Міністерство освіти і науки України

Луцький національний технічний університет
(повне найменування закладу вищої освіти)

Факультет комп'ютерних та інформаційних технологій
(повне найменування факультету)

Кафедра комп'ютерної інженерії та кібербезпеки
(повне найменування кафедри)

КВАЛІФІКАЦІЙНА РОБОТА ЗА СТУПЕНЕМ ВИЩОЇ ОСВІТИ «БАКАЛАВР»

ПРОГРАМНИЙ ІНТЕРФЕЙС ДЛЯ ПЕРЕГЛЯДУ ЕКСПОНАТІВ ІСТОРИЧНОГО МУЗЕЮ

SOFTWARE FOR VIEWING HISTORICAL MUSEUM EXHIBITS

спеціальність 123 Комп'ютерна інженерія
(шифр і назва спеціальності)

освітня програма Комп'ютерна інженерія
(назва освітньої програми)

Виконав: здобувач вищої освіти
групи КІ-41
Крулік Юрій Олександрович

(підпис)

Керівник:
к.т.н., доцент
Лавренчук Світлана Василівна

(підпис)

Кваліфікаційну роботу
допущено до захисту
« _____ » червня 2023 р.
Гарант освітньої програми:
к.т.н., доцент
Лавренчук Світлана Василівна

(підпис)

Луцьк – 2023 року

ЛУЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

Факультет *комп'ютерних та інформаційних технологій*
Кафедра *комп'ютерної інженерії та кібербезпеки*
Ступінь вищої освіти: *бакалавр*
Галузь знань: *12 Інформаційні технології*
Спеціальність: *123 Комп'ютерна інженерія*
Освітня програма: *«Комп'ютерна інженерія»*

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ проф. Н.Чернящук
«_____» _____ 2023 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧУ ВИЩОЇ ОСВІТИ

Круліку Юрію Олександровичу
(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи *Програмний інтерфейс для перегляду експонатів історичного музею*

Керівник роботи *к.т.н., доцент Лавренчук Світлана Василівна*

затверджені наказом закладу вищої освіти від «28» грудня 2022 року № 982/01-02
2. Строк подання здобувачем вищої освіти кваліфікаційної роботи 01.06.2023р.

3. Вихідні дані до роботи *Джерелом розробки є науково-технічна література та публікації в періодичних виданнях з даного питання, опубліковані зарубіжні та вітчизняні роботи в даній області та різні інтернет-ресурси технічного спрямування*

4. Зміст пояснювальної записки (перелік питань, які потрібно розробити):
Вступ
Аналіз предметної області з питань розробки програмних інтерфейсів
Дослідження та порівняння найпопулярніших технологій у розробці інтерфейсів
Розробка базового дизайну інтерфейсу
Розробка інтерфейсу
Висновки

5. Перелік графічного (ілюстративного) матеріалу:

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис	
		завдання видав	завдання прийняв
Аналіз проблеми за темою роботи та постановка завдань дослідження	*Лавренчук С.В.*		
Теоретичне дослідження та практична реалізація	*Лавренчук С.В.*		
Практична реалізація об'єкта проектування	*Лавренчук С.В.*		
Висновки			

7. Дата видачі завдання 01.11.2022 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	*Обгрунтування теми*	До 15.11.2022 р.	Виконано
2.	*Літературний огляд найновіших наукових досліджень у сфері розробки інтерфейсів*	До 15.12.2022 р.	Виконано
3.	*Аналіз загальної проблеми розробки програмних інтерфейсів*	До 02.02.2023 р.	Виконано
4.	*Дослідження технологій для розробки додатку*	До 02.03.2023 р.	Виконано
5.	*Налаштування робочого середовища та розгортання проєкту*	До 02.04.2023 р.	Виконано
6.	*Розробка програмного інтерфейсу*	До 15.04.2023 р.	Виконано
7.	*Формування списку використаних джерел*	До 02.05.2023 р.	Виконано
8.	*Нормоконтроль*	До 25.05.2023 р.	Виконано
9.	*Інструментальна перевірка на академічний плагіат*	До 01.6.2023 р	Виконано
10.	*Представлення кваліфікаційної роботи бакалавра до захисту*	До 07.06.2023 р.	Виконано

Здобувач вищої освіти _____ Крулік Ю.О.
(підпис) (прізвище, ініціали)

Керівник кваліфікаційної роботи _____ Лавренчук С.В.
(підпис) (прізвище, ініціали)

АНОТАЦІЯ

Крулік Ю.О. Програмний інтерфейс для перегляду експонатів історичного музею. Рукопис.

Кваліфікаційна робота бакалавра ОП «Комп'ютерна інженерія» спеціальності 123 Комп'ютерна інженерія. Луцький національний технічний університет. Луцьк, 2023.

Кваліфікаційна робота написана обсягом 62 сторінок і містить 12 ілюстрацій, 3 додатки та 16 джерел за переліком посилань.

Кваліфікаційна робота складається з вступу, трьох розділів, висновків, списку використаних джерел (згідно структури кваліфікаційної роботи, затвердженої кафедрою).

Перший розділ присвячено аналітичному огляду програмних інтерфейсів, а саме: аналіз інтерфейсів популярних музеїв, порівняння технологій, які можуть бути використані у розробці програмних інтерфейсів. У кінці розділу було проведено обґрунтований вибір технологій для розробки програмного інтерфейсу музею на основі порівняльної характеристики інструментів.

У другому розділі було сформовано дизайн інтерфейсу на основі потреб користувачів та досліджено загальні методи для реалізації програмного інтерфейсу музею. Наведені способи для вирішення проблеми адаптації користувацького інтерфейсу на пристроях різних масштабів.

У третьому розділі на основі проведених досліджень було реалізовано функціональний, динамічний та адаптивний інтерфейс.

Ключові слова: програмний інтерфейс, адаптація, фреймворк, компонент, стилі.

ANNOTATION

Krulik Y.O. Software for viewing historical museum exhibits. Manuscript.

Bachelor's qualification work in the specialty 123 Computer Engineering of the Educational Program «Computer Engineering». Lutsk National Technical University. Lutsk, 2023.

The qualification work is written on 62 pages and contains 12 illustrations, 3 appendices, and 16 sources in the list of references. The qualification work consists of an introduction, three sections, conclusions, a list of used sources (according to the structure of the qualification work, approved by the department).

The first section is dedicated to an analytical review of software interfaces, namely: the analysis of interfaces of popular museums, comparison of technologies that can be used in the development of software interfaces. At the end of the section, a justified choice of technologies for the development of the museum software interface was made based on a comparative characteristic of tools.

The second section formed the interface design based on user needs and investigated general methods for implementing the museum software interface. Solutions are provided to solve the problem of adapting the user interface on devices of different scales.

In the third section, based on the conducted studies, a functional, dynamic, and adaptive interface was implemented.

Keywords: software interface, adaptation, framework, component, styles.

ЗМІСТ

ВСТУП

Актуальність теми. Тема «Програмний інтерфейс для перегляду експонатів історичного музею» є актуальною у зв'язку з розвитком технологій та зростанням інтересу до культурної спадщини. В сучасному світі, де відбувається швидкий технологічний розвиток, інформаційні технології стають важливим інструментом для популяризації культури та історії. Використання інтерактивних експозицій дозволяє залучити більше відвідувачів до музею, зробити експозиції більш зрозумілими та доступними, зберегти та популяризувати культурну спадщину.

Метою роботи є розробка функціонального та естетичного інтерфейсу, який дозволить користувачам зручно знайти, переглянути та дослідити експонати історичного музею.

Об'єкт дослідження – процес розробки програмного інтерфейсу історичного музею на основі сучасних технологій та підходів.

Предмет дослідження – аспекти, пов'язані з користувачами, які використовують програмний інтерфейс та технології для розробки програмного інтерфейсу

Завдання, які необхідно виконати:

– Провести аналіз потреб та очікувань користувачів: необхідно зрозуміти, які функції має містити програмний інтерфейс для того, щоб задовольняти потреби відвідувачів музею.

– Розробити вимоги до програмного інтерфейсу;

– Дослідити технології, які найкраще підходять для розробки програмного інтерфейсу;

– Спроектувати дизайну інтерфейсу: на основі вимог до нього;

– Розробити програмне забезпечення;

Апробація роботи: Кваліфікаційна робота виконувалась в межах Договору про співпрацю згідно Міжнародного Проекту «Готський шлях: спільна історична реконструкція та віртуальна подорож у минуле» (укладений між ЛНТУ та Виконавчим комітетом Володимир-Волинської міської ради від 07 лютого 2022

року). Подано до публікації статтю: Христинець Н.А., Лавренчук С.В., Євсюк В.М., Крулік Ю.О. Функціональні адаптивні інтерфейси з динамічними компонентами для підсистем зберігання мультимедійного контенту. Науковий журнал «Комп'ютерно-інтегровані технології: освіта, наука, виробництво». Луцьк, 2023. № 51. С. 87-93.

РОЗДІЛ 1
АНАЛІТИЧНИЙ ОГЛЯД ПИТАНЬ ФОРМУВАННЯ ПРОГРАМНИХ ІНТЕРФЕЙСІВ

1.1 Аналіз інтерфейсів популярних музеїв

Для розробки програмного інтерфейсу історичного музею корисно провести аналіз інтерфейсів популярних музеїв. Цей аналіз допоможе зрозуміти, які елементи інтерфейсу приваблюють відвідувачів та що можна запозичити для покращення власного інтерфейсу.

Потрібно обрати декілька популярних музеїв з різних країн та проаналізувати їх інтерфейси. При цьому слід звернути увагу на такі аспекти:

– Дизайн інтерфейсу. Які кольори та шрифти використовуються? Які елементи дизайну привертають увагу відвідувачів? Які елементи дизайну є спільними для багатьох музеїв?

– Навігація. Як легко знайти інформацію на сайті? Які розділи та підрозділи є на сайті? Чи є можливість швидко повернутися на головну сторінку сайту?

– Контент. Які матеріали доступні для перегляду на сайті? Чи є можливість переглядати експонати онлайн? Які особливості презентації матеріалів на сайті?

– Функціонал. Які можливості є на сайті? Чи є можливість зареєструватися на сайті? Чи є можливість відправити запитання до адміністратора сайту?

– Адаптивність. Як сайт відображається на різних пристроях?Чи є можливість переглядати сайт на мобільних пристроях?

Аналіз інтерфейсів популярних музеїв допоможе зрозуміти, які елементи веб-інтерфейсу є важливими для відвідувачів, та як їх краще розташовувати на сайті для забезпечення зручного та ефективного користування. Крім того, цей аналіз допоможе виявити нові ідеї та можливості для покращення власного програмного інтерфейсу.

Провівши цей було зроблено такі висновки:

– Більшість популярних музеїв мають зручний та зрозумілий інтерфейс для користувачів, що допомагає швидко знайти потрібну інформацію та експонати.

– Навігаційні панелі та меню зверху або з боку сторінки забезпечують зручний та легкий доступ до різних розділів веб-сайту. Це дозволяє користувачам швидко знайти необхідний контент без необхідності багато розгортати чи скролити веб-сторінку. Також, користувачам легко змінювати розділи та переглядати експонати в будь-який момент

– Фотографії та відеоматеріали про експонати є невід'ємною частиною багатьох сайтів музеїв, що допомагає користувачам краще зрозуміти та оцінити експонати. Як правило, фотографії та відеоматеріали є головним елементом інтерфейсу музею.

– Для зручного ознайомлення з експонатами медіа контент часто представляють у вигляді каруселі.

– Більшість сайтів музеїв мають адаптацію для мобільних пристроїв, що забезпечує зручний та ефективний доступ до інформації та експонатів навіть з маленького екрану.

Загальний аналіз інтерфейсів популярних музеїв дозволив мені зрозуміти, які елементи веб-інтерфейсу є ключовими для забезпечення зручного та ефективного користування інтерфейсом історичного музею. Ці висновки будуть використані при розробці програмного інтерфейсу для нашого музею з метою покращення досвіду користувачів та приваблення нових відвідувачів.

1.2 Огляд методів та технологій створення веб-інтерфейсів

1.2.1 Створення програмних інтерфейсів з використанням HTML та CSS

Створення програмних інтерфейсів з використанням HTML та CSS є одним з найпоширеніших методів розробки веб-інтерфейсів. HTML (Hypertext Markup Language) – це стандартна мова розмітки для створення веб-сторінок, яка описує структуру та зміст веб-документів [1]. CSS (Cascading Style Sheets) – це мова опису стилів, яка використовується для визначення зовнішнього вигляду веб-сторінок, таких як кольори, шрифти, розміри та інше [2]. Основна перевага використання HTML та CSS полягає в їх простоті та доступності. Обидві мови досить легко

вивчити, тому що вони мають простий синтаксис та їх легко зрозуміти. Крім того, HTML та CSS є стандартними мовами для створення веб-інтерфейсів, тому більшість браузерів та веб-додатків підтримують їх. Однак, HTML та CSS мають свої обмеження, особливо для більш складних веб-інтерфейсів, які потребують більш складних функцій та динамічного контенту. Це може створювати проблеми для веб-розробників, які хочуть створити більш складний та функціональний веб-інтерфейс. Також варто зазначити, що HTML та CSS не є мовами програмування, а лише мовами розмітки та опису стилів. Для більш складних функцій та динамічного контенту, необхідно використовувати інші мови програмування, такі як JavaScript.

1.2.2 Препроцесори CSS

У сучасному світі веб-розробки важливо створювати ефективні та гнучкі стилі CSS, щоб відповідати вимогам дизайну і створювати користувацький інтерфейс високої якості. Препроцесори CSS є інструментами, які дозволяють розробникам створювати більш структуровані, модульні та гнучкі стилі CSS, використовуючи зручні та потужні функції, такі як змінні, змішування, вкладеність тощо. Застосування препроцесорів CSS може підвищити продуктивність, полегшити підтримку та забезпечити більшу гнучкість у розробці веб-сайту [4]. Розглянемо переваги та недоліки найбільш популярних препроцесорів CSS:

Переваги Sass:

– Синтаксис: Sass пропонує два синтаксиси – SCSS та індентаційний Sass, що дозволяє розробникам вибрати стиль, який їм більше подобається.

– Функціональність: Sass має багатий набір функцій, таких як змінні, змішування, вкладеність, умовні оператори, цикли та багато іншого.

– Спільнота та підтримка: Sass має велику спільноту, що означає широкий спектр ресурсів, бібліотек, плагінів та підтримки.

Недоліки Sass:

– Зовнішній компілятор: Sass вимагає встановлення Ruby або використання зовнішнього компілятора для перетворення свого коду у CSS.

– Повільніша компіляція: Sass може мати повільніші часи компіляції порівняно з Less.

Переваги Less:

– Легкість використання: Less має простіший синтаксис та менше функцій, що може полегшити вивченення та розробку для нових користувачів

– Браузерна компіляція: Less може компілюватись безпосередньо у браузері, що полегшує налаштування та розробку.

– Швидкість компіляції: Less відомий своєю швидкою компіляцією порівняно з Sass та Stylus.

Недоліки Less:

– Обмежені функції: Less має менший набір функцій, що може обмежувати розробників, які звикли до розширеної функціональності Sass або Stylus.

– Менша спільнота: Less має меншу спільноту, порівняно з Sass, що може зменшити доступ до ресурсів, плагінів та підтримки.

Переваги Stylus:

– Гнучкість синтаксису: Stylus має високу гнучкість синтаксису, що дозволяє розробникам використовувати кілька стилів написання коду.

– Експресивність: Stylus пропонує багато експресивних можливостей, таких як аліаси, динамічні типи даних, перегрузка функцій та інше.

– Підтримка JavaScript: Stylus дозволяє використовувати JavaScript безпосередньо у своєму коді, що надає ще більше потужності та гнучкості.

Недоліки Stylus:

– Менш популярний: Stylus менш популярний порівняно з Sass та Less, що може призвести до меншої кількості ресурсів, бібліотек та плагінів.

– Крива навчання: Stylus може мати трохи вищу криву навчання через його гнучкий синтаксис та експресивні можливості.

– Повільніша компіляція: Через більшу кількість функцій та експресивність, Stylus може мати повільніші часи компіляції порівняно з Less і дещо повільніше порівняно з Sass.

При виборі препроцесора CSS для свого проекту, розробники повинні розглянути свої власні вимоги, навички та пріоритети. Всі три препроцесори можуть бути ефективними інструментами для створення гнучких та легко підтримуваних

стилів CSS. Окрім препроцесорів, варто також звернути увагу на бібліотеки стилів, такі як Tailwind CSS та Bootstrap, які можуть спростити та прискорити процес розробки інтерфейсів.

1.2.3 Бібліотеки стилів

Бібліотеки стилів є важливим інструментом у сучасній веб-розробці, який полегшує створення естетичних, адаптивних та зручних для користувачів інтерфейсів. Вони включають набори готових компонентів, стилів та функцій, що значно прискорюють процес розробки, забезпечують сумісність з різними браузерами та пропонують кастомізацію для задоволення індивідуальних вимог дизайну. Найбільш популярними бібліотеками стилів є Tailwind CSS та Bootstrap, розглянемо їх детальніше:

Tailwind CSS – це утилітарна бібліотека стилів, яка пропонує гнучкі класи для швидкого та зручного створення відповідних інтерфейсів без потреби писати власні стилі [5].

Переваги Tailwind CSS:

– Гнучкість: Tailwind дозволяє легко створювати відповідні дизайни без написання багатьох CSS-стилів.

– Швидкість розробки: Наявність готових класів прискорює процес розробки інтерфейсів.

– Кастомізація: Tailwind дозволяє кастомізувати свій дизайн, використовуючи конфігураційний файл.

– Чистий код: Утилітарний підхід сприяє створенню чистого коду без переплетення структури та стилів.

Недоліки Tailwind CSS:

– Класи в HTML: Використання класів безпосередньо в HTML може ускладнити читання коду.

– Відсутність важливих компонентів: на відміну від Bootstrap, Tailwind не має багатьох стилізованих компонентів. Це означає, що розробнику доведеться вручну додавати такі функції, як заголовки, кнопки та панелі навігації для веб-додатків.

– Крива навчання: Може зайняти час для освоєння утилітарного підходу та запам'ятовування класів.

Bootstrap – це найпопулярніша бібліотека стилів, яка пропонує готові компоненти, стилі та JavaScript – функції для створення адаптивних інтерфейсів [6].

Переваги Bootstrap:

– Швидкість розробки: Bootstrap пропонує готові компоненти та стилі, що прискорюють процес розробки.

– Адаптивність: Bootstrap має вбудовані стилі та сітки, що сприяють адаптивному дизайну.

– Сумісність з браузерами: Bootstrap підтримує всі сучасні браузери, гарантуючи сумісність і стабільність.

– Велика спільнота: Bootstrap має велику спільноту, що забезпечує підтримку, ресурси та допомогу у вирішенні проблем.

Недоліки Bootstrap:

– Однотипність: Багато сайтів, що використовують Bootstrap, можуть мати схожий вигляд, якщо не вносити значних змін у дизайн.

– Розмір файлу: Bootstrap може мати великий розмір файлу через кількість вбудованих стилів і компонентів.

– Надлишковість: Часто в проекті не використовуються всі компоненти та стилі, що присутні в Bootstrap, що може призвести до надлишкового коду.

– Залежність від jQuery: Bootstrap залежить від jQuery для деяких своїх JavaScript компонентів, що може додати зайву вагу та залежності до проекту.

1.2.4 JavaScript

JavaScript є однією з найпоширеніших технологій для створення веб-додатків. Він дозволяє розробникам створювати динамічні та інтерактивні елементи на веб-сторінках, такі як валідація форм, анімації, взаємодія з користувачем та багато іншого.

JavaScript – це скриптова мова програмування, що використовується для створення веб-додатків, яка працює на бічному (клієнтському) боці веб-сторінки. Це означає, що код JavaScript виконується в браузері користувача, і він може

взаємодіяти зі структурою HTML та CSS, змінювати їх та динамічно міняти відображення веб-сторінки відповідно до дій користувача [3].

JavaScript має декілька фреймворків та бібліотек, таких як React, Angular, Vue.js та інші, які спрощують процес розробки та надають багато корисних інструментів для розробки веб-додатків. Ці фреймворки та бібліотеки забезпечують швидкий розвиток, підтримують багато функціональних можливостей та допомагають уникнути повторного використання коду.

Однак, як і в усіх технологіях, у JavaScript є свої обмеження та проблеми, такі як проблеми з безпекою та швидкість виконання. Тому, при виборі використання JavaScript у веб-додатку, важливо зрозуміти його обмеження та знайти спосіб їх уникнення або пом'якшення.

1.2.5 Огляд фреймворків веб-розробки

Фреймворки веб-розробки є важливою складовою сучасної веб-розробки, які дозволяють швидко та ефективно створювати веб-додатки зі складною функціональністю. Найбільш популярними фреймворками веб-розробки є Angular, React та Vue.js. Розглянемо кожен з них детальніше:

– Angular – це фреймворк веб-розробки, створений компанією Google. Він базується на мові програмування TypeScript та пропонує широкий спектр можливостей для створення високопродуктивних веб-додатків. Angular має велику спільноту розробників та широкий набір інструментів, що дозволяє ефективно створювати веб-додатки будь-якої складності. Однак, використання Angular може бути складним для початківців та вимагає певного часу на вивчення фреймворку [7].

– React – це бібліотека JavaScript, розроблена компанією Facebook. Вона дозволяє створювати високопродуктивні веб-додатки зі складною функціональністю та забезпечує ефективну роботу зі станом компонентів. React має широку підтримку від розробників та велику кількість сторонніх бібліотек, що дозволяє значно спростити розробку веб-додатків. Однак, використання React вимагає розуміння JavaScript та певних концепцій веб-розробки [7].

– Vue.js – це прогресивний фреймворк веб-розробки з простою та інтуїтивно зрозумілою структурою. Він дозволяє швидко створювати високоякісні веб-додатки

та має легку інтеграцію з іншими бібліотеками та фреймворками. Vue.js також надає широкий спектр можливостей для розробки веб-додатків будь-якої складності, включаючи SPA (Single Page Applications) та SSR (Server-Side Rendering) [7].

Кожен фреймворк веб-розробки має свої переваги та недоліки. Розглянемо їх детальніше:

Переваги Angular:

– Масштабованість: Angular має потужні архітектурні засади, які забезпечують стабільність та масштабованість застосунків.

– Двостороннє зв'язування даних: Angular пропонує потужний механізм двостороннього зв'язування даних, який автоматично синхронізує зміни між моделлю та представленням.

– Зручна організація коду: Angular підтримує модульну архітектуру та компонентний підхід, що сприяє зручній організації коду.

– Вбудовані інструменти: Angular має велику кількість вбудованих інструментів, таких як HttpClient, Forms, Routing, інтернаціоналізація та ін., що полегшують розробку застосунків.

– Спільнота: Angular має велику та активну спільноту, що забезпечує доступ до ресурсів, підтримки та готових рішень.

Недоліки Angular:

– Висока крива навчання: Angular вважається складнішим для вивчення порівняно з React та Vue.js через свою непросту структуру та поняття.

– Продуктивність: Angular може бути менш продуктивним через важкий розмір фреймворка та складність оптимізації продуктивності.

– Потреба у специфічних знаннях: Angular вимагає від розробників знання специфічних мов програмування та підходів, таких як TypeScript та ReactiveX (RxJS). Це може додати додатковий рівень навчання та складності для розробників, які не знайомі з цими технологіями.

– Версії: Міграція між версіями Angular може бути складною задачею через непостійність API та розбіжності між версіями.

– Актуальність документації: З огляду на часті оновлення та зміни у фреймворку, документація Angular може залишатися застарілою або неактуальною. Це може ускладнити вивчення та розуміння найкращих практик розробки.

Переваги React:

– Легкість вивчення: React пропонує простий синтаксис і структуру, що полегшує вивчення для нових розробників.

– Віртуальний DOM: React використовує віртуальний DOM для підвищення продуктивності, оптимізації рендеринга та забезпечення швидкодії.

– Спільнота: React має широку підтримку від спільноти розробників та велику кількість сторонніх бібліотек, що дозволяє значно спростити розробку веб-додатків.

– Компонентний підхід: Застосунки, створені на React, складаються з компонентів, які можна легко повторно використовувати та комбінувати

Недоліки React:

– Лише представлення: React забезпечує лише рівень представлення, тому розробники повинні використовувати додаткові бібліотеки або фреймворки для реалізації архітектури програми.

– Недостатня підтримка з боку Facebook та відсутність строго контролю за використанням React можуть призвести до проблем із сумісністю та безпекою коду.

– Часті оновлення: React постійно оновлюється, що може призвести до несумісності версій і необхідності рефакторингу коду.

Переваги Vue.js:

– Легкість вивчення: Vue.js має легку та просту для вивчення структуру, що робить його ідеальним для початківців.

– Гнучкість: Vue.js може використовуватися як для створення невеликих віджетів, так і для розробки великих масштабованих застосунків.

– Двустороннє зв'язування даних: Vue.js пропонує просту реалізацію двостороннього зв'язування даних, що полегшує роботу з формами та динамічними інтерфейсами.

– Легкість інтеграції: Vue.js легко інтегрується з іншими бібліотеками, фреймворками та існуючими проектами.

– Vue.js має низький розмір, що забезпечує швидке завантаження сторінок веб-додатків.

Недоліки Vue.js:

– Розмір спільноти: Хоча спільнота Vue.js росте, вона все ще менша порівняно зі спільнотами React та Angular.

– Обмежений ресурс: Попри зростання популярності Vue.js, доступ до ресурсів, таких як плагіни та готові рішення, є обмеженим порівняно з React та Angular.

– Відсутність корпоративної підтримки: На відміну від React (Facebook) та Angular (Google), Vue.js розробляється та підтримується незалежною групою розробників. Це може викликати опіки та сумніви в інвестиціях у фреймворк на корпоративному рівні, що може обмежити наявність ресурсів та підтримки.

– Робота з великими проектами: Vue.js може бути менш зручним для роботи з дуже великими та складними проектами через відсутність строгих правил структуризації коду. У таких ситуаціях, більш суворі архітектурні вимоги Angular можуть стати перевагою.

– Документація та мовна підтримка: Хоча Vue.js має досить хорошу документацію, вона може бути неповною або недостатньою для певних специфічних випадків використання. Крім того, частину документації та ресурсів може бути доступно лише англійською мовою, що може утруднити вивчення та використання Vue.js для носіїв інших мов.

1.2.6 Огляд систем контролю версій

Системи контролю версій (Version Control Systems, VCS) є важливим інструментом для будь-якого розробника, незалежно від того, працює він в команді чи самостійно. Вони допомагають стежити за змінами у коді, відновлювати попередні версії, легко співпрацювати з іншими розробниками, вирішувати конфлікти та ефективно організовувати роботу над проектами[8]. Розглянемо найпопулярніші системи контролю версій детальніше:

Git – найбільш популярна розподілена система контролю версій, що використовується в більшості проектів сьогодні [9].

Переваги Git:

– Швидкість: Git пропонує високу швидкість операцій та створення гілок.

– Розподілена система: Кожен розробник має повну копію репозиторію, що сприяє незалежності від централізованого сервера.

– Гнучкість: Git підтримує різні моделі гілкування та робочі процеси.

– Інтеграція: Git має велику кількість графічних клієнтів, плагінів та підтримку в сервісах, таких як GitHub, GitLab, та Bitbucket.

Недоліки Git:

– Крива навчання: Git може бути складним для новачків через численні команди та концепції.

– Більші репозиторії: Git може мати проблеми зі швидкістю та ефективністю великих репозиторіїв.

SVN – це централізована система контролю версій, що була дуже популярна до появи Git [9].

Переваги SVN:

– Простота: SVN має простішу модель роботи, що полегшує навчання для новачків.

– Централізований сервер: SVN забезпечує єдине місце зберігання коду, спрощуючи його адміністрування та керування доступом.

– Лінійність версій: Версії в SVN представлені послідовними числами, що полегшує сприйняття хронології змін.

– Часткова отримка: SVN дозволяє отримувати лише частину репозиторію, якщо потрібно працювати тільки з певною частиною коду.

Недоліки SVN:

– Централізація: Залежність від централізованого сервера може стати проблемою при відмові сервера або поганому з'єднанні з Інтернетом.

– Відсутність гнучкого гілкування: Гілкування в SVN менш зручне, порівняно з Git, тому що воно потребує більше часу та ресурсів.

Mercurial – розподілена система контролю версій, яка схожа на Git, але пропонує спрощений користувацький інтерфейс та меншу кількість команд [9].

Переваги Mercurial:

– Простота: Mercurial має простіший інтерфейс та менше команд порівняно з Git, що полегшує навчання новачкам.

– Розподілена система: Mercurial також пропонує розподілену архітектуру, дозволяючи кожному розробнику мати повну копію репозиторію.

– Швидкість: Mercurial має хорошу швидкість операцій, хоча й не таку високу, як у Git.

Недоліки Mercurial:

– Менший рівень популярності: Mercurial менш популярний за Git, що може призвести до меншої кількості ресурсів, інтеграцій та спільноти.

– Обмежена підтримка: Mercurial може мати менше підтримки в сервісах та інструментах порівняно з Git.

1.2.7 Огляд односторінкових та багатосторінкових додатків: SPA та MPA

У сучасній веб-розробці існує дві основні архітектури додатків: односторінкові додатки (Single Page Applications, SPA) та багатосторінкові додатки (Multi-Page Applications, MPA). Обидві архітектури мають свої переваги та недоліки, а вибір між ними залежить від специфіки проекту та потреб користувачів.

Односторінкові додатки (SPA):

SPA – це веб-додатки, які завантажуються лише один раз і забезпечують динамічну інтеракцію з користувачем без перезавантаження сторінки. Усі потрібні дані завантажуються динамічно через API [10].

Переваги SPA:

– Швидкість: SPA забезпечують швидку взаємодію з користувачем, оскільки вміст завантажується тільки один раз.

– Гнучкість: SPA можуть легко адаптуватися для різних пристроїв, таких як мобільні телефони та планшети.

– Відмінний користувацький досвід: SPA забезпечують плавні переходи між сторінками та анімації, створюючи враження нативного додатка.

Недоліки SPA:

– SEO: SPA можуть мати проблеми з оптимізацією для пошукових систем через їх динамічну природу.

– Довге завантаження: SPA можуть мати довший час завантаження, оскільки всі дані та скрипти завантажуються спочатку.

Багатосторінкові додатки (MPA):

MPA – це традиційні веб-додатки, які складаються з кількох сторінок, кожна з яких відображає певний набір інформації та функціональності [10].

Переваги MPA:

– SEO: MPA легше оптимізувати для пошукових систем, оскільки кожна сторінка має унікальний URL та відповідний контент

– Стабільність: Менше залежності від JavaScript та клієнтської сторони, що забезпечує стабільність роботи додатку.

– Взаємодія з сервером: Завантаження контенту в MPA відбувається при переході на нову сторінку, що дозволяє користувачеві бачити оновлені дані.

Недоліки MPA:

– Швидкість: При переході між сторінками потрібно завантажувати нову сторінку, що може сповільнювати взаємодію з користувачем.

– Відсутність плавності: MPA не забезпечує плавні переходи та анімації, що можуть бути доступні в SPA.

– Більше коду: Зазвичай MPA вимагає більше коду для реалізації функціональності, ніж SPA.

1.3 Вибір технологій

Враховуючи аналіз попередніх розділів та відповідні переваги та недоліки різних технологій, можна зробити висновок щодо того, які інструменти та технології будуть використовуватися при розробці програмного інтерфейсу для перегляду експонатів історичного музею.

1. HTML: Як базова мова розмітки, HTML буде використовуватися для створення структури веб-сторінок та організації контенту на них. Використання семантичних тегів сприятиме покращенню SEO та доступності сайту.

2. SASS: Для забезпечення гнучкості, реюзабельності та легкої підтримки CSS стилів, використання препроцесора SASS є оптимальним варіантом. Він дозволить використовувати змінні, міксіни, вкладеність та інші функції, що спрощують процес створення стилів. Крім того, використання препроцесора допоможе уникнути повторення коду та підвищить продуктивність розробки.

3. Bootstrap: Оскільки я маю досвід роботи з Bootstrap, його використання дозволить швидко створити адаптивний та сучасний інтерфейс для користувачів. Крім того, використання цієї бібліотеки стилів забезпечує підтримку широкого спектру пристроїв, включаючи мобільні. Bootstrap також надає широкий набір готових компонентів та стилів, що сприятиме швидкій та послідовній розробці проекту.

4. JavaScript: Як базова мова програмування для клієнтської сторони, JavaScript буде використовуватися для динамічної інтеракції з користувачем, а також для реалізації різних функцій та ефектів. JavaScript дозволить розробити кастомні анімації, реагувати на події користувача, а також забезпечити можливість зміни контенту сторінки без перезавантаження. Це покращить швидкість роботи сайту та зручність використання.

5. Vue.js: З огляду на мій досвід з Vue.js та його переваги, такі як простота, гнучкість, невеликий розмір та висока швидкість, цей фреймворк буде використовуватися для реалізації клієнтської частини додатка. Vue.js дозволяє створювати модульні компоненти, що можуть бути легко повторно використані та тестовані. Невеликий розмір фреймворка забезпечує швидше завантаження сторінок та підвищує загальну продуктивність додатка. Також, з використанням Vue.js, розробник зможе легко створювати реактивні додатки та працювати з даними.

6. Git: Як система контролю версій, Git буде використовуватися для відслідковування змін у коді та координації роботи над проектом. Навіть самостійному розробнику, використання Git може принести ряд переваг, таких як

збереження різних версій коду, можливість відкату до попередніх версій у разі виникнення проблем, а також легке відновлення втраченого коду через віддалений репозиторій. Крім того, Git допомагає підтримувати структурованість робочого процесу та може бути корисним у майбутньому, якщо виникне необхідність залучити додаткових розробників для реалізації або підтримки проекту.

7. Множинні сторінки (MPA): Застосування множинних сторінок дозволяє створити структуровану та ієрархічну архітектуру для сайту. Це спрощує навігацію по сайту та дозволяє користувачам легше знаходити потрібний контент за допомогою структурованих розділів. Враховуючи характер історичного музею, MPA буде більш підходящим рішенням для реалізації головних функцій та забезпечення стабільності роботи сайту. Більше того, MPA має переваги у плані SEO (оптимізації для пошукових систем), оскільки кожна сторінка може мати власні метатеги та ключові слова, що поліпшує видимість сайту в пошукових системах та збільшує органічний трафік.

РОЗДІЛ 2
МЕТОДИ ТА ПІДХОДИ ДО РОЗРОБКИ ПРОГРАМНОГО ІНТЕРФЕЙСУ

2.1 Структура інтерфейсу

Відповідно до проведеного аналізу, на кожній сторінці сайту будуть наступні елементи:

Навігаційне меню: цей елемент розташований у верхній частині сторінки та містить посилання на основні розділи сайту. Він сприяє зручній навігації та дозволяє користувачам швидко знайти необхідну інформацію.

1. Заголовок сторінки: розташований безпосередньо під навігаційним меню, він відображає основну тему сторінки та допомагає користувачам зорієнтуватися.

2. Підзаголовок: розташований під заголовком, він доповнює заголовок, розширюючи тему та забезпечуючи додаткову інформацію для кращого розуміння контенту сторінки.

3. Основний контент сторінки: знаходиться під підзаголовком та може включати елементи, такі як каруселі з фотографіями або відео експонатів. Ці елементи надають користувачам можливість переглядати експонати візуально, що сприяє кращому розумінню та оцінці експонатів.

4. Текстовий контент: розміщений після основного контенту, цей розділ надає додаткову інформацію про експонати та може включати детальні описи, історію тощо.

5. Футер: розташований у нижній частині сторінки, він містить корисні посилання, контактну інформацію та авторські права.

Така структура інтерфейсу забезпечує зручну навігацію та доступність необхідної інформації для користувачів. Кожен елемент інтерфейсу спроектований таким чином, щоб максимально полегшити процес взаємодії користувачів з сайтом та забезпечити легке розуміння контенту.

Загалом, ця структура враховує потреби користувачів та дозволяє легко знайти потрібну інформацію та насолоджуватися відвідуванням музейного сайту. Комбінація візуальних та текстових елементів допомагає створити гармонійний та

зручний інтерфейс, який задовольняє потреби широкого кола користувачів. На рисунку 2.1 зображено базовий дизайн інтефейсу, який враховує зазначенні вимоги.

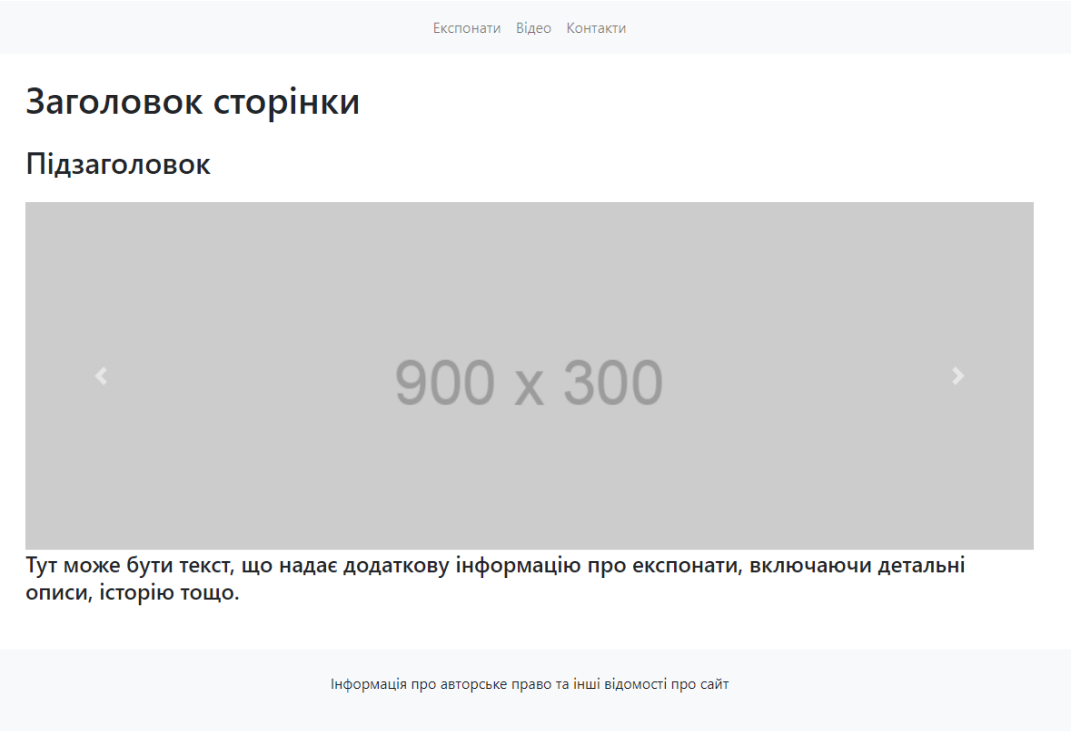


Рисунок 2.1 – Базовий дизайн інтерфейсу

2.2 Адаптація інтерфейсу

Оскільки користувачі можуть відвідувати сайт з різних пристроїв, таких як смартфони, планшети або комп'ютери, важливо розробити адаптивний дизайн, що враховує розміри екранів різних пристроїв.

Адаптивний дизайн дозволяє забезпечити оптимальний відображення інтерфейсу на різних пристроях, забезпечуючи зручність користування сайтом. Застосування адаптивного дизайну може включати наступні підходи:

2.2.1 Використання Bootstrap

Bootstrap пропонує адаптивні класи сітки, які допомагають автоматично розміщувати та масштабувати контент на різних пристроях. Сітка Bootstrap базується на 12 колонках, і ви можете використовувати різні класи, щоб

контролювати ширину елементів на різних типах екранів [11]. Нижче представлено декілька основних адаптивних класів, які надає Bootstrap:

– .col- застосовується на екранах з розміром менше 576 пікселів (мобільні пристрої).

– .col-sm- застосовується на екранах з розміром від 576 пікселів до 767 пікселів (мобільні пристрої та планшети).

– .col-md- застосовується на екранах з розміром від 768 пікселів до 991 пікселів (планшети та маленькі настільні комп'ютери).

– .col-lg- застосовується на екранах з розміром від 992 пікселів до 1199 пікселів (настільні комп'ютери).

– .col-xl- застосовується на екранах з розміром від 1200 пікселів і більше (великі настільні комп'ютери).

Для кожного класу ви можете додати номер від 1 до 12, щоб вказати, скільки колонок займає елемент. Наприклад, якщо ви хочете, щоб елемент займав половину ширини екрана на планшетах, ви можете використовувати клас .col-md-6.

Приклад використання адаптивних класів сітки Bootstrap:

```
<div class="container">
  <div class="row">
    <div class="col-sm-12 col-md-6">
      <!-- Вміст для мобільних пристроїв (1 колонка) та планшетів (половина ширини екрана) -->
    </div>
  </div>
  <div class="row">
    <div class="col-12 col-md-4 col-lg-3">
      <!-- Вміст для мобільних пристроїв (1 колонка), планшетів (одна третина ширини екрана) та настільних комп'ютерів (одна четверта ширини екрана) -->
    </div>
    <div class="col-12 col-md-8 col-lg-9">
```

```
<!-- Вміст для мобільних пристроїв (1 колонка), планшетів (дві третини ширини екрана) та настільних комп'ютерів (три четверті ширини екрана) -->
    </div>
  </div>
</div>
```

2.2.2 Використання медіа-запитів

Медіа-запити дозволяють розробнику змінювати стилі для елементів залежно від характеристик пристрою чи вікна перегляду, таких як розмір екрану, співвідношення сторін, орієнтація екрану та інше. Це дає можливість реагувати на різні ситуації, створюючи адаптивний дизайн [12].

Приклад використання медіа-запитів в CSS для адаптації контенту сайту під різні розміри екрану:

Стилі за замовчуванням для контейнера, які застосовуються для всіх розмірів екрану:

```
.container {
    padding: 15px;
    margin: 0 auto;
    background-color: white;
}
```

Адаптація контейнера до екранів з мінімальною шириною 576 пікселів:

```
@media screen and (min-width: 576px) {
    .container {
        max-width: 540px;
    }
}
```

Адаптація контейнера до екранів з мінімальною шириною 768 пікселів:

```
@media screen and (min-width: 768px) {
    .container {
        max-width: 720px;
    }
```

```
}
```

Адаптація контейнера до екранів з мінімальною шириною 992 пікселів:

```
@media screen and (min-width: 992px) {
   .container {
      max-width: 960px;
   }
}
```

У цьому коді ми використовуються медіа-запити, щоб змінити ширину елемента з класом .container залежно від ширини екрану. Коли ширина екрану менше 576 пікселів, будуть застосовані стилі за замовчуванням, і .container займе максимально можливу ширину на екрані.

Для екранів із шириною 576 пікселів і більше використовуються медіа-запити, які задають максимальну ширину елемента .container відповідно до конкретних розмірів екрану. Це дозволяє контейнеру автоматично адаптуватися до ширини екрану, забезпечуючи зручне відображення контенту на різних пристроях

2.3 Робота з медіа контентом

Робота з медіа контентом у HTML виконується через теги <img> та <video>. Розглянемо їх детальніше:

2.3.1 Тег <img>

Тег <img> використовується для вставки зображень на веб-сторінки. Він має наступні важливі атрибути:

– src: Цей атрибут вказує на URL-адресу зображення. Він є обов'язковим атрибутом для тега <img>.

– alt: Цей атрибут надає альтернативний текст для зображення, який відображається, якщо зображення не може бути завантажене. Також альтернативний текст корисний для пошукових роботів та скрінрідерів для незорових користувачів.

– width та height: Ці атрибути визначають ширину та висоту зображення відповідно. Вони можуть вказуватися як в абсолютних одиницях (пікселі), так і відносних (відсотки).

Приклад коду з тегом <img>:

<img src="path/to/image.jpg" alt="Опис зображення" width="300" height="200">

2.3.2 Тег <video>

Тег <video> використовується для вбудовування відео на веб-сторінки. Він має наступні важливі атрибути:

– src: Цей атрибут вказує на URL-адресу відеофайлу. Він є обов'язковим атрибутом для тега <video>.

– width та height: Ці атрибути визначають ширину та висоту відеоплеєра відповідно. Вони можуть вказуватися як в абсолютних одиницях (пікселі), так і відносних (відсотки).

– controls: Цей атрибут додає елементи керування відеоплеєром, такі як кнопки відтворення, гучності та повного екрану.

– autoplay: Цей атрибут дозволяє відео автоматично відтворюватися, як тільки сторінка завантажується. Однак з міркувань користувачів, браузери можуть блокувати автоматичне відтворення звуку.

– loop: Цей атрибут дозволяє відео автоматично зациклюватися та відтворюватися з початку після завершення.

– muted: Цей атрибут приглушує звук відео за замовчуванням.

– poster: Цей атрибут вказує на URL-адресу зображення, яке відображається до завантаження відео або поки користувач не натисне кнопку відтворення.

Приклад коду з тегом <video>:

```
<video width="640" height="360" controls>
  <source src="path/to/video.mp4" type="video/mp4">
  <source src="path/to/video.webm" type="video/webm">
  Ваш браузер не підтримує відео тег.
</video>
```

У прикладі вище ми використовуємо тег <source> для вказівки джерел відеофайлів. Це дозволяє нам вказати кілька форматів відео, щоб забезпечити сумісність з різними браузерами. Браузер вибере перший сумісний формат для відтворення.

2.4 Динамічні компоненти

Динамічні компоненти є ключовою особливістю у сучасній розробці програмних інтерфейсів. Вони дозволяють створювати гнучкі, інтерактивні та масштабовані веб-інтерфейси, які полегшують роботу з користувачем та розвиток проектів.

Суть динамічних компонентів полягає у тому, що вони можуть змінюватися відповідно до потреб користувача та стану додатку. Вони дозволяють розбити інтерфейс на окремі, легко керовані частини, кожна з яких відповідає за певну функціональність. Це спрощує розробку, розширення та супровід веб-додатків, оскільки робота з окремими компонентами стає набагато прозорішою та зрозумілішою.

Динамічні компоненти можуть бути реактивними, що означає, що вони автоматично оновлюються при зміні вхідних даних або стану додатку. Це забезпечує високу швидкість відгуку та плавну роботу інтерфейсу без необхідності перезавантаження сторінки.

Основні директиви Vue.js, які допомагають створювати динамічні компоненти, включають v-for, v-bind та props.

2.4.1 Директива v-for

Ця директива дозволяє відображати список елементів на основі масиву даних. Вона створює "цикл" через елементи масиву та відображає їх у відповідних HTML-елементах [13]. Наприклад:

```
<ul>
  <li v-for="item in items" :key="item.id">
    {{ item.name }}
```

```
  </li>
</ul>
```

Приклад коду вище відобразить маркований список імен з масиву items

2.4.2 Директива v-bind

Ця директива дозволяє динамічно прив'язувати атрибути HTML-елементів до даних або виразів. Вона корисна, коли потрібно змінювати значення атрибута на основі даних або стану компонента [14]. Наприклад:

```
<img v-bind:src="imageUrl" alt="Опис зображення">
```

Код вище буде зміювати зображення яке відображається залежно від посилання яке знаходиться в змінній imageUrl

2.4.3 Директива props

Props (або властивості) в динамічних компонентах – це механізм передачі даних між компонентами в ієрархії додатка. Вони дозволяють батьківському компоненту передавати дані своїм дочірнім компонентам, створюючи односторонній потік даних від верхівки ієрархії до нижніх рівнів [15].

Робота з props включає декілька основних аспектів:

– Оголошення props: Для передачі даних у дочірній компонент, спочатку необхідно оголосити властивості, які будуть приймати дані. Оголошення props зазвичай відбувається у секції props у визначенні компонента. Тут можна також вказати тип даних, який очікується, та значення за замовчуванням, якщо воно є.

– Передача даних: Після оголошення props, батьківський компонент може передавати дані своїм дочірнім компонентам, використовуючи атрибути в шаблоні компонента. Властивості передаються у форматі prop-name="value", де prop-name – назва властивості, а value – значення, яке передається.

– Використання props: У дочірньому компоненті можна використовувати передані властивості так само, як і локальні дані. Props можна використовувати в шаблонах, обчислюваних властивостях та методах компонента.

Важливо зазначити, що props є односторонніми, тобто дочірній компонент не повинен змінювати значення props. Якщо дочірньому компоненту потрібно змінити значення переданої властивості, він повинен сповістити батьківський компонент про

цю потребу, використовуючи події. Це допомагає уникнути проблем з непередбачуваною зміною даних та небажаними побічними ефектами.

2.4.4 Приклад створення компоненту засобами Vue.js

Спочатку потрібно створити новий файл Vue-компонента (наприклад, Carousel.vue). У цьому файлі будуть використовуватися три основні частини: <template>, <script> та <style>.

<template> містить структуру HTML-елементів компоненту. Розробник може використовувати Vue-директиви, які розглянуто вище, для динамічного відображення елементів залежно від даних. Наприклад, використовуючи v-for для рендерингу слайдів каруселі та v-bind для прив'язки атрибутів елементів до даних.

Тег <script> визначає поведінку нашого компонента, включаючи властивості та методи. Тут імпортуються необхідні залежності, визначається вхідні параметри (props) для компонента та створюються методи для обробки подій та взаємодії з користувачем.

Тег <style> містить стилі CSS для компоненту. Тут можна використовувати звичайний CSS або препроцесори, такі як SCSS. Використання області видимості стилів (scoped styles) дозволяє нам ізолювати стилі компоненту, для того, щоб вони не впливали на інші частини сайту.

Приклад коду компоненту каруселі:

```
<template>
  <div class="carousel">
    <div class="carousel-slide" v-for="item in items" :key="item.id">
      <img v-bind:src="item.imageUrl" alt="Зображення експонату" />
      <p>{{ item.title }}</p>
    </div>
  </div>
</template>
<script>
export default {
  props: {
```

```
    items: {
      type: Array,
      required: true,
    },
  },
  methods: {
    // Методи для обробки подій, наприклад зміни слайдів
  },
};
</script>
```

У цьому прикладі ми створюється компонент каруселі, який приймає масив items як вхідний параметр. Кожен елемент масиву містить інформацію про експонат, таку як id, imageUrl та title. За допомогою директиви v-for ми створюютсья слайди каруселі для кожного елемента в масиві. Директива v-bind використовується для динамічного прив'язування атрибута src зображення до відповідного URL з даних експонату.

Таким чином, використовуючи Vue.js та його директиви, ми можемо створювати динамічні та інтерактивні компоненти інтерфейсу, які забезпечують плавну роботу з користувачем.

2.4.5 Шаблонізатор Blade

Так як, підсистема для зберігання мультимедійного контенту виконана засобами PHP фреймворку Laravel і не всі компоненти потребують можливості JavaScript та Vue.js, у проекті буде використовуватися шаблоні затор Blade.

Шаблонізатор Blade – це потужна система шаблонів, що входить до складу фреймворку Laravel. Blade надає простий та елегантний синтаксис для взаємодії з даними, переданими у шаблон від контролера, а також для декларування директив, які дозволяють виконувати різноманітні операції з даними та генерацією HTML [16]. Для цього проекту важливими є дві речі: Директива @foreach та механізм x-slot.

Директива @foreach у Blade використовується для ітерації через колекцію або масив елементів. Це дозволяє легко генерувати HTML-код для кожного елемента колекції або масиву. Синтаксис директиви @foreach є наступним:

```
@foreach($collection as $item)
@endforeach
```

У цьому прикладі $collection – це колекція або масив, який містить елементи, і $item – це поточний елемент колекції або масиву, який обробляється всередині циклу.

x-slot – це механізм, що дозволяє передавати іменовані слоти в компоненти Blade. Іменовані слоти дозволяють вам вставляти контент у визначені частини компонента з батьківського шаблону.

Синтаксис x-slot виглядає наступним чином:

```
<x-component>
    <x-slot name="slotName">
    </x-slot>
</x-component>
```

У цьому прикладі <x-component> – це компонент Blade, в якому ви хочете використовувати іменований слот, а <x-slot name="slotName"> – це визначення іменованого слота з ім'ям slotName, весь код який буде розміщуватися усередині тегу x-slot, можна буде використати у батьківському компоненті за допомогою конструкції {{ $slotName }}.

РОЗДІЛ 3
РОЗРОБКА ПРОГРАМНОГО ІНТЕРФЕЙСУ МУЗЕЮ

3.1 Git ініціалізація проекту

Першим кроком у розробці будь-якого проекту є ініціалізація системи контролю версій, у мому випадку Git. Її використання забезпечить ефективну роботу над проектом, контроль за змінами у коді, відстеження історії змін, та збереження різних версій проекту у віддаленому репозиторії, що унеможливить втрату прогресу розробки проекту, навіть якщо втратиться доступ до робочого комп`ютера.

Перший крок – це відкрити термінал і у командному рядку перейти до дерикторії з проектом. Далі у командний рядок потрібно ввести команду git init, яка створить новий підкаталог .git у директорії проєкту, де будуть зберігатися всі файли та інформація, пов'язані з Git. Далі потрібно створити будь який файл для того щоб його можна було додати у Git репозиторій. Було створено файл header.blade.php, у якому буде описано навігаційну панель сайту. Після його створення потрібно у командній стрічці виконати команду git add . важливо звернути увагу на крапку після add, яка означає додавання усіх файлів у поточній директорії. Далі потрібно зафіксувати зміни у репозиторії для цього Git використовується коміт.

Коміт – це дія збереження набору змін у коді, який міститься у робочому каталозі. Коміт представляє собою фіксацію стану робочого каталогу на певний момент часу та дозволяє зберегти історію змін коду. Перший коміт виконується командою git commit -m "Initial commit". Далі було виконано команду git push для того щоб відправити зміни у віддалений репозиторій. Ця команда створить перший коміт у репозиторії з повідомленням: Initial commit. Далі потрібно створити нову гілку.

Гілка у системі контролю версій Git – це незалежний потік роботи, який відокремлений від інших гілок. Гілки дозволяють розробнику працювати над різними функціями, виправленнями або експериментами не впливаючи на стабільність основного коду. Кожна гілка відображає певний стан проекту в певний момент часу. Гілка main зазвичай вважається основною гілкою, що містить

стабільну версію проекту. Розробники створюють нові гілки для роботи над окремими задачами, а потім зливають зміни з цих гілок назад в основну гілку після завершення роботи над задачею. Було прийняте рішення створити гілку develop у якій і буде відбуватись процес розробки, вона, після розробки певної частини функціоналу, буде зливатись у основну гілку main – у якій буде зберігатись стабільна версія проекту. Для створення гілки develop з гілки main я зпочатку переконався, що я знаходжусь на гілці main, для цього я виконав команду git checkout main, яка перемкне робочу гілку на гілку main. Далі було створено нову гілку виконавши команду git checkout -b develop. Ця команда створює нову гілку develop, яка спочатку буде ідентичною гілці main, та одразу перемикає робочу директорію на цю нову гілку. Тепер можна розпочинати розробку і не хвилюватись про втрату прогрессу.

3.2 Навігаційне меню

У процесі створення навігаційного меню було використано Bootstrap 4, який надає готові стилі та компоненти, що спрощують процес розробки та шаблонізатор Blade.

Спочатку було створено контейнер для навігаційної панелі (тег <nav>) та задав стилі:

– navbar – базовий клас, який визначає основні стилі для навігаційної панелі;

– navbar-expand-lg – клас, який забезпечує адаптивність навігаційного меню на різних розмірах екрану, вказуючи, що меню буде розгортатися на екранах розміром large (lg) та більше;

– navbar-font – це користувацький клас, створений для задання відповідного стилю шрифту для навігаційного меню.

Далі було додано кнопку (тег <button>) з класом navbar-toggler для мобільних пристроїв, яка буде відкривати та закривати навігаційне меню. Використано наступні атрибути:

– data-bs-toggle – атрибут, який вказує на тип перемикача (у нашому випадку – collapse);

– data-bs-target – атрибут, який вказує на ідентифікатор елемента, який буде розгортатися або згортатися при натисканні кнопки;

– aria-controls – атрибут, який вказує на ідентифікатор елемента, яким керує кнопка;

– aria-expanded – атрибут, який вказує стан розгорнутого або згорнутого елемента;

– aria-label – атрибут, який вказує текстовий опис кнопки для технологій доступності, таких як екранні читачі.

У наступному кроці ствворено контейнер для елементів меню за допомогою класу collapse navbar-collapse та вказав justify-content-center для відцентрування пунктів меню. Згадані класи мають наступне призначення:

– collapse – клас, який використовується для створення елементів, які можуть згортатися та розгортатися при натисканні кнопки navbar-toggler;

– navbar-collapse – клас, що використовується для групування елементів меню, які повинні згортатися та розгортатися;

– justify-content-center – клас, який відцентровує пункти меню по горизонталі в контейнері.

Далі використано цикл @foreach для генерації пунктів меню на основі даних змінної $pages, яка містить інформацію про сторінки. Кожен пункт меню містить посилання з класом nav-link та відповідними атрибутами href та aria-current для стилізації та навігації:

– nav-link – клас, який використовується для стилізації посилань в навігаційному меню;

– active – клас, що використовується для виділення активного пункту меню;

– aria-current – атрибут, який вказує на поточний активний пункт меню для технологій доступності.

У правій частині меню розміщено блок для зміни локалізації. Знову використано цикл @foreach для генерації посилань на різні мови на основі даних

змінної $localeLinks. Використав клас nav-link для стилізації посилань та атрибут href для навігації.

Завершено структуру меню за допомогою тега <img> який відобразить зображення під навігаціним меню, для того щоб розділити навігаційне менб та основний контент. Вказав шлях до зображення за допомогою функції asset, яка генерує URL-адресу для файлів активів (наприклад, зображень). Додав клас header_image для відповідного відображення та позиціювання зображення на фоні меню. Використовуючи клас header_image, встановлено наступні стилі:

– «width: 100%;» – встановлює ширину зображення на 100%, щоб воно розтягувалося на всю ширину екрану;

– «max-height: 500px;» – встановлює максимальну висоту зображення на 500 пікселів, щоб контролювати його розмір;

– «position: relative;» – використовується для встановлення позиціонування зображення відносно його батьківського елемента;

– «top: 0;» – встановлює відстань зображення від верхнього краю батьківського елемента на 0 пікселів;

– «left: 0;» – встановлює відстань зображення від лівого краю батьківського елемента на 0 пікселів;

– «z-index: 10;» – встановлює порядок накладення зображення на інші елементи. Значення 10 означає, що зображення буде відображатися над елементами з меншим значенням z-index.

На рисунках 3.1 та 3.2 зображена навігаційна панель для великих екранів та мобільних пристроїв відповідно

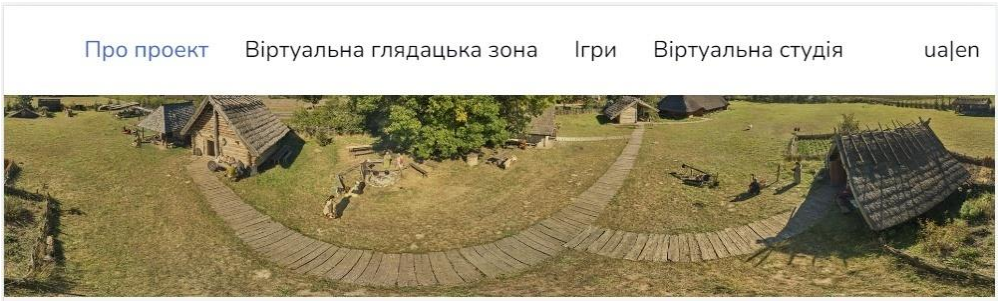


Рисунок 3.1 – Навігаційна панель для великих екранів

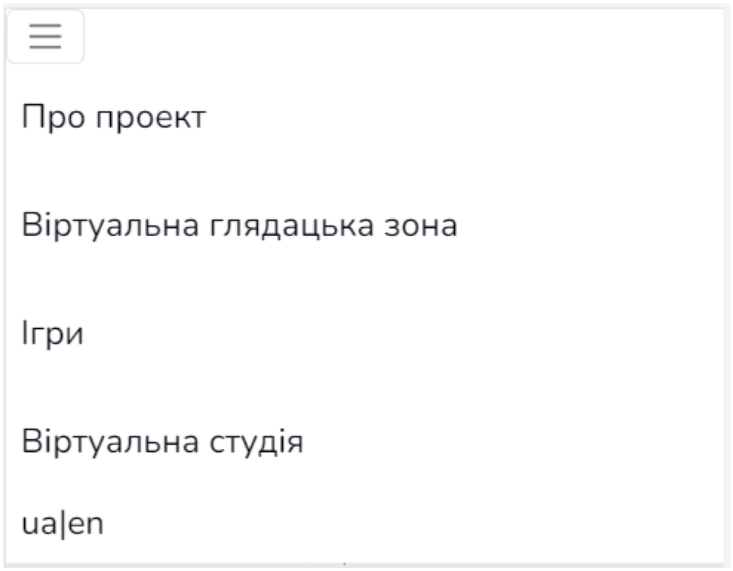


Рисунок 3.2 – Навігаціна панель для мобільних пристроїв

Повний код для навігаційної панелі можна переглянути у додатку А.

3.3 Футер

Наступним було вирішено створити футер, для того щоб завершити створення елементів які є спілиними для всіх сторінок.

Для створення футера використано HTML тег <footer> вказано клас footer. Цей клас використовує такі стилі:

– «bottom: 0;» – встановлює відстань футера від нижнього краю екрана на 0 пікселів;

– «width: 100%;» – встановлює ширину футера на 100%, щоб він займав всю ширину екрана;

– «margin-top: auto;» – встановлює автоматичний верхній відступ, щоб футер завжди був розташований внизу сторінки;

– «padding: 40px;» – встановлює внутрішній відступ на 40 пікселів, щоб відділити вміст від країв футера;

– «background-color: white;» – встановлює білий колір фону футера;

– «box-shadow: 0 0 3px 3px rgb(0 0 0 / 10%);» – додає тінь навколо футера, що створює ефект «відокремлення» від змісту сторінки.

Наступним кроком було створено елемент і задав йому Bootstrap клас container, яний задає стилі для відцентрованого контейнера, з шириною яка автоматично адаптується під різні розміри екрану.

Далі було розділено футер на два ряди за допомогую елементів з класом row.

Верхню частину розділено на 3 колонки, використовуючи клас col-md-4 для кожної колонки, розподілено доступний горизонтальний простір на три рівні частини. За допомогою md- було вказано що цей розподіл має бути застосований на екранах середнього розміру і вище. На менших екранах колонки будуть займати всю ширину рядка.

Далі було додано горизонтальну лінію між верхньою та нижньою частинами футера за допомогою тега <hr>. Це візуально розділяє контент у футері та полегшує сприйняття інформації користувачами.

У нижній частині створено одну колонку використовуючи клас col-md-12. Для роміщення авторських прав використано тег <p> з класом text-muted який зменшить насиченість кольору тексту.

Результат зображено на рисунку 3.3

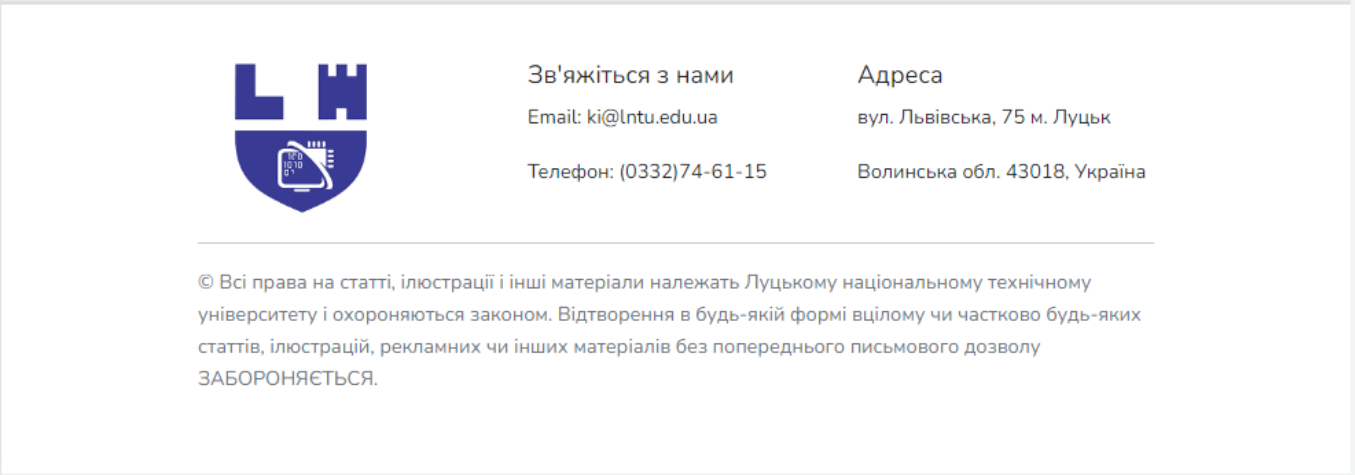


Рисунок 3.3 – Футер

Повний код для футера можна переглянути у додатку А.

3.4 Компоненти каруселі фото та відео

Для створення цих компонентів використано пакет VueSlickCarousel, який допомагає забезпечити динамічне та гнучке відображення зображень у вигляді каруселі.

Для початку було встановлено цей пакет вписавши у командну стрічку команду npm і vue-slick-carousel. Далі було створено файл PhotosComponent.vue у якому власне і буду описувати компонент з каруселлю фото. Наступним кроком було імпортовано пакет VueSlickCarousel у компонент за допомогою трьох наступних стрічок коду:

```
import VueSlickCarousel from 'vue-slick-carousel'
import 'vue-slick-carousel/dist/vue-slick-carousel.css'
import 'vue-slick-carousel/dist/vue-slick-carousel-theme.css'
```

У першому рядку імпортується сам пакет, а у двох наступних стилі до нього.

Далі було зареєстровано цей пакет як компонент використовуючи параметр components у Vue.js: “components: {VueSlickCarousel}”. Тепер є можливість використовувати цей пакет як HTML тег <VueSlickCarousel>.

Далі у props оголошено змінну у яку буде передаватися масив зображень і назвав його images.

Наступний крок це створення самого шаблону компоненту для початку було створено з класом component_content, цей клас відповідає за відцентрування компоненту та за те щоб компонент використовував повну доступну висоту.

Всередині цього блоку додано компонент VueSlickCarousel з директивою v-bind, у яку передаються налаштування каруселі. Класи carousel, col-sm-10 та col-md-7 використовуються для адаптивності каруселі та забезпечують гнучке масштабування на різних екранах.

У цьому тегу використано директиву v-for для генерації слайдів з масиву images, який отримую як параметр (prop) від батьківського компонента. Для кожного зображення використано тег <img> та прив'язую атрибут src до властивості url об`єкта забраження зображення та атрибут alt до властивості alt об`єкта

зображення. Кожному зображенню надано клас slideimage. Данний клас використовує такі стилі:

– «width: inherit;» – ця властивість забезпечує, що ширина зображення спадкується від свого батьківського елемента. Це дозволяє забезпечити правильну адаптивність зображення при зміні розміру екрана.

– «margin: auto;» – властивість відповідає за центрування зображення всередині свого контейнера. Завдяки цьому зображення буде розташоване по центру каруселі незалежно від розміру екрана.

– «max-height: 1000px;» – задає максимальну висоту зображення, яка не може перевищувати 1000 пікселів. Це забезпечує, що зображення не стане занадто великим, коли екран буде збільшуватись.

– «object-fit: contain;» – властивість відповідає за збереження пропорцій зображення при масштабуванні. Значення contain означає, що зображення буде масштабуватись таким чином, щоб його ширина та висота вміщалися всередині свого контейнера, не втрачаючи пропорцій. Це забезпечує коректне відображення зображень незалежно від розміру екрана.

Далі у скриптовій частині компоненту у секції data задано налаштування для каруселі, такі як відображення стрілок та точок, плавність анімації та інші параметри.

Компонент з каруселлю відео створенний аналогічно, головною відмінністю є те що замість тегу <img> використовується тег <video>.

Тепер ці компоненти можна підключити на будь яких сторінках які потребують карусель з фото або відео. Код цих компонентів можна переглянути у додатку А.

3.5 Головний шаблон

Так як згідно структури інтерфейсу описаної у розділі 2, на кожній сторінці повинні бути: навігаційне меню, заголовок, підзаголовок, основний контент,

текстовий контент та футер. Для цього потрібно зробити основний шаблон сторінки сайту у який за допомогою x-slot буде об`єднувати всі необхідні елементи.

Для початку створено файл app.blade.php. У верхній частині файлу додано вказівку <!doctype html>, яка повідомляє браузеру, що документ відповідає стандарту HTML5. Також вказано мову документа за допомогою атрибуту lang.

Наступним у тезі <head> додано метатеги, які визначають кодування символів, налаштування відображення для різних пристроїв, а також CSRF-токен, який захищає від певних видів атак. Заголовок та опис сторінки будуть отримуватися від компонентів спадкоємців за допомогою x-slot з атрибутами name і значенням title та description.

Далі було підключено іконки, шрифти та файл app.css у якому зберігається всі стилі проекту у мінімізованому вигляді для того аби зменшити час завантаження сторінки.

У розділі <body> були підключені раніше створенні компоненти навігаційного меню та футера за допомогою <x-site.header/> та <x-site.footer/> а та створив блоки для відображення заголовка, підзаголовка та текстового контенту, які можуть бути передані за допомогою x-slot. За допомогою директив @isset виконується перевірка, чи передано дані, і якщо так – вони виводяться на сторінці. Заголовок та підзаголовок сторінки було вирішено виводити по центру для цього було створено клас text-component у якому за допомогою стилів margin-right: auto та margin-left: auto відцентрував блок, а за допомогою класу text-center відцентрував текст у цьому блоці. Для виведення основного контенту сторінки використовується змінна $slot, у яку автоматично підставлятиметься відповідний контент з шаблонів-спадкоємців. Код цього файлу можна переглянути у додатку А.

3.6 Сторінки з каруселлю фото та відео

Для реалізації сторінки, було створено файл index.blade у каталозі images у якому за допомогою x-slot встановлюються необхідні данні. Код цього файлу приведено нижче:

```
<x-layouts.site.app>
    <x-slot name="title">{{ $page->title }}</x-slot>
    <x-slot name="description">{{ $page->description ?? }}</x-slot>
    <x-slot name="h1">{{ $page->h1 }}</x-slot>
    <x-slot name="h2">{{ $page->h2 }}</x-slot>
    <x-slot name="text">{!! $page->text !!}</x-slot>
    <photos-component :images="{{ json_encode($images) }}" />
</x-layouts.site.app>
```

Цей файл буде розширяти головний шаблон сайту. Спочатку встановлюється заголовок сторінки, який відображається у вкладці браузера. Після цього задається опис сторінки, який використовується для пошукової оптимізації та відображення у снипетах пошукової системи. Далі встановлюються заголовки сторінки h1 та h2 і також текст який буде виводитися після каруселі. Також у цей файл підключається вижче описаний компонент з каруселлю фото.

Сторінка з каруселлю відео є анологічною, єдинною відмінністю є те, що замість <photos-component> використовується <videos-component>.

ВИСНОВКИ

У кваліфікаційній роботі було розроблено програмний інтерфейс для історичного музею.

Поставлену мету роботи повністю досягнуто, у процесі розробки вирішено такі задачі:

1. Проведено аналіз потреб та очікувань користувачів на основі якого були визначенні основні функції та елементи які має включати інтефейс.
2. Розроблені вимоги до програмного інтерфейсу.
3. Проведений аналіз та порівняння найбільш сучасних технологій у розробці програмних інтерфейсів. А саме: HTML, CSS та його препроцесори, бібліотеки стилів, JavaScript, популярні фреймворки для фронт-енд розробки, системи контролю версій, односторінкові та багатосторінкові додатки.
4. На основі проведеного аналізу було обрано найкращі інструменти для реалізації проекту.
5. Розроблений дизайн інтерфейсу.
6. Було введено ряд стратегій та технологій для вирішення проблеми адаптації інтерфейсу на пристроях різних розмірів.
7. Проведено ініціалізацію проекту у системі контролю версій Git.
8. Розроблено навігаційне меню.
9. Розроблено футер.
10. Реалізовано динамічні компоненти з реактивною каруселлю для відображення фото та відео експонатів.
11. Створено загальний шаблон сторінки у якому знаходиться заголовок, підзаголовок, основний контент, та текст який розміщується після нього.
12. Розроблені стилі для правильного розміщення елементів на сторінці.
13. Реалізовані сторінки для розміщення фото та відео експонатів.

У результаті отримано інтерфейс який відповідає всім сучасним вимогам щодо дизайну, технологій та користувацького досвіду. Він є інтуїтивно зрозумілим, зручним для користувача і легко адаптованим до різних розмірів екранів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. HTML Підручник. URL: https://w3schoolsua.github.io/html/index.html#gsc.tab=0 (дата звернення: 15.04.2023).

2. CSS Підручник. URL: https://w3schoolsua.github.io/css/index.html#gsc.tab=0 (дата звернення: 16.04.2023).

3. Сучасний підручник з JavaScript. URL: https://uk.javascript.info/ (дата звернення: 17.04.2023).

4. Sass vs LESS vs Stylus: Вибір препроцесора. URL: https://codeguida.com/post/288 (дата звернення: 19.04.2023).

5. The Pros and Cons of Tailwind CSS. URL: https://www.webdesignerdepot.com/2021/09/the-pros-and-cons-of-tailwind-css/ (дата звернення: 21.04.2023).

6. Bootstrap (front-end framework). URL: https://en.wikipedia.org/wiki/Bootstrap_(front-end_framework) (дата звернення: 22.04.2023).

7. What Are The Best Frontend Frameworks To Use In 2023. URL: https://www.ideamotive.co/blog/best-frontend-frameworks (дата звернення: 22.04.2023).

8. Системи Контролю Версіями. URL: https://repository.kpi.kharkov.ua/server/api/core/bitstreams/3ae67d24-d095-4da7-9537-7a105585dda9/content (дата звернення: 24.04.2023).

9. Version Control Software Comparison: Git, Mercurial,CVS, SVN. URL: https://medium.com/@derya.cortuk/version-control-software-comparison-git-mercurial-cvs-svn-21b2a71226e4 (дата звернення: 25.04.2023).

10. Single Page Application (SPA) vs Multi Page Application (MPA): Which Is The Best. URL: https://cleancommit.io/blog/spa-vs-mpa-which-is-the-king/ (дата звернення: 28.04.2023).

11. Grid system. URL: https://getbootstrap.com/docs/4.0/layout/grid/ (дата звернення: 03.05.2023).

12. Що таке медіа запити css і для чого вони потрібні. URL: https://freehost.com.ua/ukr/faq/articles/chto-takoe-media-zaprosi-css-i-dlja-chego-oni-nuzhni/(дата звернення: 05.05.2023).

13. Рендеринг списків. URL: https://ua.vuejs.org/guide/essentials/list.html#v-for (дата звернення: 06.05.2023).

14. API – Vue.js. URL: https://v2.vuejs.org.ua/v2/api/#v-bind (дата звернення: 06.05.2023).

15. Props. URL: https://vuejs.org/guide/components/props.html (дата звернення: 07.05.2023).

16. Blade Templates. URL: https://laravel.com/docs/10.x/blade (дата звернення: 07.05.2023).

ДОДАТКИ

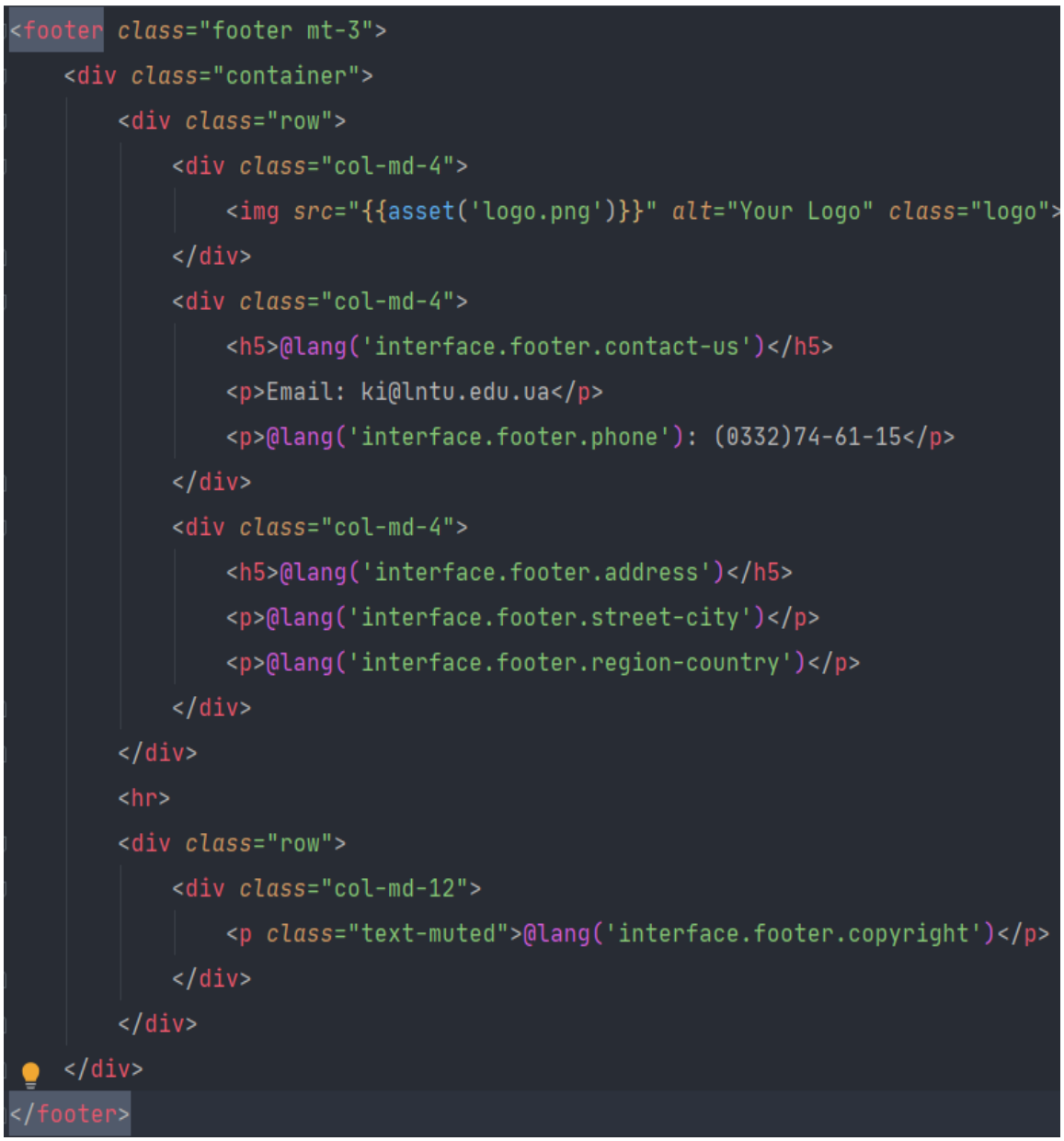
Додаток А

Код основних компонентів

```
<div>
    <nav class="navbar navbar-expand-lg navbar-font w-100 position-fixed">
        <button
            class="navbar-toggler"
            type="button" data-bs-toggle="collapse"
            data-bs-target="#navbarNav" aria-controls="navbarNav"
            aria-expanded="false"
            aria-label="Toggle navigation">
            <span class="navbar-toggler-icon"></span>
        </button>
        <div class="collapse navbar-collapse justify-content-center" id="navbarNav">
            <ul class="navbar-nav justify-content-center" style="...">
                @foreach($pages as $page)
                    <li style="...">
                        <a
                            class="nav-link active"
                            aria-current="page"
                            href="{{ $page->getUrl() }}"
                        >
                            {{ $page->name }}
                        </a>
                    </li>
                @endforeach
            </ul>
        </div>

        <div class="flex-row float-right w-auto d-flex" style="...">
            @foreach($localeLinks as $locale => $link)
                <a class="nav-link" href="{{ $link }}">{{ $locale }}</a>
                @if(! $loop->last) | @endif
            @endforeach
        </div>
    </nav>
    <img src="{{ asset('back1.jpg') }}" class="overlay_image" alt=""/>
</div>
```

Рисунок А.1 – Код навігаційної панелі header.blade.php



```
<footer class="footer mt-3">
    <div class="container">
        <div class="row">
            <div class="col-md-4">
                <img src="{{asset('logo.png')}}" alt="Your Logo" class="logo">
            </div>
            <div class="col-md-4">
                <h5>@lang('interface.footer.contact-us')</h5>
                <p>Email: ki@lntu.edu.ua</p>
                <p>@lang('interface.footer.phone'): (0332)74-61-15</p>
            </div>
            <div class="col-md-4">
                <h5>@lang('interface.footer.address')</h5>
                <p>@lang('interface.footer.street-city')</p>
                <p>@lang('interface.footer.region-country')</p>
            </div>
        </div>
        <hr>
        <div class="row">
            <div class="col-md-12">
                <p class="text-muted">@lang('interface.footer.copyright')</p>
            </div>
        </div>
    </div>
</footer>
```

Рисунок А.2 – Код футера footer.blade.php

```vue
<template>
  <div class="component_content col-sm-12 col-md-9">
    <VueSlickCarousel v-bind="settings" class="carousel col-sm-10 col-md-7">
      <div v-for="(image, index) in images" :key="index" class="slickslide">
        <img class="slideimage" :src="image.url" :alt="image.alt">
      </div>
    </VueSlickCarousel>
  </div>
</template>


<script>
  import VueSlickCarousel from 'vue-slick-carousel'
  import 'vue-slick-carousel/dist/vue-slick-carousel.css'
  import 'vue-slick-carousel/dist/vue-slick-carousel-theme.css'
  export default {
    data() {
      return {
        settings: {
          "dots": true,
          "arrows": true,
          "fade": true,
          "dotsClass": "slick-dots custom-dot-class",
          "edgeFriction": 0.35,
          "infinite": true,
          "speed": 500,
          "slidesToShow": 1,
          "slidesToScroll": 1,
          "adaptiveHeight": true,
        }
      }
    },
    props: {
      images: Array
    },
    name: "PhotosComponent",
    components: { VueSlickCarousel },
  }
</script>
```

Рисунок А.3 – Код компонента каруселі фото PhotosComponent.vue

```vue
<template>
  <div class="component_content col-sm-12 col-md-9">
    <VueSlickCarousel v-bind="settings" class="carousel col-sm-10 col-md-7">
      <div v-for="(video, index) in videos" class="slickslide">
        <video :id="'video'+index" class="slideimage" muted loop autoplay :key="index" width="700" height="400">
          <source :src="video.url">
        </video>
      </div>
    </VueSlickCarousel>
  </div>
</template>

<script>
  import ...
  // optional style for arrows & dots
  import 'vue-slick-carousel/dist/vue-slick-carousel-theme.css'
  export default {
    data() {
      return {
        settings: {
          "dots": true,
          "arrows": true,
          "dotsClass": "slick-dots custom-dot-class",
          "edgeFriction": 0.35,
          "infinite": true,
          "speed": 500,
          "slidesToShow": 1,
          "slidesToScroll": 1,
          "adaptiveHeight": true,
          "lazyLoad": 'ondemand',
        }
      }
    },
    props: {
      videos: Array
    },
    name: "VideosComponent",
    components: { VueSlickCarousel },
  }
</script>
```

Рисунок А.4 – Код компонента каруселі відео VideosComponent.vue

```blade
<!doctype html>
<html lang="{{ str_replace('_', '-', app()->getLocale()) }}">
    <head>
        <meta charset="utf-8">
        <meta name="viewport" content="width=device-width, initial-scale=1">
        <meta name="csrf-token" content="{{ csrf_token() }}">
        <title>{{ $title }}</title>
        <meta name="description" content="{{ $description }}">
        <link rel="dns-prefetch" href="//fonts.gstatic.com">
        <link href="https://fonts.googleapis.com/css?family=Nunito" rel="stylesheet">
        <link
            href="https://maxcdn.bootstrapcdn.com/font-awesome/4.1.0/css/font-awesome.min.css"
            rel="stylesheet">
        <link rel="stylesheet" href="{{ asset(mix('css/app.css')) }}">
    </head>
    <body>
        <x-site.header/>
        <div class = "text-component col--12 col-md-9 text-center mt-5" >
            {{-- Заголовок --}}
            @isset($h1)
                <h1>{{ $h1 }}</h1>
            @endisset

            {{-- Підзаголовок --}}
            @isset($h2)
                <h2>{{ $h2 }}</h2>
            @endisset
        </div>
        <div id="app">
            {{ $slot }}
        </div>
        <div class="text-component col-sm-12 col-md-9" style="font-size: 1.4em;">
            @isset($text)
                {!! $text !!}
            @endisset
        </div>
        <x-site.footer/>
        <script src="{{ asset(mix('js/app.js')) }}"></script>
    </body>
</html>
```

Рисунок А.5 – Код головного шаблона сайту app.blade.php

Додаток Б

Стилі index.scss

```
        body {
  display: flex;
  flex-direction: column;
  position: relative;
  background-color: whitesmoke;
}

html { overflow-y:scroll; }

html,body {
  height: 100%;
}

.slick-next:before {
  color: black !important;
  font-size: 2.5rem !important;
}

.slick-prev:before {
  position: relative;
  color: black !important;
  font-size: 2.5rem !important;
  left: -20px;
}

.navbar {
  background-color: white;
```

```
  box-shadow: 0 0 5px 5px rgb(0 0 0 / 10%);
  z-index: 1000;
}

.navbar-font {
  font-size: 23px !important;
}

.nav-link {
  color: #030712 !important;
}

.nav-link:hover {
  color: #3367D6 !important;
}

.dropdown-item {
  color: #3b3c93;
}

.overlay_container {
  position: relative;
  margin: 0;
  height: 100%;
}

.overlay_image {
  width: 100%;
  max-height: 500px;
  position: relative;
```

```
  top: 0;
  left: 0;
  z-index: 10;
}

.footer {
  bottom: 0;
  width: 100%;
  margin-top: auto;
  padding: 40px;
  background-color: white;
  box-shadow: 0 0 3px 3px rgb(0 0 0 / 10%);
}

.slideimage {
  width: inherit;
  margin: auto;
  max-height: 1000px;
  object-fit: contain;
}

.component_content {
  margin-right: auto;
  margin-left: auto;
  padding: 40px;
  height: 100%;
}

.text-component {
  margin-right: auto;
```

```
  margin-left: auto;
}

.carousel {
  margin: auto;
  max-height: 1000px;
}

.card {
  position: relative;
  overflow: hidden;
}

.card-img-overlay {
  position: absolute;
  top: 0;
  left: 0;
  right: 0;
  bottom: 0;
  background-color: rgba(0, 0, 0, 0.5);
  opacity: 0;
  transition: opacity 0.5s ease;
}

.card:hover .card-img-overlay {
  opacity: 1;
}

.fa-play-circle {
  font-size: 60px;
```

```
  color: rgba(255, 255, 255, 0.8);
  line-height: 70px;
  transition: transform 0.5s ease;
}

.logo {
  max-width: 150px;
  margin-top: -5px;
}

video::-webkit-media-controls-volume-slider {
  display: none;
}

video::-webkit-media-controls-mute-button {
  display: none;
}
```