

**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»**

**Інститут прикладного системного аналізу
Кафедра системного проектування**

«На правах рукопису»
УДК 004.4

«До захисту допущено»
Завідувач кафедри
А. І. Петренко
« 15 » грудня 2018 р.

**Магістерська дисертація
на здобуття ступеня магістра
зі спеціальності 122 Комп'ютерні науки
на тему: «Мікросервісний підхід до розробки клієнтської частини
веб-застосунків»**

Виконала:
студентка VI курсу, групи ДА-72мп
Петенок Олена Володимирівна

Керівник:
к.т.н., доцент кафедри системного проектування,
Булах, Б.В.

Рецензент:
к.т.н., доцент кафедри обчислювальної техніки,
Волокита, А.М.

Засвідчую, що у цій магістерській
дисертації немає запозичень з праць
інших авторів без відповідних
посилань.
Студентка _____

Київ – 2018 р.

**Національний технічний університет України
«Київський політехнічний інститут
імені Ігоря Сікорського»**

Інститут/факультет **Інститут прикладного системного аналізу**

(повна назва)

Кафедра **Кафедра системного проектування**

(повна назва)

Рівень вищої освіти – другий (магістерський) за освітньо-професійною програмою

Спеціальність (спеціалізація) – 122 «Комп'ютерні науки» («Системне проектування сервісів»)

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____ А. І. Петренко
(підпис) (ініціали, прізвище)

« 25 » Жовтня 2018р.

**ЗАВДАННЯ
на магістерську дисертацію студенту
Петенок Олені Володимирівні**

(прізвище, ім'я, по батькові)

1. Тема дисертації **Мікросервісний підхід до розробки клієнтської частини веб-застосунків**
науковий керівник дисертації **Булах Богдан Вікторович, к.т.н., доцент,**
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом по університету від **«07» 11 2018 р. № 4121-с**
2. Термін подання студентом дисертації **20 грудня 2018**
3. Об'єкт дослідження **Розробка веб-застосунків**
4. Вихідні дані **1. Приклади застосування мікросервісного підходу до розробки клієнтської частини веб-застосунків.**
2. JavaScript фреймворки.

5. Перелік завдань, які потрібно розробити _____

1 ОГЛЯД ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Аналіз монолітного підходу до розробки клієнтської частини веб-застосувань

1.2 Аналіз компонентного підходу до розробки клієнтської частини веб-застосувань

1.3 Аналіз мікросервісного підходу до розробки клієнтської частини веб-застосувань

1.4 Огляд варіантів імплементування реалізації клієнтської частини веб-застосувань за мікросервісного підходу розробки

1.5 Огляд реалізацій клієнтської частини веб-застосувань за мікросервісного підходу розробки

1.6 Висновки

2 ОПИС ЗАСОБІВ РЕАЛІЗАЦІЇ

2.1 Опис існуючих популярних JavaScript фреймворків для реалізації клієнтської частини веб-застосувань

2.2 Опис обраних засобів реалізації

2.3 Опис архітектури клієнтської частини власного веб-застосування

2.4 Висновки

3 РЕАЛІЗАЦІЯ ТА АНАЛІЗ РЕЗУЛЬТАТІВ

3.1 Опис реалізованої клієнтської частини веб-застосування

3.2 Аналіз результатів

3.3 Висновки

4 РОЗРОБКА СТАРТАП-ПРОЕКТУ

4.1 Опис ідеї стартап-проекту

4.2 Технологічний аудит ідеї стартап-проекту

4.3 Аналіз ринкових можливостей запуску стартап-проекту

4.4 Розробка ринкової стратегії стартап-проекту

4.5 Розробка маркетингової програми стартап-проекту

4.6 Висновки

6. Орієнтовний перелік графічного (ілюстративного) матеріалу _____

Презентація дипломного проекту

7. Орієнтовний перелік публікацій _____

8. Консультанти розділів дисертації*

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

9. Дата видачі завдання **25 Жовтня 2018р**

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації.	Строк виконання етапів магістерської дисертації	Примітка
1.	Огляд предметної області та опис засобів реалізації.	25.10.2018	
2.	Реалізація та аналіз результатів.	20.11.2018	
3.	Розробка стартап-проекту.	05.12.2018	
4.	Оформлення матеріалів магістерської дисертації.	15.12.2018	

Студент

(підпис)

О.В., Петенок

(ініціали, прізвище)

Науковий керівник дисертації

(підпис)

Б.В., Булах

(ініціали, прізвище)

*Консультантом не може бути зазначено наукового керівника

РЕФЕРАТ

магістерської дисертації Петенок Олени Володимирівни на тему
«Мікросервісний підхід до розробки клієнтської частини веб-застосувань»

У магістерській дисертації досліджується застосування мікросервісного підходу до розробки клієнтської частини веб-застосувань. Тема є актуальною, бо сучасні веб-застосування більше спираються на клієнтську частину, ніж на серверну, вимоги до функціоналу веб-застосувань постійно збільшуються, а вже існуючі функціонал та контент потребують постійної підтримки та оновлення, у зв'язку з чим важливо підібрати доцільний підхід до розробки.

Метою роботи є дослідження застосування мікросервісного підходу до розробки клієнтської частини веб-застосувань. Об'єктом дослідження є розробка веб-застосувань. Предметом дослідження є застосування та аналіз мікросервісного підходу до розробки клієнтської частини веб-застосувань.

Було виконано огляд та аналіз предметної області, спроектовано архітектуру власного веб-застосування за мікросервісного підходу до розробки, реалізовано декілька варіантів власного веб-застосування за розробки різних сервісів різними засобами з однаковим інтерфейсом користувача, протестовано швидкість завантаження всіх реалізованих варіантів власного веб-застосування, проаналізовано отримані результати та зроблено висновки. Потенційне застосування результатів магістерської дисертації: проектування та розробка клієнтської частини веб-застосувань за мікросервісного підходу, маючи інформацію про переваги та недоліки даного підходу та можливі варіанти імплементування реалізації сервісів.

Загальний обсяг роботи: 90 сторінок, 34 рисунки, 24 таблиці та 28 бібліографічних найменувань.

Ключові слова: мікросервісний підхід, веб-застосування, клієнтська частина веб-застосування.

ABSTRACT

for master's thesis of Olena Volodymyrivna Petenok

On "Microservices approach to development of client part of web applications"

In this master's thesis the application of microservices approach to development of client part of web applications is investigated. The theme is relevant because modern web applications are more client-centric than server-centric, the requirements to functionality of web applications are constantly increasing, and existing functionalities and content require constant support and upgrade, which is why it is important to choose an appropriate approach of development.

The purpose of the work is the investigation of the application of the microservices approach to the development of the client part of web applications. The object of research is the development of web applications. The subject of research is the application and analysis of the microservicetic approach to the development of the client part of web applications.

A review and analysis of the subject area was made, the architecture of own Web application was developed in a micro-service approach to development, several variants of own web application were implemented with the development of various services by various means with the same user interface, the speed of loading of all implemented variants of the own web application was tested, the obtained results were analysed and conclusions were made. Potential application of the results of the master's thesis: designing and developing a client part of web applications using the microservices approach with information on the advantages and disadvantages of this approach and possible options for implementing of the services.

Total amount of research: 90 pages, 34 figures, 24 tables, 28 bibliographical names.

Key words: microservices approach, web application, client part of web application.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ, ТЕРМІНІВ.....	10
ВСТУП	12
1 ОГЛЯД ПРЕДМЕТНОЇ ОБЛАСТІ.....	14
1.1 Аналіз монолітного підходу до розробки клієнтської частини веб-застосунків.....	14
1.1.1 Загальний аналіз підходу	14
1.1.2 Аналіз підходу з точки зору розробки клієнтської частини веб-застосунків.....	14
1.2 Аналіз компонентного підходу до розробки клієнтської частини веб-застосунків.....	15
1.2.1 Загальний аналіз підходу	15
1.2.2 Аналіз підходу з точки зору розробки клієнтської частини веб-застосунків.....	17
1.3 Аналіз мікросервісного підходу до розробки клієнтської частини веб-застосунків.....	18
1.3.1 Загальний аналіз підходу	18
1.3.2 Аналіз підходу з точки зору розробки клієнтської частини веб-застосунків.....	19
1.4 Огляд варіантів імплементування реалізації клієнтської частини веб-застосунків за мікросервісного підходу розробки	22
1.4.1 Використання компонентів як інтеграційного шару	22
1.4.2 Використання спільних подій як інтеграційного шару	22
1.4.3 Використання сторонніх бібліотек та фреймворків як інтеграційного шару.....	22

1.5 Огляд реалізацій клієнтської частини веб-застосувань за мікросервісного підходу розробки	23
1.5.1 Приклад з ресурсу micro-frontends.org	23
1.5.2 Приклад з ресурсу allegro.pl	24
1.6 Висновки.....	25
2 ОПИС ЗАСОБІВ РЕАЛІЗАЦІЇ.....	29
2.1 Опис існуючих популярних JavaScript фреймворків для реалізації клієнтської частини веб-застосувань	29
2.1.1 React	29
2.1.2 Angular	29
2.1.3 Vue	29
2.2 Опис обраних засобів реалізації	29
2.3 Опис архітектури клієнтської частини власного веб-застосування	30
2.4 Висновки.....	31
3 РЕАЛІЗАЦІЯ ТА АНАЛІЗ РЕЗУЛЬТАТІВ	32
3.1 Опис реалізованої клієнтської частини веб-застосування	32
3.1.1 Загальний опис	32
3.1.2 Перший варіант реалізації: Header: HTML, Footer: HTML, Фільтр: React, Галерея: React	34
3.1.3 Другий варіант реалізації: Header: React, Footer: React, Фільтр: React, Галерея: React	34
3.1.4 Третій варіант реалізації: Header: HTML, Footer: HTML, Фільтр: React, Галерея: Angular	34
3.1.5 Четвертий варіант реалізації: Header: React, Footer: React, Фільтр: React, Галерея: Angular	35
3.2 Аналіз результатів	35

3.2.1	Опис засобів аналізу результатів	35
3.2.2	Перший варіант реалізації: Header: HTML, Footer: HTML, Фільтр: React, Галерея: React.....	35
3.2.3	Другий варіант реалізації: Header: React, Footer: React, Фільтр: React, Галерея: React.....	38
3.2.4	Третій варіант реалізації: Header: HTML, Footer: HTML, Фільтр: React, Галерея: Angular	41
3.2.5	Четвертий варіант реалізації: Header: React, Footer: React, Фільтр: React, Галерея: Angular	43
3.3	Висновки.....	46
4	РОЗРОБКА СТАРТАП-ПРОЕКТУ	49
4.1	Опис ідеї стартап-проекту	49
4.2	Технологічний аудит ідеї стартап-проекту.....	51
4.3	Аналіз ринкових можливостей запуску стартап-проекту	52
4.4	Розробка ринкової стратегії стартап-проекту.....	61
4.5	Розробка маркетингової програми стартап-проекту	67
4.6	Висновки.....	72
	ВИСНОВКИ.....	74
	ПЕРЕЛІК ПОСИЛАНЬ.....	79
	ДОДАТОК А.....	82
	ДОДАТОК Б	88

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ, ТЕРМІНІВ

API	Application programming interface (Інтерфейс програмування застосувань).
UI	User interface (Інтерфейс користувача).
UX	User Experience (Користувальницький досвід) – це емоції та ставлення людини до використання певного продукту, системи чи послуги.
DOM	Document Object Model (Об'єктна модель документа).
XML	Extensible Markup Language (Розширювана мова розмітки).
HTML	Hyper Text Markup Language (Мова розмітки гіпертекстових документів).
AJAX	Asynchronous JavaScript and XML (Асинхронний JavaScript та XML) – підхід до побудови користувацьких інтерфейсів веб-застосувань, за якого веб-сторінка у фоновому режимі надсилає запити на сервер і довантажує дані.
MVC	Model View Controller (Модель Представлення Контроллер) – архітектурний паттерн, який

розподіляє застосування на три взаємопов'язаних частини: Модель Представлення та Контроллер.

URL	Uniform Resource Locator (Уніфікований локатор ресурсу) – посилання на веб-ресурс, що визначає його розташування в комп'ютерній мережі та механізм його отримання.
SPA	Single-page application (Односторінкове застосування) – веб-застосування, що взаємодіє з користувачем, динамічно перезаписуючи поточну сторінку замість завантаження нових сторінок з сервера.
SSI	Server Side Includes (Включення серверної сторони) – інтерпретована на сервері скриптова мова.
Micro Frontends	Термін вперше з'явився наприкінці 2016 року, він розширює поняття мікро-сервісної архітектури на розробку клієнтської частини веб-застосувань.
Shadow DOM	Shadow Document Object Model (Тіньова об'єктна модель документа) забезпечує спосіб приєднання прихованого дерева DOM до звичайного дерева DOM.

ВСТУП

Метою магістерської дисертації є: дослідити застосування мікросервісного підходу до розробки клієнтської частини веб-застосувань.

Поставлена мета передбачає вирішення наступних задач:

- огляд та аналіз предметної області;
- проектування архітектури власного веб-застосування за мікросервісного підходу до розробки;
- реалізація декількох варіантів власного веб-застосування за розробки різних сервісів різними засобами з однаковим інтерфейсом користувача;
- тестування швидкості завантаження всіх реалізованих варіантів власного веб-застосування та аналіз отриманих результатів.

Об'єкт дослідження: розробка веб-застосувань.

Предмет дослідження: застосування та аналіз мікросервісного підходу до розробки клієнтської частини веб-застосувань.

Актуальність дослідження:

- сучасні веб-застосування більше спираються на клієнтську частину, ніж на серверну;
- вимоги до функціоналу веб-застосувань постійно збільшуються, а вже існуючі функціонал та контент потребують постійної підтримки та оновлення, тому дуже важливо підібрати доцільний підхід до розробки.

Практична цінність результатів дослідження у:

- аналізі сучасних підходів до розробки клієнтської частини веб-застосувань;
- визначенні переваг та недоліків існуючих підходів до розробки клієнтської частини веб-застосувань;

- дослідженні швидкості завантаження клієнтської частини веб-застосунків за різної реалізації мікросервісного підходу.

Потенційне застосування результатів дослідження: проектування та розробка клієнтської частини веб-застосунків за мікросервісного підходу, маючи інформацію про переваги та недоліки даного підходу та можливі варіанти імплементування реалізації сервісів.

1 ОГЛЯД ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Аналіз монолітного підходу до розробки клієнтської частини веб-застосунків

1.1.1 Загальний аналіз підходу

За монолітного підходу застосування будується як єдине ціле. Моноліт – єдиний логічний виконуваний файл. Щоб внести будь-які зміни в систему, розробник повинен збудувати та розгорнути оновлену версію програми.

За розробки невеликих застосунків монолітний підхід може бути цілком виправданим, але для великих застосунків виникають очевидні проблеми:

- монолітні програми можуть перетворитися на "велику кулю бруду" (ситуацію, коли жоден розробник (або група розробників) не розуміє всієї програми);
- масштабування монолітних додатків часто може бути складним завданням;
- монолітні програми реалізуються за допомогою одного стеку розробки, що може обмежити доступність «правильного» інструментарію для роботи.

1.1.2 Аналіз підходу з точки зору розробки клієнтської частини веб-застосунків

Переваги монолітного підходу:

- легше мати великий і послідовний UX;
- єдина кодова база;
- простіше тестування, у порівнянні з іншими підходами;
- легко і швидко налаштувати та впроваджувати проект в роботу;
- обмін знаннями в одній команді відбувається набагато легше.

Недоліки монолітного підходу:

- монолітні застосування можуть перетворитися на "велику кулю бруду" (ситуацію, коли розробники не розуміють всієї програми);
- масштабування може бути складним завданням;
- єдиний стек розробки може обмежити доступність «правильного» інструментарію для роботи.

Монолітний підхід підходить для невеликих або майже статичних веб-застосувань. Перше в сучасній веб-розробці зустрічається досить часто, а друге майже не зустрічається. Дані обмеження накладаються, бо, за проектування великих нестатичних веб-застосувань, виникає необхідність рефакторингу коду, оновлення функціоналу, оптимізації завантаження сторінок, а це все, в свою чергу, передбачає перехід до модульності в тій чи іншій формі.

1.2 Аналіз компонентного підходу до розробки клієнтської частини веб-застосувань

1.2.1 Загальний аналіз підходу

Компонентний підхід до розробки програмного забезпечення є гілкою розробки програмного забезпечення, що акцентує увагу на поділі функціональних можливостей, доступних в даній програмній системі. Це, заснований на повторному використанні, підхід до визначення, впровадження та компонування слабо пов'язаних незалежних компонентів в системах.

Індивідуальний компонент програмного забезпечення – це програмний пакет, веб-сервіс, веб-ресурс або модуль, який інкапсулює набір пов'язаних функцій або даних.

Всі системні процеси розміщуються в окремі компоненти, так що всі дані та функції усередині кожного компонента семантично пов'язані (так само як і з

вмістом класів), тому часто говорять, що компоненти є модульними та згуртованими.

З точки зору системної координації: компоненти взаємодіють один з одним через інтерфейси. Коли компонент пропонує послуги для решти системи, він приймає наданий інтерфейс, який визначає послуги, які інші компоненти можуть використовувати, і як вони можуть це зробити. Цей інтерфейс можна розглядати як підпис компоненту – клієнтові не потрібно знати про внутрішню роботу компонента (реалізацію), щоб використовувати його. Коли компонент повинен використовувати інший компонент для функціонування, він приймає використовуваний інтерфейс, який визначає послуги, які йому потрібні.

Іншим важливим атрибутом компонентів є те, що вони є замінюваними, тобто компонент може замінити інший (на час проектування або час виконання), якщо компонент-замінник відповідає вимогам початкового компонента (вираженим через інтерфейси). В результаті, компоненти можна замінювати альтернативними або оновленими версіями, не порушуючи систему, в якій вони працюють. Компонент В може негайно замінити компонент А, якщо компонент В забезпечує щонайменше те, що компонент А і використовує не більше, ніж компонент А.

Можливість повторного (багаторазового) використання – важлива характеристика високоякісного програмного компонента. Програмісти повинні розробляти та впроваджувати компоненти програмного забезпечення таким чином, щоб багато різних програм могли їх повторно використовувати.

Сучасні компоненти багаторазового використання інкапсулюють як структури даних, так і алгоритми, які застосовуються до структур даних.

1.2.2 Аналіз підходу з точки зору розробки клієнтської частини веб-застосувань

Компонентний підхід передбачає проектування клієнтської частини веб-застосування як ціле з компонентів, кожен з яких виконує невелику функцію, що складає частину користувацького інтерфейсу.

Кожен з компонентів існує в єдиному просторі, але взаємодіє незалежно від інших. Компоненти мають власні структуру, методи та API. Компоненти також є багаторазовими і можуть бути "вставлені" в інтерфейси за бажанням. Незалежний характер компонентів дозволяє розробникам створювати UI з багатьма різними нестатичними частинами.

Компоненти надбудовуються над концепцією запитів AJAX, в яких запити на сервер здійснюються безпосередньо з клієнтської сторони, що дозволяє DOM бути динамічно оновленим без необхідності оновлення сторінки. Кожен компонент має власні інтерфейси, які можуть здійснювати запити на сервер та оновлюватись. Оскільки компоненти незалежні, один компонент може оновлюватися, не впливаючи на інші компоненти або UI у цілому. Компонентний підхід вимагає, щоб всі методи та API, що відносяться до одного компонента, існували в структурі цього компонента.

У фреймворків MVC-структури є шаблони, які представляють UI, маршрути, які визначають, які шаблони рендерити, а також служби, які визначають допоміжні функції. Навіть якщо шаблон має маршрути та пов'язані методи, всі вони існують на різних рівнях архітектури програми. За компонентного підходу відповідальність розподіляється на компоненти, тобто проектування, логіка та допоміжні методи існують на одному рівні архітектури (як правило, у View), а все, що стосується певного компонента, визначається в класі цього компонента.

Компонентний підхід заохочує багаторазове використання коду та єдину відповідальність, але часто може призвести до роздутого та забрудненого View.

Мета MVC полягає в тому, щоб гарантувати, що кожен рівень застосування має свою окрему відповідальність, а мета компонентного підходу – це втілення всіх обов'язків в одному просторі, тому, за використання багатьох компонентів, існує можливість зниження читабельності коду.

Однією з найбільш очевидних проблем компонентного підходу за розробки клієнтської частини веб-застосунків є схильність до надмірної інженерії. Можливо "вкраплювати" компоненти в декількох різних частинах UI, але багато розробників проектують кожен аспект UI як компонент, що може бути надлишковим.

Переваги компонентного підходу:

- компоненти є модульними, згуртованими, замінюваними, повторно використовуваними, слабо пов'язаними, що дозволяє легко вносити зміни в систему, добудовувати новий функціонал.

Недоліки компонентного підходу:

- схильність до надмірної інженерії: проектування кожного аспекту інтерфейсу користувача як компонента може бути надлишковим.

1.3 Аналіз мікросервісного підходу до розробки клієнтської частини веб-застосунків

1.3.1 Загальний аналіз підходу

Мікросервісний підхід являє собою техніку розробки програмного забезпечення (сервіс-орієнтованого стилю архітектури), який структурує програму як сукупність слабо пов'язаних сервісів. У мікросервісній архітектурі послуги є дрібнозернистими, а протоколи – легкими.

Переваги декомпозиції програми на різні менші сервіси полягають в тому, що це покращує модульність, полегшує розуміння, розробку та тестування і

підвищує стійкість до ерозії архітектури. Це паралелізує розвиток, дозволяючи невеликим автономним командам самостійно розробляти, розгортати та масштабувати свої відповідні сервіси. Це також дозволяє архітектурі індивідуального сервісу виникати шляхом постійного рефакторингу.

Філософія архітектури мікросервісів – "Виконай одне і виконай якісно":

- кожен сервіс дрібний (для виконання єдиної функції);
- сервіси повинні враховувати дефекти та відмови;
- кожен сервіс гнучкий, стійкий до відмов, komponується з іншими сервісами, функціонально мінімальний та повний;
- організаційна культура охоплює автоматизацію тестування та розгортання.

Основні властивості мікросервісного підходу:

- незалежна розробка;
- незалежне розгортання;
- незалежне масштабування;
- простота заміни однієї реалізації сервісу іншою;
- простота додавання нового функціоналу в систему;
- еластичність: вихід з ладу одного сервісу зазвичай не призводить до виходу з ладу всієї системи;
- кожен сервіс може бути реалізований різним технологічним стеком;
- сервіси архітектурно побудовані за симетричним принципом (виробник-споживач).

1.3.2 Аналіз підходу з точки зору розробки клієнтської частини веб-застосувань

Термін Micro Frontends вперше з'явився наприкінці 2016 року, він розширює поняття мікро-сервісної архітектури на розробку клієнтської частини веб-застосувань. Ідея цього виникла через те, що, з плином часу, клієнтська

частина великих веб-застосувань, що часто розробляється командою, зростає, ускладнюється, стає важчою у підтримці та потребує рефакторингу.

Підхід полягає в тому, щоб думати про веб-застосування як про складову функцій, які належать незалежним групам розробників. Кожна команда має визначену сферу чи завдання, на якій вона спеціалізується. Команди розвивають функціонал повною мірою: від бази даних до UI.

Основні властивості мікросервісного підходу:

- бути незалежними від технологій;
- кожна команда має можливість обирати та оновлювати свій стек технологій без узгодження з іншими командами;
- ізолювати код команди;
- встановити незмінні конвенції щодо найменувань, де ізоляція неможлива: простір імен CSS, події, локальні сховища та файли cookie.

На рисунку 1.1 наведено приклад архітектури клієнтської частини веб-застосування за мікросервісного підходу до розробки.

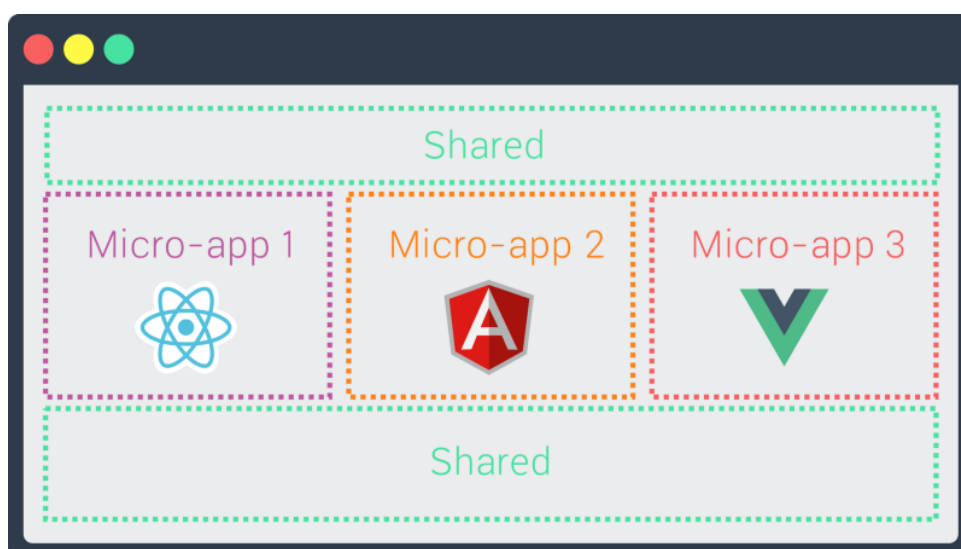


Рис. 1.1 – Можлива архітектура клієнтської частини веб-застосування за мікросервісного підходу до розробки

Основні вимоги до мікросервісного підходу:

- незалежність від стеку розробки;
- команди обирають та оновлюють стек розробки без узгодження з іншими командами;
- ізолювання коду команд;
- можливість використання різних БД незалежними сервісами;
- встановлення незмінної конвенції щодо найменувань, де ізоляція неможлива: простір імен CSS, події, локальні сховища та файли cookie.

Переваги мікросервісного підходу:

- покращує модульність, полегшує розуміння, розробку та тестування, підвищує стійкість до ерозії архітектури;
- паралелізує розробку, дозволяючи автономним командам самостійно розробляти, розгортати та масштабувати сервіси;
- незалежність від стеку розробки, що дозволяє добудовувати новий функціонал у старі великі проекти без рефакторингу старого функціоналу.

Комерційні переваги мікросервісного підходу:

- можливість розширення штату компанії без прив'язки до стеку розробки;
- можливість аутсорсингу без ризиків внесення змін в існуючу продакшн версію застосування;
- відсутня необхідність міграції з фреймворку на фреймворк.

Недоліки мікросервісного підходу:

- схильність до надмірної інженерії: проектування кожного аспекту інтерфейсу користувача як сервісу може бути навіть більш надлишковим, ніж проектування кожного аспекту як компонента;

- необхідність чіткого поділу завдань між групами розробки на початкових стадіях розробки;
- необхідність встановлення незмінної конвенції щодо найменувань, де ізоляція неможлива: простір імен CSS, події, локальні сховища та файли cookie.

1.4 Огляд варіантів імплементування реалізації клієнтської частини веб-застосунків за мікросервісного підходу розробки

1.4.1 Використання компонентів як інтеграційного шару

Інтеграційний шар створюється за рахунок використання одного або комбінації з наступних компонентів:

- Shadow DOM (забезпечує спосіб приєднання прихованого дерева DOM до звичайного дерева DOM);
- Custom Elements (наприклад: `<books-list></books-list>`);
- HTML Imports, HTML Templates (SSI – інтерпретована на сервері скриптова мова).

1.4.2 Використання спільних подій як інтеграційного шару

Сервіси незалежні та обробляють спільні вхідні та вихідні події (тобто спільна шина подій, імплементування шаблону проектування Publish-Subscribe).

1.4.3 Використання сторонніх бібліотек та фреймворків як інтеграційного шару

1.4.3.1 Мета-фреймворк Single-SPA

Мета-фреймворк об'єднує декілька фреймворків на одній сторінці без оновлення сторінки.

Надає можливість:

- використовувати кілька фреймворків на сторінці, не оновлюючи сторінку;
- писати код за допомогою нового фреймворку, не переписуючи існуючу програму;
- використовувати Lazy load code для покращення первинного часу завантаження.

1.4.3.2 Набір бібліотек Project Mosaic

Project Mosaic являє собою набір сервісів, бібліотек разом із специфікацією, які визначають, як їх компоненти взаємодіють між собою, для підтримки мікросервісної архітектури для великомасштабних веб-сайтів.

1.5 Огляд реалізацій клієнтської частини веб-застосувань за мікросервісного підходу розробки

1.5.1 Приклад з ресурсу micro-frontends.org

У прикладі реалізації клієнтської частини веб-застосування за мікросервісного підходу розробки на рисунку 1.2 клієнтська частина веб-застосування розділена на окремі компоненти, що належать трьом командам:

- команда Checkout (синій) відповідає за все, що стосується процесу покупки: кнопка та кошик;
- команда Inspire (зелений) керує рекомендаціями про продукт;
- команда Product (червоний) вирішує, яка функціональність включена і де вона розташована: містить інформацію, яку надає команда Product (назва продукту, зображення та доступні варіанти) і включає компоненти інших команд.

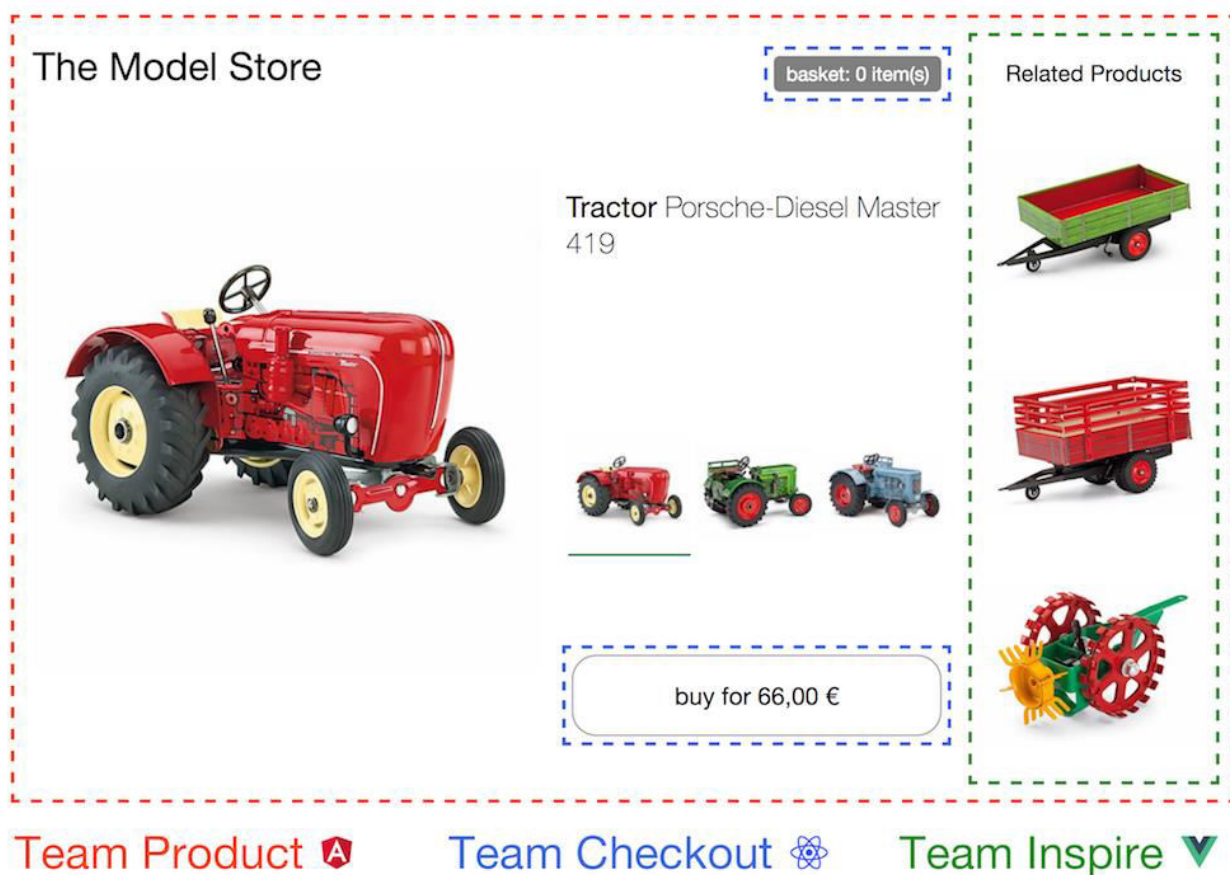


Рис. 1.2 – Приклад реалізації клієнтської частини веб-застосування за мікросервісного підходу розробки з ресурсу micro-frontends.org

1.5.2 Приклад з ресурсу allegro.pl

На рисунку 1.3 зображено приклад реалізації клієнтської частини веб-застосування за мікросервісного підходу розробки, де клієнтська частина веб-застосування розділена на окремі компоненти, що належать чотирьом командам.

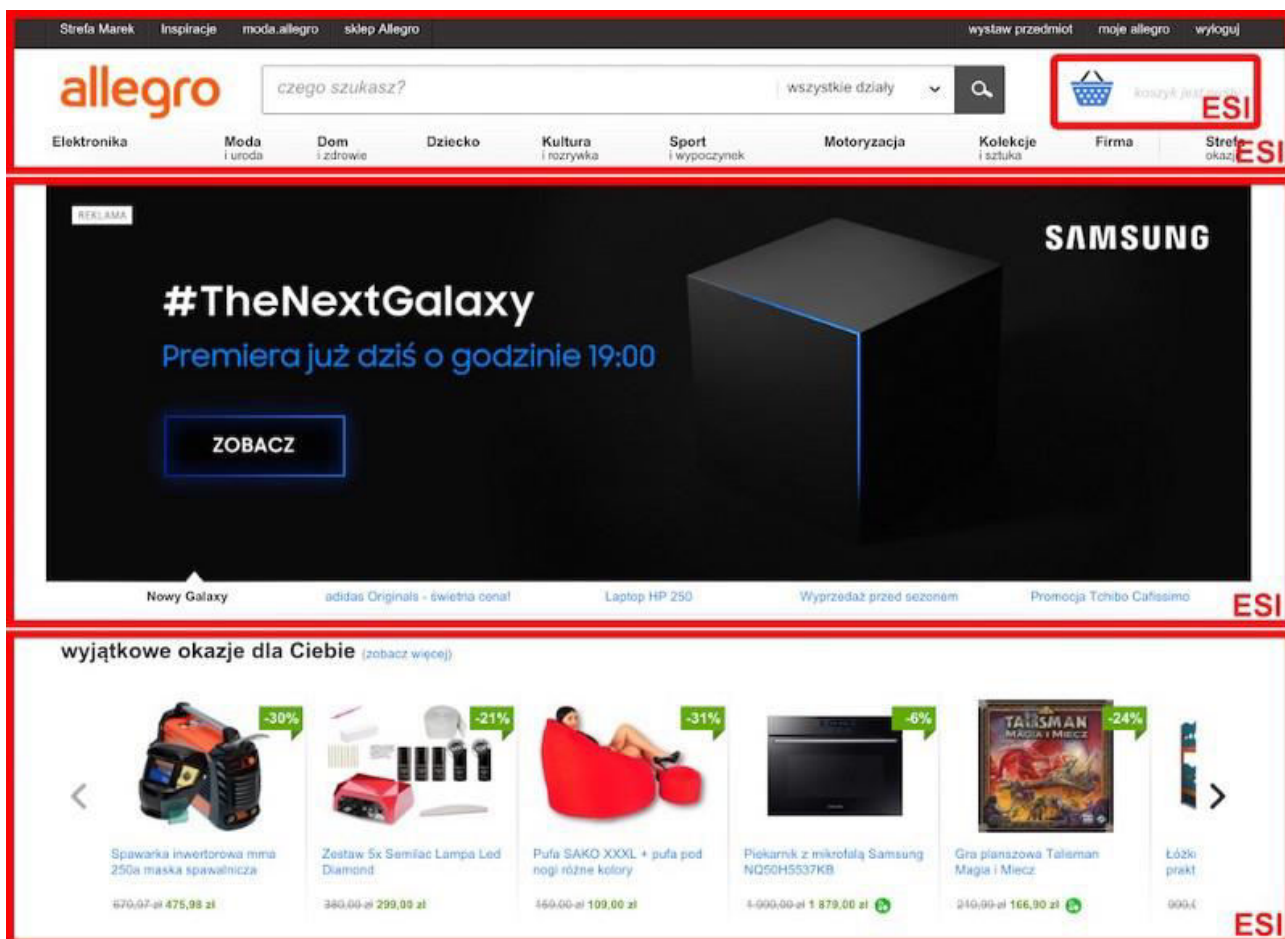


Рис. 1.3 – Приклад реалізації клієнтської частини веб-застосування за мікросервісного підходу розробки з ресурсу allegro.pl

1.6 Висновки

У даному розділі проведено огляд предметної області:

- проведено аналіз монолітного, компонентного та мікросервісного підходів (як в загальному, так і з точки зору розборки клієнтської частини веб-застосувань);
- зазначено переваги та недоліки монолітного, компонентного та мікросервісного підходів з точки зору розборки клієнтської частини веб-застосувань;
- досліджено варіанти імплементування реалізації клієнтської частини веб-застосувань за мікросервісного підходу розробки (використання

компонентів, спільних подій або сторонніх бібліотек в якості інтеграційного шару);

- представлено варіанти існуючих реалізацій клієнтської частини веб-застосувань за мікросервісного підходу до розробки.

Переваги монолітного підходу:

- легше мати великий і послідовний UX;
- єдина кодова база;
- простіше тестування, у порівнянні з іншими підходами;
- легко і швидко налаштувати та впроваджувати проект в роботу;
- обмін знаннями в одній команді відбувається набагато легше.

Недоліки монолітного підходу:

- монолітні застосування можуть перетворитися на "велику кулю бруду" (ситуацію, коли розробники не розуміють всієї програми);
- масштабування може бути складним завданням;
- єдиний стек розробки може обмежити доступність «правильного» інструментарію для роботи.

Переваги компонентного підходу:

- компоненти є модульними, згуртованими, замінюваними, повторно використовуваними, слабо пов'язаними, що дозволяє легко вносити зміни в систему, добудовувати новий функціонал.

Недоліки компонентного підходу:

- схильність до надмірної інженерії: проектування кожного аспекту інтерфейсу користувача як компонента може бути надлишковим.

Основні вимоги до мікросервісного підходу:

- незалежність від стеку розробки;

- команди обирають та оновлюють стек розробки без узгодження з іншими командами;
- ізолювання коду команд;
- можливість використання різних БД незалежними сервісами;
- встановлення незмінної конвенції щодо найменувань, де ізоляція неможлива: простір імен CSS, події, локальні сховища та файли cookie.

Переваги мікросервісного підходу:

- покращує модульність, полегшує розуміння, розробку та тестування, підвищує стійкість до ерозії архітектури;
- паралелізує розробку, дозволяючи автономним командам самостійно розробляти, розгортати та масштабувати сервіси;
- незалежність від стеку розробки, що дозволяє добудовувати новий функціонал у старі великі проекти без рефакторингу старого функціоналу.

Комерційні переваги мікросервісного підходу:

- можливість розширення штату компанії без прив'язки до стеку розробки;
- можливість аутсорсингу без ризиків внесення змін в існуючу продакшн версію застосування;
- відсутня необхідність міграції з фреймворку на фреймворк.

Недоліки мікросервісного підходу:

- схильність до надмірної інженерії: проектування кожного аспекту інтерфейсу користувача як сервісу може бути навіть більш надлишковим, ніж проектування кожного аспекту як компонента;
- необхідність чіткого поділу завдань між групами розробки на початкових стадіях розробки;

- необхідність встановлення незмінної конвенції щодо найменувань, де ізоляція неможлива: простір імен CSS, події, локальні сховища та файли cookie.

2 ОПИС ЗАСОБІВ РЕАЛІЗАЦІЇ

2.1 Опис існуючих популярних JavaScript фреймворків для реалізації клієнтської частини веб-застосувань

2.1.1 React

React обробляє компоненти надзвичайно ефективним способом. React використовує віртуальний DOM, що використовує алгоритм "відмінності" для виявлення змін у компоненті і рендерить лише ці зміни замість повторного рендерингу всього компонента.

2.1.2 Angular

Angular спирається на TypeScript, що забезпечує узгодженість у споріднених задачах та проектах з відкритим вихідним кодом. Є поняття декораторів та статичних типів. Статичні типи корисні для розумних інструментів розробки (таких як автоматичний рефакторинг), крім того, вони зменшують кількість помилок у програмі.

2.1.3 Vue

Vue дуже швидко набирає популярності, бо є відносно простим в опануванні і легковісним (тому підходить для невеликих проектів краще, ніж React чи Angular). Розробники також відмічають, що Vue схожий на ранні версії Angular з додаванням сучасних фіч.

2.2 Опис обраних засобів реалізації

Обрано JavaScript фреймворки React (версії 16.5) та Angular (версії 1.6.4).

Також обрано фреймворк Uikit (версії 3.0.0) для візуалізації.

2.3 Опис архітектури клієнтської частини власного веб-застосування

Власне веб-застосування складається з чотирьох сервісів:

- header;
- footer;
- фільтр (ширини відображення списків зображень в галереї);
- галерея (відображення зображень).

Сервіси header, footer та фільтр є повністю незалежними.

Сервіс галерея є частково залежним від сервісу фільтр, тобто: залежність не пряма, а через пошук елементів `` сервісу фільтр по дереву DOM, після чого знайдені елементи порівнюються з допустимими значеннями `id` для `width` сервісу галерея (дані про які завчасно присутні у сервісі галерея). У випадку допустимих значень `id` для `width` сервісу фільтр, сервіс галерея рендериться зі значеннями `value` для `width`, відповідними до `id` та додає обробники події натисканням на елемент до елементів ``.

В результаті такої часткової залежності, у елементів `` сервісу фільтр існує по два обробники подій натискання на елемент (один, що обробляється у власному сервісі та відповідає за відображення активного елемента `` у батьківському елементі `` сервісу фільтр, та інший, що обробляється у сервісі галереї та відповідає за відображення ширини зображень в галереї).

Таким чином, обидва обробники подій спричиняють ререндерінг сервісів фільтру та галереї без впливу на інші сервіси та без загального перезавантаження сторінки.

Також часткова незалежність сервісу галереї від сервісу фільтру у тому, що, у випадку заміни сервісу фільтру іншим (але таким, що структура дерева DOM не зміниться), сервіс галереї працюватиме так само як і до заміни.

2.4 Висновки

У даному розділі представлено опис засобів реалізації:

- проведено огляд існуючих популярних JavaScript фреймворків для реалізації клієнтської частини веб-застосувань;
- обрано засоби реалізації клієнтської частини власного веб-застосування;
- описано архітектуру клієнтської частини власного веб-застосування.

Архітектура власного веб-застосування складається з чотирьох сервісів:

- header;
- footer;
- фільтр (ширини відображення списків зображень в галереї);
- галерея (відображення зображень).

Сервіси header, footer та фільтр є повністю незалежними.

Сервіс галерея є залежним від сервісу фільтр за результуючим деревом DOM, але незалежним від реалізації сервісу фільтр.

У елементів `` сервісу фільтр існує по два обробники подій натискання на елемент (один обробляється у сервісі фільтру та відповідає за відображення елемента ``, а інший обробляється у сервісі галереї та відповідає за відображення ширини зображень).

3 РЕАЛІЗАЦІЯ ТА АНАЛІЗ РЕЗУЛЬТАТІВ

3.1 Опис реалізованої клієнтської частини веб-застосування

3.1.1 Загальний опис

Реалізовано чотири веб-застосування різними засобами з однаковим інтерфейсом користувача, що відповідають архітектурі, описаній у Розділі 2.3.

На рисунках 3.1 та 3.2 зображено інтерфейс користувача клієнтської частини розроблених веб-застосувань за замовчанням.

На рисунку 3.3 зображено інтерфейс користувача клієнтської частини розроблених веб-застосувань після натискання на елемент фільтру зі значенням Normal.

На рисунку 3.4 зображено інтерфейс користувача клієнтської частини розроблених веб-застосувань після натискання на елемент фільтру зі значенням Big.

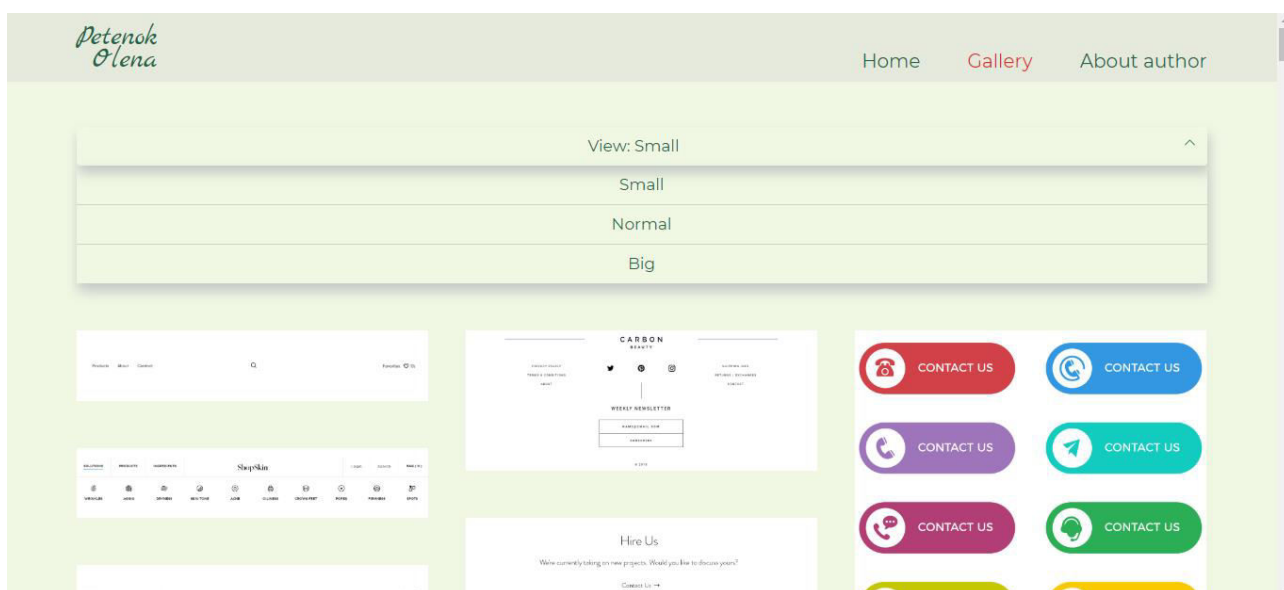


Рис. 3.1 – Інтерфейс користувача клієнтської частини розроблених веб-застосувань за замовчанням

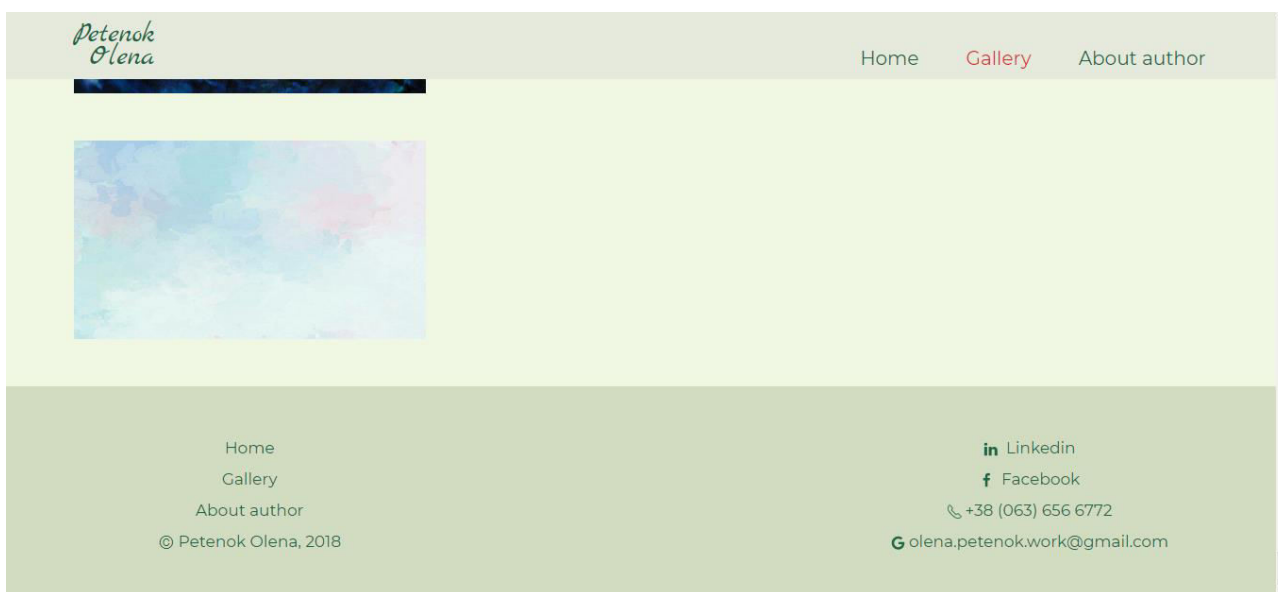


Рис. 3.2 – Інтерфейс користувача клієнтської частини розроблених веб-застосунків за замовчанням

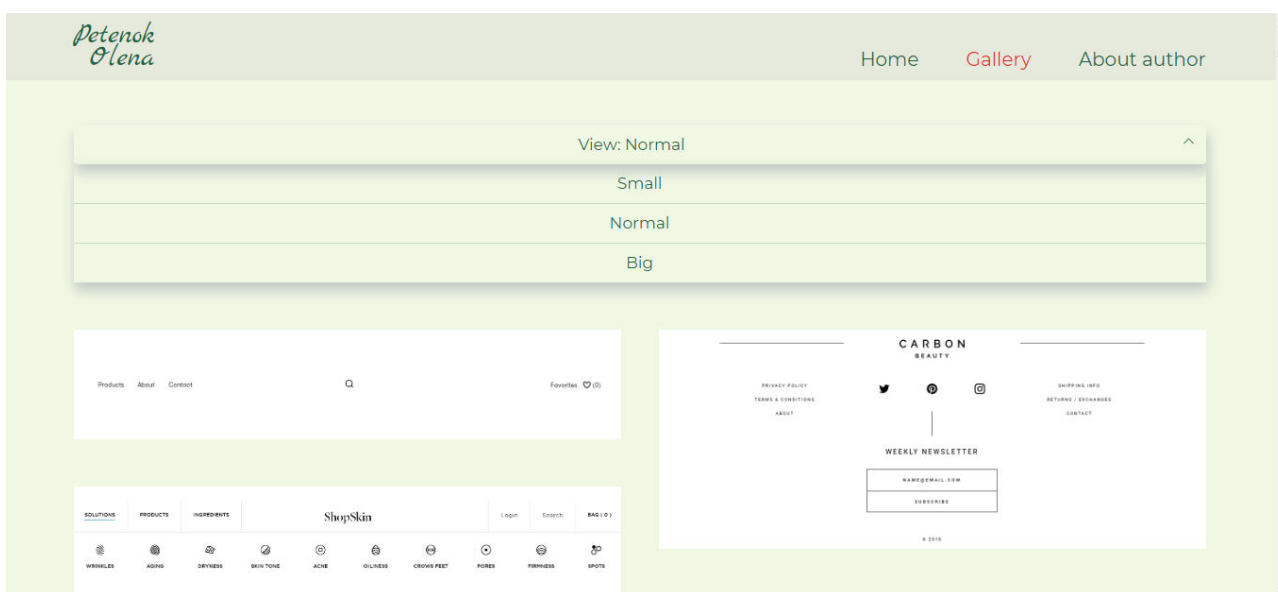


Рис. 3.3 – Інтерфейс користувача клієнтської частини розроблених веб-застосунків після натискання на елемент фільтру зі значенням Normal

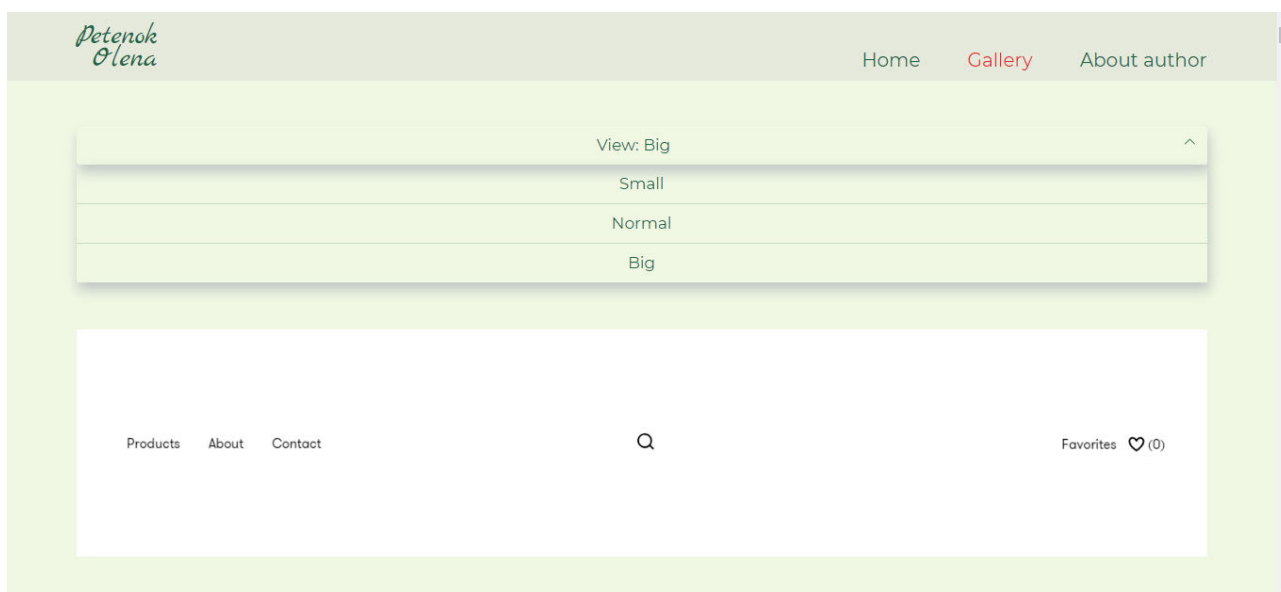


Рис. 3.4 – Інтерфейс користувача клієнтської частини розроблених веб-застосунків після натискання на елемент фільтру зі значенням Big

3.1.2 Перший варіант реалізації: Header: HTML, Footer: HTML, Фільтр: React, Галерея: React

Перший варіант реалізації відтворює частково компонентний підхід до розробки: статичні компоненти Header та Footer написані на HTML, а динамічні компоненти Фільтр та Галерея написані на React.

3.1.3 Другий варіант реалізації: Header: React, Footer: React, Фільтр: React, Галерея: React

Другий варіант реалізації відтворює повністю компонентний підхід до розробки: всі компоненти написані на React.

3.1.4 Третій варіант реалізації: Header: HTML, Footer: HTML, Фільтр: React, Галерея: Angular

Третій варіант реалізації відтворює мікросервісний підхід до розробки: статичні компоненти Header та Footer написані на HTML, а динамічні компоненти Фільтр та Галерея написані на React та Angular відповідно.

3.1.5 Четвертий варіант реалізації: Header: React, Footer: React, Фільтр: React, Галерея: Angular

Четвертий варіант реалізації відтворює мікросервісний підхід до розробки: статичні компоненти Header та Footer написані на React, а динамічні компоненти Фільтр та Галерея написані на React та Angular відповідно.

3.2 Аналіз результатів

3.2.1 Опис засобів аналізу результатів

Аналіз результатів проведений з точки зору швидкості завантаження веб-застосувань, розроблених різними засобами, але з однаковим інтерфейсом користувача.

Для аналізу всі веб-застосування було завантажено на хостинг.

Аналіз проведено наступними веб-застосуваннями:

- Page Speed Insights [26];
- Pingdom Website Speed Test [27] (тести з: Німеччини, Франкфурт; Великобританій, Лондон; США, Вашингтон).

3.2.2 Перший варіант реалізації: Header: HTML, Footer: HTML, Фільтр: React, Галерея: React

На Рисунках 3.5 та 3.6 зображено аналіз швидкості завантаження веб-застосування засобами PageSpeed Insights для мобільної версії.

На Рисунках 3.7 та 3.8 зображено аналіз швидкості завантаження веб-застосування засобами PageSpeed Insights для десктопної версії.

На Рисунках 3.9 – 3.11 зображено аналіз швидкості завантаження веб-застосування засобами Pingdom Website Speed Test.

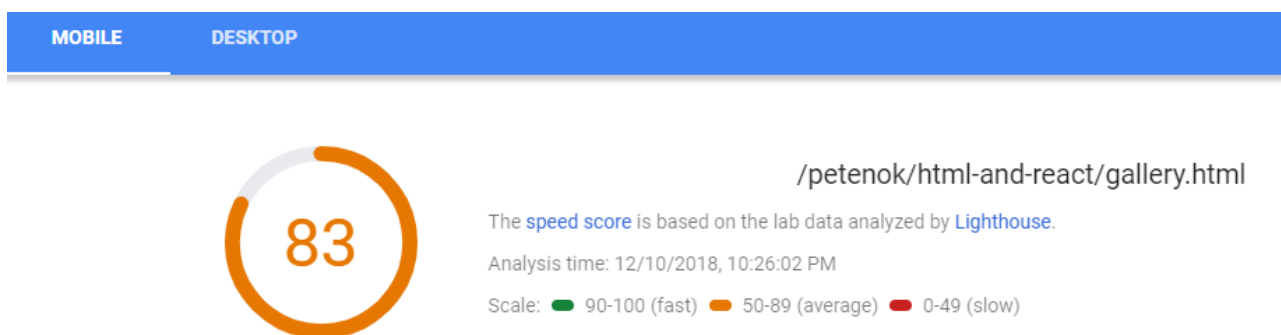


Рис. 3.5 – Аналіз швидкості завантаження веб-застосування засобами PageSpeed Insights для мобільної версії

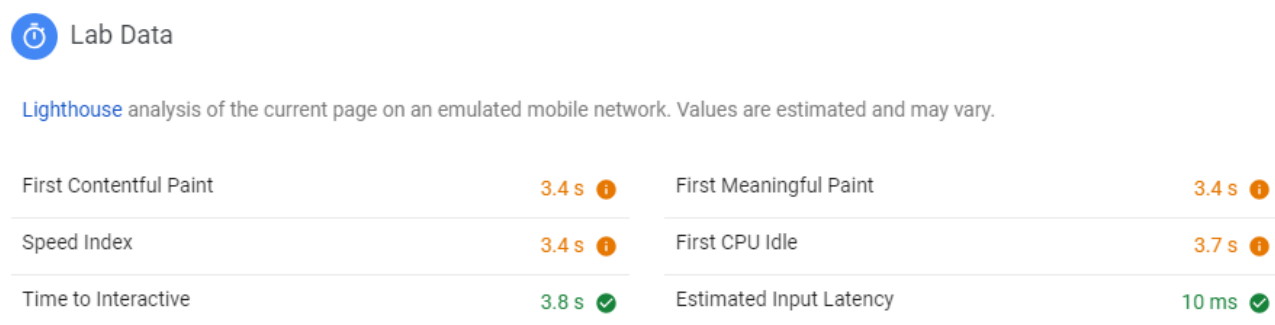


Рис. 3.6 – Аналіз швидкості завантаження веб-застосування засобами PageSpeed Insights для мобільної версії

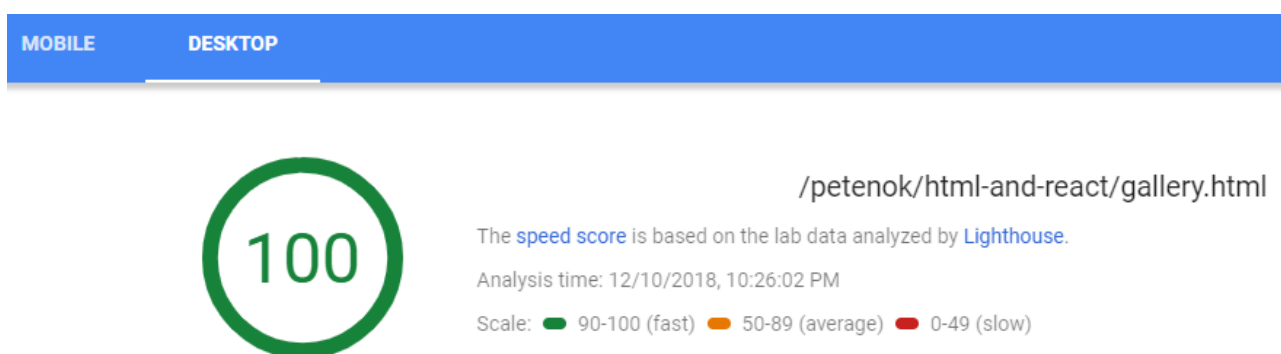



Рис. 3.7 – Аналіз швидкості завантаження веб-застосування засобами PageSpeed Insights для десктопної версії

 Lab Data

Lighthouse analysis of the current page on an emulated mobile network. Values are estimated and may vary.

First Contentful Paint	0.8 s ✓	First Meaningful Paint	0.8 s ✓
Speed Index	1.4 s ✓	First CPU Idle	0.8 s ✓
Time to Interactive	0.8 s ✓	Estimated Input Latency	10 ms ✓

Рис. 3.8 – Аналіз швидкості завантаження веб-застосування засобами PageSpeed Insights для десктопної версії

Performance grade B 81	Page size 12.2 MB
Load time 1.33 s	Requests 61

Рис. 3.9 – Аналіз швидкості завантаження веб-застосування засобами Pingdom Website Speed Test (Germany, Frankfurt)

Performance grade B 81	Page size 12.2 MB
Load time 1.65 s	Requests 61

Рис. 3.10 – Аналіз швидкості завантаження веб-застосування засобами Pingdom Website Speed Test (UK, London)

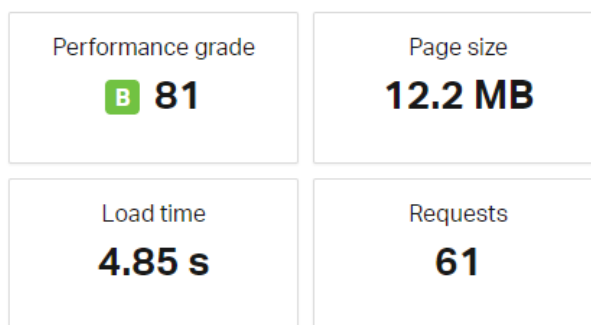


Рис. 3.11 – Аналіз швидкості завантаження веб-застосування засобами Pingdom Website Speed Test (USA, Washington)

3.2.3 Другий варіант реалізації: Header: React, Footer: React, Фільтр: React, Галерея: React

На Рисунках 3.12 та 3.13 зображено аналіз швидкості завантаження веб-застосування засобами PageSpeed Insights для мобільної версії.

На Рисунках 3.14 та 3.15 зображено аналіз швидкості завантаження веб-застосування засобами PageSpeed Insights для десктопної версії.

На Рисунках 3.16 – 3.18 зображено аналіз швидкості завантаження веб-застосування засобами Pingdom Website Speed Test.

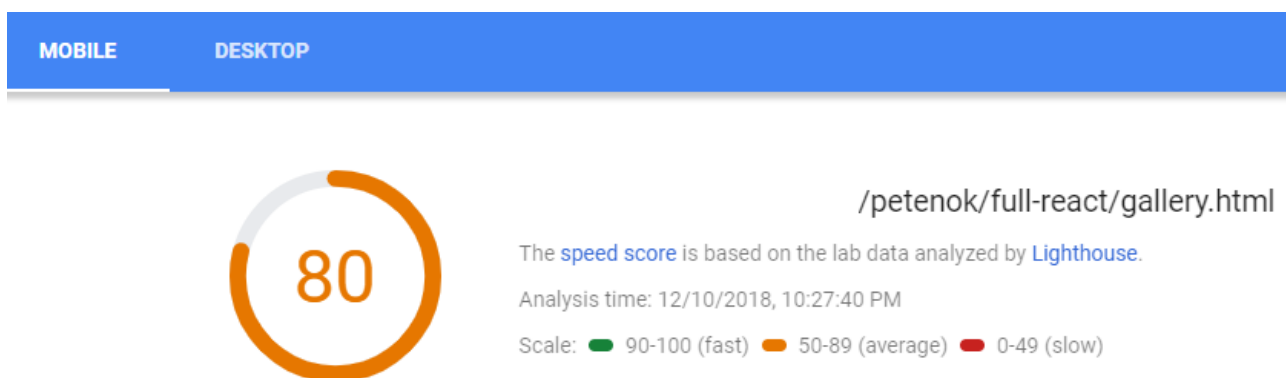



Рис. 3.12 – Аналіз швидкості завантаження веб-застосування засобами PageSpeed Insights для мобільної версії

 Lab Data

[Lighthouse](#) analysis of the current page on an emulated mobile network. Values are estimated and may vary.

First Contentful Paint	3.6 s ⓘ	First Meaningful Paint	3.6 s ⓘ
Speed Index	3.6 s ⓘ	First CPU Idle	3.8 s ⓘ
Time to Interactive	4.0 s ⓘ	Estimated Input Latency	90 ms ⓘ

Рис. 3.13 – Аналіз швидкості завантаження веб-застосування засобами PageSpeed Insights для мобільної версії

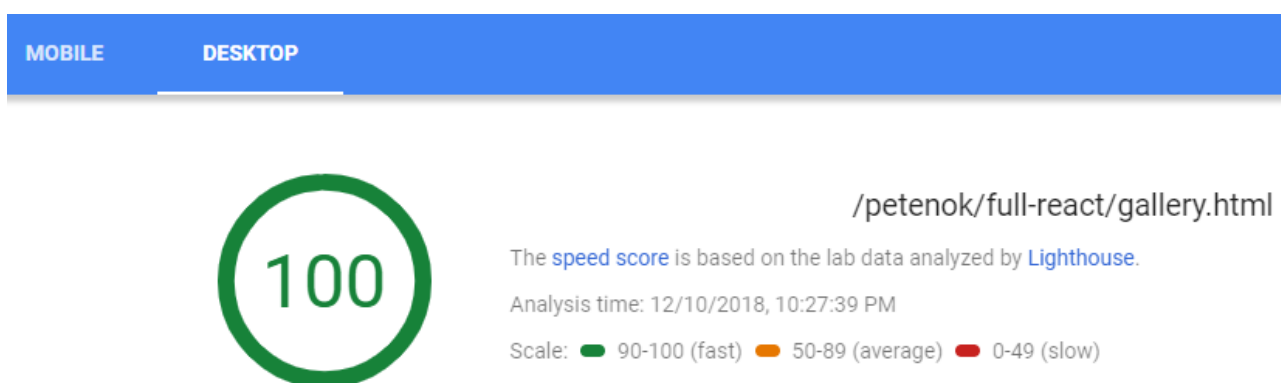



Рис. 3.14 – Аналіз швидкості завантаження веб-застосування засобами PageSpeed Insights для десктопної версії

 Lab Data

[Lighthouse](#) analysis of the current page on an emulated mobile network. Values are estimated and may vary.

First Contentful Paint	0.9 s ✓	First Meaningful Paint	0.9 s ✓
Speed Index	1.3 s ✓	First CPU Idle	0.9 s ✓
Time to Interactive	0.9 s ✓	Estimated Input Latency	10 ms ✓

Рис. 3.15 – Аналіз швидкості завантаження веб-застосування засобами PageSpeed Insights для десктопної версії

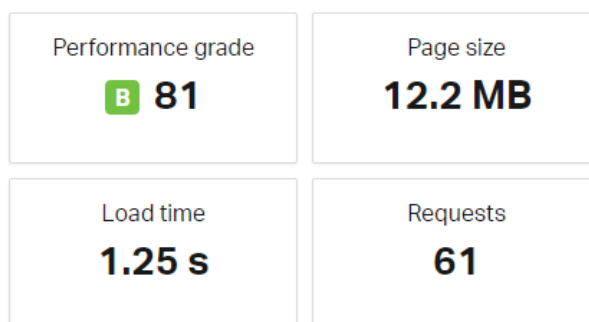


Рис. 3.16 – Аналіз швидкості завантаження веб-застосування засобами Pingdom
Website Speed Test (Germany, Frankfurt)

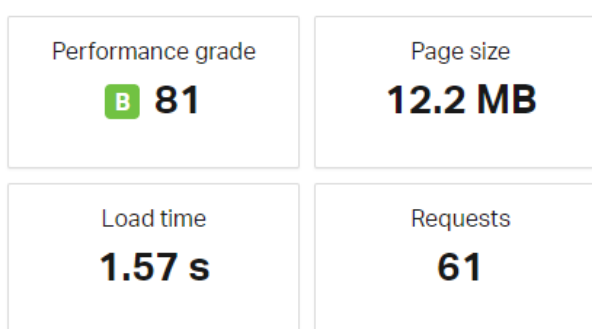


Рис. 3.17 – Аналіз швидкості завантаження веб-застосування засобами Pingdom
Website Speed Test (UK, London)

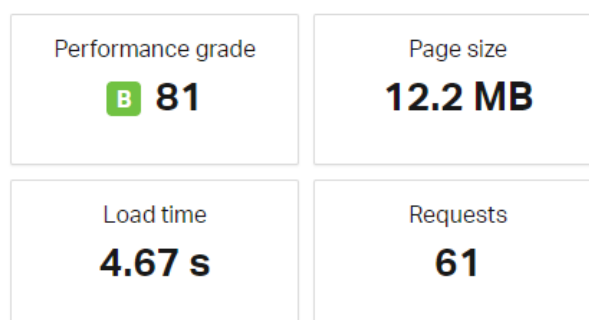


Рис. 3.18 – Аналіз швидкості завантаження веб-застосування засобами Pingdom
Website Speed Test (USA, Washington)

3.2.4 Третій варіант реалізації: Header: HTML, Footer: HTML, Фільтр: React, Галерея: Angular

На Рисунках 3.19 та 3.20 зображено аналіз швидкості завантаження веб-застосування засобами PageSpeed Insights для мобільної версії.

На Рисунках 3.21 та 3.22 зображено аналіз швидкості завантаження веб-застосування засобами PageSpeed Insights для десктопної версії.

На Рисунках 3.23 – 3.25 зображено аналіз швидкості завантаження веб-застосування засобами Pingdom Website Speed Test.

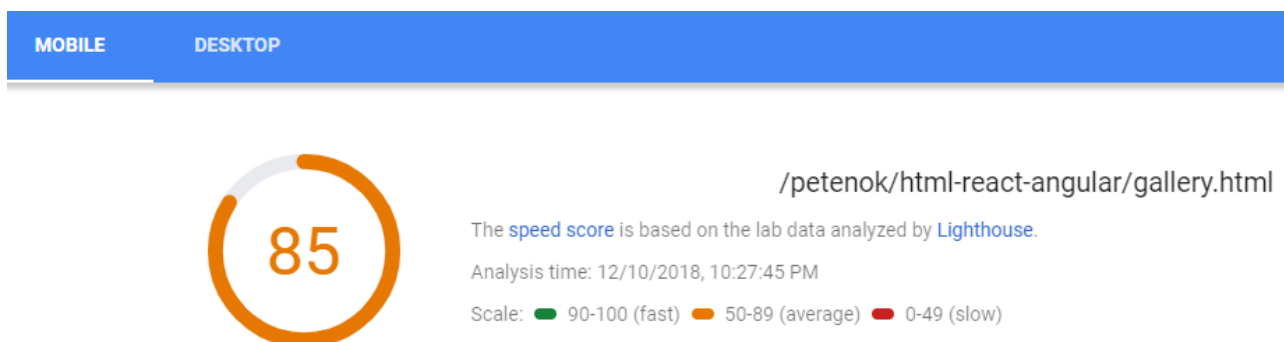


Рис. 3.19 – Аналіз швидкості завантаження веб-застосування засобами PageSpeed Insights для мобільної версії

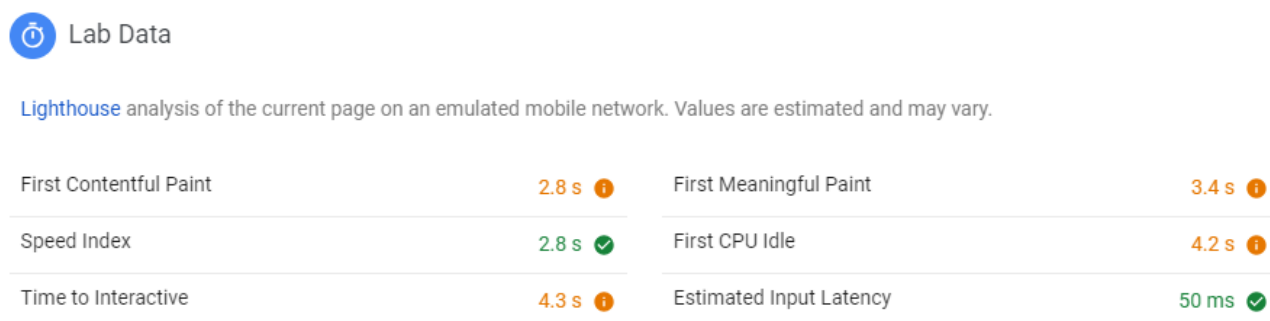


Рис. 3.20 – Аналіз швидкості завантаження веб-застосування засобами PageSpeed Insights для мобільної версії

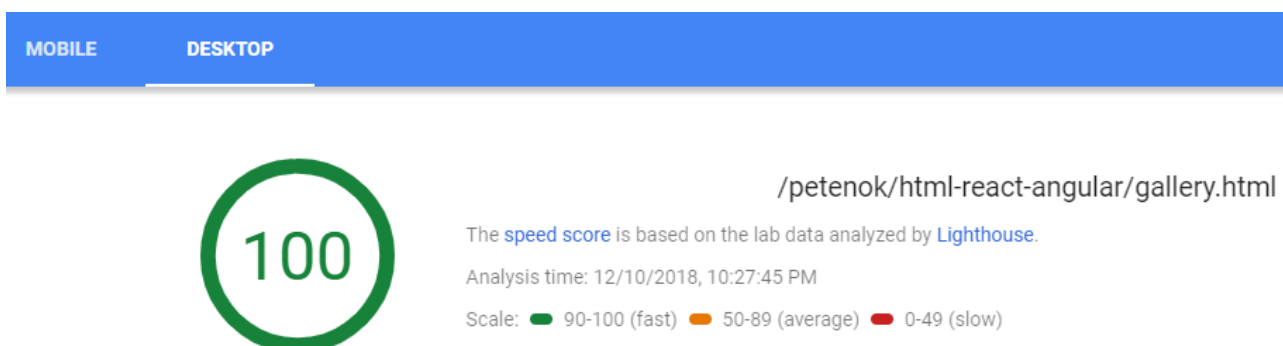


Рис. 3.21 – Аналіз швидкості завантаження веб-застосування засобами PageSpeed Insights для десктопної версії

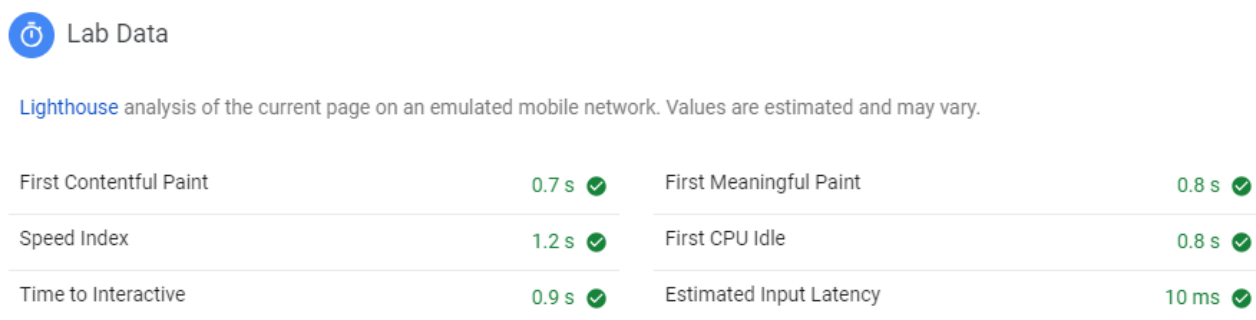


Рис. 3.22 – Аналіз швидкості завантаження веб-застосування засобами PageSpeed Insights для десктопної версії

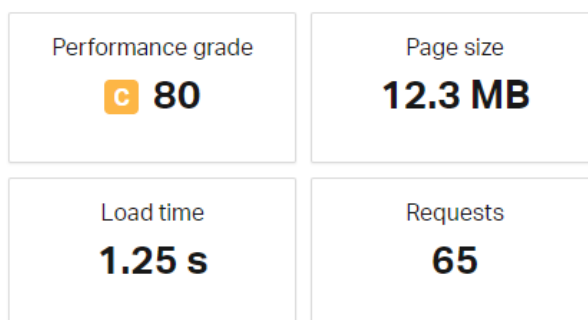


Рис. 3.23 – Аналіз швидкості завантаження веб-застосування засобами Pingdom Website Speed Test (Germany, Frankfurt)

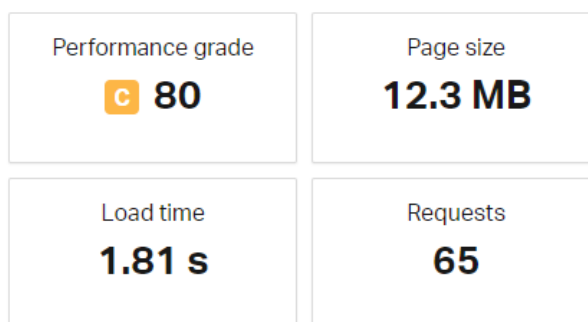


Рис. 3.24 – Аналіз швидкості завантаження веб-застосування засобами Pingdom Website Speed Test (UK, London)

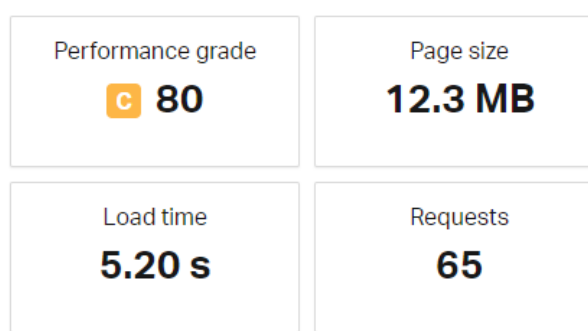


Рис. 3.25 – Аналіз швидкості завантаження веб-застосування засобами Pingdom Website Speed Test (USA, Washington)

3.2.5 Четвертий варіант реалізації: Header: React, Footer: React, Фільтр: React, Галерея: Angular

На Рисунках 3.26 та 3.27 зображено аналіз швидкості завантаження веб-застосування засобами PageSpeed Insights для мобільної версії.

На Рисунках 3.28 та 3.29 зображено аналіз швидкості завантаження веб-застосування засобами PageSpeed Insights для десктопної версії.

На Рисунках 3.30 – 3.32 зображено аналіз швидкості завантаження веб-застосування засобами Pingdom Website Speed Test.

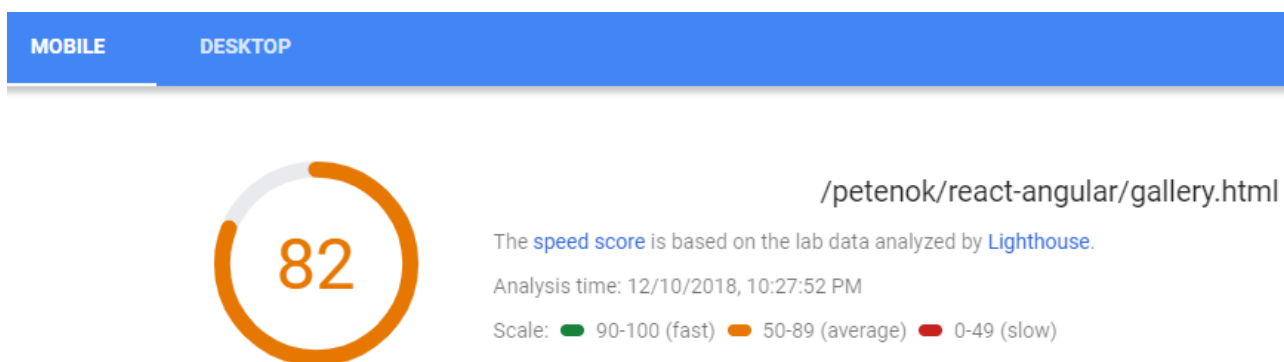


Рис. 3.26 – Аналіз швидкості завантаження веб-застосування засобами PageSpeed Insights для мобільної версії

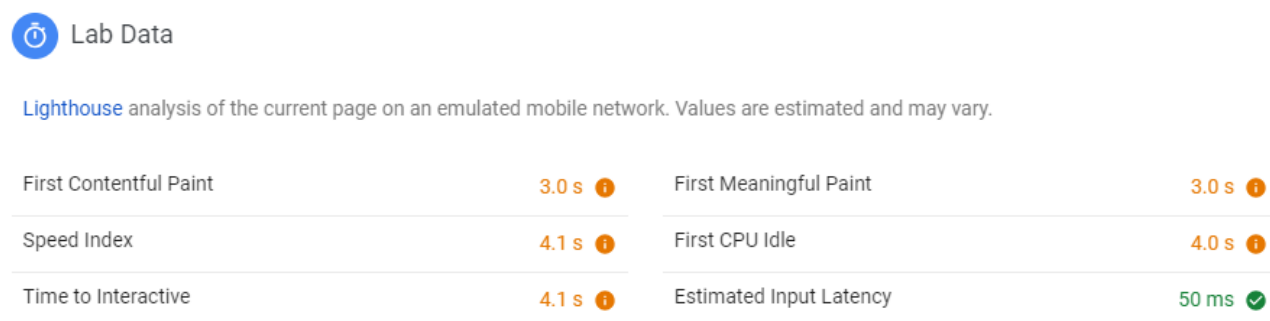


Рис. 3.27 – Аналіз швидкості завантаження веб-застосування засобами PageSpeed Insights для мобільної версії

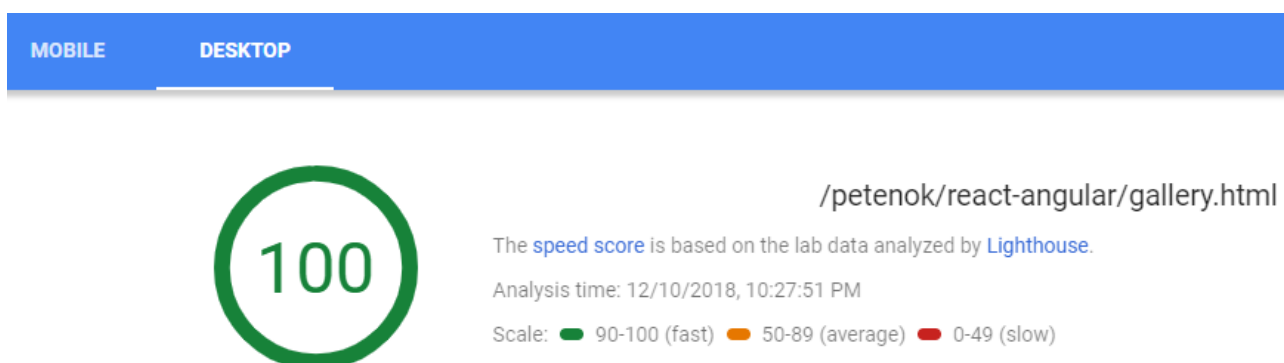



Рис. 3.28 – Аналіз швидкості завантаження веб-застосування засобами PageSpeed Insights для десктопної версії

 Lab Data

[Lighthouse](#) analysis of the current page on an emulated mobile network. Values are estimated and may vary.

First Contentful Paint	0.7 s ✓	First Meaningful Paint	0.7 s ✓
Speed Index	1.4 s ✓	First CPU Idle	0.8 s ✓
Time to Interactive	0.9 s ✓	Estimated Input Latency	10 ms ✓

Рис. 3.29 – Аналіз швидкості завантаження веб-застосування засобами PageSpeed Insights для десктопної версії

Performance grade C 80	Page size 12.3 MB
Load time 1.17 s	Requests 65

Рис. 3.30 – Аналіз швидкості завантаження веб-застосування засобами Pingdom Website Speed Test (Germany, Frankfurt)

Performance grade B 81	Page size 12.2 MB
Load time 2.29 s	Requests 61

Рис. 3.31 – Аналіз швидкості завантаження веб-застосування засобами Pingdom Website Speed Test (UK, London)

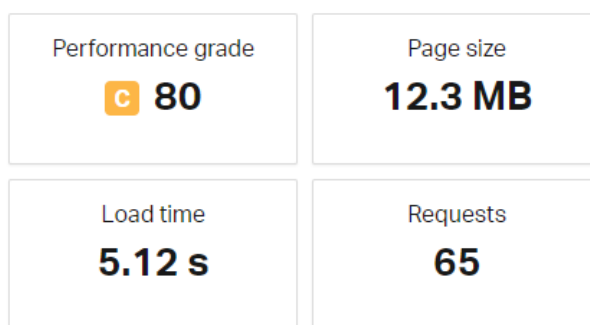


Рис. 3.32 – Аналіз швидкості завантаження веб-застосування засобами Pingdom Website Speed Test (USA, Washington)

3.3 Висновки

У даному розділі представлено:

- опис реалізованої клієнтської частини веб-застосувань (реалізовано чотири різні веб-застосування різними засобами, але з однаковим інтерфейсом користувача);
- аналіз швидкості завантаження веб-застосувань, розроблених різними засобами, але з однаковим інтерфейсом користувача.

Засоби реалізації веб-застосувань зведено в таблицю 3.1.

Таблиця 3.1 – Засоби реалізації веб-застосувань

№	Засіб реалізації сервісу Header	Засіб реалізації сервісу Footer	Засіб реалізації сервісу Фільтр	Засіб реалізації сервісу Галерея
1	HTML	HTML	React	React
2	React	React	React	React
3	HTML	HTML	React	Angular
4	React	React	React	Angular

Для аналізу всі веб-застосування було завантажено на хостинг. Аналіз проведено веб-застосуваннями: Page Speed Insights; Pingdom Website Speed Test. Результати аналізу зведено в таблицю 3.2.

Таблиця 3.2 – Результати аналізу швидкості завантаження реалізованих веб-застосувань

All data is presented in seconds (s)	Page Speed Insights:		Pingdom Website Speed Test, desktop			Average
	Mobile	Desktop	Germany, Frankfurt	UK, London	USA, Washington	
HTML, React	3.80	0.80	1.33	1.65	4.85	2,486
React	4.00	0.90	1.25	1.57	4.67	2,478
HTML, React, Angular	4.30	0.90	1.25	1.81	5.20	2,692
React, Angular	4.10	0.90	1.17	2.29	5.12	2,71

Висновки за результатами аналізу швидкості завантаження реалізованих веб-застосувань:

- використання виключно HTML в статичних сервісах інколи дає виграв у швидкості завантаження, а інколи – ні, тому рішення про доцільність його використання можна приймати на початку розробки проекту, в залежності від архітектури розроблюваного веб-застосування;
- використання однакового фреймворку для різних сервісів щоразу давало невеликий виграв у швидкості, тобто веб-застосування, реалізовані компонентним підходом працюють трохи швидше за веб-застосування, реалізовані мікросервісним підходом;

- виграш у швидкості компонентного підходу над мікросервісним зовсім невеликий, а «свободу дій» для команд розробників мікросервісний підхід дає величезну в порівнянні з компонентрим, тому обидва підходи валідні, а вибір використовуваного залежить від політики компанії та величини проекту (чим більший проект, тим більша ймовірність схилитись на користь мікросервісного підходу).

4 РОЗРОБКА СТАРТАП-ПРОЕКТУ

4.1 Опис ідеї стартап-проекту

Розділ описує розробку стартап-проекту на тему “Мікросервісний підхід до розробки клієнтської частини веб-застосунків”.

Метою розділу є формування інноваційного мислення, підприємницького духу та формування здатностей щодо оцінювання ринкових перспектив і можливостей комерціалізації основних науково-технічних розробок, сформованих у попередній частині магістерської дисертації у вигляді розроблення концепції стартап-проекту в умовах висококонкурентної ринкової економіки глобалізаційних процесів.

Опис ідеї стартап-проекту “Мікросервісний підхід до розробки клієнтської частини веб-застосунків” наведено у Таблиці 4.1.

Таблиця 4.1 – Опис ідеї стартап-проекту

Зміст ідеї	Напрямки застосування	Вигоди для користувача
Надавання послуг розробки клієнтської частини веб-застосунків за мікросервісного підходу.	Розробка веб-застосунків.	Розміщення в мережі Інтернет необхідної інформації у структурованому вигляді
	Частковий рефакторинг існуючих веб-застосунків.	Можливість додавати новий функціонал до веб-застосунків, майже не проводячи рефакторинг старого функціоналу.

Аналіз потенційних техніко-економічних переваг ідеї порівняно із пропозиціями конкурентів передбачає:

- визначення переліку техніко-економічних властивостей та характеристик ідеї;
- визначення попереднього кола конкурентів, проектів-конкурентів, товарів-замінників чи товарів-аналогів, що вже існують на ринку;
- збір інформації щодо значень техніко-економічних показників для ідеї власного проекту та проектів-конкурентів.

Відповідно до визначеного вище переліку проводиться порівняльний аналіз показників: для власної ідеї визначаються показники, що мають:

- гірші значення (W, слабкі);
- аналогічні (N, нейтральні) значення;
- кращі значення (S, сильні).

Визначення сильних, слабких та нейтральних характеристик ідеї стартап-проекту “Мікросервісний підхід до розробки клієнтської частини веб-застосувань” наведено у Таблиці 4.2.

Таблиця 4.2 – Визначення сильних, слабких та нейтральних характеристик ідеї стартап-проекту

№ п/ п	Техніко- економічні характеристик и ідеї	(Потенційні) товари/концепції конкурентів			W	N	S
		Мій проект	Конкурент 1	Конкурент 2			
1.	Форма виконання	Надавання послуг	Надавання послуг	Надавання послуг		+	
2.	Собівартість	Низька	Висока	Середня		+	
3.	Наявність індивідуального підходу до кожної задачі	Є	Нема	Є			+
4.	Потреба в обчислюваних ресурсах	Є	Нема	Є	+		
5.	Швидкість виконання	Середня	Низька	Середня		+	

Визначений перелік слабких, сильних та нейтральних характеристик та властивостей ідеї потенційного товару є підґрунтям для формування його конкурентоспроможності.

4.2 Технологічний аудит ідеї стартап-проекту

В межах даного підрозділу необхідно провести аудит технології, за допомогою якої можливо реалізувати ідею проекту (технології створення товару).

Визначення технологічної здійсненності ідеї проекту передбачає аналіз таких складових:

- за якою технологією буде виготовлено товар згідно ідеї проекту;
- чи існують такі технології, чи їх потрібно розробити/добробити;
- чи доступні такі технології авторам проекту?

Технологічну здійсненність ідеї стартап-проекту “Мікросервісний підхід до розробки клієнтської частини веб-застосувачів” наведено у Таблиці 4.3.

Таблиця 4.3 – Технологічна здійсненність ідеї стартап-проекту

№ п/п	Ідея проекту	Технології реалізації	Наявність технології	Доступність технології
	Надавання послуг розробки клієнтської частини веб-застосувачів за мікросервісного підходу.	Сучасні фреймворки для розробки веб-застосувачів: React, Angular, Vue.	Наявна	Доступна
Обрано технології реалізації ідеї проекту: фреймворки React, Angular, Vue – безкоштовно та доступно.				

4.3 Аналіз ринкових можливостей запуску стартап-проекту

Визначення ринкових можливостей, які можна використати під час ринкового впровадження проекту, та ринкових загроз, які можуть зашкодити реалізації проекту, дозволяє спланувати напрями розвитку проекту з урахуванням стану ринкового середовища, потреб потенційних клієнтів та пропозицій проектів-конкурентів.

В ході таких досліджень вивчаються особливості і перспективи розвитку попиту на конкретні товари, позиції конкурентів на ринку, їх сильні і слабкі сторони, динаміку цін і т.д. Стартап-проекту важливо знати, чи буде обсяг продажів його товарів достатнім для компенсації зусиль щодо виходу на ринок, тому важливою характеристикою ринку є його ємність, під якою розуміють максимально можливий обсяг продажу певного товару протягом року, виражений в натуральних і вартісних одиницях.

Попит на більшість товару, який визначає місткість ринку, характеризується нестабільністю. Тому кожне підприємство прагне мати достовірний прогноз попиту на свій товар. З метою стимулювання збільшення попиту на товар необхідно вивчити і проаналізувати думки і потреби споживачів певного товару.

Попередню характеристику потенційного ринку стартап-проекту “Мікросервісний підхід до розробки клієнтської частини веб-застосунків” наведено у Таблиці 4.4.

Таблиця 4.4 – Попередня характеристика потенційного ринку стартап-проекту

Показники стану ринку (найменування)	Характеристика
Кількість головних гравців, од	5000
Загальний обсяг продажу, грн/ум.од	25000 грн/у.о
Динаміка ринку (якісна оцінка)	Зростає
Наявність обмежень для входу (вказати характер обмежень)	Відсутні
Специфічні вимоги до стандартизації та сертифікації	Відсутні
Середня норма рентабельності в галузі (або по ринку), %	10%

Отже, проаналізовано наявність попиту, обсяг, динаміку розвитку ринку. Обмеження для входу на ринок відсутні, динаміка ринку зростає, галузь є рентабельною.

Далі визначаються потенційні групи клієнтів, їх характеристики, та формується орієнтовний перелік вимог до товару для кожної групи.

Характеристику потенційних клієнтів стартап-проекту “Мікросервісний підхід до розробки клієнтської частини веб-застосувань” наведено у Таблиці 4.5.

Таблиця 4.5. – Характеристика потенційних клієнтів стартап-проекту

№ п/п	Потреба, що формує ринок	Цільова аудиторія (цільові сегменти ринку)	Відмінності у поведінці різних потенційних цільових груп клієнтів	Вимоги споживачів до товару
1	Отримати якісно написане та швидко працююче веб- застосування.	Аудиторія: клієнти, що потребують веб- застосувань. Сегменти: індивідуальні користувачі, підприємства будь-якого розміру.	Для сегменту дрібних користувачів більш характерні невеликі веб- застосування. Підприємства зацікавлені у великих веб-застосуваннях, їх постійному оновленні та підтримці.	Усім споживачам важливі якість та швидкість роботи веб- застосувань.

Після визначення потенційних груп клієнтів проводиться аналіз ринкового середовища: складаються таблиці факторів, що сприяють ринковому впровадженню проекту, та факторів, що йому перешкоджають. Фактори в таблицях подають в порядку зменшення значущості.

Ринкові можливості – це сприятливі обставини, які підприємство може використовувати для отримання переваг. Слід зазначити, що можливостями з погляду SWOT-аналізу є не всі можливості, які існують на ринку, а тільки ті, які можна використовувати.

Ринкові загрози – події, настання яких може несприятливо вплинути на підприємство.

Фактори загроз стартап-проекту “Мікросервісний підхід до розробки клієнтської частини веб-застосувань” наведено у Таблиці 4.6.

Фактори можливостей стартап-проекту “Мікросервісний підхід до розробки клієнтської частини веб-застосувань” наведено у Таблиці 4.7.

Таблиця 4.6. - Фактори загроз

№ п/п	Фактор	Зміст загрози	Можлива реакція компанії
1	Зростаюча конкуренція.	Зі зростом попиту на розробку веб-застосувань зростає і пропозиція.	Розробляти веб-застосування високої якості.

Таблиця 4.7 – Фактори можливостей

№ п/п	Фактор	Зміст можливості	Можлива реакція компанії
1	Зростаючий попит.	Збільшення попиту на веб-застосування.	Надавати високоякісні рішення, займати нішу ринку.
2	Оптимізація швидкості завантаження.	Оптимізація швидкості завантаження веб-застосувань.	Оптимізація швидкості завантаження за рахунок рефакторингу, асинхронності, мінімізації файлів кінцевого веб-застосування та оптимізації стиснення зображень.

Ступеневий аналіз конкуренції на ринку стартап-проекту “Мікросервісний підхід до розробки клієнтської частини веб-застосувань” наведено у Таблиці 4.8.

Таблиця 4.8 – Ступеневий аналіз конкуренції на ринку

Особливості конкурентного середовища	В чому проявляється дана характеристика	Вплив на діяльність підприємства (можливі дії компанії, щоб бути конкурентоспроможною)
1. Вказати тип конкуренції: чиста.	Існує величезна кількість конкурентів на ринку.	Якісно провести рекламу.
2. За рівнем конкурентної боротьби: локальний.	На компанію впливатиме конкуренція в Україні.	Розробляти якісний продукт.
3. За галузевою ознакою: міжгалузева.	Веб-застосування розробляються незалежно від галузі їх призначення.	Залучення будь-яких клієнтів.
4. Конкуренція за видами товарів: між бажаннями.	Існує багато варіацій створення веб-застосувань.	Прислухатись до побажань клієнта.
5. За характером конкурентних переваг: цінова.	Цінова категорія сильно варіюється.	Враховувати ціни конкурентних компаній на початкових етапах.
6. За інтенсивністю: не марочна.	Значення мають технології і ціни, а не бренд.	Використовувати сучасні технології.

Аналіз конкуренції в галузі за М. Портером стартап-проекту “Мікросервісний підхід до розробки клієнтської частини веб-застосувань” наведено у Таблиці 4.9.

Таблиця 4.9 – Аналіз конкуренції в галузі за М. Портером

Складові аналізу	Прямі конкуренти	Потенційні конкуренти	Постачальники	Клієнти	Замінники
	Веб-студії.	Великі компанії, що створюють відділи веб-розробки.	Відсутні.	Важливою є швидкість завантаження веб-застосувань.	Сервіси з шаблонами веб-застосувань.
Висновки	Велика інтенсивність конкуренції.	Потенційно, при розширенні, компанії створюють веб-відділи.	Постачальники не мають впливу.	Клієнти мають сильний вплив на роботу на ринку.	Замінники пропонують більш доступну і менш якісну технологію.

Обґрунтування факторів конкурентоспроможності стартап-проекту “Мікросервісний підхід до розробки клієнтської частини веб-застосувань” наведено у Таблиці 4.10.

Таблиця 4.10 – Обґрунтування факторів конкурентоспроможності

№ п/п	Фактор конкурентоспроможності	Обґрунтування (наведення чинників, що роблять фактор для порівняння конкурентних проектів значущим)
1	Технологічність	Пропонування якісніших, краще оптимізованих, з полегшеним внесенням змін веб-застосувань.
2	Мікросервісний підхід до розробки.	Запровадження мікросервісного підходу до розробки, яким ще не насичений ринок.

Порівняльний аналіз сильних та слабких сторін стартап-проекту “Мікросервісний підхід до розробки клієнтської частини веб-застосувань” наведено у Таблиці 4.11.

Таблиця 4.11 – Порівняльний аналіз сильних та слабких сторін

№ п/п	Фактор конкурентоспроможності	Бали 1-20	Рейтинг товарів-конкурентів						
			-3	-2	-1	0	+1	+2	+3
1	Технологічність	15					✓		
2	Мікросервісний підхід до розробки	6		✓					

Перелік ринкових загроз та ринкових можливостей складається на основі аналізу факторів загроз та факторів можливостей маркетингового середовища.

Ринкові загрози та ринкові можливості є наслідками (прогнозованими результатами) впливу факторів, і, на відміну від них, ще не є реалізованими на ринку та мають певну ймовірність здійснення. Наприклад: зниження доходів потенційних споживачів – фактор загрози, на основі якого можна зробити прогноз щодо посилення значущості цінового фактору при виборі товару та відповідно, – цінової конкуренції (а це вже – ринкова загроза).

SWOT аналіз стартап-проекту “Мікросервісний підхід до розробки клієнтської частини веб-застосунків” наведено у Таблиці 4.12.

Таблиця 4.12 – SWOT аналіз

<p>Сильні сторони: розробка якісного продукту: оптимізація швидкості завантаження за рахунок рефакторингу, асинхронності, мінімізації файлів кінцевого веб-застосування та оптимізації стиснення зображень.</p>	<p>Слабкі сторони: дуже насичений ринок.</p>
<p>Можливості: насичення ринку новим підходом до розробки; різноманітна клієнтура.</p>	<p>Загрози: конкуренція.</p>

Альтернативи ринкового впровадження стартап-проекту “Мікросервісний підхід до розробки клієнтської частини веб-застосунків” наведено у Таблиці 4.13.

Таблиця 4.13 – Альтернативи ринкового впровадження стартап-проекту

№ п/п	Альтернатива (орієнтовний комплекс заходів) ринкової поведінки	Ймовірність отримання ресурсів	Строки реалізації
1	Розробка веб-застосунків за мікросервісного підходу.	80%	6 місяців
2	Розробка веб-застосунків за компонентного підходу.	80%	6 місяців
3	Розробка веб-застосунків з орієнтуванням на серверний рендерінг.	30%	12 місяців

Обрано першу та другу альтернативи, бо вони мають більшу ймовірність отримання ресурсів та менші строки реалізації.

4.4 Розробка ринкової стратегії стартап-проекту

Розробка ринкової стратегії передбачає визначення стратегії охоплення ринку: опис цільових груп потенційних споживачів.

Вибір цільових груп потенційних споживачів стартап-проекту “Мікросервісний підхід до розробки клієнтської частини веб-застосунків” наведено у Таблиці 4.14.

Таблиця 4.14 – Вибір цільових груп потенційних споживачів

№ п / п	Опис профілю цільової групи потенційних клієнтів	Готовність споживачів сприйняти продукт	Орієнтовний попит в межах цільової групи (сегменту)	Інтенсивн ість конкурен ції в сегменті	Простота входу у сегмент
1	Великі компанії.	Середня: велика конкуренція і можливість власних веб-відділів.	Середній.	Велика.	Важко.
2	Маленькі компанії.	Велика.	Високий.	Середня.	Досить легко.
3	Приватні особи.	Низька. Приватні особи воліють продукт за найменшу ціну і не обов'язково якісний.	Низький.	Середня.	Середня.

Обрано цільову групу: маленькі компанії.

За результатами аналізу потенційних груп споживачів (сегментів) обирають цільові групи, для яких пропонуватимуть товар, та визначають стратегію охоплення ринку:

- якщо компанія зосереджується на одному сегменті – вона обирає стратегію концентрованого маркетингу;
- якщо працює із кількома сегментами, розробляючи для них окремо програми ринкового впливу – вона використовує стратегію диференційованого маркетингу;
- якщо компанія працює із всім ринком, пропонуючи стандартизовану програму (включно із характеристиками товару/послуги) – вона використовує масовий маркетинг.

Обрано стратегію концентрованого маркетингу.

Визначення базової стратегії розвитку стартап-проекту “Мікросервісний підхід до розробки клієнтської частини веб-застосувань” наведено у Таблиці 4.15.

Таблиця 4.15 – Визначення базової стратегії розвитку

№ п/п	Обрана альтернатива розвитку проекту	Стратегія охоплення ринку	Ключові конкурентоспроможні позиції відповідно до обраної альтернативи
1	Розробка веб-застосувань за мікросервісного підходу.	Ринкове позиціонування на маленькі компанії.	Швидкодія, якість продукту.

Наступним кроком є вибір стратегії конкурентної поведінки.

Стратегія лідера. Залежно від міри сформованості товарного (галузевого) ринку, характеру конкурентної боротьби компанії-лідери обирають одну з трьох стратегій: розширення первинного попиту, оборонну або наступальну стратегію або ж застосувати демаркетинг або диверсифікацію.

Стратегія розширення первинного попиту доцільна у разі, якщо фірмі-лідеріві недоцільно розмінюватися на боротьбу з невеликими конкурентами, вона може отримати велику економічну віддачу від розширення первинного рівня попиту. В цьому випадку компанія займається реалізацією заходів по формуванню попиту (навчанню споживачів користуванню товаром, формування регулярного попиту, збільшення разового споживання), також пропаганду нових напрямів застосувань існуючих товарів, виявлень нових груп споживачів. У міру зростання ринку, його становлення позиції компанії-новатора починають атакувати конкуренти-імітатори. В цьому випадку, компанія може вибрати оборонну стратегію, метою якої є захист власної ринкової долі. Наступальна стратегія припускає збільшення своєї частки ринку. При цьому переслідувана мета полягає в подальшому підвищенні прибутковості роботи компанії на ринку за рахунок максимального використання ефекту масштабу. Якщо фірма потрапляє під дію антимонопольного законодавства, вона може удатися до стратегії демаркетинга, що припускає скорочення своєї частки ринку, зниження рівня попиту на деяких сегментах за рахунок підвищення ціни. При цьому ставиться завдання недопущення на ці сегменти конкурентів, а компенсація втрат прибутку через зменшення обсягів виробництва компенсується встановленням надвисоких цін.

Стратегія виклик у лідера. Стратегію виклику лідеріві найчастіше вибирають компанії, які є другими, третіми на ринку, але бажають стати лідером ринку. Теоретично, ці компанії можуть прийняти два стратегічні

рішення: атакувати лідера у боротьбі за частку ринку або ж йти за лідером. Рішення атакувати лідера є досить ризикованим. Для його реалізації потрібні значні фінансові витрати, know – how, краще співвідношення «ціна- якість», переваги в системі розподілу і просування і т. д. У разі не реалізації цієї стратегії, компанія може бути відкинута на аутсайдерські позиції на досить довгий час. Залежно від цього компанія може вибрати одну з альтернативних стратегій: фронтальної або флангової атаки. Стратегія наслідування лідера у . Компанії, що приймають слідування за лідером – це підприємства з невеликою часткою ринку, які вибирають адаптивну лінію поведінки на ринку, усвідомлюють своє місце на ній і йдуть у фарватері фірм-лідерів. Головна перевага такої стратегії – економія фінансових ресурсів, пов'язаних з необхідністю розширення товарного(галузевого) ринку, постійними інноваціями, витратами на утримання домінуючого положення.

Стратегія заняття конкурентної ніші. При прийнятті стратегії зайняття конкурентної ніші (інші назви – стратегія фахівця або нішера) компанія в якості цільового ринку вибирає один або декілька ринкових сегментів. Головна особливість – малий розмір сегментів/сегменту. Ця конкурентна стратегія являється похідною від такої базової стратегії компанії, як концентрація. Головне завдання для компаній, що вибирають стратегію нішера або фахівця, – це постійна турбота про підтримку і розвиток своєї конкурентної переваги, формування лояльності і прихильності споживачів, підтримка вхідних бар'єрів.

Визначення базової стратегії конкурентної поведінки стартап-проекту “Мікросервісний підхід до розробки клієнтської частини веб-застосунків” наведено у Таблиці 4.16.

Таблиця 4.16 – Визначення базової стратегії конкурентної поведінки

№ п / п	Чи є проект «першопрохідцем» на ринку?	Чи буде компанія шукати нових споживачів, або забирати існуючих у конкурентів?	Чи буде компанія копіювати основні характеристики товару конкурента, і які?	Стратегія Конкурентної поведінки
1	Ні.	Компанія буде шукати нових споживачів та забирати існуючих у конкурентів.	Компанія пропонує новий підхід до створення продукту.	Якісний продукт з високою швидкодією.

На основі вимог споживачів з обраних сегментів до постачальника (стартап-компанії) та до продукту, а також в залежності від обраної базової стратегії розвитку та стратегії конкурентної поведінки розробляється стратегія позиціонування, що полягає у формуванні ринкової позиції (комплексу асоціацій), за яким споживачі мають ідентифікувати торгівельну марку/проект.

Визначення стратегії позиціонування стартап-проекту “Мікросервісний підхід до розробки клієнтської частини веб-застосувачів” наведено у Таблиці 4.17.

Таблиця 4.17 – Визначення стратегії позиціонування

№ п/п	Вимоги до товару цільової аудиторії	Базова стратегія розвитку	Ключові конкуренто- спроможні позиції власного стартап-проекту	Вибір асоціацій, які мають сформувати комплексну позицію власного проекту (три ключових)
1	Споживачам важливі якість та швидкість роботи веб-застосунків.	Диференціація.	Висока якість продукту: оптимізація швидкості завантаження за рахунок рефакторингу, асинхронності, мінімізації файлів кінцевого веб-застосування та оптимізації стиснення зображень.	Швидкість завантаження, оптимізація, легке внесення змін в готовий проект.

4.5 Розробка маркетингової програми стартап-проекту

Першим кроком є формування маркетингової концепції товару, який отримає споживач.

Визначення ключових переваг концепції потенційного товару стартап-проекту “Мікросервісний підхід до розробки клієнтської частини веб-застосунків” наведено у Таблиці 4.18.

Таблиця 4.18 – Визначення ключових переваг концепції потенційного товару

№ п/п	Потреба	Вигода, яку пропонує товар	Ключові переваги перед конкурентами (існуючі або такі, що потрібно створити)
1	Якісні веб-застосування із швидким завантаженням.	Високі якість та швидкість роботи веб-застосовань.	Переваги у швидкості та якості розробки веб-застосовань.

Надалі розробляється трирівнева маркетингова модель товару: уточнюється ідея продукту та / або послуги, його фізичні складові, особливості процесу його надання.

1-й рівень. При формуванні задуму товару вирішується питання щодо того, засобом вирішення якої потреби і / або проблеми буде даний товар, яка його основна вигода. Дане питання безпосередньо пов'язане з формуванням технічного завдання в процесі розробки конструкторської документації на виріб.

2-й рівень. Цей рівень являє рішення того, як буде реалізований товар в реальності, включає в себе якість, властивості, дизайн, упаковку, ціну.

3-й рівень. Товар з підкріпленням (супроводом) – додаткові послуги та переваги для споживача, що створюються на основі товару за задумом і товару в реальному виконанні (гарантії якості , доставка, умови оплати та ін).

Опис трьох рівнів моделі товару стартап-проекту “Мікросервісний підхід до розробки клієнтської частини веб-застосовань” наведено у Таблиці 4.19.

Таблиця 4.19 – Опис трьох рівнів моделі товару

Рівні товару	Сутність та складові		
1.Товар за задумом	Використання мікросервісного підходу до розробки клієнтської частини веб-застосувань. (З точки зору користувача, товар – веб-застосування.)		
2. Товар у реальному виконанні	Властивості/характеристики	М/Нм	Вр/Тх /Тл/Е/Ор
	1. Індивідуальний підхід.	1.Нм	1.Технологічна
	2. Швидке завантаження.	2.Нм	2.Технологічна
	3. Легке внесення змін у готовий продукт.	3.Нм	3.Технологічна
	Якість: згідно зі стандартом ISO 4444 буде проведено тестування.		
	Маркування відсутнє.		
	Моя компанія: “PetenokOlena”.		
3. Товар із підкріпленням	Постійна підтримка для користувачів.		
За рахунок чого потенційний товар буде захищено від копіювання: ноу-хау.			

Після формування маркетингової моделі товару слід особливо відмітити – чим саме проект буде захищено від копіювання. Захист може бути організовано за рахунок захисту ідеї товару (захист інтелектуальної власності), або ноу-хау,

чи комплексне поєднання властивостей і характеристик, закладене на другому та третьому рівнях товару.

Наступним кроком є визначення цінових меж, якими необхідно керуватись при встановленні ціни на потенційний товар (остаточне визначення ціни відбувається під час фінансово-економічного аналізу проекту), яке передбачає аналіз ціни на товари-аналоги або товари субституту, а також аналіз рівня доходів цільової групи споживачів. Аналіз проводиться експертним методом.

Визначення меж встановлення ціни стартап-проекту “Мікросервісний підхід до розробки клієнтської частини веб-застосувань” наведено у Таблиці 4.20.

Таблиця 4.20 – Визначення меж встановлення ціни

№ п/п	Рівень цін на товари замінники, грн	Рівень цін на товари аналоги, грн.	Рівень доходів цільової групи споживачів, грн.	Верхня та нижня межі встановлення ціни на товар/послугу, грн.
1.	5000	30000	15000 – 60000	5000 – 80000

Наступним кроком є визначення оптимальної системи збуту, в межах якого приймається рішення:

- проводити збут власними силами або залучати сторонніх посередників (власна або залучена система збуту);
- вибір та обґрунтування оптимальної глибини каналу збуту;
- вибір та обґрунтування виду посередників.

Формування системи збуту стартап-проекту “Мікросервісний підхід до розробки клієнтської частини веб-застосунків” наведено у Таблиці 4.21.

Таблиця 4.21 – Формування системи збуту

№ п/п	Специфіка закупівельної поведінки цільових клієнтів	Функції збуту, які має виконувати постачальник товару	Глибина каналу збуту	Оптимальна система збуту
1.	Купують індивідуальні веб-застосунки різного масштабу та функціоналу, в залежності від потреб. Можуть також купувати місячну підтримку веб-застосунків, розроблених компанією.	Продаж.	0 (напрям), 1 (через одного посередника)	Власна та через посередників.

Останньою складовою маркетингової програми є розроблення концепції маркетингових комунікацій, що спирається на попередньо обрану основу для позиціонування, визначену специфіку поведінки клієнтів.

Концепція маркетингових комунікацій стартап-проекту “Мікросервісний підхід до розробки клієнтської частини веб-застосунків” наведено у Таблиці 4.22.

Таблиця 4.22 – Концепція маркетингових комунікацій

№ п/п	Специфіка поведінки цільових клієнтів	Канали комунікацій, якими користуютьс я цільові клієнти	Ключові позиції, обрані для позиціонуван ня	Завдання рекламног о повідомлен ня	Концепція Рекламног о звернення
1	Замовляють і купують після зустрічі з менеджером та домовленості щодо контенту та функціоналу.	Інтернет, мобільний зв'язок з менеджером.	Швидкодія, якість, легкість внесення змін в готове веб-застосування.	Показати переваги мікросервісного підходу розробки веб-застосувань.	Пояснення переваг мікросервісного підходу до розробки веб-застосувань.

4.6 Висновки

У даному розділі проведено повний аналіз стартап-проекту “Мікросервісний підхід до розробки клієнтської частини веб-застосувань”:

- описано ідею стартап-проекту;
- проведено технологічний аудит ідеї стартап-проекту;
- виконано аналіз ринкових можливостей запуску стартап-проекту;
- розроблено ринкову стратегію стартап-проекту;
- розроблено маркетингову програму стартап-проекту.

В результаті проведеного аналізу, можна стверджувати, що:

- є можливість ринкової комерціалізації проекту (наявний попит на послуги);
- є перспективи впровадження з огляду на потенційні групи клієнтів, бар'єри входження, стан конкуренції, конкурентоспроможність проекту;
- для ринкової реалізації проекту доцільно обрати впровадження розробки веб-застосунків за мікросервісного або компонентного підходу;
- подальша імплементація проекту є доцільною.

ВИСНОВКИ

В ході магістерської дисертації було:

- досліджено застосування мікросервісного підходу до розробки клієнтської частини веб-застосувань;
- створено власне веб-застосування за мікросервісного підходу до розробки;
- протестовано швидкість завантаження декількох варіантів власного веб-застосування за розробки різних сервісів різними засобами з однаковим інтерфейсом користувача та проаналізовано отримані результати.

Для цього було вирішено наступні задачі:

- проведено огляд та аналіз предметної області;
- спроектовано архітектуру власного веб-застосування за мікросервісного підходу до розробки;
- реалізовано декількох варіантів власного веб-застосування за розробки різних сервісів різними засобами з однаковим інтерфейсом користувача;
- протестовано швидкість завантаження всіх реалізованих варіантів власного веб-застосування та проаналізовано отримані результати.

У розділі огляду предметної області було:

- проведено аналіз монолітного, компонентного та мікросервісного підходів (як в загальному, так і з точки зору розборки клієнтської частини веб-застосувань);
- зазначено переваги та недоліки монолітного, компонентного та мікросервісного підходів з точки зору розборки клієнтської частини веб-застосувань;
- досліджено варіанти імплементування реалізації клієнтської частини веб-застосувань за мікросервісного підходу розробки (використання

компонентів, спільних подій або сторонніх бібліотек в якості інтеграційного шару);

- представлено варіанти існуючих реалізацій клієнтської частини веб-застосувань за мікросервісного підходу до розробки.

У розділі опису засобів реалізації було:

- проведено огляд існуючих популярних JavaScript фреймворків для реалізації клієнтської частини веб-застосувань;
- обрано засоби реалізації клієнтської частини власного веб-застосування;
- описано архітектуру клієнтської частини власного веб-застосування.

Архітектура власного веб-застосування складається з чотирьох сервісів:

- header;
- footer;
- фільтр (ширини відображення списків зображень в галереї);
- галерея (відображення зображень).

Сервіси header, footer та фільтр є повністю незалежними.

Сервіс галерея є залежним від сервісу фільтр за результируючим деревом DOM, але незалежним від реалізації сервісу фільтр.

У елементів `` сервісу фільтр існує по два обробники подій натискання на елемент (один обробляється у сервісі фільтру та відповідає за відображення елемента ``, а інший обробляється у сервісі галереї та відповідає за відображення ширини зображень).

У розділі реалізації та аналізу результатів було представлено:

- опис реалізованої клієнтської частини веб-застосувань (реалізовано чотири різні веб-застосування різними засобами, але з однаковим інтерфейсом користувача);
- аналіз швидкості завантаження веб-застосувань, розроблених різними засобами, але з однаковим інтерфейсом користувача.

Засоби реалізації веб-застосувань було зведено в таблицю 3.1.

Для аналізу всі веб-застосування було завантажено на хостинг. Аналіз проведено веб-застосуваннями: Page Speed Insights; Pingdom Website Speed Test. Результати аналізу було зведено в таблицю 3.2.

Висновки за результатами аналізу швидкості завантаження реалізованих веб-застосувань:

- використання виключно HTML в статичних сервісах інколи дає виграш у швидкості завантаження, а інколи – ні, тому рішення про доцільність його використання можна приймати на початку розробки проекту, в залежності від архітектури розроблюваного веб-застосування;
- використання однакового фреймворку для різних сервісів щоразу давало невеликий виграш у швидкості, тобто веб-застосування, реалізовані компонентним підходом працюють трохи швидше за веб-застосування, реалізовані мікросервісним підходом;
- виграш у швидкості компонентного підходу над мікросервісним зовсім невеликий, а «свободу дій» для команд розробників мікросервісний підхід дає величезну в порівнянні з компонентрим, тому обидва підходи валідні, а вибір використовуваного залежить від політики компанії та величини проекту (чим більший проект, тим більша ймовірність схилитись на користь мікросервісного підходу).

У розділі розробки стартап-проекту було проведено повний аналіз стартап-проекту “Мікросервісний підхід до розробки клієнтської частини веб-застосунків”:

- описано ідею стартап-проекту;
- проведено технологічний аудит ідеї стартап-проекту;
- виконано аналіз ринкових можливостей запуску стартап-проекту;
- розроблено ринкову стратегію стартап-проекту;
- розроблено маркетингову програму стартап-проекту.

В результаті проведеного аналізу стартап-проекту, можна стверджувати, що:

- є можливість ринкової комерціалізації проекту (наявний попит на послуги);
- є перспективи впровадження з огляду на потенційні групи клієнтів, бар'єри входження, стан конкуренції, конкурентоспроможність проекту;
- для ринкової реалізації проекту доцільно обрати впровадження розробки веб-застосунків за мікросервісного або компонентного підходу;
- подальша імплементація проекту є доцільною.

Практична цінність результатів магістерської дисертації у:

- аналізі сучасних підходів до розробки клієнтської частини веб-застосунків;
- визначенні переваг та недоліків існуючих підходів до розробки клієнтської частини веб-застосунків;
- дослідженні швидкості завантаження клієнтської частини веб-застосунків за різної реалізації мікросервісного підходу.

Потенційне застосування результатів магістерської дисертації: проектування та розробка клієнтської частини веб-застосунків за мікросервісного підходу, маючи інформацію про переваги та недоліки даного підходу та можливі варіанти імплементування реалізації сервісів.

ПЕРЕЛІК ПОСИЛАНЬ

1. Microservices vs Monolithic Architecture. – Режим доступу:
<https://www.mulesoft.com/resources/api/microservices-vs-monolithic>. – Дата доступу : 04.09.2018.
2. Understanding Component-Based Architecture. – Режим доступу:
<https://medium.com/@dan.shapiro1210/understanding-component-based-architecture-3ff48ec0c238>. – Дата доступу : 06.09.2018.
3. Micro frontends – a microservice approach to front-end web development. – Режим доступу: <https://medium.com/@tomsoderlund/micro-frontends-a-microservice-approach-to-front-end-web-development-f325ebdad16>. – Дата доступу : 07.09.2018.
4. Micro frontends. – Режим доступу: <https://micro-frontends.org/>. – Дата доступу : 10.09.2018.
5. Building UIs in DevOps / microservices environment part 1—frontend monoliths and where the story begins. – Режим доступу:
<https://blog.coffeeapplied.com/building-uis-in-devops-microservices-environment-part-1-frontend-monoliths-and-where-the-story-7860633756c7>. – Дата доступу : 11.09.2018.
6. Building UIs in DevOps / microservices environment part 2— micro-frontends and composite UIs. – Режим доступу: <https://blog.coffeeapplied.com/building-uis-in-devops-microservices-environment-part-2-micro-frontends-and-composite-uis-ab3d4ac394ed>. – Дата доступу : 11.09.2018.
7. Building UIs in DevOps / microservices environment part 3— headless microservices, bounded contexts and conclusions. – Режим доступу:
<https://blog.coffeeapplied.com/building-uis-in-devops-microservices-environment-part-3-headless-microservices-bounded-contexts-1ade60f2350b>. – Дата доступу : 11.09.2018.

8. Using shadow DOM. – Режим доступу: https://developer.mozilla.org/en-US/docs/Web/Web_Components/Using_shadow_DOM. – Дата доступу : 13.09.2018.
9. A javascript metaframework single-spa. – Режим доступу: <https://github.com/CanopyTax/single-spa>. – Дата доступу : 14.09.2018.
10. Project Mosaic. – Режим доступу: <https://www.mosaic9.org/>. – Дата доступу : 14.09.2018.
11. Including Front-End Web Components Into Microservices. – Режим доступу: <https://technologyconversations.com/2015/08/09/including-front-end-web-components-into-microservices/>. – Дата доступу : 17.09.2018.
12. Managing Frontend in the Microservices Architecture. – Режим доступу: <https://allegro.tech/2016/03/Managing-Frontend-in-the-microservices-architecture.html>. – Дата доступу : 18.09.2018.
13. Choosing a framework: React vs Angular vs Vue vs Ember vs... – Режим доступу: <https://medium.com/@Blockium/choosing-a-framework-react-vs-angular-vs-vue-vs-ember-vs-4664062d60d6>. – Дата доступу : 20.09.2018.
14. Angular vs. React vs. Vue: A 2017 comparison. – Режим доступу: <https://medium.com/unicorn-supplies/angular-vs-react-vs-vue-a-2017-comparison-c5c52d620176>. – Дата доступу : 21.09.2018.
15. React. – Режим доступу: <https://reactjs.org/>. – Дата доступу : 24.09.2018.
16. Angular. – Режим доступу: <https://angular.io/>. – Дата доступу : 24.09.2018.
17. Vue. – Режим доступу: <https://vuejs.org/>. – Дата доступу : 27.09.2018.
18. Стаття Вікіпедії «Component-based software engineering». – Режим доступу: https://en.wikipedia.org/wiki/Component-based_software_engineering. – Дата доступу : 01.10.2018.
19. Стаття Вікіпедії «Microservices». – Режим доступу: <https://en.wikipedia.org/wiki/Microservices>. – Дата доступу : 01.10.2018.
20. Стаття Вікіпедії «AJAX». – Режим доступу: [https://en.wikipedia.org/wiki/Ajax_\(programming\)](https://en.wikipedia.org/wiki/Ajax_(programming)). – Дата доступу : 01.10.2018.

21. Стаття Вікіпедії «XML». – Режим доступу:
<https://en.wikipedia.org/wiki/XML>. – Дата доступу : 01.10.2018.
22. Стаття Вікіпедії «HTML». – Режим доступу:
<https://en.wikipedia.org/wiki/HTML>. – Дата доступу : 01.10.2018.
23. Стаття Вікіпедії «URL». – Режим доступу: <https://en.wikipedia.org/wiki/URL>.
– Дата доступу : 01.10.2018.
24. Стаття Вікіпедії «SPA». – Режим доступу:
https://en.wikipedia.org/wiki/Single-page_application. – Дата доступу :
01.10.2018.
25. Стаття Вікіпедії «Server Side Includes». – Режим доступу:
https://en.wikipedia.org/wiki/Server_Side_Includes. – Дата доступу : 01.10.2018.
26. PageSpeed Insights. – Режим доступу:
<https://developers.google.com/speed/pagespeed/insights/>. – Дата доступу :
10.12.2018.
27. Pingdom Website Speed Test. – Режим доступу: <https://tools.pingdom.com/>. –
Дата доступу : 10.12.2018.
28. Розроблення стартап-проекту : Методичні рекомендації до виконання
розділу магістерських дисертацій для студентів інженерних спеціальностей /
За заг. ред. О.А. Гавриша. – Київ : НТУУ «КПІ» , 2016. – 28 с.

ДОДАТОК А

Лістинг gallery.html варіанту веб-застосування на React:

```

1 <!DOCTYPE html>
2 <html lang="en">
3   <head>
4     <meta charset="utf-8">
5     <meta http-equiv="X-UA-Compatible" content="IE=edge">
6     <meta name="viewport" content="width=device-width, initial-scale=1">
7
8     <title>Petenok Olena</title>
9
10    <link href="./images/favicon.png" rel="shortcut icon" type="image/png">
11
12    <!-- fonts -->
13    <link href="https://fonts.googleapis.com/css?family=Marck+Script|Montserrat" rel="stylesheet">
14
15    <!-- uikit -->
16    <link rel="stylesheet" href="./lib/uikit-3-0-0-rc-9/css/uikit.min.css">
17
18    <!-- react (change this in production) -->
19    <script src="https://unpkg.com/react@16/umd/react.development.js"></script>
20    <script src="https://unpkg.com/react-dom@16/umd/react-dom.development.js"></script>
21
22    <!-- custom -->
23    <link rel="stylesheet" href="./css/style.css">
24  </head>
25
26  <body>
27
28    <header class="header-background" id="header"></header>
29    <section id="header-padding"></section>
30    <section class="main-background gallery" id="gallery-select"></section>
31    <section class="main-background gallery" id="gallery-image"></section>
32    <footer class="footer-background" id="footer"></footer>
33
34    <!-- uikit -->
35    <script src="./lib/uikit-3-0-0-rc-9/js/uikit.min.js"></script>
36    <script src="./lib/uikit-3-0-0-rc-9/js/uikit-icons.min.js"></script>
37
38    <!-- custom -->
39    <script src="./js/react-out/react.js"></script>
40
41  </body>
42 </html>

```

Лістинг react.js варіанту веб-застосування на React:

```

1  /* header----- */
2  function HeaderLink(props) {
3    return (
4      <li className={`menu-item ${props.active}`}>
5        <a href={props.link} className="menu-item-link">{props.value}</a>
6      </li>
7    );
8  }
9
10 function HeaderMenu(props) {
11   const links = props.links.map((item) =>
12     <HeaderLink key={item.id} active={item.active} link={item.link} value={item.value} />
13   );
14
15   return (
16     <nav>
17       <ul className="uk-flex uk-flex-center@m uk-flex-right@l">
18         {links}
19       </ul>
20     </nav>
21   );
22 }
23
24 function HeaderLogoLink(props) {
25   return (
26     <li id={props.id}>
27       <a href={props.link} className="logo-link">{props.value}</a>
28     </li>
29   );
30 }
31
32 function HeaderLogo(props) {
33   const links = props.links.map((item) =>
34     <HeaderLogoLink key={item.id} id={item.id} link={item.link} value={item.value} />
35   );
36
37   return <ul>{links}</ul>;
38 }

```

```

39
40 function Header(props) {
41   return (
42     <div className="uk-container">
43       <div className="uk-grid">
44         <div className="uk-width-2-5">
45           <HeaderLogo links={props.logoLinks} />
46         </div>
47         <div className="uk-width-3-5">
48           <HeaderMenu links={props.links} />
49         </div>
50       </div>
51     </div>
52   );
53 }
54
55 /* footer----- */
56 function FooterLeftLink(props) {
57   return (
58     <li className="footer-item">
59       <a href={props.link} className="footer-item-link">{props.value}</a>
60     </li>
61   );
62 }
63
64 function FooterLeft(props) {
65   const links = props.links.map((item) =>
66     <FooterLeftLink key={item.id} link={item.link} value={item.value} />
67   );
68
69   return (
70     <ul>
71       {links}
72       <li className="footer-item not-link">&#x2488; Petenok Olena, 2018</li>
73     </ul>
74   );
75 }
76

```

```

77 function FooterRightLink(props) {
78   return (
79     <li className="footer-item">
80       <a href={props.link} target="_blank" className="footer-item-link">
81         <span className="icon" uk-icon={props.icon}></span>
82         {props.value}
83       </a>
84     </li>
85   );
86 }
87
88 function FooterRightElement(props) {
89   return (
90     <li className="footer-item not-link">
91       <span className="icon" uk-icon={props.icon}></span>
92       {props.value}
93     </li>
94   );
95 }
96
97 function FooterRight(props) {
98   const links = props.links.map((item) =>
99     <FooterRightLink key={item.id} link={item.link} icon={item.icon} value={item.value} />
100   );
101   const elements = props.elements.map((item) =>
102     <FooterRightElement key={item.id} icon={item.icon} value={item.value} />
103   );
104
105   return (
106     <ul>
107       {links}
108       {elements}
109     </ul>
110   );
111 }
112

```

```

112
113 function Footer(props) {
114   return (
115     <div className="uk-container">
116       <div className="uk-grid">
117         <div className="uk-width-1-3"><FooterLeft links={menuLinks} /></div>
118         <div className="uk-width-1-3"></div>
119         <div className="uk-width-1-3"><FooterRight links={contactLinks} elements={contactElements} /></div>
120       </div>
121     </div>
122   );
123 }
124
125 /* gallery-select----- */
126 class SelectElement extends React.Component {
127   render() {
128     return <li onClick={this.props.onClick}>{this.props.value}</li>;
129   }
130 }
131
132 class SelectBlock extends React.Component {
133   constructor(props) {
134     super(props);
135
136     this.state = {
137       expand: true,
138       expandIcon: 'up',
139       activeElement: this.props.elements[0].value
140     };
141
142     this.elements = this.props.elements.map(function (item) {
143       return <SelectElement key={item.id} value={item.value} onClick={this.onElementClick.bind(this, {item})} />;
144     }).bind(this);
145   };
146
147   this.onBlockClick = this.onBlockClick.bind(this);
148   this.onElementClick = this.onElementClick.bind(this);
149 }

```

```

151   onBlockClick() {
152     let expandIcon = '';
153     if (this.state.expand) {
154       let hidden = document.querySelector(`#${this.props.value}`);
155       hidden.classList.add("hidden");
156       expandIcon = 'down';
157     }
158   } else {
159     let visible = document.querySelector(`#${this.props.value}`);
160     visible.classList.remove("hidden");
161     expandIcon = 'up';
162   }
163
164   this.setState(previousState => ({
165     expand: !previousState.expand,
166     expandIcon: expandIcon
167   }));
168 }
169
170 onElementClick(item, event) {
171   this.setState({
172     activeElement: event.target.innerHTML
173   });
174 }
175

```

```

176 render() {
177   return (
178     <div className="uk-width-1-1">
179       <div className="dropdown-block">
180         <div className="select-block block-shadow" onClick={this.onBlockClick}>
181           <a href="#" className="select">
182             `${this.props.value}: ${this.state.activeElement}`
183             <span className="icon" uk-icon={`icon: chevron-${this.state.expandIcon}; ratio: 0.85`} /></span>
184           </a>
185         </div>
186         <div className="select-dropdown block-shadow" id={this.props.value}>
187           <ul>{this.elements}</ul>
188         </div>
189       </div>
190     </div>
191   )
192 }
193
194
195 function GallerySelect(props) {
196   const selectArray = props.select.map((item) =>
197     <SelectBlock key={item.id} value={item.value} elements={item.elements} />
198   );
199
200   return (
201     <div className="uk-container">
202       <div className="uk-grid">
203         {selectArray}
204       </div>
205     </div>
206   );
207 }
208

```

```

209  /* gallery-image-block----- */
210  function GalleryImageElement(props) {
211    return ( <li className="image-block"><img src={props.src} alt={props.alt} /></li> );
212  }
213
214  function GalleryImageBlock(props) {
215    const elements = props.elements.map((item) =>
216      <GalleryImageElement key={item.id} src={item.src} alt={item.alt} />
217    );
218
219    return (
220      <div className={props.width}>
221        <ul>{elements}</ul>
222      </div>
223    );
224  }
225
226  class GalleryImage extends React.Component {
227    constructor(props) {
228      super(props);
229
230      /* get state of SelectBlock components */
231      let select = [];
232      let selectQuery = document.querySelectorAll("#gallery-select a.select");
233      selectQuery.forEach( (item) => {
234        select.push(item.innerHTML.split('<', 1)[0].split(' ', 2)[1]);
235      });
236
237      /* initialize state of component depending on state of SelectBlock components */
238      this.props.width.forEach( (item) => {
239        if (item.id == select[0]) {
240          this.state = {
241            width: item.value,
242            imageBlockArray: ''
243          };
244        }
245      });

```

```

247  /* initialize changes of view by click depending on SelectBlock components change */
248  this.selectItems = document.querySelectorAll("#gallery-select .select-dropdown li");
249
250  for (let i = 0; i < this.selectItems.length; i++) {
251    this.selectItems[i].onclick = this.selectItems[i].onclick.bind(this);
252    this.selectItems[i].onclick = (this, function (event) {
253      let current = this.selectItems[i].innerHTML;
254
255      /* initialize width filter */
256      this.props.width.forEach ( (item) => {
257        if (current == item.id) {
258          /* alert('second for: item=' + item + ', item.id=' + item.id + ', item.value=' + item.value); */
259          this.setState({ width: item.value });
260        }
261      });
262
263    }).bind(this);
264  }
265
266  this.shouldComponentUpdate = this.shouldComponentUpdate.bind(this);
267  }
268
269  shouldComponentUpdate(nextProps, nextState) {
270    return !( this.state.width == nextState.width );
271  }
272
273  render() {
274    this.state.imageBlockArray = this.props.gallery.map((item) =>
275      <GalleryImageBlock key={item.id} width={this.state.width} elements={item.elements} />
276    );
277
278    return (
279      <div className="uk-container">
280        <div className="uk-grid">
281          {this.state.imageBlockArray}
282        </div>
283      </div>
284    )

```



```

289 const logoLinks = [
290   {id: 'petenok', link: 'index.html', value: 'Petenok'},
291   {id: 'olena', link: 'index.html', value: 'Olena'}
292 ];
293
294 const menuLinks = [
295   {id: '1', active: '', link: 'index.html', value: 'Home'},
296   {id: '2', active: 'active', link: 'gallery.html', value: 'Gallery'},
297   {id: '3', active: '', link: 'about-author.html', value: 'About aut
298 ];
299
300 const contactLinks = [
301   {id: '1', icon: 'icon: linkedin; ratio: 0.85', value: 'Linkedin'},
302   {id: '2', icon: 'icon: facebook; ratio: 0.85', value: 'Facebook'},
303 ];
304
305 const contactElements = [
306   {id: 'phone', icon: 'icon: receiver; ratio: 0.85', value: '+38 (06
307   {id: 'email', icon: 'icon: google; ratio: 0.85', value: 'olena.pet
308 ];
309
310 const select = [
311   {
312     id: '1',
313     value: 'View',
314     elements: [
315       {id: '1', value: 'Small'},
316       {id: '2', value: 'Normal'},
317       {id: '3', value: 'Big'}
318     ]
319   }
320 ];
321

```

```

356 const widthData = [
357   {id: 'Small', value: 'uk-width-1-3'},
358   {id: 'Normal', value: 'uk-width-1-2'},
359   {id: 'Big', value: 'uk-width-1-1'}
360 ];
361
362 const gallery = [
363   {
364     id: 'Header',
365     elements: [
366       {id: '1', src: './images/gallery/header/1.png', alt: 'header-image' },
367       {id: '2', src: './images/gallery/header/2.png', alt: 'header-image' },
368       {id: '3', src: './images/gallery/header/3.png', alt: 'header-image' },
369       {id: '4', src: './images/gallery/header/4.png', alt: 'header-image' },
370       {id: '5', src: './images/gallery/header/5.png', alt: 'header-image' },
371       {id: '6', src: './images/gallery/header/6.png', alt: 'header-image' },
372       {id: '7', src: './images/gallery/header/7.png', alt: 'header-image' },
373       {id: '8', src: './images/gallery/header/8.png', alt: 'header-image' }
374     ]
375   },
376   {
377     id: 'Footer',
378     elements: [
379       {id: '1', src: './images/gallery/footer/1.png', alt: 'footer-image' },
380       {id: '2', src: './images/gallery/footer/2.png', alt: 'footer-image' },
381       {id: '3', src: './images/gallery/footer/3.png', alt: 'footer-image' }
382     ]
383   }
384 ];

```

```

432 /* render----- */
433 ReactDOM.render(
434   <Header logoLinks={logoLinks} links={menuLinks} />,
435   document.getElementById('header')
436 );
437
438 ReactDOM.render(
439   <Footer menuLinks="menuLinks" contactLinks="contactLinks" elements="contactElements" />,
440   document.getElementById('footer')
441 );
442
443 ReactDOM.render(
444   <GallerySelect select={select} />,
445   document.getElementById('gallery-select')
446 );
447
448 ReactDOM.render(
449   <GalleryImage gallery={gallery} width={widthData} />,
450   document.getElementById('gallery-image')
451 );

```

ДОДАТОК Б

Лістинг gallery.html варіанту веб-застосування на React та Angular:

```

1 <!DOCTYPE html>
2 <html lang="en">
3   <head>
4     <meta charset="utf-8">
5     <meta http-equiv="X-UA-Compatible" content="IE=edge">
6     <meta name="viewport" content="width=device-width, initial-scale=1">
7
8     <title>Petenok Olena</title>
9
10    <link href="/images/favicon.png" rel="shortcut icon" type="image/png">
11
12    <!-- fonts -->
13    <link href="https://fonts.googleapis.com/css?family=Marck+Script|Montserrat" rel="stylesheet">
14
15    <!-- uikit -->
16    <link rel="stylesheet" href="/lib/uikit-3-0-0-rc-9/css/uikit.min.css">
17
18    <!-- react (change this in production) -->
19    <script src="https://unpkg.com/react@16/umd/react.development.js"></script>
20    <script src="https://unpkg.com/react-dom@16/umd/react-dom.development.js"></script>
21
22    <!-- custom css -->
23    <link rel="stylesheet" href="/css/style.css">
24  </head>
25
26  <body ng-app="App" ng-controller="GalleryImage">
27
28    <header class="header-background" id="header"></header>
29    <section id="header-padding"></section>
30    <section class="main-background gallery" id="gallery-select"></section>

```

```

32    <section>
33      <div class="main-background gallery">
34        <div class="uk-container">
35          <div uk-grid>
36            <div ng-class="width" ng-repeat="lists in gallery">
37              <ul ng-repeat="item in lists.elements">
38                <li class="image-block"></li>
39              </ul>
40            </div>
41          </div>
42        </div>
43      </div>
44    </section>
45
46    <footer class="footer-background" id="footer"></footer>
47
48    <!-- uikit -->
49    <script src="/lib/uikit-3-0-0-rc-9/js/uikit.min.js"></script>
50    <script src="/lib/uikit-3-0-0-rc-9/js/uikit-icons.min.js"></script>
51
52    <!-- custom react -->
53    <script src="/js/react-out/react.js"></script>
54
55    <!-- old angular -->
56    <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.4/angular.min.js"></script>
57    <script src="js/angular/BasicScope.js"></script>
58    <script src="js/angular/GalleryImage.js"></script>
59
60  </body>
61 </html>

```


Лістинг BasicScope.js варіанту веб-застосування на React та Angular:

```
1  angular.module('App',[])
2    .controller ("Basic", function($scope, $http) {
3
4    });
```

Лістинг GalleryImage.js варіанту веб-застосування на React та Angular:

```
1  angular.module('App')
2    .controller ("GalleryImage", function($scope, $http) {
3
4    $scope.widthData = [
5      {id: 'Small', value: 'uk-width-1-3'},
6      {id: 'Normal', value: 'uk-width-1-2'},
7      {id: 'Big', value: 'uk-width-1-1'}
8    ];
9
10   $scope.width = '';
11   $scope.widthElementsList = document.getElementById("gallery-select").getElementsByTagName("LI");
12
13   $scope.widthElementsListListeners = function () {
14     for (var i = 0; i < $scope.widthElementsList.length; i++) {
15       $scope.widthElementsList[i].addEventListener ("click", function() {
16         var id = this.innerHTML;
17
18         $scope.widthData.forEach( (item) => {
19           if (item.id == id) {
20             $scope.width = item.value;
21             $scope.$digest();
22           }
23         });
24       });
25     }
26   }
27
28   $scope.getDefaultWidth = function () {
29     var defaultWidthID = $scope.widthElementsList[0].innerHTML;
30     var result = '';
31     $scope.widthData.forEach( (item) => {
32       if (item.id == defaultWidthID) {
33         result = item.value;
34       }
35     });
36
37     return result;
38   });
```

```

40     $scope.setWidth = function () {
41         if ($scope.width == '') {
42             $scope.widthElementsListListeners();
43             return $scope.getDefaultWidth();
44         } else {
45             return $scope.width;
46         }
47     };
48
49     $scope.width = $scope.setWidth();
50
51     $scope.gallery = [
52         {
53             id: 'Header',
54             elements: [
55                 {id: '1', src: './images/gallery/header/1.png', alt: 'header-image' },
56                 {id: '2', src: './images/gallery/header/2.png', alt: 'header-image' },
57                 {id: '3', src: './images/gallery/header/3.png', alt: 'header-image' },
58                 {id: '4', src: './images/gallery/header/4.png', alt: 'header-image' },
59                 {id: '5', src: './images/gallery/header/5.png', alt: 'header-image' },
60                 {id: '6', src: './images/gallery/header/6.png', alt: 'header-image' },
61                 {id: '7', src: './images/gallery/header/7.png', alt: 'header-image' },
62                 {id: '8', src: './images/gallery/header/8.png', alt: 'header-image' }
63             ]
64         },
65         {
66             id: 'Footer',
67             elements: [
68                 {id: '1', src: './images/gallery/footer/1.png', alt: 'footer-image' },
69                 {id: '2', src: './images/gallery/footer/2.png', alt: 'footer-image' },
70                 {id: '3', src: './images/gallery/footer/3.png', alt: 'footer-image' }
71             ]
72         },
73         {
74             id: 'Background',
75             elements: [
76                 {id: '1', src: './images/gallery/background/1.jpg', alt: 'background-image' },
77                 {id: '2', src: './images/gallery/background/2.jpg', alt: 'background-image' },
78                 {id: '3', src: './images/gallery/background/3.jpg', alt: 'background-image' },
79                 {id: '4', src: './images/gallery/background/4.jpg', alt: 'background-image' },
80                 {id: '5', src: './images/gallery/background/5.jpg', alt: 'background-image' },
81                 {id: '6', src: './images/gallery/background/6.jpg', alt: 'background-image' },
82                 {id: '7', src: './images/gallery/background/7.jpg', alt: 'background-image' },
83                 {id: '8', src: './images/gallery/background/8.jpg', alt: 'background-image' },
84                 {id: '9', src: './images/gallery/background/9.jpg', alt: 'background-image' },
85                 {id: '10', src: './images/gallery/background/10.jpg', alt: 'background-image' },
86                 {id: '11', src: './images/gallery/background/11.jpg', alt: 'background-image' },
87                 {id: '12', src: './images/gallery/background/12.jpg', alt: 'background-image' },
88                 {id: '13', src: './images/gallery/background/13.jpg', alt: 'background-image' },
89                 {id: '14', src: './images/gallery/background/14.jpg', alt: 'background-image' },
90                 {id: '15', src: './images/gallery/background/15.jpg', alt: 'background-image' },
91                 {id: '16', src: './images/gallery/background/16.jpg', alt: 'background-image' },
92                 {id: '17', src: './images/gallery/background/17.jpg', alt: 'background-image' }
93             ]
94         }
95     ];
96
97     $scope.setWidth();
98
99     $scope.setWidth();
100
101     $scope.setWidth();
102
103     $scope.setWidth();
104
105     $scope.setWidth();
106
107     $scope.setWidth();
108
109     $scope.setWidth();
110
111     $scope.setWidth();
112
113     $scope.setWidth();
114     {id: '15', src: './images/gallery/background/15.jpg', alt: 'background-image' },
115     {id: '16', src: './images/gallery/background/16.jpg', alt: 'background-image' },
116     {id: '17', src: './images/gallery/background/17.jpg', alt: 'background-image' }
117     ]
118     }
119     ];
120
121     });
122
123     });

```