МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ РАДІОЕЛЕКТРОНІКИ

ЗВІТ
З НАУКОВО-ДОСЛІДНОЇ ПРАКТИКИ МАГІСТРІВ
Місце проходження практики кафедра Програмної інженерії ХНУРЕ
у період з 25.01.2022 по 26.03.2022 р.

Тема індивідуального завдання:
Дослідження методів аналізу ефективності використання фремворків React та Vue при розв'язанні задач планування часу

Ст. ІПЗздм-20-2
Овсянников М.Ю.

Керівник практики
доц. Лановий О.Ф.
Робота захищена з оцінкою
_____
«__»_____2022 р.

Керівник атестаційної роботи
доц. Лановий О.Ф.

_____
(рекомендована оцінка)

_____
(підпис)

Комісія:
проф. Шубін І.Ю.
доц. Лановий О.Ф.
доц. Назаров О.С.

Харків 2022

РЕФЕРАТ / ABSTRACT

Звіт з науково-дослідної практики магістрів: 34 с., 11 рис., 14 джерел.

HTTP, JAVASCRIPT, JS, ANGULAR, REACR, VUE, ВЕБ-ДОДАТОК, ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ, ФРЕЙМВОРК, ЕФЕКТИВНІСТЬ

Об’єкт – дослідження методів використання фреймворків при розробці веб-орієнтованих додатків для розв'язанні задач планування часу.

Мета роботи – порівняти ефективність застосування різних JavaScript-фреймворків для розв’язання одного класу задач на підставі визначених метрих ефефктивності.

Методи дослідження. У роботі використовувалися різні методи застосування JavaScript-фреймворків, моделювання для дослідження ефективності веб-орієнтованих додаків.

HTTP, JAVASCRIPT, JS, ANGULAR, REACR, VUE, WEB APPLICATION, SOFTWARE, FRAMEWORK, EFFICIENCY

Object – study of methods of using frameworks in the development of web-based applications for solving time planning problems.

The aim is to compare the effectiveness of using different JavaScript frameworks to solve one class of problems based on certain performance metrics.

Research methods. The paper used various methods of using JavaScript-frameworks, modeling to study the effectiveness of web-based applications.

ЗМІСТ

ВСТУП 4
1 ОПИС ПІДПРИЄМСТВА 6
1.1 Загальні відомості про кафедру Програмної інженерії 6
1.2 Опис організаційної структури 7
2 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ 9
2.1 Концепції розробки веб-орієнтованих програмних систем 9
2.2 Шаблони веб-орієнтованих систем 12
2.3 Можливі шляхи вирішення задачі 17
2.4 Постановка задачі 20
3 ДОСЛІДЖЕННЯ МЕТОДІВ РОЗВ'ЯЗАННЯ ЗАДАЧІ 22
3.1 Аналіз та порівняння поширених фреймворків JavaScript 22
3.1.1 React 24
3.1.2 Angular 2+ 26
3.1.3 Vue 27
3.2 Метрики продуктивності 29
ВИСНОВКИ 32
ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ 33

ВСТУП

На теперішній час стрімке зростання кількості засобів розробки програмних систем досить суттєво ускладнює життя програмістів, оскільки вимушує їх постійно знаходитись у пошуку нових та більш ефективних засобів розробки. Збільшення кількості бібліотек та фреймворків JavaScript, що були розроблені останнім часом, вимагає від розробників порівнювати їх ефективність. Розробникам не вистачає результатів порівняльного аналізу таких бібліотек між собою, а також – відсутність стандартизованого процесу їх оцінювання для отримання впевненості, що їх структура відповідає потребам та масштабу проекту.

За останні роки JavaScript стає однією з основних мов програмування на веб-сайтах, що призвело до зростання його популярності та інтересу до його вивчення, коли йдеться про використання його фреймворків та бібліотек. Привабливість JavaScript обумовлена тим, що пропонує безліч фреймворків та бібліотек, більшість з яких створено самими розробниками. Використання фреймворків та бібліотек суттєво спрощує процес розробки клієнтських веб-додатків завдяки чистому та структурованому коду.

В рамках проходження науково-дослідної практики було проведено дослідження методів порівняння і оцінювання різних платформ та бібліотек, що дозволить визначити, яка з них більш відповідає їх потребам та може бути більш ефективною для практичної реалізації проекту. Також такий підхід дозволить більш ефективно розробляти структуру проекту, враховувати існуючий досвід та переваги при розробці програмних систем.

Дослідження, що були проведені під час проходження науково-дослідної практики, базуються на порівняльному аналізі ефективності додатків, призначених для планування часу (Todo-списки), розробку яких було виконано з використанням фремворків React та Vue. Під час дослідження фреймворк та бібліотеки оцінювались за декількома параметрами, до яких відносяться якість

документації, якості підтримки, частоти оновлення тощо. В результаті дослідження було виявлено домінуючі фактори для Vue та React, що стало підґрунтям для проведення порівняльного аналізу з метою виявлення відмінностей та подібностей між ними. В результаті були виявлені домінуючі фактори, а саме: React має детальну документацію, значну підтримку з боку спільноти, гнучкість у використанні зовнішніх бібліотек та може використовуватись як професіоналами, так і новачками у розробці JavaScript-додатків. На відміну від нього, Vue вимагає дотримання чітких правил використання цього фреймворку, що є досить ефективним у першу чергу для новачків.

Запропонований в результаті проходження науково-дослідної практики процес оцінювання дає інструментарій для виявлення розбіжностей між структурами, які розроблено з використанням різних інструментів програмування. Дослідження вмісту різних фреймворків вимагає від програмістів, щоб вони використовували їх у правильних умовах та цілях. Знаючи про відмінності фреймворків не буде витрачатись час на адаптацію до відповідного фреймворку або бібліотеки, які можуть не відповідати критеріям проекту. Ще однією з переваг є опис факторів, які впливають на фреймворк, та той факт, що розробник повинен мати уявлення про рівень необхідних знань та складність фреймворку, перш ніж обрати його.

Таким чином можна сказати, що метою цієї науково-дослідної роботи є дослідження методів аналізу ефективності використання фремворків React та Vue при розв'язанні задач планування часу.

.

1 ОПИС ПІДПРИЄМСТВА

1.1 Загальні відомості про кафедру Програмної інженерії

Кафедра програмної інженерії ХНУРЕ була заснована у 1963 році. На той час вона мала назву «кафедра Обчислювальної техніки». З 2011 року кафедра отримала назву Програмної інженерії. На сьогодні кафедра займається як науковою діяльністю, так і підготовкою бакалаврів за освітньо-професійною програмою «Програмна інженерія», магістрів зі спеціальності «Інженерія програмного забезпечення». Освітньо-професійна програма першого (бакалаврського) рівня вищої освіти «Програмна інженерія» - єдина освітня програма в ХНУРЕ, яка повністю відповідає європейському стандарту Software Engineering.

За час свого існування на кафедрі підготовлено понад 500 кандидатів та докторів наук.

В галузі освіти та науки кафедра ПІ співпрацює з Ліннеус університетом м. Вакхо (Linnaeus University), Швеція, а також з Університетом Економіки (WSG) м. Бидгощ, Польща.

Основні наукові дослідження кафедри виконуються за напрямами: створення нового покоління обчислювальних систем і технологій; педагогіка в нових освітніх середовищах, дистанційне навчання; проектування систем штучного інтелекту; розробка математичних моделей механізмів людського інтелекту (зору, слуху, сприйняття, пізнавання тощо); розробка формального апарату методів логіки, алгебри, лінгвістичної алгебри і системи логічної підтримки проектування нових інформаційних технологій; сучасні технології інтеграції гетерогенних розподілених джерел даних і парадигма якості програмного забезпечення автоматизованих систем обробки інформації; програмні засоби автоматизованого формування інформаційного простору навчального процесу; інформаційні технології дистанційного навчання та електронної комерції; побудова, розробка і реалізації білінгових систем;

інтелектуальний аналіз даних; розвиток основ теорії сегментації та ідентифікації геометричних об’єктів у режимі реального часу для прикладних завдань обробки цифрової інформації; розробка моделей, методів і алгоритмів розпізнавання для біометричних систем; розробка підсистем аналізу зображень для системи обробки й аналізу технічної інформації в галузі медичного прогнозування; розробка моделі навчання і програмного середовища для проведення навчання та перевірки знань у довільній предметній галузі тощо.

1.2 Опис організаційної структури

На кафедрі Програмної інженерії працюють понад 50 викладачів, серед яких 6 докторів наук, 12 професорів, 27 доцентів, 33 кандидати наук.

Завідувач кафедри: Дудар Зоя Володимирівна, кандидат технічних наук, професор.

До складу кафедри входять 7 лабораторій.

Науково-навчальна дослідна лабораторія «Мозкоподібних систем». Науковий керівник – професор Шубін Ігор Юрійович. Лабораторія проводить наукові дослідження в галузі дискретної математики, штучного інтелекту та математичного моделювання інтелектуальних процесів.

Науково-навчальний дослідний центр «Математичне моделювання». Керівник лабораторії – професор Четвериков Григорій Григорович. Завданням лабораторії є здійснення науково-дослідних робіт у галузі розробки моделей та методів побудови логічних мереж для створення на їх засадах нових ЕОМ паралельної дії, методів, що дозволять збільшити швидкість обробки інформації.

Міжнародний науково-дослідний центр Математичної та прикладної лінгвістики. Завідувач МНДЦ Математичної та прикладної лінгвістики - професор Четвериков Григорій Григорович. Центр було утворено з метою розвитку фундаментальних та прикладних досліджень у галузі нових

інформаційних технологій, математичного моделювання, інформатизації та автоматизації навчального процесу, штучного інтелекту, для сприяння найбільш ефективному використанню науково-педагогічного потенціалу університету.

Науково-дослідна і навчальна лабораторія «Інформаційні технології в системах навчання і машинного зору». Завідуюча лабораторією – професор Білоус Наталія Валентинівна. До наукових інтересів належать такі напрями, як системи комп'ютерного зору, медичні та діагностичні автоматизовані системи, біометричні інформаційні системи, аналіз сцен і побудова 3D моделей, методології розробки, технології інтеграції та інтелектуального аналізу даних в корпоративних інформаційних системах, знання-орієнтовані технології класифікації, діагностики та прогнозування ситуацій, інтелектуальні системи безпеки, інтегровані системи комп'ютерного тестування знань і навчання.

Навчально-наукова лабораторія інтелектуальних програмно-апаратних систем (ІПАС). Керівник лабораторії – професор Єрохін Андрій Леонідович. Мета діяльності – дослідження сучасних інтелектуальних методів, моделей та засобів для створення програмно-апаратних систем з інтелектуальними функціями.

Науково-дослідна лабораторія програмного забезпечення автоматизованих систем. Керівник лабораторії – професор Дудар Зоя Володимирівна. Лабораторія проводить наукові дослідження в галузі інтелектуального аналізу даних, штучного інтелекту, математичного моделювання інтелектуальних процесів, Big Data, Data Science та розробку інтелектуальних програмних систем.

Навчально-наукова лабораторія розробки ігрових застосувань. Керівник лабораторії – старший викладач Новіков Юрій Сергійович. Лабораторія має чотири напрямки, серед яких дисциплінарні курси з області розробки ігор, допомога студентам в реалізації і супроводі власних ігрових проектів, вивчення і розробка систем штучного інтелекту в іграх та надання можливості ігровим компаніям вплинути на навчальний процес в сфері розробки ігор.

2 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

2.1 Концепції розробки веб-орієнтованих програмних систем

Сучасні технології розробки веб-додатків розвивалися дуже швидко протягом останніх кількох років. Традиційна модель веб-програми — це багатосторінкова програма, яка від початку домінувала у світі WWW. Одним з найважливіших недоліків традиційних веб-додатків (багатосторінкових додатків) є погана чутливість: коли користувач змінює сторінку, браузеру потрібен час, щоб отримати новий HTML-документ із сервера. Внутрішня обробка сервера також може тривати деякий час. На теперішній час користувацькі пристрої у світі WWW постійно мають більшу обчислювальну потужність і великий обсяг пам'яті. Завдяки цим фактам більшу частину логіки та обробки програм можна передати кінцевому пристрої, наприклад, настільному ПК або мобільному телефону. Це звільнить сервер від використання великої кількості ресурсів кожного клієнта. До цієї концепції краще підходить модель так званої односторінкової програми (SPA). Оскільки останнім часом швидкість передачі даних також покращилася, модель SPA пропонує значне поліпшення досвіду користувача. У SPA весь вміст програми завантажується відразу, тому початкове завантаження сторінки зазвичай довше, але останні зміни сторінки відбуваються миттєво.

Традиційна архітектура веб-додатків складається з клієнта та веб-сервера. Клієнтом може бути, наприклад, браузер на настільному ПК або мобільному пристрої. Клієнт відправляє запити HTTP на веб-сервер, який потім виконує деякі функції на основі запиту і, можливо, повертає деякий результат. На рис.2.1 зображено традиційну модель передачі даних веб-програми. У цій моделі клієнт браузера запитує веб-сторінку у веб-сервера, а веб-сервер, у свою чергу, запитує деякі дані з бази даних і, нарешті, повертає HTML-документ разом із стилями CSS в браузер користувача.

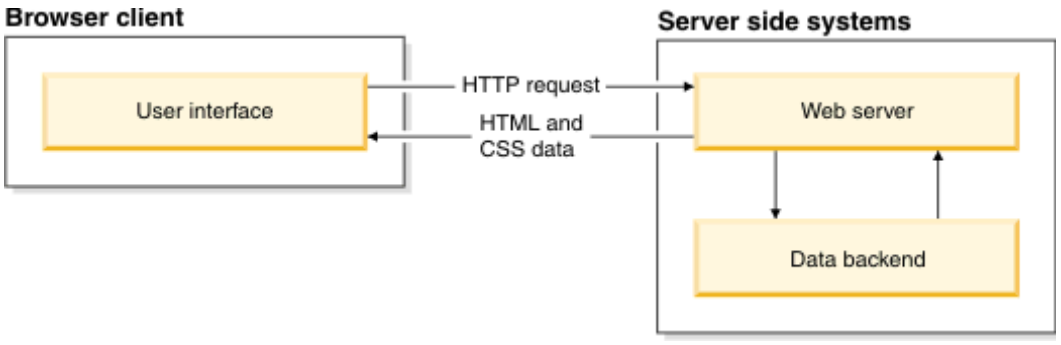

Рис.2.1 – Традиційна модель клієнт-серверної взаємодії

Багатосторінковий додаток зазвичай складається з декількох сторінок. Коли користувач натискає посилання на сторінці, він перенаправляє браузер на інший HTML-файл, який розміщено на сервері. Одночасно завантажуються файли таблиць стилів, якщо вони ще не збережені в кеші браузера. Крім повернення HTML-документів, сервер має традиційно й інші завдання. Сервер може автентифікувати користувача, вносити зміни до бази даних або, наприклад, надсилати електронні листи.

В архітектурі веб-додатків під маршрутизацією розуміють процес, в якому кожна окрема URL-адреса відповідає певній сторінці або вмісту. У моделі багатосторінкового додатка маршрутизація є частиною програми, оскільки сторінки також розділені у файловій системі. Використовуючи URL-адреси та маршрути, браузери можуть створювати закладки, ділитися посиланнями та переміщуватися веб-сайтам за допомогою кнопок «назад» та «вперед».

Деяка обробка може відбуватися і на боці клієнта. Обробка в браузері зазвичай використовує JavaScript і містить такі завдання, як зміна моделі DOM, створення інтерактивних візуалізацій або відображення попереджень на екрані. HTML DOM (об'єктна модель документа) – це стандартна деревоподібна структура або модель, яку браузер використовує для доступу до різних елементів. Існує багато видів налаштувань, але сама основа поділу задач зазвичай однакова у традиційних багатосторінкових веб-додатках.

У міру того, як односторінкові програми стають популярними і все більше обробки зосереджено на стороні клієнта, мови програмування зовнішнього

інтерфейсу (наприклад, JavaScript) також мають розвиватися. Середовище програм, що визначається як великий повторно використовуваний

Набір бібліотек для реалізації основної структури програми також стала тенденцією в розробці JavaScript. Фреймворки JavaScript мають розширити можливості розробника та спростити розробку JavaScript, надаючи готові оптимізовані функції для більш складних функцій. З погляду компанії, можна використовувати деяку структуру JavaScript поверх власного JavaScript, оскільки це прискорює процес розробки інтерфейсу у довгостроковій перспективі.

Поняття фреймворку тісно пов’язано з самим процесом розробки веб-орієнтованого програмного додатку – фактично це каркас веб-застосунку. З цієї причини розробник програмної системи повинен будувати її архітектуру відповідно до чітко визначеної логіки. Зазвичай середовище розробки надає відповідний інструментарій для реалізації окремих подій, побудови сховищ і з'єднання даних. Фактично, проводячи аналогії з машинобудуванням, майбутня машина безпосередньо залежить від принципів реалізації каркасу, кузову і двигуна. Змінюючи конфігурацію, існує можливість додавати, знімати або замінювати певні деталі за умови, що автомобіль залишається у працездатному стані.

Каркаси знаходяться на більш високому рівні абстракції, у порівнянні з бібліотеками, і дозволяють спростити процес розробки веб-орієнтованих програмних додатків.

Інструментальні засоби завжди значно спрощують процес розробки програм. Наприклад, досить ефективним є використання препроцесору Sass замість чистого CSS, оскільки він забезпечує значну кількість додаткових засобів програмування, не пов’язані безпосередньо зі стилями подання даних – це використання циклів, функцій, локальних змінних тощо. Разом з тим існує суттєва проблема – браузери не «розуміють» синтаксис Sass/SCSS, тому код необхідно перекласти на CSS.

Одним з найбільш популярних фреймворків з готовими стилями і скриптами є Bootstrap. Цей фреймворк вимагає перезаписати необхідні класи і

атрибути елементів HTML. Завдяки цьому досить легко за допомогою Bootstrap створювати мобільні проекти – це дозволяє досить легко налаштовувати будь-яку сторінку та відображати її на будь-якому пристрої. Особливість використання фреймворку Bootstrap полягає у тому, що це CSS framework – тобто набір файлів з готовим кодом, який підключається до html-сторінки в головному розділі, після чого може використовуватися елемент з цієї структури.

Аналогічно тому, як використовується Bootstrap, застосовують і JavaScript фреймворки – фактично це інструментарій для побудови значної кількості веб-орієнтованих, мобільних та настільних додатків на Javascript. Використання розробниками js-фреймворків обумовлено у першу чергу тим, що їх доцільно застосовувати там, де неможливо або складно виконувати програмування звичайними засобами розробки, особливо для веб-додатків – найбільше поширення вони знайшли при реалізації Single Page Applications.

2.2 Шаблони веб-орієнтованих систем

Особливістю Single Page Applications (SPA) слід вважати той факт, що він вміщує на одній сторінці увесь код – як HTML, так і JavaScript, та CSS, які завантажуються або одночасно із завантаженням сторінки, або динамічно – довантажуються у відповідь на дії користувача. Ще однією з особливостей SPA-додатків є той факт, що сторінка не підлягає оновленню і не перенаправляє користувача на інші сторінки. Частіш за все взаємодія із застосунком реалізується через динамічний зв'язок з веб-сервером.

SPA-додаток працює зовсім по-іншому з точки зору передачі або маршрутизації даних у порівнянні з багатосторінковим додатком. Коли клієнт запитує веб-сторінку, він отримує повний код веб-сайту. Отриманий контент містить лише частково завершений HTML-документ, який динамічно доповнюється на стороні клієнта, зазвичай, за допомогою JavaScript. Іншими

словами, видиме подання створюється серією функцій JavaScript, які доповнюють HTML DOM під час виконання. Жодних додаткових запитів не потрібно, але весь видимий контент створюється та модифікується за допомогою JavaScript. Таким чином, наприклад, зміна сторінки SPA насправді не є зміною сторінки, а швидше заміною вмісту поточної сторінки новим вмістом.

JavaScript використовується для розробки як повних веб-сайтів, так і окремих функціональних модулів, до яких відносяться онлайн-інструменти. Зазвичай, повні кадри більш підходять саме для розв’язання першої задачі, а для інших доцільно використання більш світлих кадрів або бібліотек.

В рамках застосування JavaScript однією з вимог сучасних технологій програмування є дотримання моделі обробки даних. Найбільш поширеними моделями на теперішній час вважаються MVC (Model-View-Controller), Model-View-Presenter (MVP) та Model-View-ViewModel (MVVM). Розглянемо їх відмінності та порівняємо між собою, визначивши для цього критерії. З цією метою визначимо наступні ознаки гарної архітектури, які і будемо використовувати у якості критеріїв:

– збалансований розподіл обов'язків між сутностями із жорсткими ролями;

– тестування;

– простота використання та низька вартість обслуговування.

Загальна спрямованість концепцій MVC, MVP та MVV полягає у тому, що у будь-яких додатках, побудованих за даними моделям, має забезпечуватися поділ даних від зовнішнього інтерфейсу. Ці концепції передбачають призначення сутностей додатку за однією з трьох категорій:

Models – відповідає за домен або шар доступу до даних, за допомогою яких виконується маніпулювання даними;

Views – відповідає за рівень уявлення (GUI);

Controller/Presenter/ViewModel – посередник між Model та View; загалом відповідає за зміни Model, реагуючи на дії користувача, що виконані на View, та оновлює View, використовуючи зміни з Model.

Шаблон MVC. Controller є посередником між View та Model, тому ці компоненти не мають уявлення про існування один одного (рис.2.2). Саме тому Controller важко перевикористати. У цій концепції Controller тісно пов'язані з життєвим циклом View, тому досить важко зробити висновок, що він є окремою сутністю. Є можливість розвантажити частину бізнес-логіки та перетворення даних з Controller на Model, але, коли справа доходить до відвантаження роботи у View, існує не так багато варіантів.

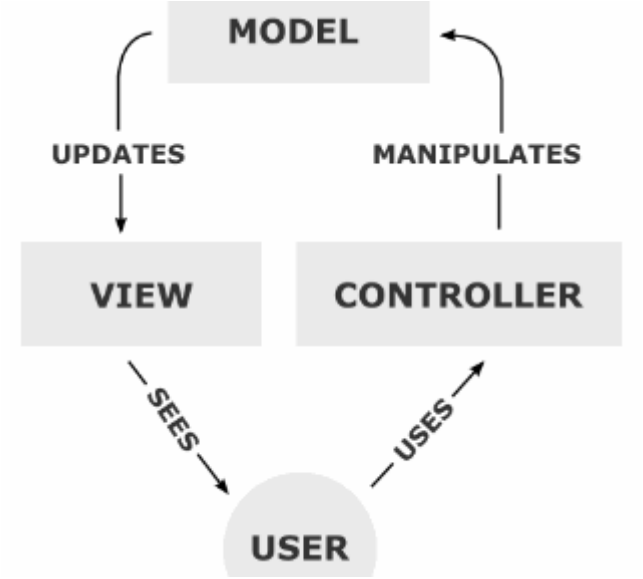

Рис. 2.2 – Шаблон MVC

Частіш за все вся відповідальність View полягає у тому, щоб відправити дії до Controller. В результаті View Controller стає делегатом і джерелом даних, а також – місцем запуску та скасування серверних запитів і, загалом, усього чого завгодно.

Оцінемо MVC з погляду на критерії архітектури, що були визначені вище:

– розподіл: View та Model насправді розділені, але View та Controller тісно пов'язані між собою;

– тестування: через поганий розподіл можливо тестуванню підлягає тільки Model;

– простота використання: найменша кількість коду серед інших патернів програмування, крім того його досить легко може підтримувати навіть недосвідчений розробник.

Шаблон MVP. MVP (рис.2.3) є похідною від MVC, яка використовується в основному для побудови інтерфейсів користувача. Елемент Presenter в даному шаблоні бере на себе функціональність посередника (аналогічно тому, як використовується Controller в MVC) і відповідає за управління подіями інтерфейсу користувача так само, як в інших шаблонах зазвичай відповідає уявлення.

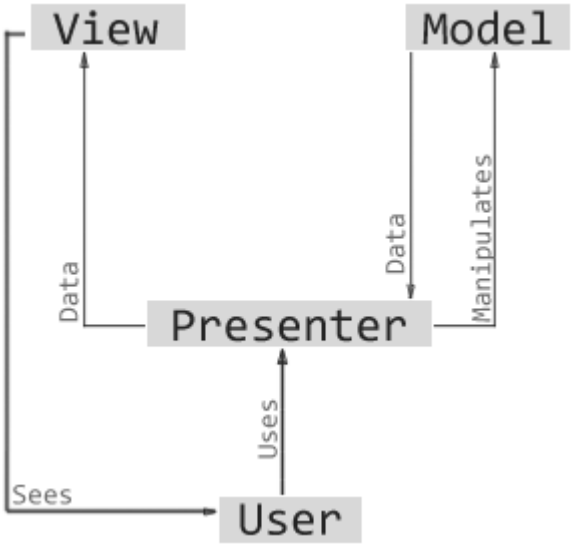

Рис. 2.3 – Шаблон MVP

Шар Presenter тут є абстракцією над View, який управляє відображенням, але не містить деталі реалізації відображення. Також, Presenter є посередником між даними та відображенням.

Виділимо критерії, що притаманні архітектурі MVP:

– розподіл: більша частина відповідальності розділена між Presenter та Model, а View нічого не робить;

– тестування: відмінна – можемо перевірити більшу частину бізнес-логіки завдяки бездіяльності View;

– простота використання: кількість коду вдвічі більша у порівнянні з MVC, але, водночас, ідея MVP дуже проста.

Шаблон MVVM. Цей шаблон дизайну архітектури додатків був представлений у 2005 році Джоном Госсманом (John Gossman) як модифікація шаблону Presentation Model. Цей шаблон орієнтований на сучасні платформи розробки.

В рамках цього шаблону ViewModel викликає зміни у Model і самостійно оновлюється з вже оновленим Model. І оскільки зв'язування відбувається між View та ViewModel, то перша, відповідно, також оновлюється (рис.2.4).

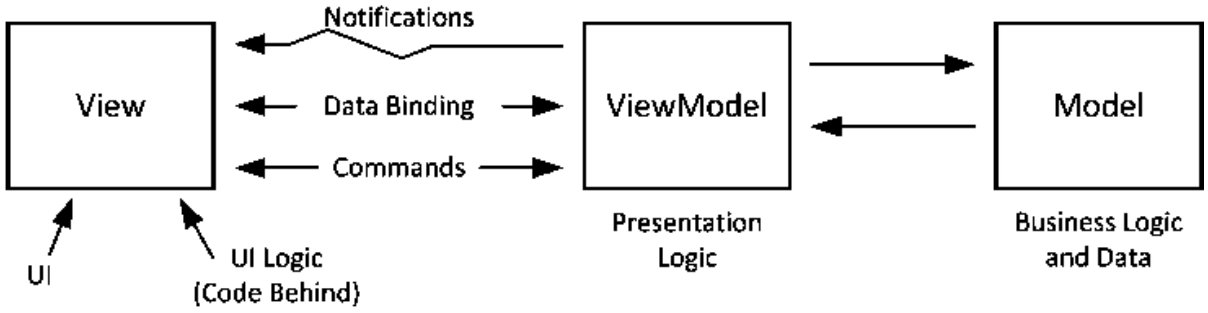

Рис. 2.4 – Шаблон MVVM

Виділимо критерії, що притаманні архітектурі MVVM:

– розподіл: MVVM View має більше обов'язків, ніж View з MVP. Тому що перша оновлює свій стан з ViewModel за рахунок встановлення зв’язування, тоді як друга – спрямовує всі події в Presenter і не оновлює себе (це робить Presenter);

– тестування: ViewModel нічого не знає про уявлення – це дозволяє досить легко тестувати її. View також можна тестувати;

– простота використання: у реальному додатку, де необхідно буде спрямовувати всі події з View в Presenter та оновлювати View вручну, в MVVM буде набагато менше коду (якщо використовувати зв’язування).

MVVM є дуже привабливим патерном, тому що він поєднує в собі переваги вищезгаданих підходів і не вимагає додаткового коду для оновлення View при зв'язуванні на стороні View, тестування також знаходиться на хорошому рівні.

Разом з тим при проектуванні програмної системи, яку будемо розробляти в рамках виконання кваліфікаційної роботи, будемо дотримуватись концепції MVC.

2.3 Можливі шляхи вирішення задачі

У SPA вся програма, включаючи всі його сторінки, завантажується за одне початкове завантаження сторінки. Перша відповідь сервера містить повний веб-сайт, HTML, JavaScript та таблиці стилів. Через це обсяг даних у першому запиті, звичайно, великий, але час завантаження все ще досить малий через поточні високі швидкості широкосмугової технології.

Однак, деякі дані необхідно отримати після початкового завантаження сторінки. Деяку частину сторінки, наприклад, у стрічках соціальних мереж або рекламних об'явах, може знадобитися оновити, наприклад, через дії користувача. В односторінкових програмах такі дані витягуються за допомогою запитів Ajax. Ajax-запити – це асинхронні методи, які дозволяють обмінюватися даними із сервером без перезавантаження сторінки. На малюнку нижче верхній рядок малюнку 3 відображає перший запит, а нижній рядок представляє останні запити. Як правило, дані передаються як об'єкт JSON. JSON (позначення об'єктів JavaScript) - це поширений, легкий, легкочитаний формат об'єктів JavaScript.

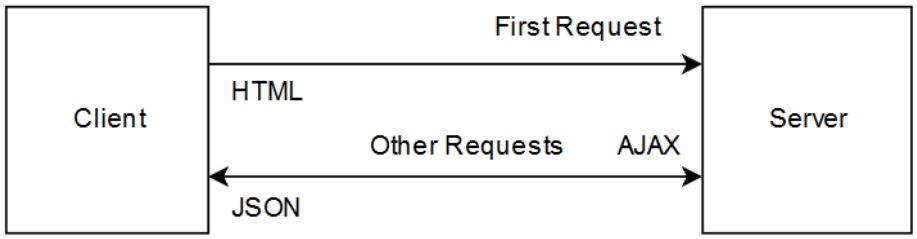

Рис. 2.5 – Передача даних в SPA-додатках

Логіка на стороні сервера зазвичай набагато простіша в моделі односторінкового додатка і вимагає меншого обсягу обробки та логіки. Сервер досить часто є лише веб-службою без збереження стану або (в деяких випадках) з повним збереженням стану. RESTful веб-служби досить поширені серед односторінкових додатків. Веб-сервіси RESTful (репрезентативна передача стану) – це концепція уніфікованих операцій без збереження стану доступу до ресурсів. Різні методи HTTP підключаються до певних ресурсів, і програма запитує ці дані окремо.

У випадку односторінкових програм вся програма спочатку має лише одну URL-адресу. З погляду файлової системи це пов'язано з тим, що вихідний HTML-документ усієї програми включений в один файл. Функції маршрутизації, такі як створення закладок або навігація, повинні бути реалізовані програмно. Використовуються два поширені типи маршрутизації: статична маршрутизація та динамічна маршрутизація.

Статична маршрутизація найчастіше використовують у односторінкових додатках. У статичній маршрутизації маршрути визначаються як частина ініціалізації програми. Перш ніж відбудеться будь-який рендеринг, маршрути вже відомі додатку. На рис.2.6 представлена схема потоку даних моделі статичної маршрутизації.

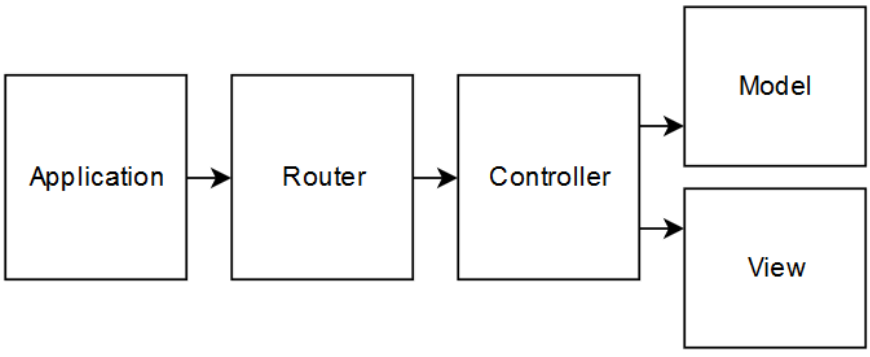

Рис. 2.6 – Схема потоку даних моделі статичної маршрутизації

На рис. 2.6 маршрутизатор розташований між додатком та контролерами. Це означає, що маршрути відомі контролеру на етапі ініціалізації і не можуть

бути згодом змінені. Існує кілька відомих недоліків використання статичної маршрутизації, до яких відноситься той факт, що оскільки маршрути статичні, то змінити маршрут під час виконання неможливо.

Динамічна маршрутизація – ще один варіант маршрутизації SPA. При динамічній маршрутизації маршрут може бути змінено в залежності від переважаючого середовища на стороні клієнта. На рис.2.6 представлена модель динамічної маршрутизації, в якій маршрутизатор розміщується поруч з контролером.

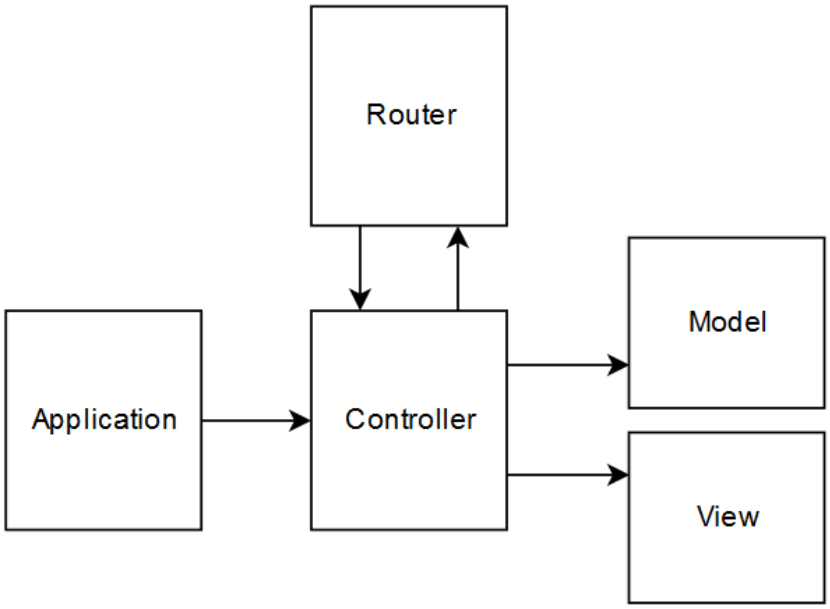

Рис. 2.7 – Потоки даних при динамічній маршрутизації

З рис.2.7 зрозуміло, як контролер постійно працює з маршрутизатором, саме тому він може передавати йому інформацію та отримувати нову інформацію про маршрут. Таким чином, динамічна маршрутизація дозволяє змінювати маршрути в будь-який час, залежно від переважаючого середовища клієнта, наприклад, розміру вікна браузера.

В SPA-моделі програми одночасне завантаження всього вмісту та динамічне створення сторінки значно підвищує загальну швидкість відгуку програми. Браузеру не потрібно завантажувати додатковий контент із сервера, коли користувач змінює сторінки, не потрібно перезавантажувати сторінки, тому

модель SPA також пропонує офлайн-можливості. Це робить зовнішній вигляд односторінкових програм більш схожим на настільну програму. Загальна продуктивність в односторінкових додатках також вища, оскільки ресурси обробки розподіляються більш ефективно. Клієнтські пристрої зазвичай мають значну обчислювальну потужність, яка використовується у моделі SPA.

Одним із факторів, що уповільнюють розробку моделі односторінкових програм, є пошукова оптимізація. Донедавна пошукові системи не могли проіндексувати односторінкові сайти, що призводило до поганої видимості сайту. У жовтні 2015 року Google вніс деякі покращення, почавши індексувати динамічні сторінки [4]. Модель односторінкового застосунку також має деякі проблеми з безпекою з точки зору атак міжсайтового скриптингу (XSS). Шкідливий код JavaScript відносно легко впровадити на веб-сайті SPA [5]. Однак цього можна запобігти, але це потребує особливої уваги з боку розробника. Ще один недолік односторінкових програм полягає в тому, що вони також повністю залежать від підтримки JavaScript у браузері.

2.4 Постановка задачі

На основі проведеного аналізу предметної галузі можна сформулювати постановку задачі та визначити необхідні вимоги до об’єкту дослідження.

Метою дослідження є розробка методів порівняння ефективності використання JavaScript-фреймворків для розробки односторінкових веб-додатків.

Система, що буде створена в рамках дослідження, повинна надавати користувачеві можливість виконувати задачі тайм-планування. Розробка систем буде проводитись з використанням різних фреймворків, але забезпечувати однаковий функціонал.

Головною проблемою та завданням побудови цієї системи є порівняння та аналіз продуктивності систем, що розроблені з використанням різних фреймворків.

Для досягнення мети дослідження необхідно вирішити ряд локальних задач, а саме:

– дослідити технологічні рішення, що використовуються у поширених JavaScript-фреймворках;

– проаналізувати та виявити методи оцінювання ефективності використання фреймворків для побудови веб-орієнтованих додатків;

– дослідити шляхи досягнення кращої продуктивності у динамічних та частково-динамічних веб-додатках;

– виконати дослідження продуктивності веб-додатку на реальному прикладі реалізації архітектурно однакового ToDo-додатку за допомогою різних фреймворків.

– зробити висновки щодо ефективності досліджуваних фреймворків для досягнення мети дослідження.

У ході розробки програмного забезпечення буде розроблено алгоритм реалізації додатку для планування часу. Цей алгоритм планується уніфікувати для можливості його реалізації за допомогою різних фреймворків без внесення суттєвих відмінностей в практичну реалізацію.

Технології, які будуть використанні при розробці програмного забезпечення:

– мова програмування JavaScript;

– фреймворки React.js та Vue.js;

– браузери Google Chrome v. 99.0.4844.84 та Mozilla Firefox v. 98.0.2.

Основною перевагою використання саме цього стеку технологій є те, що вони не потребують додаткового обладнання. Технології, які будуть використовуватися для розробки програмного продукту, буде розглянуто більш детально у наступних розділах.

3 ДОСЛІДЖЕННЯ МЕТОДІВ РОЗВ'ЯЗАННЯ ЗАДАЧІ

3.1 Аналіз та порівняння поширених фреймворків JavaScript

Популярність JavaScript зростає день у день. Кількість і якість технологій та фреймворків, які побудовані на цій мові, постійно зростає та покращується open-source спільнотою. Але існує значна кількість проблем, пов'язаних з проектуванням та розробкою веб-додатків. Серед них найбільш суттєвими є наступні:

- ускладнена масштабованість коду;
- невисока швидкість розробки та впровадження;
- нехтування принципами SOLID, що веде до повторення коду;
- чутливість UI на стороні клієнта;
- кількість ресурсів, які використовуються додатком.

Одним з ефективних шляхів подолання цих недоліків стала розробка фреймворків (серед них – Vue, AngularJS, React), які частково вирішують ці проблеми, але роблять це кожний по-своєму. Ці фреймворки є каркасом для окремих сторінок веб-додатку, що дозволяє розробникам менше приділяти увагу структурі коду або його підтримці у той час, коли вони зосереджені на створенні складних елементів інтерфейсу.

Серед переваг використання JavaScript фреймворків слід відзначити:

а) Ефективність – проекти, реалізація яких раніше зайняла б багато часу та умістилась би у сотні рядків коду, зараз можуть бути реалізовані набагато швидше з добре структурованими готовими шаблонами та функціями;

б) Безпека – кращі JavaScript фреймворки мають фірмову систему безпеки та підтримуються великою спільнотою, члени якої (да і прості користувачі також) виступають у ролі тестувальників;

в) Витрати – більшість фреймворків з відкритим кодом та є безкоштовними. За рахунок того, що вони допомагають програмістам підвищити

швидкість розробки користувальницьких рішень, кінцева вартість розробки веб-додатків буде нижчою.

Разом з тим, отримуючи всі переваги використання фреймворків, розробники вимушені жертвувати продуктивністю та кількістю споживаних ресурсів. Для невеликих веб-додатків це не є критичним, але у разі розробки масштабного проекту, який розрахований на використання мільйонами людей, то кожен мегабайт пам’яті і кожна мілісекунда, що витрачена на запуск і роботу фреймворку, є вкрай важливими.

На теперішній час React, Angular та Vue є одними з найпопулярніших фреймворків JavaScript. Швидка поява значної кількості нових фреймворків примушує розробників обирати кращі серед них. Саме тому доцільною є спроба порівняти JavaScript фреймворки за фактичними показниками, які є критичними для розробників [2]. До таких показників слід віднести:

– рендерінг: відображення кінцевого результату;

– архітектура компонентів;

– спрямованість та класи залежностей;

– зворотня сумісність;

– документація та підтримка.

Не кожен фреймворк на основі JavaScript є реальним фреймворком. Але шляхом додавання до пакету декількох бібліотек ці не зовсім фреймворки становяться схожими на справжні – це відноситься саме до фреймворків React та Vue, які є бібліотеками JS для рівня представлення; та Backbone, який також лише частково реалізує архітектуру MVC (див.рис.3.1).

Сучасне середовище JavaScript має відповідати декільком вимогам.

По-перше, сучасні інтерфейсні JavaScript-фреймворки повиння відповідати специфікації веб-компонентів. В сучасній клієнтській розробці найбільшу увагу привертає створення користувацьких HTML-елементів.

По-друге, надійний JavaScript-фреймворк повинен мати власну екосистему. Готові рішення призначені для розв’язання різних проблем розробки на стороні клієнта таких як маршрутизація, управління станом програми,

взаємодія із серверною частиною тощо. Angular, Backbone, React, Vue та Ember відповідають останній специфікації JavaScript ES6+; також вони всі мають власні екосистеми.

Name	Type	Shadow DOM EcmasScript 6+	Relative Popularity	Difficulty ofLearning
React	Library	Supported	*****	*****
Angular	Framework	Supported	***	*****
Ember	Framework	Supported	*	*****
Vue	Library	Supported	**	***
Backbone	Framework	Supported	*	***

Рис.3.1 – Популярні фреймворки

3.1.1 React

React на теперішній час є одним з найбільш поширених фреймворків JavaScript. React заохочує використання реактивного підходу та парадигми функціонального програмування. React також представив безліч власних концепцій, що визначають його унікальний (на момент створення) підхід до веб-розробці зовнішнього інтерфейсу. Для того, щоб ефективно використовувати React, необхідно освоїти компонентну архітектуру, JSX та односпрямований потік даних (рис.3.2).

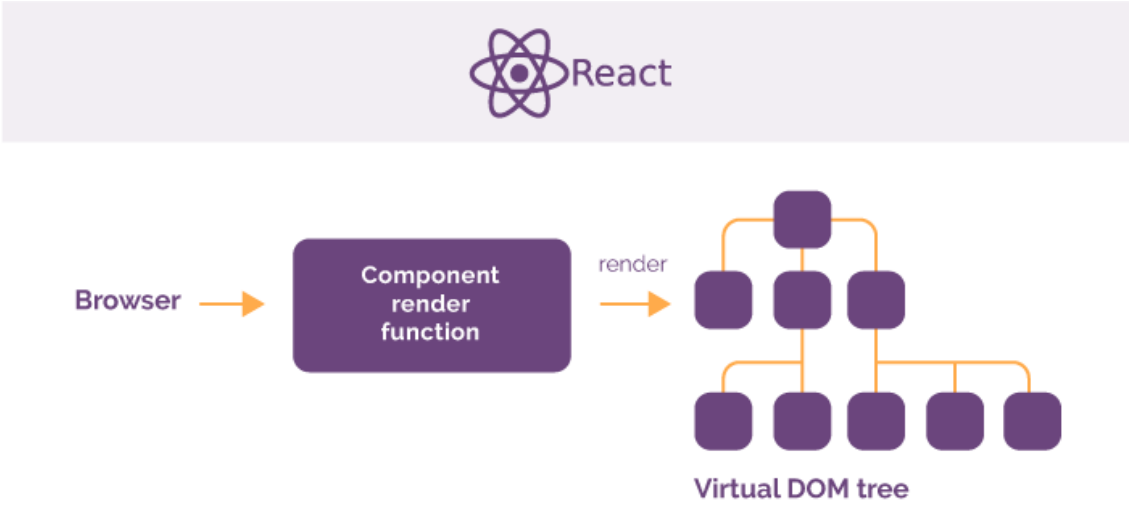

Рис.3.2 – React

Програми React розділені на декілька компонентів. Один файл компоненти містить бізнес-логіку та HTML-розмітку (фактично це розмітка JSX, перетворена на функції JavaScript). Для зв'язків між компонентами можна використовувати або Flux, або аналогічну JS-бібліотеку. У React також з'явилися спеціальні об'єкти – state та props. Використовуючи об'єкти стану та реквізиту, ви можете просто передавати дані з компоненти у макету (використовуючи об'єкт стану) або від батьківського компонента – до дочірньої компоненти (використовуючи об'єкт реквізиту).

React має багато додаткових інструментів для досягнення високої гнучкості. Наприклад, замість Flux можна вибрати MobX, Redux, Fluxy, Fluxible або RefluxJS. І цей список бібліотек управління станом для React не вичерпний. Теж саме стосується і бібліотек HTTP: React може працювати з jQuery AJAX, fetch API, Superagent і Axios.

Незважаючи на те, що гнучкість є його головною перевагою, React саме через його гнучкість має проблеми: коли доводиться вибирати з багатьох додаткових бібліотек, то досить часто виникає дилема – що саме вам слід використовувати з React. Досі немає надійного робочого процесу розробки з React.

React намагається встановити галузеві стандарти. Наприклад, Redux вважається найкращою та єдиною бібліотекою для програм React корпоративного рівня. У той же час Redux може серйозно знизити продуктивність розробки, оскільки Redux ускладнює роботу, коли необхідно впровадження нової функції, а в результаті необхідно змінити занадто багато речей у всьому додатку, що вимагає створення власного робочого процесу із React.

3.1.2 Angular 2+

Поява Angular 2+ знаменує собою поворотний момент історії фреймворку Angular. Почавши як серйозний конкурент Backbone, AngularJS (версія Angular 1.x) майже застаріла, коли вийшов React. Angular істотно змінив свою архітектуру, щоб пристосуватися до React.

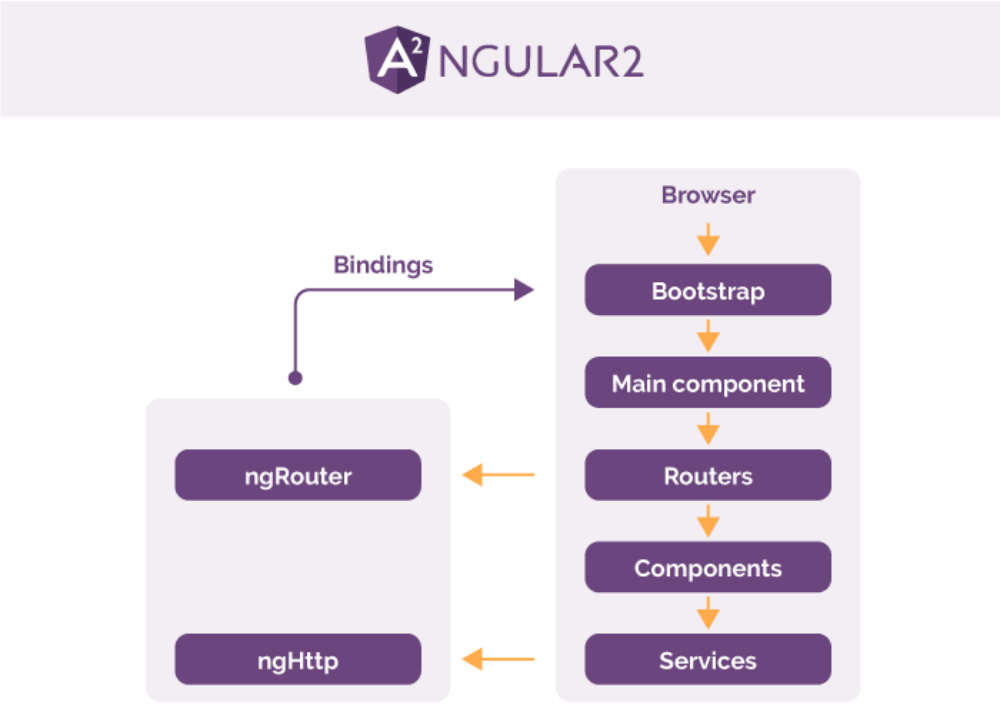

Рис.3.3 – Angular 2+

Головною зміною Angular став перехід від архітектури Model-View-Whatever (одна з форм шаблону MVC) до архітектури, заснованої на компонентах – тим самим Angular став модульним. Якщо раніше в основний HTML-файл можна було вставити посилання на бібліотеку AngularJS, то тепер з’явилась можливість встановлювати окремі модулі, а не всі разом: можна легко відмовитися від встановлення деяких модулів Angular, наприклад @angular/forms або @angular/http, якщо вони не потрібні.Під час розробки програми за допомогою Angular необхідно поділити її на кілька компонентів із вкладеними або дочірніми компонентами. Кожен компонент Angular розбитий на три файли: бізнес-логіка для компонента записується в основний файл, тоді як макет та стилі, пов'язані з цим компонентом, записуються у два інших файли.

Крім компонентів в Angular присутні сервіси – структури застосування залежностей (в React застосування залежностей відсутнє) і директив (спеціальних атрибутів HTML). Фактично, Angular бажає, щоб розробка інтерфейсної частини програми велась запропонованим чином, на відміну від React.

Angular має пару додаткових проблем. Ми майже зобов'язані використовувати TypeScript для безпеки типів у додатках Angular, а наявність TypeScript робить фреймворк Angular 2+ не таким приємним у роботі.

3.1.3 Vue

На перший погляд може здатись, що бібліотека Vue – це суміш Angular і React. Фактично, Еван Ю, автор Vue, запозичив концепції з Angular та React. Наприклад, Vue хоче, щоб ви зберігали логіку компонентів та макети разом із таблицями стилів в одному файлі, так саме, як працює React без таблиць стилів. Щоб дозволити компонентам Vue спілкуватися один з одним, Vue використовує

властивості та об'єкти стану. Цей підхід також існував у React до того, як його прийняв Vue.

Подібно до Angular, Vue хоче, щоб ви змішували макети HTML з JavaScript. Ви повинні використовувати директиви Vue, такі як v-bind або v-if, для інтерполяції значень із логіки компонентів шаблону.

Одна з причин, чому Vue варто розглядати замість React, полягає в бібліотеці Redux, яка часто використовується у великомасштабних додатках React. Як пояснюється в розділі React, коли програма React+Redux стає більшою, витрачаєте багато часу на внесення невеликих змін у кілька файлів замість фактичної роботи над функціями. Бібліотека Vuex – схожий на Flux інструмент управління станом, розроблений саме для Vue – є менш громіздким, ніж Redux.

Що стосується вибору між Vue та Angular, то причини вибору Vue замість Angular можна звести до наступного: Angular – це надто складний, повноцінний фреймворк з обмежувальним характером; Vue набагато простіше і менш обмежений, ніж Angular.

Одним з недоліків Vue є те, що він все ще набагато менш популярний, ніж React чи Angular, в результаті чого у Vue набагато менше готових рішень

Екосистема VueJS складається з:

– Vue як бібліотека перегляду;

– Vue-завантажувач для компонентів;

– Vuex, спеціальна бібліотека для керування станом програми за допомогою Vue;

– інструменти розробки Vue.js для Chrome та Firefox;

– Axios та vue-resource для зв'язку між Vue та серверною частиною;

– Nuxt.js - проект, призначений для створення серверних програм за допомогою Vue; Nuxt.js є конкурентом Angular Universal;

– Weex — бібліотека JS, що підтримує синтаксис Vue та використовується для розробки мобільних програм.

Vue надзвичайно близький з погляду робочого процесу до інших фреймворків. Вибір Vue може бути обумовлений тим, що він менш складний,

ніж React та Angular 2+. З іншого боку, наявність надійної інфраструктури (Angular) або надійного вибору бібліотек (React та його екосистема) значно переважує простоту розробки програм корпоративного рівня. Саме тому дослідження ефективності застосування фреймворків Vue та React для розв'язання однієї й тієї ж самої задачі дозволить порівняти їх між собою в практичному аспекті.

3.2 Метрики продуктивності

Для порівняння ефективності застосування того чи іншого фреймворку для розвязання конкретної проблеми необхідно визначитись з метриками, за якими ми будемо оцінювати продуктивність та кількість споживаних ресурсів (див. рис.3.4).

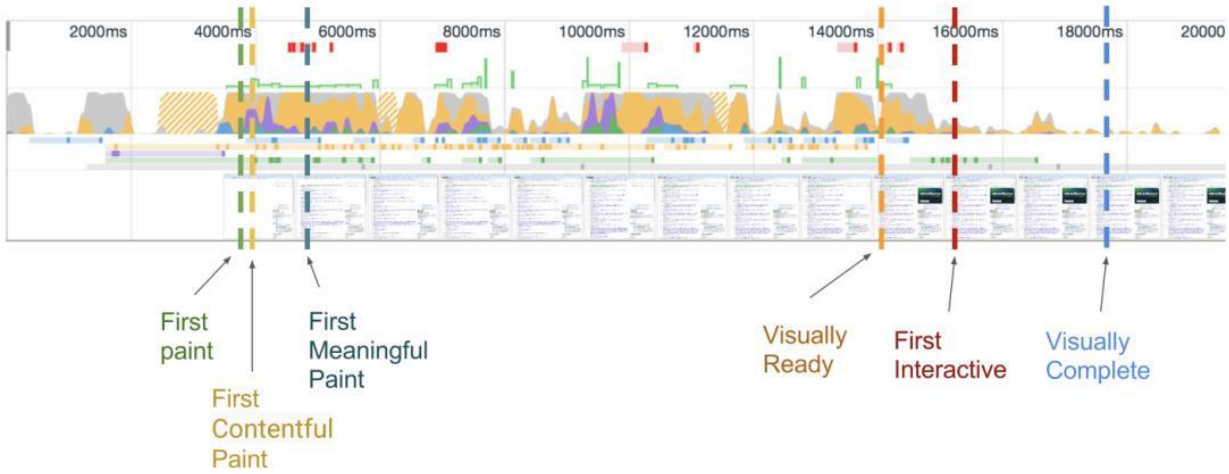

Рис.3.4 – Pipeline завантаження веб-сторінок

Розглянемо кожний з цих показників окремо.

Час до першого байта (Time to First Byte). Час між кліком на посилання та надходженням першого фрагмента вмісту на клієнтську частину, зазвичай це веб-браузер.

Перше відображення (First Paint). Той момент, коли сторінка тільки почала відображатися, тобто це час, коли користувач бачить порожню сторінку вперше. Скрипти ще не завантажилися, і рендеринг на стороні клієнта ще не відбувся.

Перше відображення контенту (First Contentful Paint). Це той час, коли користувач бачить щось корисне, відображене на сторінці – те, що відрізняється від порожньої сторінки. Це може бути все, що завгодно: перше відображення тексту, перше відображення SVG або перше зображення Canvas. Якщо перше відображення контенту займає дуже багато часу, то це може бути пов’язано або з проблемами на рівні з'єднання, або з проблемами ресурсів (наприклад, сама HTML-сторінка дуже «важка» і для її доставки потрібен час).

Перше значиме зображення (First Meaningful Paint). Це той момент часу, коли рендерінг усього головногу контенту завершився і з'явився на сторінці. Для кожної ситуації він різний, але головним контентом слід вважати:

− шапку та текст блогу;

− вміст для пошукових систем;

− критичні для інтернет-магазинів зображення тощо.

Перша взаємодія (First Interactive). Це момент часу, коли веб-додаток стає інтерактивним та починає реагувати на дії користувача і обробляти інформацію на стороні клієнта. Перша взаємодія вважається такою, що відбулась, якщо виконані наступні умови:

− відбулось перше значиме зображення;

− спрацював DOMContentLoaded;

− сторінка візуально готова приблизно на 85%.

Метрика першої взаємодії у свою чергу розділена на дві метрики:

− час до першої взаємодії (TTFI);

− час до першої послідовної взаємодії (TTCI).

Перша послідовна взаємодія (Time to First Consistently Interactive). Використовуючи реверсивний аналіз, який має на увазі аналіз трейсу з кінця, визначається період часу, коли завантаження ресурсів є неактивним протягом 5 секунд, і в цей період відсутні довгі завдання. Такий період часу ще має назву «тихе» вікно (quiet window). Час після тихого вікна і перед першим, з кінця, довгим завданням, буде часом до першої послідовної взаємодії.

Час до першої взаємодії (Time to First Interactive). Визначення цієї метрики відрізняється від першої послідовної взаємодії. Трейс аналізується з монету старту та до кінця. Після того, як відбулося перше значиме зображення, шукають «тихе» вікно довжиною у 3 секунди – цього достатньо, щоб сказати, що сторінка є інтерактивною. Але іноді в трейсі можуть бути самостійні завдання під час або після завершення «тихого» вікна.

Поодинокі завдання виконуються значно пізніше після першого значимого зображення та ізольовані періодом виконання: 250 мс (envelope size) та одна секунда тиші до і після цього періоду. Іноді самотні завдання, час виконання яких займає понад 250 мс, можуть значно впливати на швидкодію сторінки.

Затримка введення (Estimated Input Latency). Ця метрика призначена для оцінювання часу, який необхідний додатку для відповіді на введення користувача протягом найбільш завантаженого 5-секундного вікна завантаження сторінки. Часом цього аудиту є ділянка від першого значимого зображення до кінця трейсу, що становить приблизно 5 секунд після часу до інтерактивності. Якщо затримка перевищує 50 мс, користувачі можуть сприймати веб-додаток як занадто повільний.

ВИСНОВКИ

В результаті виконання програми науково-дослідної практики було проведено ознайомлення з базою практики, вивчено структуру наукові напрямки досліджень, що проводяться на кафедру програмної інженерії ХНУРЕ. Також в рамках написання кваліфікаційної роботи магістра було виконано дослідження методів розробки веб-орієнтованих додатків із застосуванням фреймворків JavaScript, визначено переваги та недоліки застосування більшості JavaScript-фреймворків, проаналізовано наукові дослідження, що проводились вітчизняними та закордонними дослідниками та практичними фахівцями стосовно шляхів підвищення ефективності розробки веб-орієнтованих додатків.

Під час виконання програми науково-дослідної практики було виконано постановку задачі дослідження, визначено перелік локальних задач, розв’язання яких безпосередньо пов’язані з метою дослідження. На підставі проведеного аналізу першоджерел було визначено метрики ефективності, що будуть покладені в основу порівняння ефективності використання веб-орієнтованих додатків планування часу, визначено перелік фреймворків, які будуть використовуватись для розробки програмної системи.

На підсумку слід зазначити, що програму науково-дослідної практики виконано у повному обсязі у відподності до плану досліджень.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Berd R. Pearls of Functional Algorithm Design. Tutorial – DMK-Press, 2013. – 454 p.

2. Graham I. Object-Oriented: Principles & Practice. Tutorial – Williams, 2004. – 880 p.

3. Klements P., Norsrop L. Software Product Lines: Practices and Patterns. Tutorial – Addison-Wesley, 2002. – 608 p.

4. Ахо А., Хопкрофт В., Ульман Д. Структуры данных и алгоритмы. Пособие – Вильямс, 2000. – 384 с.

5. Бурханов К.С., Родионов В.В. Сравнение производительности Vue, React, Angular [Текст] // Сборник научных трудов по материалам XX Международной научно-практической конференции «Актуальные вопросы науки и практики». – 2020. – С. 93.

6. Гамма Э., Хелм Р., Джонсон Р. Приемы объектно-ориентированного проектирования. Паттерны проектирования. Пособие – Питер, 2007. – 366 с.

7. Макконнелл С. Совершенный код. Пособие – Питер, 2005. – 896 с.

8. Мухортов В., Рылов В. Объектно-ориентированное программирование, анализ и дизайн. Пособие – Новосибирск, 2010. – 211 с.

9. Орлов С. Технології розробки програмного забезпечення. Пособие – Питер, 2008. – 464 с.

10. A 2021 Comparison of the Best Frontend JavaScript Frameworks // [Електроний ресурс]. – Режим доступу: https://javascript.plainenglish.io/best-frontend-javascript-framework-96ecef9791fa (дата перегляду: 28.01.2022)

11. The Best JS Frameworks for Front End // [Електроний ресурс]. – Режим доступу: https://rubygarage.org/blog/best-javascript-frameworks-for-front-end (дата перегляду: 27.01.2022)

12. Определения понятий Stateful и Stateless в контексте веб-сервисов // [Електроний ресурс]. – Режим доступу:

https://medium.com/@ermakovichdmitriy/определения-понятий-stateful-и-stateless-в-контексте-веб-сервисов-перевод-18a910a226a1 (дата перегляду: 07.02.2022)

13. Серверный и клиентский рендеринг в вебе [Електроний ресурс]. – Режим доступу: https://tproger.ru/translations/rendering-on-the-web/ (дата перегляду: 04.02.2022)

14. Тестирование производительности JavaScript фреймворков [Електроний ресурс]. – Режим доступу: https://github.com/krausest/js-framework-benchmark (дата перегляду: 27.01.2022)