА ОПТИЧНОМУ НОСІЇ

Ім’я файла	**Опис**
Пояснювальні документи	
Диплом_Степанов.docx	Пояснювальна записка до магістерської роботи. Документ Word.
Диплом_Степанов.pdf	Пояснювальна записка до магістерської роботи в форматі PDF.
Програма	
Program.rar	Архів. Містить вихідні коди додатку.
Презентація	
Презентація_Степанов.ppt	Презентація магістерської роботи.