Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет інформаційних систем та програмної інженерії
(повна назва факультету)
Кафедра програмної інженерії
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

бакалавр
(назва освітнього ступеня)

на тему: Дослідження та реалізація адаптивного дизайну веб-сайту для оптимального відображення на різних пристроях з використанням бібліотеки React.js та її екосистеми

Виконав(ла):
студент(ка) 4 курсу групи СП-42
спеціальності 121
«Інженерія програмного забезпечення»
(шифр і назва спеціальності)

(підпис) Глух О. М. (прізвище та ініціали)

Керівник (підпис) Сиротюк О.Б. (прізвище та ініціали)

Нормоконтроль (підпис) Стоянов Ю. А. (прізвище та ініціали)

Завідувач кафедри (підпис) Петрик М. Р. (прізвище та ініціали)

Рецензент (підпис) (прізвище та ініціали)

Тернопіль
2024

АНОТАЦІЯ

Кваліфікаційна робота бакалавра на тему «Дослідження та реалізація адаптивного дизайну веб-сайту для оптимального відображення на різних пристроях з використання бібліотеки React.js та її екосистеми» виконана Глухом Олегом Миколайовичом, студентом Тернопільського національного технічного університету імені Івана Пулюя, Факультет комп'ютерно-інформаційних систем і програмної інженерії, кафедра програмної інженерії, група СП-42.

Відомості про обсяг: сторінок – 83, рисунків – 18, частин – 4, додатків – 2, посилань – 14

Робота присвячена дослідженню адаптивного дизайну вебсайтів та використанню бібліотеки React для реалізації адаптивних вебдодатків. Реалізація адаптивного дизайну є необхідністю у сучасному світі, де користувачі активно використовують різні пристрої для доступу до інтернету.

У вступі вказано мету роботи: вивчити принципи адаптивного дизайну та розробити прототип вебсайту за допомогою React для забезпечення оптимального відображення на різних пристроях. Завданнями роботи є вивчення історії, основних принципів адаптивного дизайну, можливостей React, розробка та тестування адаптивного вебсайту.

Перший розділ охоплює теоретичні основи адаптивного дизайну, описує його історію, основні принципи, такі як гнучкі макети, зображення, медійні запити та інші сучасні методи. Також проводиться порівняння адаптивного, гнучкого та флюїдного дизайну.

Другий розділ детально розглядає бібліотеку React, її основні концепції, такі як компонентний підхід, хуки, віртуальний DOM та односпрямований потік даних. Окрім того, аналізуються основні елементи екосистеми React, включаючи React Router, React Hook Form, Redux Toolkit та інші бібліотеки.

Третій розділ містить практичну частину роботи: опис прототипу проєкт "Nuegas", концептуальний дизайн якого допомагає користувачам керувати завданнями та взаємодіяти з менторами. Продемонстровано вибір редактора коду, організацію структури проєкту, створення базового прототипу, опис структури та загальних підходів до адаптивного дизайну на конкретних прикладах реалізації платформи, та тестування вебсайту на різних пристроях і браузерах.

ANNOTATION

The diploma thesis is dedicated to the investigation of adaptive web design and the use of the React library for implementing adaptive web applications. The implementation of adaptive design is a necessity in the modern world, where users actively use various devices to access the internet.

The bachelor's thesis on the topic "Research and Implementation of Adaptive Website Design for Optimal Display on Different Devices using React.js Library and its Ecosystem" was carried out by Ole Mykolayovych Glukh, a student of Ivano-Frankivsk National Technical University of Oil and Gas, Faculty of Computer and Information Systems and Software Engineering, Department of Software Engineering, group SP-42.

Information about the scope: pages - 80, figures - 18, sections - 4, appendices - 2, references - 14.

The introduction states the purpose of the work: to study the principles of adaptive design and develop a website prototype using React to ensure optimal display on different devices. The tasks of the thesis include studying the history, basic principles of adaptive design, React capabilities, development, and testing of an adaptive website.

The first chapter covers the theoretical foundations of adaptive design, describing its history, basic principles such as flexible layouts, images, media queries, and other modern methods. A comparison of adaptive, flexible, and fluid design is also carried out.

The second chapter thoroughly examines the React library, its main concepts such as a component-based approach, hooks, virtual DOM, and unidirectional data flow. In addition,

the main elements of the React ecosystem are analyzed, including React Router, React Hook Form, Redux Toolkit, and other libraries.

The third chapter contains the practical part of the work: a description of the "Nuegas" project prototype, the conceptual design of which helps users manage tasks and interact with mentors. It demonstrates the choice of code editor, project structure organization, creation of a basic prototype, description of the structure and general approaches to adaptive design with specific examples of platform implementation, and website testing on various devices and browsers.

ЗМІСТ

ВСТУП

Зі стрімким розвитком інформаційних технологій і зростанням популярності мобільних пристроїв адаптивний дизайн стає важливою частиною сучасної веброзробки. Згідно з дослідженнями, більша частина сьогоднішнього інтернет-трафіку генерується мобільними пристроями. Користувачі очікують, що вебсайти будуть простими у використанні та однаково функціональними на різних платформах, від смартфонів і планшетів до настільних ПК і великих екранів.

Адаптивний дизайн дозволяє вебсайту автоматично адаптуватися до різних розмірів екрана та орієнтації пристрою, забезпечуючи кращу взаємодію з користувачем. Для досягнення цієї мети він використовує гнучкі макети, гнучкі зображення, медіазапити та інші сучасні методи.

На ринку з’являються нові інструменти та бібліотеки, які дозволяють розробникам ефективніше створювати адаптивні вебсайти. Однією з них є бібліотека React, яка пропонує високу продуктивність, гнучкість і легкість розширення. Експерти з дизайну React і адаптивного дизайну сьогодні мають великий попит, і ця технологія стає все більш популярною серед багатьох компаній для розробки масштабованих, гнучких і адаптивних вебдодатків.

Мета роботи – вивчити принципи адаптивного дизайну та реалізації сайту за допомогою бібліотеки React для забезпечення оптимального відображення на різних пристроях та розмірах екрана.

Завдання роботи:

1. Вивчити історію та розвиток концепцій адаптивного дизайну.

2. Ознайомитися з основними принципами адаптивного дизайну: гнучкі макети, гнучкі зображення, медійні запити.

3. Дослідити можливості та переваги використання бібліотеки React у контексті адаптивного дизайну.

4. Розробити прототип вебсайту з використанням React, застосовуючи принципи адаптивного дизайну.

5. Виконати оптимізацію зображень та мультимедійного контенту для гнучкої адаптації до різних розмірів екранів.

6. Провести тестування вебсайту на різних пристроях та екранах, проаналізувати результати та внести необхідні корективи.

Методологія дослідження базується на використанні як теоретичних, так і практичних методів.

1. Аналіз наукових та технічних літературних джерел з питань адаптивного дизайну. Вивчення статей, книг, документів, що описують методи та підходи до реалізації адаптивного дизайну, технологій HTML, CSS та JavaScript.

2. Практичний аналіз з використанням бібліотеки React та її екосистеми, включаючи інструменти для адаптивного дизайну, такі як медійні запити, гнучкі зображення, Flexbox і Grid Layout.

3. Емпіричні дослідження для тестування та оцінки вебсайту на різних пристроях. Використання інструментів розробки, таких як Chrome DevTools, для перевірки роботи сайту на різних розмірах екрана та пристроях.

4. Практична розробка прототипу вебсайту, включаючи створення дизайну, розмітки та стилізації, а також інтеграцію функціональних компонентів для забезпечення адаптивності сайту.

Наукова новизна заходу полягає в комплексному підході до дослідження та реалізації адаптивного дизайну, інтеграції сучасних методів веброзробки з використанням бібліотеки React. Це дослідження пропонує нові методи та підходи для створення адаптивних вебсайтів з урахуванням властивостей і можливостей сучасних бібліотек і фреймворків.

Практичне значення дослідження полягає у створенні вебсайту, який демонструє найбільш ефективні методи адаптивного дизайну з використанням бібліотеки React. Створений прототип можна використовувати як платформу для потенційної комерційної чи некомерційної діяльності. Крім того, надані правила та приклади коду можуть бути дуже корисними для розробників, які хочуть розширити свої можливості у створенні адаптивних вебсайтів.

Дипломна робота складається з трьох основних розділів та висновків.

1. Теоретичні основи адаптивного дизайну – історія розвитку, основні принципи, порівняння адаптивного та гнучкого дизайну.
2. Аналіз бібліотеки React та її екосистеми – основні концепції React, елементи екосистеми та інші популярні бібліотеки.
3. Реалізація адаптивного дизайну в проєкті на React – створення базового проєкту, використання медійних запитів, реалізація гнучких макетів, оптимізація зображень та тестування на різних пристроях.

РОЗДІЛ 1. ТЕОРЕТИЧНІ ОСНОВИ АДАПТИВНОГО ДИЗАЙНУ

1.1 Історія та розвиток адаптивного дизайну

Адаптивний дизайн є відносно новою концепцією в історії веброзробки, яка виникла у відповідь на зростаючу різноманітність пристроїв, що використовуються для перегляду вебсайтів. Початково, веброзробники створювали сайти, орієнтовані переважно на настільні комп'ютери з великою роздільною здатністю екранів. Проте з появою смартфонів, планшетів і інших мобільних пристроїв стала необхідність у гнучких підходах, які забезпечують комфортне використання незалежно від розміру екрана.

Перші спроби пристосувати контент до різних розмірів екранів включали створення окремих мобільних версій сайтів. Однак цей підхід мав суттєві обмеження, пов'язані з необхідністю підтримки декількох версій одного й того ж сайту, що вимагало значних ресурсів і створювало додаткові труднощі для розробників.

Виникнення концепції адаптивного дизайну

Перші спроби створити єдине рішення для різних пристроїв започаткували концепцію адаптивного дизайну, а початок цьому руху поклав Ітан Маркотт (Ethan Marcotte) у 2010 році своїм відомим есе "Responsive Web Design". Він запропонував нові підходи до розробки сайтів, використовуючи гнучкі сітки (flexible grids), гнучкі зображення (flexible images) та медійні запити (media queries), які дозволяють вебсторінкам динамічно підлаштовуватися під різні розміри екранів без потреби створення окремих версій для кожного пристрою [4].

Гнучкі сітки — використовують відсоткові або відносні одиниці виміру, такі як em або rem, замість фіксованих одиниць виміру, що дозволяє елементам інтерфейсу змінювати свої розміри відповідно до розміру екрана.

Гнучкі зображення — забезпечують можливість масштабування зображень разом зі зміною розмірів контейнера, в якому вони розміщені, завдяки використанню відсоткових значень ширини замість фіксованих пікселів.

Медійні запити — дозволяють змінювати стилі CSS залежно від характеристик пристрою, таких як ширина, висота, роздільна здатність та орієнтація екрана. Це дозволяє створювати різні макети для різних умов перегляду, забезпечуючи оптимальне відображення на будь-якому пристрої.

Подальший розвиток адаптивного дизайну

З часом концепція адаптивного дизайну отримала значний розвиток та визнання у світовій спільноті веброзробників. Сучасні вебтехнології підтримують нові можливості, що ще більше спрощують реалізацію адаптивного дизайну.

Наприклад, такі інструменти як CSS Flexbox та CSS Grid значно полегшують створення гнучких макетів та розміщення елементів на сторінці. Вони дозволяють більш точно контролювати розмір і розташування елементів, покращуючи користувацький досвід на різних пристроях.

CSS Flexbox - система компонування, що дозволяє створювати прості гнучкі макети, які автоматично адаптуються до розміру контейнера. Вона особливо корисна для створення адаптивних навігаційних панелей, каркасів сторінок і контейнерів для контенту [5].

CSS Grid - більш потужна система компонування, яка надає значно більші можливості для організації елементів на сторінці. Вона дозволяє створювати складні макети з кількома рядами і колонками, які можуть динамічно адаптуватися до різних розмірів екранів.

Подальше дослідження адаптивного дизайну зосереджується на поліпшенні інструментів і методів, що дозволяють досягти вищого рівня адаптивності та зручності використання вебсайтів.

Дослідження нових технік включає використання змінних CSS для створення динамічних стилів, які можуть змінюватися залежно від контексту. Також застосовуються нові одиниці виміру, такі як vw (viewport width), vh (viewport height), vmin і vmax, які дозволяють більш точно контролювати розмір елементів на основі розміру вікна перегляду. Поліпшення технік оптимізації зображень включає використання сучасних форматів зображень, таких як WebP, і технік завантаження зображень, таких як lazy loading, для покращення продуктивності та зменшення часу завантаження сторінок.

В результаті, адаптивний дизайн продовжує еволюціонувати, забезпечуючи користувачам найкращий досвід перегляду вебсайтів незалежно від пристрою, яким вони користуються.

1.2 Основні принципи адаптивного дизайну

Адаптивний дизайн належить до методології створення вебсайтів, які автоматично підлаштовуються під різні розміри екранів і орієнтації пристроїв, забезпечуючи кращий користувацький досвід. Основні принципи адаптивного дизайну включають гнучкі макети, гнучкі зображення, медійні запити та інші важливі концепції. Розгляньмо кожен з них детальніше.

1.2.1 Гнучкі макети

Адаптивний дизайн базується на використанні гнучких макетів, які застосовують відсоткові або відносні одиниці вимірювання, такі як em, rem, vw та vh, замість фіксованих пікселів для налаштування розмірів елементів. Це дозволяє інтерфейсним елементам динамічно змінювати свої розміри відповідно до розміру екрана, що є ключовим аспектом у створенні адаптивного дизайну.

Відсоткові одиниці вимірювання забезпечують високу гнучкість при створенні макетів, оскільки дозволяють елементам масштабуватися відносно розміру їхнього контейнера. Наприклад, коли використовуються відсоткові значення для ширини блоків, вони автоматично підлаштовуються під ширину вікна браузера, сприяючи оптимальному відображенню контенту на різних пристроях.

Відносні одиниці вимірювання, такі як em і rem, дозволяють налаштовувати розміри елементів залежно від розміру шрифту або кореневого елемента. Це забезпечує додаткову гнучкість та зручність у застосуванні стилів, оскільки зміна розміру шрифту на одному рівні може автоматично вплинути на розміри всіх підлеглих елементів. Таким чином, забезпечується консистентність і узгодженість дизайну на всіх рівнях ієрархії елементів.

Використання сучасних технологій CSS, таких як Flexbox та Grid Layout, значно полегшує процес створення гнучких макетів. CSS Flexbox забезпечує зручний спосіб розташування елементів в одному вимірі, що забезпечує гнучкість при організації контейнера і його вмісту з урахуванням усіх можливих змін розмірів екрана. Своєю чергою, CSS Grid дозволяє створювати складні двомірні макети з кількома рядами й

колонками, що надає можливість більш точно контролювати розташування елементів у просторі сторінки [6].

В цілому, гнучкі макети сприяють створенню адаптивних вебдизайнів, які легко підлаштовуються під будь-який розмір екрана, забезпечуючи зручність та інтуїтивність користування вебресурсу на різних пристроях. Це особливо актуально у сучасних умовах, коли роль мобільних пристроїв у споживанні інтернет-контенту постійно зростає. Завдяки цьому підходу до вебдизайну, користувачі можуть насолоджуватись комфортним переглядом і взаємодією з вебресурсами, незалежно від типу пристрою, який вони використовують.

1.2.2 Гнучкі зображення

Однією з ключових проблем в адаптивному дизайні є забезпечення коректного відображення зображень на різних пристроях і розмірах екранів. Гнучкі зображення відіграють вирішальну роль у цьому процесі, дозволяючи зображенням автоматично масштабуватися та підлаштовуватися під розмір контейнера [7].

Одним із ключових методів для досягнення цієї мети є використання відсоткової ширини. Стиль max-width: 100% дозволяє зображенням масштабуватися відносно розміру свого контейнера без втрати пропорцій, що забезпечує їхнє коректне відображення на різних пристроях. Це сприяє гармонійному вписанню зображень у структуру сайту, забезпечуючи оптимальне користувацьке враження.

Важливим аспектом оптимізації зображень є використання сучасних форматів зображень, таких як WebP та AVIF. Ці формати забезпечують високу якість зображень при значно меншому розмірі файлів у порівнянні з традиційними форматами, такими

як JPEG чи PNG. Застосування таких форматів позитивно впливає на продуктивність вебсайтів, зменшуючи час завантаження сторінок і споживання трафіку.

Ще однією важливою технікою є використання елемента <picture> і атрибута srcset в HTML5. Ці інструменти дозволяють завантажувати різні версії зображення залежно від роздільної здатності або інших характеристик пристрою. Зокрема, атрибут srcset дозволяє визначати кілька варіантів одного зображення, що дає змогу браузера обирати оптимальну версію для конкретного пристрою. Елемент <picture> дає можливість ще детальніше налаштовувати вибір зображень, враховуючи не лише роздільну здатність, але й інші параметри, такі як орієнтація екрана.

Загалом, інтеграція гнучких зображень є важливим елементом адаптивного дизайну, що забезпечує високу якість користувацького досвіду на різних пристроях. Завдяки таким підходам, вебресурси можуть ефективно масштабувати контент, забезпечуючи його коректне та оптимальне відображення незалежно від параметрів пристроїв, які використовуються користувачами.

1.2.3 Медійні запити

Медійні запити (media queries) є невід'ємною частиною адаптивного дизайну, що дозволяють змінювати стилі CSS відповідно до характеристик пристрою, таких як ширина, висота, роздільна здатність чи орієнтація екрана.

Базові медійні запити є одним з основних інструментів для створення адаптивного дизайну. Вони дозволяють застосовувати стилі до певних елементів залежно від розмірів вікна перегляду. Наприклад, простий медійний запит з

параметром @media (max-width: 768px) застосовує стилі до елементів класу container лише тоді, коли ширина вікна перегляду є меншою або дорівнює 768 пікселям. Це дозволяє додавати відступи до контейнера для кращого відображення на мобільних пристроях.

Медійні запити також можуть бути комбінованими для створення ще більш гнучких макетів. Наприклад, комбінація запитів @media (min-width: 600px) and (max-width: 1200px) приховує елемент з класом sidebar, коли ширина вікна знаходиться в діапазоні від 600 до 1200 пікселів. Це підхід дозволяє гнучкіше реагувати на різні розміри екранів і специфічні вимоги до дизайну [8].

Сучасні одиниці вимірювання, такі як vw (viewport width) та vh (viewport height), також надають додаткову гнучкість при створенні адаптивних стилів. Вони дозволяють задавати розміри елементів відносно розміру вікна перегляду, що сприяє кращій адаптації інтерфейсу до різноманітних пристроїв.

Використання функції clamp() у CSS забезпечує ще один рівень гнучкості. Ця функція дозволяє обмежувати значення властивостей між мінімальним і максимальним значенням, що є особливо корисним при створенні адаптивних типографічних шкал та розмірів елементів. Наприклад, синтаксис font-size: clamp(1rem, 2.5vw, 2rem) задає мінімальний розмір шрифту 1rem, максимальний розмір 2rem, але дозволяє їм масштабуватися відносно ширини вікна перегляду. Це зменшує потребу в додаткових медійних запитах та підвищує гнучкість стилів [9].

Запити до контейнерів (Container Queries) є майбутнім стандартом CSS, що дозволяє застосовувати стилі до елементів на основі розмірів їхніх контейнерів. Ідея полягає в тому, щоб стилі застосовувалися до елемента залежно від його безпосереднього контейнера, що забезпечує більшу гнучкість при створенні комплексних компонентів. Наприклад, запит @container (min-width: 300px) змінює стилі компонента card залежно від розміру його контейнера.

Цей підхід підвищує модульність та повторне використання коду. Завдяки адаптивності окремих компонентів незалежно від контексту їх використання, створення адаптивний вебдизайн стає більш ефективним і зручним.

Основні принципи адаптивного дизайну – це передумови для створення вебсайту, які здатні забезпечити високий рівень зручності та функціональності на будь-яких пристроях. Гнучкі макети, гнучкі зображення, медійні запити та новітні техніки, такі як clamp() та запити до контейнерів, допомагають розробникам створювати сайти, що є доступними та зручними для користувачів незалежно від того, на яких пристроях вони переглядаються.

1.3 Основні підходи до вебдизайну

Даний розділ висвітлює різні підходи до дизайну вебсайтів і їх оптимізацію для різних пристроїв. Один із таких підходів - "Mobile First". Окрім того, ми розглянемо адаптивний, гнучкий та флюїдний дизайн.

1.3.1 Підхід "Mobile First"

Mobile First – це підхід до вебдизайну та розробки, де спочатку створюється дизайн для мобільних пристроїв, а потім він поступово розширюється до планшетів, настільних комп’ютерів та інших більших екранів. Ідея цього підходу полягає в тому, щоб спочатку розв'язувати важливі проблеми обмежених пристроїв, таких як

смартфони, а потім додавати додаткову функціональність та покращення для більших екранів [10].

Цей підхід зосереджується на ряді особливостей. Прогресивне покращення передбачає початок розробки зі створення базового функціонала та дизайну для мобільних пристроїв з подальшим додаванням функціональних можливостей і стилів для більших екранів. Ще однією важливою характеристикою є врахування обмежених ресурсів мобільних пристроїв, таких як швидкість завантаження, процесор, пам'ять і трафік, з акцентом на оптимізацію для сенсорних пристроїв та інтерфейсів, адаптованих для використання пальцями.

Підхід Mobile First пропонує декілька переваг. Він покращує користувацький досвід на мобільних пристроях, забезпечуючи спрощений інтерфейс і підвищену продуктивність. Також цей підхід забезпечує послідовний користувацький досвід на різних пристроях завдяки уніфікованому заповненню контентом. Спрощення дизайну та функціональності для мобільних користувачів сприяє швидшому завантаженню та обробці даних.

Втім, існують і певні недоліки підходу Mobile First. Початок з мобільного дизайну може ускладнити реалізацію складних функцій для настільних комп'ютерів, і може потребувати більше часу для задоволення потреб користувачів різних пристроїв.

1.3.2 Адаптивний дизайн (Adaptive design)

Адаптивний дизайн ґрунтується на використанні декількох фіксованих макетів, кожен з яких призначений для певного діапазону розмірів екранів. Вебсайт автоматично вибирає найбільш відповідний макет залежно від розміру вікна перегляду пристрою. Основні особливості цього підходу включають використання

фіксованих макетів для різних точок перелому (breakpoints), застосування медійних запитів для перемикання між макетами, а також забезпечення оптимального відображення на різних пристроях.

Перевагами адаптивного дизайну є підвищений контроль над дизайном для кожного типу пристрою та оптимізація продуктивності шляхом завантаження конкретних макетів для кожного пристрою. Це дозволяє створити дизайн, що найкраще відповідає можливостям та обмеженням кожного пристрою. Втім, адаптивний дизайн має свої недоліки. Зокрема, виникає необхідність створення та підтримки окремих макетів для кожного типу пристрою, що може бути досить трудомістким. Крім того, можливі розриви між макетами, якщо розмір вікна перегляду відрізняється від передбачених точок перелому.

1.3.3 Гнучкий дизайн (Responsive design)

Гнучкий дизайн базується на використанні гнучких сіток, гнучких зображень і медійних запитів для створення єдиного макета, який адаптується до будь-якого розміру екрана. Основні особливості Гнучкого дизайну включають використання відносних одиниць вимірювання, таких як відсотки, em та rem. Гнучкі зображення автоматично масштабуються відповідно до розміру контейнера, а медійні запити дозволяють адаптувати стилі до різних розмірів екранів.

Перевагами гнучкого дизайну є його універсальність – один макет адаптується до будь-якого розміру екрана, значно зменшуючи обсяг роботи з підтримки, оскільки немає потреби створювати декілька окремих макетів. Однак, гнучкий дизайн може бути складнішим у налаштуванні порівняно з адаптивним дизайном через необхідність точного налаштування параметрів для кожного можливого розміру

екрана. Крім того, можуть виникати проблеми з продуктивністю при завантаженні великого обсягу стилів, необхідного для забезпечення гнучкості макета.

1.3.4 Флюїдний дизайн (Fluid Design)

Флюїдний дизайн використовує відсоткові одиниці вимірювання для встановлення всіх розмірів, що дозволяє елементам автоматично підлаштовуватися під розмір екрана. Основні особливості цього підходу включають використання відсоткових значень для ширини, висоти та інших розмірів елементів, що дозволяє елементам змінюватися пропорційно до розмірів контейнера чи вікна браузера. Це забезпечує плавні переходи між різними розмірами екранів.

Серед переваг флюїдного дизайну можна виділити плавність змін розмірів елементів, що забезпечує приємний користувацький досвід, а також значне спрощення процесу дизайну, знижуючи кількість медійних запитів шляхом використання відсоткових значень. Водночас цей підхід має свої недоліки. Наприклад, складніше забезпечити точне розташування елементів на всіх розмірах екранів, і можливі конфлікти стилів та розриви у дизайні на дуже великих або дуже малих екранах.

1.4 Порівняння підходів.

Порівнюючи різних підходів до вебдизайну можна відзначити кілька ключових відмінностей і особливостей. Підхід Mobile First починається з розробки дизайну для

мобільних пристроїв і поступово додає стилі для більших екранів. Своєю чергою, адаптивний дизайн використовує фіксовані макети для різних точок перелому, що дозволяє краще контролювати вигляд сайту на конкретних пристроях. Гнучкий дизайн створює один гнучкий макет, який плавно адаптується до будь-якої ширини екрана, використовуючи відносні одиниці вимірювання, гнучкі зображення і медійні запити. Флюїдний дизайн використовує відсоткові одиниці вимірювання для встановлення всіх розмірів елементів, забезпечуючи плавність переходів між різними розмірами екранів.

Кожен із підходів має різний ступінь контролю над дизайном. Mobile First забезпечує послідовний і оптимізований дизайн для мобільних пристроїв. Адаптивний дизайн дає більше контролю над виглядом сайту на конкретних пристроях, тоді як гнучкий дизайн забезпечує більш універсальний підхід, який потребує точного налаштування для кожного компонента. Флюїдний дизайн забезпечує плавність змін розмірів, але може потребувати додаткових налаштувань для коректного відображення.

Гнучкість та адаптація також відрізняються між підходами. Mobile First починає з базового дизайну і поступово додає функції для більших екранів. Адаптивний дизайн може недосконало виглядати на розмірах екранів, які не відповідають заданим точкам перелому. Гнучкий дизайн забезпечує плавну адаптацію незалежно від розміру екрана. Флюїдний дизайн дозволяє змінювати розміри елементів плавно, але може потребувати додаткових налаштувань для конкретних розмірів екранів [11].

З погляду продуктивності, Mobile First забезпечує високу продуктивність для мобільних пристроїв, оптимізуючи завантаження ресурсу. Адаптивний дизайн має перевагу у можливості завантаження тільки необхідного макету для кожного пристрою, що знижує витрати ресурсів. Гнучкий дизайн, з іншого боку, завантажує один макет, що може вплинути на продуктивність, якщо він не оптимізований. Флюїдний дизайн вимагає мінімальних ресурсів для зміни розмірів елементів, хоча й

може потребувати додаткових налаштувань для забезпечення коректного відображення на різних розмірах екранів.

Вибір між Mobile First, адаптивним, гнучким та флюїдним підходами залежить від конкретних вимог проекту, бюджету, ресурсів та цільової аудиторії. Mobile First забезпечує оптимізований досвід для мобільних користувачів і послідовний перехід до більших екранів. Адаптивний дизайн забезпечує більший контроль і оптимізацію для конкретних пристроїв, але вимагає більше зусиль для розробки та підтримки. Гнучкий дизайн пропонує універсальність і простішу підтримку, але може бути складнішим у розробці дійсно гнучких макетів. Флюїдний дизайн додає плавності та гнучкості, але може вимагати додаткових налаштувань для забезпечення коректного відображення на всіх розмірах екранів.

Таким чином, сучасні веброзробники часто використовують комбінації цих підходів для досягнення оптимального результату, забезпечуючи зручність та доступність вебсайтів для максимально широкої аудиторії.

РОЗДІЛ 2. REACT ТА ЙОГО ЕКОСИСТЕМА

2.1 Знайомство з React

React є однією з найпопулярніших бібліотек для розробки користувацьких інтерфейсів, створеною компанією Facebook у 2013 році. Вона забезпечує високу продуктивність, гнучкість та легкість у розширенні, що робить її ідеальним інструментом для створення сучасних вебдодатків.

2.1.1 Основні концепції React

React базується на компонентному підході, що означає, що користувацький інтерфейс складається з незалежних, багаторазово використовуваних блоків – компонентів. Кожен компонент інкапсулює власну логіку та частину відображення, що дозволяє розробникам створювати складні додатки із простих і ізольованих елементів.

Функціональні компоненти є простими JavaScript-функціями, які приймають props (властивості) як аргумент і повертають JSX-елементи.

JSX (JavaScript XML) – це синтаксичне розширення JavaScript, яке дозволяє писати HTML-подібний код усередині JavaScript. Він дозволяє розробникам більш інтуїтивно описувати користувацькі інтерфейси, а також інтегрувати динамічні дані та логіку безпосередньо у шаблони.

```
function Welcome(props: WelcomeProps) : JSX.Element  {
  return <h1>Привіт, {props.name}</h1>;
}
```

Рисунок 2.1 - Приклад компонента із використання прорсів(props)

React-компоненти мають важливі поняття, які допомагають їм керувати даними: props (властивості) та state (стан).

Props - це дані, які компонент отримує ззовні, зазвичай від його батьківського компонента. Їх роль полягає у передачі інформації в компонент для відображення. Потрібно зауважити, що props є незмінними (immutable), тобто після їх отримання компонент не може змінити їх [12].

Наприклад, у компоненті Welcome(Рисунок 2.1) використовується властивість props.name для вітання користувача.

State - це внутрішній стан компонента, який може змінюватися під час взаємодії користувача чи інших подій. Він дозволяє компоненту зберігати та оновлювати дані під час його життєвого циклу.

Наприклад, у компоненті Counter(Рисунок 2.2) використовується стан, який включає лічильник (count) та функцію setCount для зміни цього лічильника. При кожному натисканні кнопки значення лічильника збільшується на одиницю.

Таким чином, використання props і state допомагає React-компонентам ефективно керувати даними та їх відображенням.

```
function Counter() {  Show usages
  // Використовуємо useState для створення стану з початковим значенням 0
  const [count : number , setCount] = useState( initialState: 0);

  // Функція для збільшення значення state (лічильника) на одиницю
  const increaseCount = () : void  => {  Show usages
    setCount( value: count + 1);
  };

  return (
    <div>
      <p>Ви натиснули кнопку {count} разів</p>
      <button onClick={increaseCount}>Натисни мене</button>
    </div>
  );
}
```

Рисунок 2.2 Приклад компоненту із використання стану(state).

React Hooks

В React, розробники широко використовують поняття хуків, які дозволяють досягати певних функціональних можливостей у функціональних компонентах. Серед найпоширеніших хуків є useState, useEffect, useContext, useReducer, useMemo, useCallback і useRef.

Хук useState забезпечує можливість створення внутрішніх станів у компонентах, що дозволяє їм зберігати та оновлювати дані. useEffect використовується для роботи з побічними ефектами та виконання певних операцій після рендерингу компонента.

Хук useContext дозволяє отримувати доступ до значень з контексту, спрощуючи передачу даних через ієрархію компонентів. useReducer надає зручний спосіб керування складними структурами станів у компонентах.

Хуки useMemo та useCallback допомагають в оптимізації продуктивності програм шляхом кешування обчислень та колбек-функцій відповідно. useRef використовується для створення посилань на DOM-елементи або сталих значень у компонентах.

Ці хуки допомагають розробникам впроваджувати функціональності та підвищувати продуктивність функціональних компонентів у React, що робить їх невід'ємною складовою платформи для створення динамічних та ефективних вебдодатків.

Віртуальний DOM

React використовує концепцію віртуального DOM для оптимізації оновлень інтерфейсу користувача, що призводить до покращення продуктивності вебдодатків. Віртуальний DOM являє собою легку копію реального DOM, що зберігається у пам'яті та використовується для ефективного визначення оптимальних шляхів впровадження змін до реального DOM.

Використання віртуального DOM дозволяє досягти швидких і ефективних оновлень частини сторінки без необхідності маніпулювання реальним DOM напряму. Це призводить до зниження кількості операцій, необхідних для оновлення вебсторінки, що своєю чергою поліпшує продуктивність вебдодатків та забезпечує кращий досвід користувача.

React дотримується концепції односпрямованого потоку даних, що означає, що дані передаються від батьківського компонента до дочірнього через props. Це забезпечує більш передбачувану і легко керовану структуру даних додатка. Односпрямований потік даних дозволяє легше відстежувати та налагоджувати додаток, оскільки стає зрозумілим, де знаходяться дані та як вони змінюються.

Життєвий цикл компонента

У функціональних компонентах React можна керувати життєвим циклом компонентів за допомогою хуків, зокрема useEffect. Цей хук надає можливість визначати поведінку компонента на різних етапах його життєвого циклу.

Етапи життєвого циклу компонента включають.

- Монтування це етап відбувається, коли компонент вперше додається до DOM. Під час цього етапу можна виконувати дії, такі як ініціалізація стану чи виконання запитів до сервера.
- Оновлення. Компонент уже знаходиться в DOM і оновлюється у відповідь на зміни даних чи props. За допомогою useEffect можна визначати дії, які мають відбутися при кожному оновленні компонента.
- Демонтування. На цьому етапі компонент видаляється з DOM. Викликаючи певні функції в useEffect з певними налаштуваннями, можна забезпечити очищення ресурсів, підписок або скасовувати запущених процесів перед видаленням компонента.

Контролюючи ці етапи за допомогою useEffect та інших хуків, розробники можуть ефективно керувати поведінкою компонентів на різних етапах їх життєвого циклу.

2.1.2 Порівняння з конкурентами

React, з Angular та Vue.js, є одним із найпопулярніших інструментів для розробки користувацьких інтерфейсів. Проте, варто відзначити, що він не протиставляється іншим фреймворкам, а також має свої конкуренти у формі Angular та Vue.js.

Angular, розроблений компанією Google, є повноцінним фреймворком для створення вебдодатків. Він пропонує повний підхід до архітектури через MVC (Model-View-Controller), що включає двонаправлену прив'язку даних. Angular використовує TypeScript для узагальнення та пропонує широкий спектр вбудованих функцій. Навчання Angular може вимагати більше часу через його складність та швидкі зміни в версіях.

Vue.js, створений Evan You, є прогресивним фреймворком для створення користувацьких інтерфейсів. Підтримуючи компонентний підхід, він схожий на React, та також підтримує двонаправлену прив'язку даних. Vue.js відрізняється легкістю вивчення, простим синтаксисом і здатністю легко інтегруватися з іншими проєктами.

Основні відмінності між цими інструментами полягають у підходах до архітектури, використання мов програмування, складності та продуктивності. React відзначається високою продуктивністю завдяки віртуальному DOM, Angular пропонує масштабованість та включає велику кількість вбудованих інструментів, тоді як Vue.js є мінімалістичним та ефективним фреймворком.

Крім того, кожен інструмент має свою спільноту та підтримку, де React та Angular мають широка спільнота та регулярну підтримку від відомих компаній, а Vue.js визначається швидким зростанням спільноти та активною підтримкою [13].

Вибір між React, Angular і Vue.js залежить від конкретних потреб проєкт, досвіду команди розробки та особистих вподобань. React виділяється своєю гнучкістю, простотою інтеграції та розширюваністю, що робить його ідеальним вибором для багатьох сучасних вебдодатків. Angular пропонує повний набір інструментів для масштабних додатків, тоді як Vue.js забезпечує простоту вивчення і легкість інтеграції в наявний проєкт.

Наступним кроком ми розглянемо елементи екосистеми React, які доповнюють і розширюють основну функціональність бібліотеки.

2.2 Елементи екосистеми React

React надає розробникам можливість створювати багатофункціональні інтерфейси користувача, але він також має велике оточення додаткових бібліотек та інструментів, що розширюють його можливості та спрощують розробку складних додатків. Нижче ми розглянемо найважливіші елементи екосистеми React.

React Router - це потужна бібліотека для маршрутизації в React-додатках, яка дозволяє створювати односторінкові застосунки з динамічними маршрутами. Основні можливості React Router включають декларативну маршрутизацію, підтримку динамічних маршрутів, захист маршрутів та можливість створення вкладених маршрутів для ефективної організації додатків.

React Hook Form - це бібліотека для управління формами у React-додатках за допомогою хуків. Вона надає просту інтеграцію з компонентами та бібліотеками UI, підтримку валідації та високу продуктивність завдяки мінімальній кількості повторних рендерів компонентів форми.

Redux Toolkit - інструмент для управління станом у React-додатках, який надає декларативне управління станом, вбудовані інструменти для скорочення коду та зручну інтеграцію з асинхронними операціями. RTK Query, частина Redux Toolkit, надає можливість виконання запитів на сервер з підтримкою кешування, мутацій та інтеграції з Redux Toolkit.

React Table - гнучка бібліотека для створення таблиць у React-додатках, з підтримкою сортування, фільтрації та пагінації. Ця бібліотека дозволяє легко побудувати таблиці з високою гнучкістю, кастомізацією та інтерактивними можливостями.

Ці інструменти та бібліотеки допомагають розробникам React-додатків покращити продуктивність, швидкість розробки та користувацький досвід у їх проєктах.

React надає потужний набір можливостей для створення сучасних користувацьких інтерфейсів, який також підкріплюється широкою екосистемою додаткових бібліотек та інструментів. React Router, React Hook Form, Redux Toolkit, RTK Query і React Table є лише деякими з багатьох інструментів, що допомагають розробникам створювати ефективні, гнучкі та добре структуровані веб-додатки. Розуміння та вміння використовувати ці інструменти дозволяє значно спростити процес розробки та забезпечити високу продуктивність додатків.

2.3 Стилізація в React

Стилізація є важливою частиною розробки користувацьких інтерфейсів, і в React існує кілька підходів до застосування стилів у компонентах. Кожен підхід має свої переваги та недоліки, і вибір залежить від конкретних потреб проєкту. У цьому підрозділі ми розглянемо найбільш поширені методи стилізації в React: Inline стилі, CSS, CSS Modules, Styled-components та інші CSS-in-JS рішення.

Різні підходи до стилізації в React проєктах не тільки впливають на зручність розробки, продуктивність та підтримку коду, але і на оптимізацію та швидкодію вебдодатків.

Inline стилі задаються безпосередньо в компоненті, використовуючи атрибути стилів у JSX. Цей підхід простий у використанні для невеликих чи тимчасових стилів, які легко зрозуміти та швидко налаштувати. Однак його основний недолік — відсутність підтримки псевдокласів та медіазапит, а також складність підтримки

складних стилів. Щодо швидкодії, велика кількість inline-стилів може призвести до збільшення обсягу HTML, що негативно впливає на час завантаження сторінок.

Традиційна методика CSS, яка передбачає використання окремих CSS-файлів зі звичайними класами та селекторами, забезпечує високу гнучкість і потужність у використанні CSS специфікацій. Цей підхід підтримує медіазапити, анімації та псевдокласи. Основні недоліки — можливі конфлікти імен класів і складне управління стилями у великих проєктах. Оптимізація тут можлива за допомогою методів, таких як BEM або CSS Minification, що знижує обсяг CSS-файлів і покращує швидкодію.

CSS Modules забезпечують локальну обмеженість стилів, уникаючи конфліктів завдяки автоматичному створенню унікальних класів. Стилі застосовуються локально до компонентів, підтримують медіазапити та псевдокласи, що знижує ймовірність конфліктів. Однак додаткова складність у налаштуванні збірки може бути мінусом. CSS Modules покращують оптимізацію, оскільки стилі інкапсульовані та мінімізуються для кращої швидкодії.

Styled-components дозволяють писати CSS безпосередньо у JavaScript за допомогою шаблонних літералів і базуються на концепції CSS-in-JS. Вони забезпечують локально обмежені та динамічні стилі з легкістю передачі пропсів, а також підтримку темізації. Настроювання проєкту для використання шаблонних літералів і складний підхід для новачків можуть бути викликом. Перевагою є автоматична оптимізація стилів та вилучення невикористовуваних правил, що покращує швидкодію.

Material-UI (MUI) пропонує популярний набір компонентів для React, спрощуючи створення сучасних інтерфейсів за принципами Material Design від Google. Бібліотека містить широкий вибір готових компонентів, підтримує темізацію і пропонує високу налаштованість. Однак MUI може призвести до збільшення розміру додатка, що впливає на продуктивність, і вимагає більше часу на освоєння. Щодо

оптимізації, MUI використовує tree-shaking і lazy loading, що знижує розмір бандла і покращує час завантаження.

Tailwind CSS — утилітарно-орієнтована бібліотека, яка дозволяє швидко створювати складні макети з використанням класів службових стилів. Це забезпечує високу швидкість розробки завдяки утилітарним класам, високу конфігурованість та модульність. Початківцям цей підхід може здатися незвичним і вимагати більше часу на налаштування. Оптимізація стилів проводиться через "tree shaking", що видаляє невикористані стилі й покращує швидкодію.

Stylex є CSS-in-JS бібліотекою, розробленою Facebook для великих React проєктів, яка забезпечує високу продуктивність та мінімізацію зусиль завдяки ефективному стисненню стилів і підтримці статичного аналізу. Ця бібліотека може бути складною для освоєння і має обмежену документацію. Використовуючи техніки стиснення та мінімізації, Stylex забезпечує поліпшену швидкодію завдяки оптимальному використанню ресурсів.

Кожен з цих підходів має свої особливості в плані оптимізації й швидкодії, що важливо враховувати при виборі методів стилізації для конкретного проекту.

РОЗДІЛ 3. РЕАЛІЗАЦІЯ АДАПТИВНОГО ДИЗАЙНУ НА REACT

3.1. Опис прототипу проєкту

Платформа Nuegas допомагає користувачам залишатися організованими, підвищувати продуктивність та ефективно взаємодіяти з наставниками, надаючи своєчасну підтримку. Це робить її корисним інструментом для особистого та професійного розвитку у різних сферах. Проєкт передбачає розробку лише прототипу інтерфейсу користувача (UI), що дозволить оцінити та вдосконалити візуальну та функціональну складову платформи.

Сайт "Nuegas" являє собою платформу для управління завданнями та взаємодії з менторами. Основними розділами платформи є головна сторінка користувача, сторінка завдань та сторінка менторів.

Головна сторінка користувача включає привітання користувача, його поточні завдання та прогрес їх виконання, а також календар для планування. На цій сторінці розташовано блок "Monthly Mentors", де представлені рекомендовані ментори для поточного місяця. Крім того, на головній сторінці знаходиться блок з майбутніми завданнями, який відображає їх статус та залишковий час до виконання.

Сторінка завдань містить інформацію про поточні завдання, де відображені активні проєкти та їхній прогрес. Також тут розміщений розділ нових завдань, які потрібно виконати в найближчому майбутньому. Цей розділ забезпечує користувачів необхідною інформацією для ефективного управління своїми завданнями.

Сторінка менторів має являти собою список доступних менторів з інформацією про їхні рейтинги та кількість виконаних завдань. Для зручності користувачів передбачено фільтри для пошуку менторів за категоріями та популярністю. Крім того, на основі активності користувача система пропонує рекомендованих менторів, що дозволяє оптимізувати процес взаємодії та підвищити ефективність навчання та виконання завдань.

Таким чином, сайт "Nuegas" забезпечує користувачам зручний та інтуїтивно зрозумілий інтерфейс для управління завданнями та взаємодії з менторами, сприяючи підвищенню продуктивності та ефективності роботи.

3.2. Проєктування платформи Nuegas.

Проєктування є ключовим етапом розробки програмного забезпечення, що включає визначення структури системи, взаємодії між її компонентами та специфікацію функціональних вимог. Основною метою проєктування є створення структурованої моделі, яка визначає, як різні частини системи співпрацюють для досягнення поставлених завдань.

Сайт "Nuegas" являти собою інтерактивну платформу для управління завданнями та взаємодії з менторами. Основний акцент платформи зосереджено на зручному користувацькому інтерфейсі (UI), що дозволяє користувачам легко створювати та редагувати завдання, відстежувати прогрес та отримувати підтримку від менторів. В процесі проєктування було враховано необхідність інтуїтивно зрозумілого інтерфейсу, який забезпечує ефективну взаємодію користувачів з системою.

3.2.1 Розробка моделі предметної області

Модель предметної області відображає основні сутності системи та їх взаємозв'язки. Вона допомагає зрозуміти, які дані будуть оброблятися і як ці дані

взаємодіятимуть між собою. В рамках платформи "Nuegas" виділяються наступні основні сутності.

Користувач (User):

- userID: унікальний ідентифікатор користувача.
- username: ім'я користувача.
- password: пароль користувача.
- email: електронна пошта користувача.
- Опис: Користувач є основним актором у системі, який може створювати, редагувати та видаляти завдання, а також взаємодіяти з менторами.

Ментор (Mentor):

- mentorID: унікальний ідентифікатор ментора.
- name: ім'я ментора.
- expertise: область експертизи ментора.
- Опис: Ментор допомагає користувачам у виконанні їх завдань, надаючи консультації та рекомендації.

Завдання (Task):

- taskID: унікальний ідентифікатор завдання.
- userID: ідентифікатор користувача, який створив завдання.
- title: заголовок завдання.

- description: опис завдання.
- dueDate: дата завершення завдання.
- status: статус завдання (наприклад, "виконується", "завершено").

Прогрес (Progress):

- progressID: унікальний ідентифікатор прогресу.
- taskID: ідентифікатор завдання.
- status: статус прогресу завдання.
- lastUpdated: дата останнього оновлення прогресу.
- Опис: Прогрес відображає статус виконання завдання.

3.2.2. Розробка бізнес-моделі

Бізнес-модель визначає, як користувачі взаємодіють із системою для досягнення своїх цілей. Для візуалізації цього використовуються діаграми варіантів використання (Use Case діаграми).

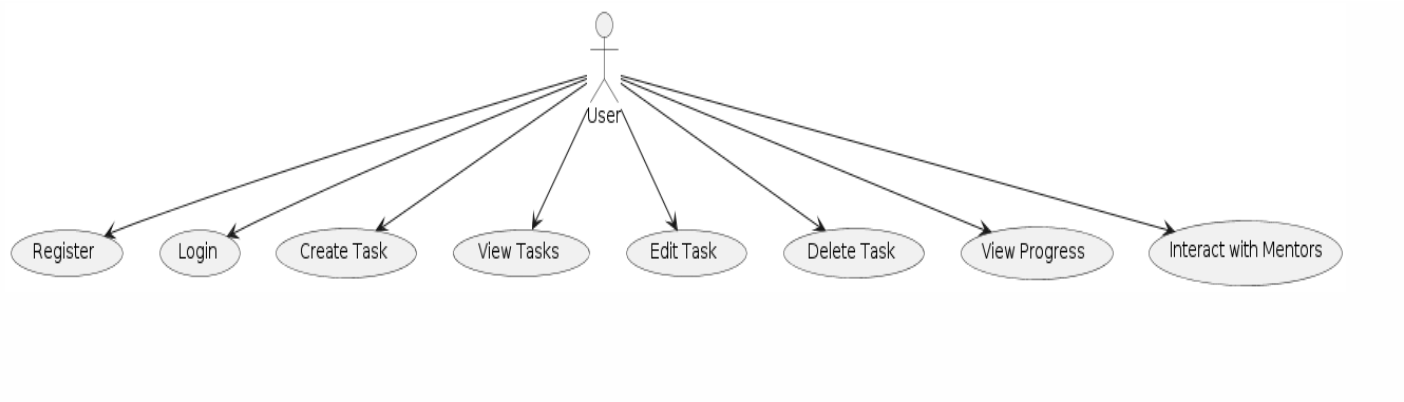

Рисунок 3.1 Діаграма use case для користувача

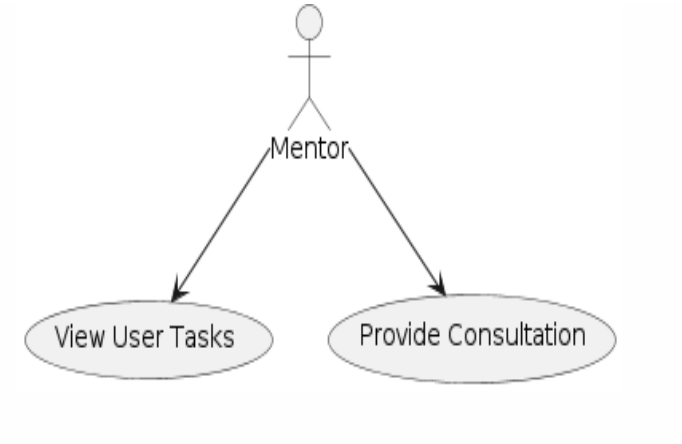

Рисунок 3.2 Діаграма use case для наставника

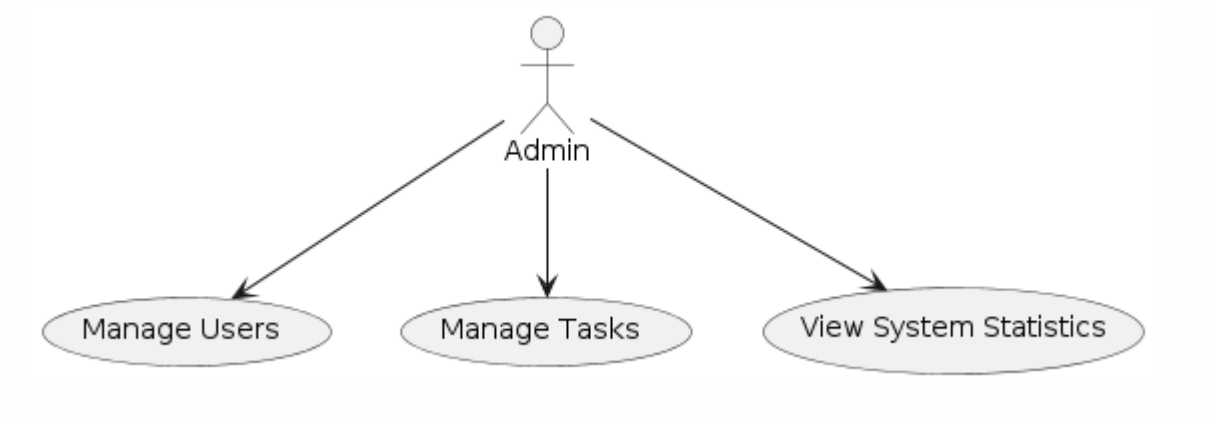

Рисунок 3.3 Діаграма use case для адміна

3.2.3 Проєктування архітектури.

Проєктування архітектури включає розробку детальних моделей, таких як діаграми класів, які визначають основні компоненти платформи, їх атрибути, методи, а також взаємозв'язки між ними.

Діаграма класів для платформи "Nuegas" відображає основні класи системи, їх атрибути та методи, а також взаємозв'язки між ними.

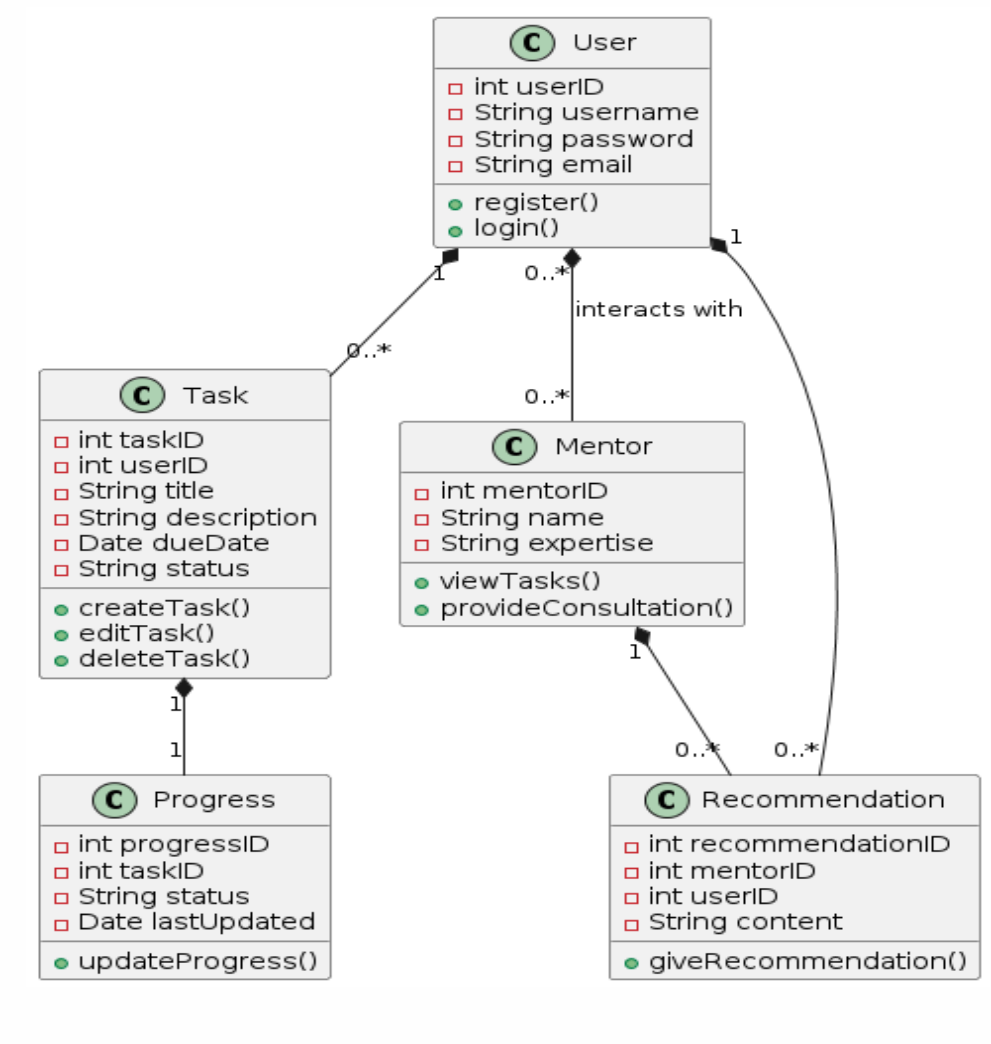

Рисунок 3.4 Діаграма класів.

Клас User має такі атрибути: userID, що є унікальним ідентифікатором користувача; username, що зберігає ім'я користувача; password, що містить пароль користувача; та email, що зберігає електронну пошту користувача. Методи цього класу включають register(), який забезпечує реєстрацію нового користувача, та login(), що відповідає за авторизацію користувача в системі.

Клас Task містить такі атрибути: taskID, що є унікальним ідентифікатором завдання; userID, що ідентифікує користувача, який створив завдання; title, що

зберігає заголовок завдання; description, що містить опис завдання; dueDate, що вказує на дату завершення завдання; та status, що відображає статус завдання, наприклад, "виконується" чи "завершено". Методи цього класу включають createTask() для створення нового завдання, editTask() для редагування існуючого завдання та deleteTask() для його видалення.

Клас Mentor має такі атрибути: mentorID, що є унікальним ідентифікатором ментора; name, що зберігає ім'я ментора; та expertise, що відображає область експертизи ментора. Методи цього класу включають viewTasks(), який дозволяє переглядати завдання користувачів, та provideConsultation(), що забезпечує надання консультацій користувачам.

Клас Progress включає такі атрибути: progressID, що є унікальним ідентифікатором прогресу; taskID, що ідентифікує завдання; status, що відображає статус прогресу завдання; та lastUpdated, що вказує на дату останнього оновлення прогресу. Метод updateProgress() забезпечує оновлення статусу прогресу завдання.

Клас Recommendation має такі атрибути: recommendationID, що є унікальним ідентифікатором рекомендації; mentorID, що ідентифікує ментора; userID, що ідентифікує користувача; та content, що містить зміст рекомендації. Метод giveRecommendation() забезпечує надання рекомендації користувачу.

3.3 Вибір редактора коду

Для розробки нашого проєкту на React ми будемо використовувати WebStorm від компанії JetBrains.

WebStorm – це комерційний редактор коду, спеціалізований на роботі з JavaScript та підтримці широкого набору інструментів для веброзробки. Він надає

значні переваги для розробників, завдяки своїм інтелектуальним можливостям. Наприклад, потужне автозаповнення, рефакторинг та навігація по коду дозволяють значно підвищити продуктивність розробки. Цей редактор також має вбудовану підтримку різноманітних систем контролю версій, включаючи Git, GitHub та SVN, що спрощує управління версіями коду.

WebStorm забезпечує спеціалізовану підтримку для популярних фреймворків, таких як React, Vue.js та Angular, що робить його особливо корисним для сучасних веброзробників. Вбудований дебагер дозволяє швидко знаходити та виправляти помилки в коді, що підвищує ефективність розробки. Крім того, редактор легко інтегрується з різноманітними інструментами розробки, включаючи тестові фреймворки, такі як Jest, що необхідно для повсякденної роботи розробника.

Особливо корисними функціями для розробки з React є можливість швидкого створення шаблонів коду. WebStorm дозволяє швидко створювати компоненти та інші частини проєкту, використовуючи шаблони, що значно спрощує та прискорює процес розробки. Додатково, вбудована підтримка інструментів статичного аналізу коду та форматування, таких як ESLint та Prettier, забезпечує високу якість коду, що є критично важливим для успішних проєктів.

3.4 Створення базового проєкту на React

Встановлення Vite з підтримкою TypeScript: Для створення нового проєкту використаємо наступні команди:

```
npm create vite@latest my-react-app --template react
cd my-react-app
npm install
```

Організація проєкту за модульним принципом дозволяє легко підтримувати та масштабувати додаток. Кожний модуль містить всі необхідні файли й логіку, що дозволяє легко розробляти та підтримувати код.

Структура проєкту. Ось приклад структури проєкту, організованої за модульним принципом з використанням TypeScript і MUI.

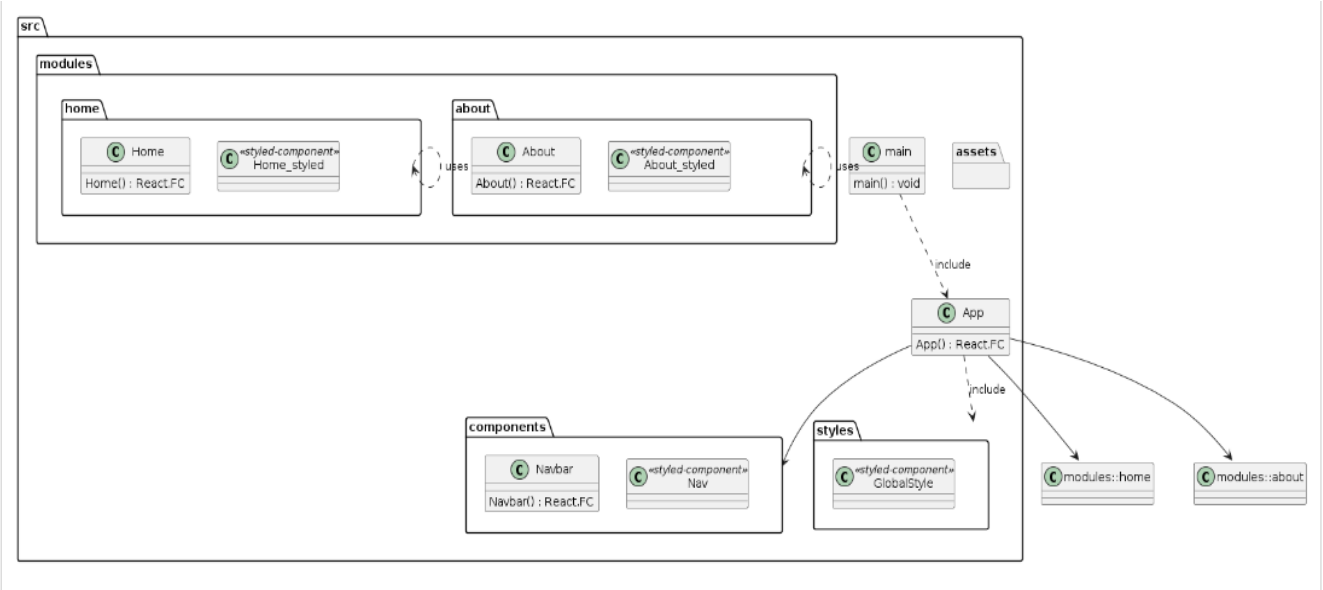

Рисунок 3.5 – базова структура проєкту в uml діаграмі.

3.5 Конструювання прототипу платформи “Nuegas”

У цьому розділі розглядається детальний процес розробки вебсайту з використанням принципів адаптивного дизайну та бібліотеки React. Мета даного розділу — представити повний огляд архітектури проєкту, вибраних технологій та інструментів, а також методології реалізації ключових компонентів і функцій. Описані підходи до стилізації за допомогою Material-UI, застосовані методи

оптимізації зображень та мультимедійного контенту, а також проведені тестування вебсайту на різних пристроях та розмірах екранів.

Цей розділ охоплює всі аспекти конструювання проєкту від початкового планування до кінцевого тестування, надаючи вичерпну інформацію про технічні рішення та кроки, здійснені під час розробки. Основний акцент ставиться на забезпечення адаптивності та зручності користувацького досвіду незалежно від типу пристрою, що використовується для доступу до сайту.

3.5.1 Налаштування базової маршрутизації

```
const routes : Router  = createBrowserRouter( routes: [
  {
    path: "/",
    element: <Layout />,
    errorElement: <ErrorPage />,
    children: [
      {
        path: "/",
        element: <Dashboard />,
      },
      {
        path: '/tasks',
        element: <Tasks />,
      },
      {
        path: '/mentors',
        element: <Mentors />,
      },
      {
        path: '/messages',
        element: <Messages />,
      },
      {
        path: '/settings',
        element: <Settings />,
      }
    ],
  },
])
```

Рисунок 3.6 - Створення базової маршрутизації.

Код (Рисунок 3.6) демонструє процес налаштування маршрутизації в додатку React за допомогою бібліотеки react-router-dom. Основна мета цього коду полягає в організації ефективного керування маршрутами в веб-додатку, забезпечуючи плавний перехід між різними сторінками та компонентами.

Основу маршрутизації складає функція createBrowserRouter, яка створює об'єкт маршрутизатора, що визначає структуру маршруту. Центральним маршрутом є кореневий шлях ("/"), для якого встановлено компонент Layout, який виконує роль контейнера для всіх дочірніх маршрутів. Крім того, вказано компонент ErrorPage для обробки помилкових ситуацій в маршрутизації.

В межах кореневого маршруту визначені кілька дочірніх маршрутів, кожен з яких відповідає певному шляху та компоненту. Це дозволяє відображати відповідний компонент при переході за певним URL. Наприклад, маршрут з шляхом "/tasks" відображає компонент Tasks, а шлях "/mentors" відповідає компоненту Mentors.

Функція Routes забезпечує інтеграцію маршрутизатора з додатком, обертаючи налаштування маршрутизації в компонент RouterProvider і передаючи створений маршрутизатор як параметр. Це дозволяє додатку використовувати визначені маршрути для навігації між різними компонентами.

Таким чином, даний код забезпечує гнучку і зручну систему маршрутизації в додатку React, використовуючи потужні можливості бібліотеки react-router-dom. Це забезпечує ефективне керування переходами між сторінками й покращує загальний користувацький досвід, створюючи інтерактивний і динамічний вебдодаток.

3.5.2 Створення основних компонентів

Базовий компонент навігації.

```
function Navigation() : JSX.Element  {  Show usages
  const location : Location<any>  = useLocation();

  return (
    <List>
      <Box component='nav'>
        {links.map((link : Link ) => {
          const isSelected : boolean  = location.pathname === link.link;

          return (
              <ListItem key={link.link} component={Link} to={link.link} aria-selected={isSelected}>
                <ListItemIcon>
                  <link.icon />
                </ListItemIcon>
                <ListItemText>
                  <Typography color={'text.primary'}>
                    {link.title}
                  </Typography>
                </ListItemText>
              </ListItem>
          )
        })}
      </Box>
    </List>
  )
}
```

Рисунок 3.6 - Лістинг компонента навігації

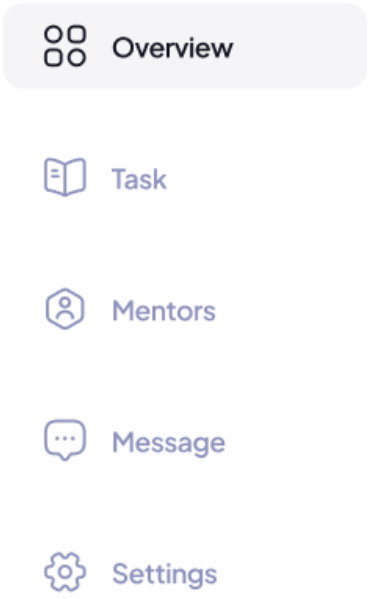

Рисунок 3.7 - Зовнішній вигляд компонента навігації

Базовий компонент навігації, код (Рисунок 3.6) компонент Navigation, який розроблений для побудови навігаційного списку у вебдодатку з використанням бібліотек MUI (Material-UI) та React Router.

У вказівному коді використовуються компоненти з бібліотек MUI, такі як List, Box, ListItem, ListItemIcon, ListItemText та Typography, а також компоненти Link та useLocation з фреймворку React Router.

Компонент Navigation отримує поточний шлях (location) за допомогою хука useLocation. Після чого він малює список List з навігаційними елементами, які отримані з масиву links. Кінцевий компонент Navigation експортується з метою подальшого використання в інших частинах вебдодатку.

Компонент шапки сайту.

```
import useMediaQuery from '@mui/material/useMediaQuery';

import MainHeader from './MainHeader';
import MobileHeader from './MobileHeader';

function Header() : JSX.Element  {  Show usages
  const isMobile : boolean  = useMediaQuery( queryInput: (theme) => theme.breakpoints.down( key: 'sm'));

  return (
    <header>
      {isMobile ? <MobileHeader /> : <MainHeader />}
    </header>
  );
}

export default Header;  Show usages
```

Рисунок 3.8 - Лістинг компонента “Header”.

У даному підході (Рисунок 3.8) до реалізації адаптивного дизайну, завдяки використанню функції useMediaQuery з параметром медіазапиту для малих і середніх екранів ('sm'), забезпечується те, що для користувачів з мобільними пристроями буде завантажуватися лише мобільна версія (MobileHeader). Це приклад адаптивного підходу до реалізації.

Це покращує оптимізацію, оскільки десктопна версія (MainHeader) не буде завантажуватися і відображатися на мобільних пристроях, що дозволяє зменшити обсяг завантажуваних даних і покращує швидкість завантаження сторінки для користувачів з обмеженим доступом до мережі або на повільних мережах.

Такий підхід допомагає створити оптимальний досвід користувача, де інтерфейс адаптується до конкретного пристрою, забезпечуючи зручність і швидкість роботи навіть на мобільних пристроях.

Базовий шаблон для використання на більшості сторінок.

```jsx
function Layout() : JSX.Element  {  Show usages
  return (
    <Container>
      <Header />
      <SideBar />
      <main>
        <Outlet />
      </main>
    </Container>
  )
}

export default Layout  Show usages
```

Рисунок 3.9 - Лістинг “Layout”

```jsx
export const Container : StyledComponent<MUIStyledCommo...  = styled( tag: 'div')(({ theme : Theme  }) : {...}  ⇒ ({  St
  display: 'grid',
  gridTemplateAreas:
  `
    'header header header',
    'sidebar main main'
  `,
  gridTemplateColumns: '200px 1fr',
  gridTemplateRows: 'auto 1fr',
  minHeight: '100vh',

  [theme.breakpoints.down( key: 'sm')]: {
    gridTemplateAreas:`
      'header',
      'main',
      `,
    gridTemplateColumns: '1fr',
    gridTemplateRows: 'auto 1fr',
  },
}));
```

Рисунок 3.10 - Лістинг стилів для компонента “Layout”.

Код (Рисунок 3.9) це компонент Layout включає контейнер, компоненти Header, SideBar, <main> та <Outlet>. Ця структура дозволяє організувати загальний макет сторінки, забезпечуючи розміщення верхньої частини, бічного меню та основного

контенту відповідно. Layout може бути використано для побудови загального макета у додатках на React.

У наданому фрагменті коду (Рисунок 3.10) використовується styled-components для стилізації компонента Container. Визначаються стилі, які встановлюють контейнер як сітку за допомогою властивостей gridTemplateAreas, gridTemplateColumns та gridTemplateRows.

Також в коді присутній вкладений медіазапит, який застосовує зміни стилів для екранів з розміром до 'sm'. У цьому медіазапиті змінюються gridTemplateAreas, gridTemplateColumns та gridTemplateRows для підлаштування вигляду контейнера під менший екран, забезпечуючи вертикальне розташування елементів блоку.

Цей підхід надає можливість динамічно змінювати розміщення елементів на сторінці залежно від розміру екрана, що сприяє створенню адаптивного дизайну та покращує користувацький досвід на різних пристроях.

Базовий компонент для відображення зображень.

```
type Size = 'small' | 'medium' | 'large';

type ImageSources = {
  [size in Size]: string;
};

interface ImageProps {  Show usages
  sources: ImageSources;
  alt: string;
}

export const Image: FC<ImageProps> = ({ sources : ImageSources , alt : string  }) => {  no usages
  const getImageSource = (size: Size): string => sources[size];  Show usages

  return (
    <picture>
      {Object.keys(sources).map((size : string  ) => (
        <source key={size} media={`(max-width: ${size})`} srcSet={getImageSource(size as Size)} />
      ))}
      <img src={getImageSource( size: 'large')} alt={alt} />
    </picture>
  );
}
```

Рисунок 3.11 - Компонент “Image”

Під час відображення <source> елементів у компоненті <picture>, ми використовуємо динамічний підхід для створення декількох <source> елементів на основі ключів об'єкта sources. Кожен <source> вказує на зображення відповідного розміру.

Image дозволить передавати об'єкт з розмірами та посиланнями на зображення для відображення залежно від розміру екрана. Це дозволить оптимізувати зображення для різних розмірів екрана.

Компонент для заголовків в якому показано переваги використання css функції clamp().

```
export const TitleStyled : StyledComponent<MUIStyledCommo... = styled( tag: 'h1') (() : {...} ⇒ (
  fontSize: 'clamp(1.5rem, 2.5vw, 2rem)',
  color: '#333',
  fontWeight: 500,
  margin: 0,
  padding: 0,
  textAlign: 'center',
  textTransform: 'capitalize',
}));
```

Рисунок 3.12 - Компонент для заголовків.

У даному конкретному випадку(Рисунок 3.12), TitleStyled застосовує флюїдний підхід до визначення розміру шрифту. Використання флюїдного (рідкого) підходу дозволяє заголовку автоматично змінювати свій розмір залежно від ширини вікна браузера користувача. Це досягається через функцію CSS clamp(), яка встановлює мінімальне, бажане та максимальне значення font-size.

В результаті, розмір шрифту буде знаходитись у діапазоні від 1.5rem до 2rem, з 2.5vw в якості базового значення, яке залежить від ширини вікна. Цей підхід

забезпечує адаптивність вебсайту та покращує його читабельність на будь-яких пристроях, будь-то мобільні телефони, планшети чи десктопи.

3.6 Тестування UI прототипу платформи “Nuegas”

Метою тестування прототипу платформи “Nuegas” є забезпечення його оптимального відображення та функціонування незалежно від використовуваного пристрою. Це завдання включає перевірку адаптивності дизайну, що гарантує коректну зміну макета та вигляду вебсайт відповідно до розмірів і орієнтації екрана. Важливо, щоб вебсайт залишався зручним для користувачів як на великих настільних моніторах, так і на маленьких екранах смартфонів.

Крім того, тестування спрямоване на оцінку функціональності. Це включає перевірку всіх інтерактивних елементів, таких як кнопки, форми, посилання та меню, на коректність їх роботи на різних пристроях. Не менш важливим є забезпечення візуальної цілісності, яка передбачає чітке та без викривлень відображення всіх компонентів, включаючи тексти, зображення. Вебсайт повинен залишатися читабельним і привабливим незалежно від типу пристрою.

Окрему увагу слід приділити перевірці на сумісність, що має на меті забезпечити коректне відображення та функціонування вебсайт у різних веббраузерах (Chrome, Firefox, Safari, Edge) та операційних системах (Windows, macOS, iOS, Android). Це дозволяє уникнути потенційних проблем з сумісністю, які можуть виникнути через специфічні обмеження або особливості браузерів та платформ.

Загалом, основною метою є забезпечення зручності та приємного користувацького досвіду на всіх типах пристроїв, на яких може відображатися вебсайт. Успішне тестування дає змогу виявити можливі проблеми на ранніх етапах

розробки, що дозволяє оперативно вирішувати їх та випустити високоякісний кінцевий продукт.

План тестування:

1. Перевірити адаптивність дизайну, а саме коректне змінення макета та вигляду вебсайту залежно від розміру та орієнтації екрана для зручності на будь-якому пристрої.

2. Функціональність. Перевірити коректну роботу всіх інтерактивних елементів (кнопки, форми, посилання, меню) на різних пристроях.

3. Візуальна цілісність. Забезпечити чітке та без викривлень відображення всіх компонентів вебсайту, збереження читабельності та привабливості контенту.

4. Перевірка на сумісність. Переконатися у правильному відображенні та функціонуванні вебсайту в різних браузерах і операційних системах, уникнути проблем з сумісністю.

Під час тестування головної сторінки вебсайту (Рисунок 3.13) проводилася ретельна перевірка її адаптивності, функціональності, візуальної цілісності та сумісності на різних пристроях і розмірах екранів. Тестування включало відкриття вебсайту на настільних комп'ютерах, ноутбуках, планшетах та смартфонах, а також використання інструментів розробника у браузерах, таких як Google Chrome DevTools, для симуляції різних розмірів екранів.

Було перевірено, чи правильно змінюється макет і вигляд вебсайту при зміні розміру екрана, забезпечуючи зручну навігацію незалежно від типу пристрою. Також оцінювалася робота інтерактивних елементів (кнопок, форм, посилань і меню) для запобігання випадкам, коли вони стають недоступними або не реагують на дії користувачів.

Під час тестування зверталася увага на те, чи відображаються всі компоненти (тексти, зображення, відео, графіка) чітко та без викривлень. Проводилася перевірка вебсайту у різних веббраузерах (Chrome, Firefox, Safari, Edge) та операційних системах (Windows, macOS, iOS, Android) для виявлення та усунення потенційних проблем із сумісністю.

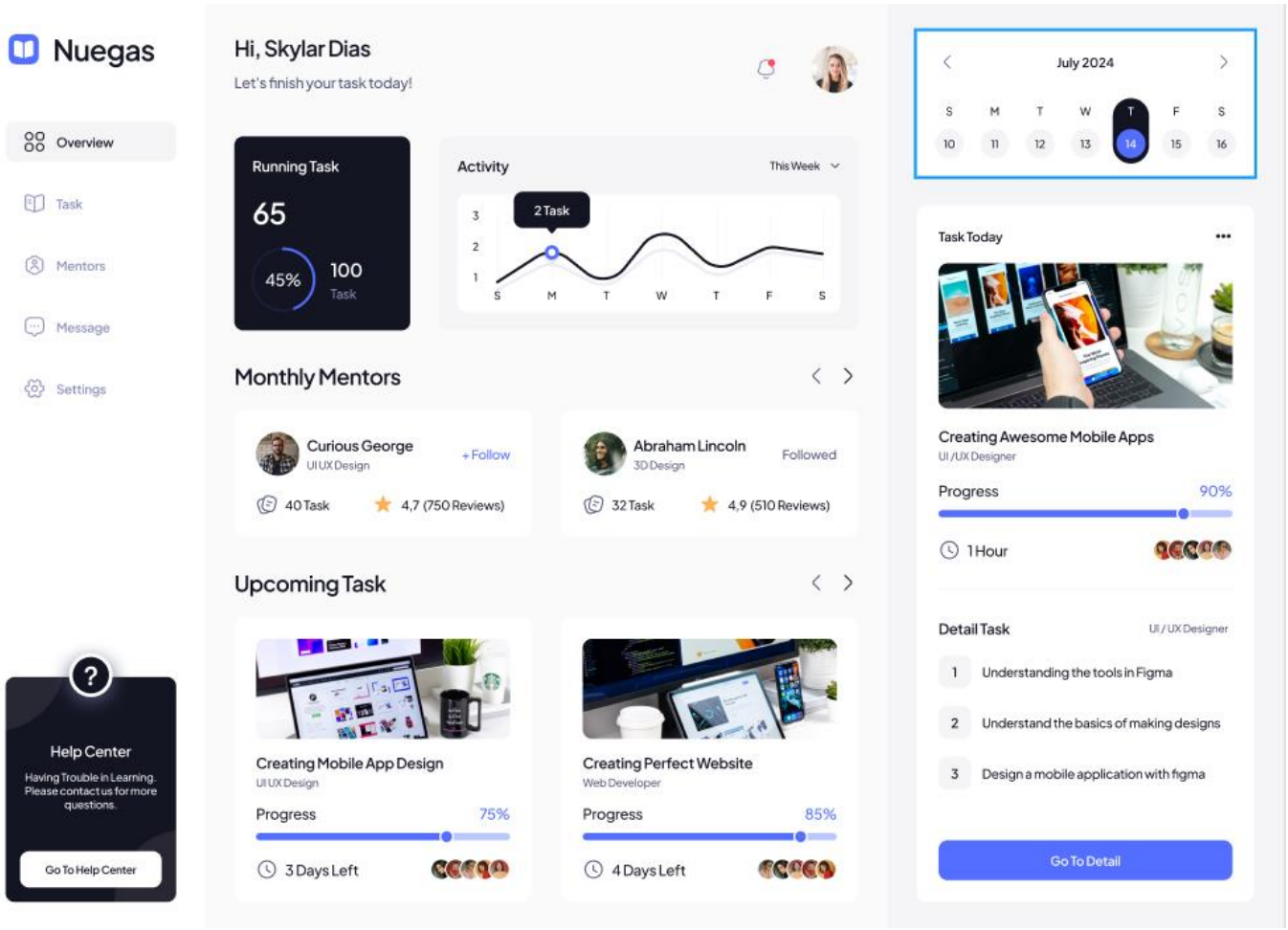

Рисунок 3.13 - Сторінка Overview. Deskop

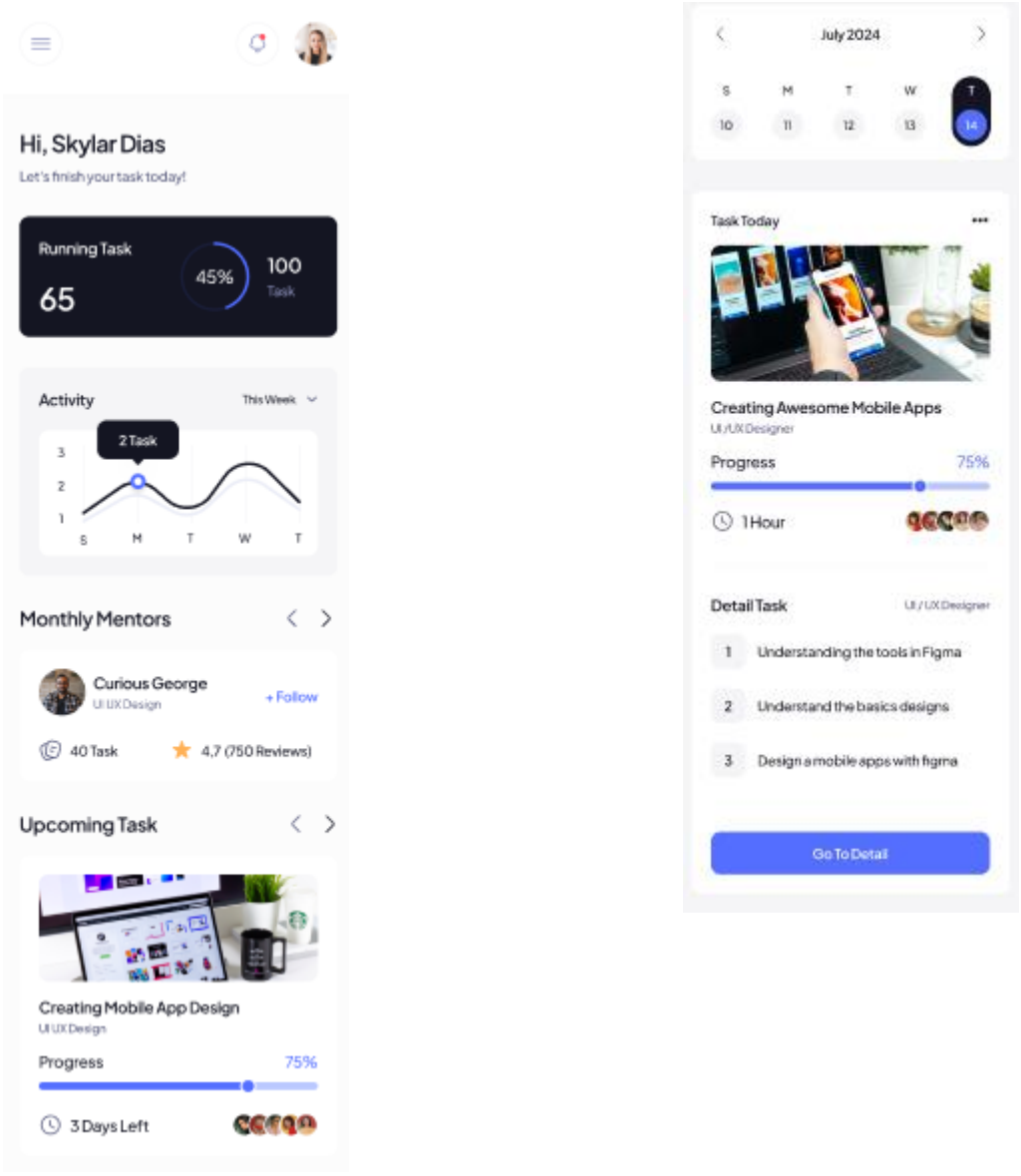

Рисунок 3.14 - Сторінка Overview. Mobile.

Тестування сторінки “Task” проводилося аналогічно до головної сторінки, було проведено перевірку її адаптивності, функціональності, візуальної цілісності та сумісності на різних пристроях і розмірах екранів. Аналогічно до тестування головної

сторінки, перевірка включала відкриття сторінки "Task" на настільних комп'ютерах, ноутбуках, планшетах та смартфонах, а також використання інструментів розробника для симуляції різних розмірів екранів.

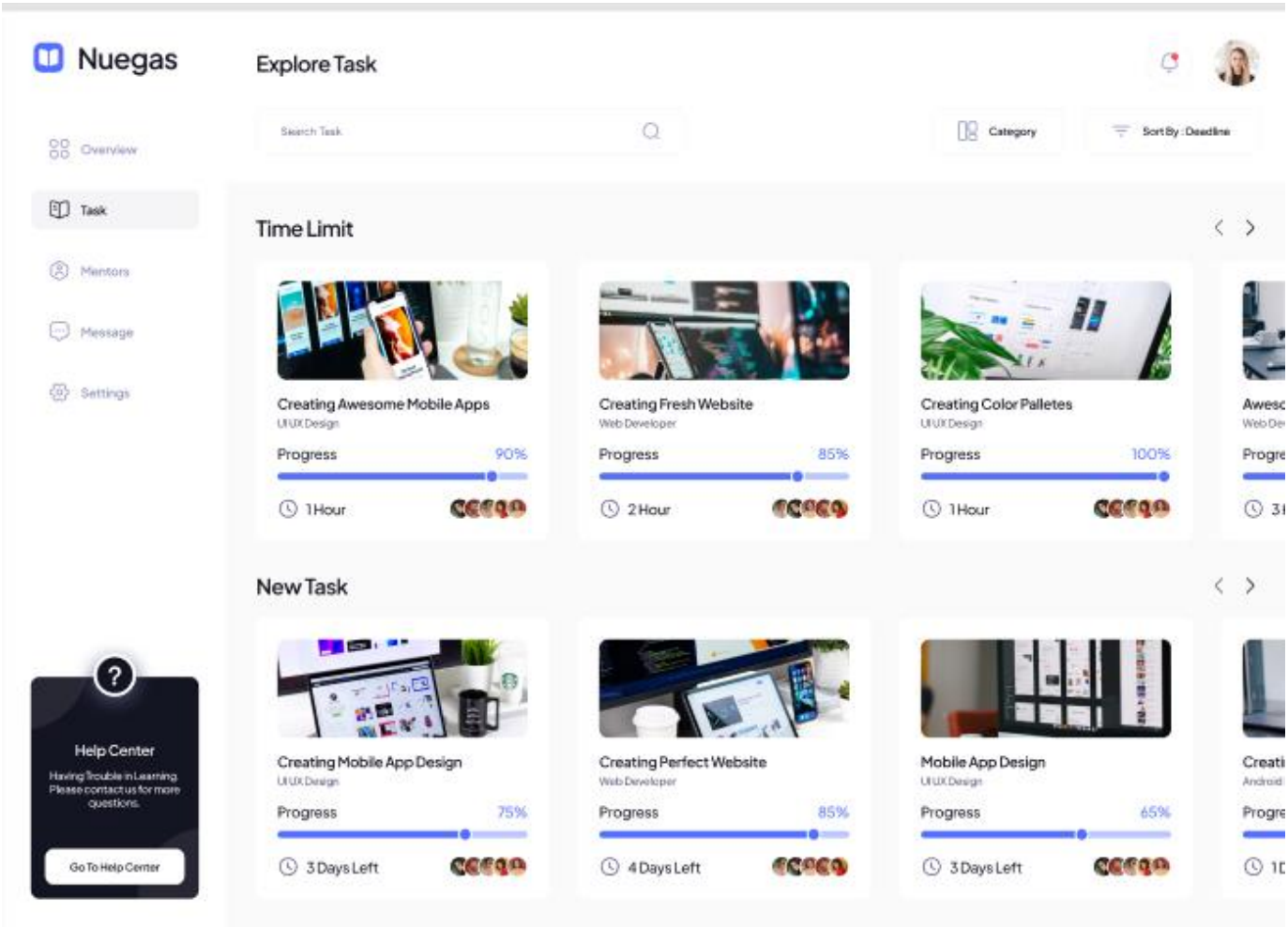

Рисунок 3.15 - Сторінка “Task”. Desktop.

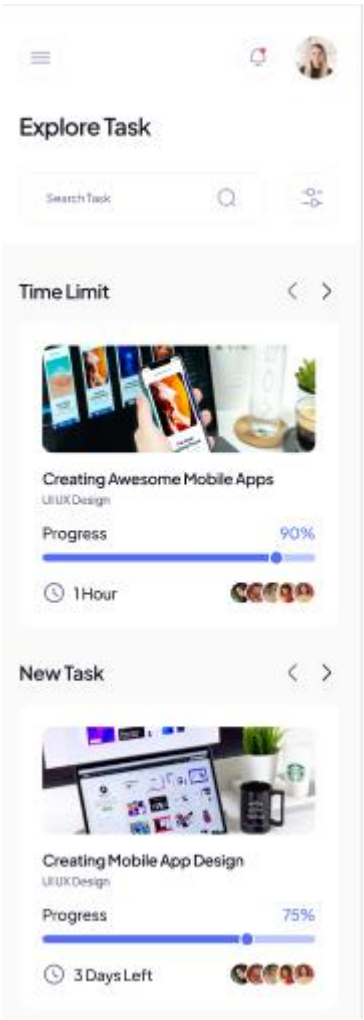

Рисунок 3.16 - Сторінка “Task”. Mobile.

У процесі тестування сторінок вебсайту на різних пристроях та розмірах екранів було виявлено, що всі елементи відображаються коректно. Дизайн веб-сайту адаптується до будь-яких розмірів екранів, забезпечуючи зручну навігацію та приємний користувацький досвід. Текст, зображення та інтерактивні елементи, такі як кнопки та посилання, зберігають свою цілісність і функціональність на настільних комп'ютерах, ноутбуках, планшетах та смартфонах. Адаптивність макета забезпечує, що контент залишається читабельним і привабливим незалежно від типу пристрою.

Однак у процесі тестування у веббраузері Firefox було виявлено проблеми з відображенням іконок. Деякі іконки виглядали некоректно, що знижувало якість користувацького досвіду. Це питання потребувало додаткового аналізу та виправлення, щоб забезпечити коректне відображення всіх елементів вебсайту у цьому браузері.

Після виявлення проблеми були виконані додаткові роботи для її усунення, включаючи перевірку шрифтів, використаних для іконок, та коректність шляхів до файлів. Це дозволило виправити відображення іконок у Firefox і забезпечити їх коректну роботу аналогічно до інших браузерів.

Таким чином, результати тестування показують вебсайт відповідає вимогам щодо адаптивного дизайну та сумісності з різними пристроями.

РОЗДІЛ 4. ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ

4.1 Долікарська допомога при обмороженнях.

Відхилення температури навколишнього середовища від допустимих значень становить небезпеку для людини. Ця небезпека тим більша, чим більше відхилення температури середовища чи речей, з якими контактує людина, від допустимих значень. Як низькі, так і високі температури навколишнього середовища призводять до порушення процесів терморегуляції організму і розладу функцій життєво важливих систем. Контакт окремих ділянок тіла з гарячими або холодними речами та предметами викликає травмування цих ділянок, яке зветься опіком чи обмороженням. Контакт з деякими хімічними речовинами та сполуками також призводить до травмування, яке зветься хімічним опіком.

Переохолодження. Розвивається внаслідок порушення процесів терморегуляції при дії на організм низьких температур. Погіршенню самопочуття сприяють втома, малорухомість, алкогольне сп'яніння. Ознаки. На початковому етапі потерпілого морозить, прискорюються дихання і пульс, підвищується артеріальний тиск, потім настає переохолодження, рідшає пульс та дихання, знижується температура тіла. При зниженні температури тіла від 34 до 32 °С затьмарюється свідомість, припиняється довільне дихання, мова стає неусвідомленою. Після припинення дихання серце може ще деякий час (від 5 до 45 хвилин) скорочуватися, а потім зупиняється, і настає смерть [2].

Порядок надання домедичної допомоги постраждалим при загальному переохолодженні та/або відмороженні

1. Цей Порядок визначає механізм надання домедичної допомоги постраждалим при загальному переохолодженні які не мають медичної освіти, але за своїми службовими обов’язками повинні надавати домедичну допомогу.

2. У цьому Порядку терміни вживаються у такому значенні.

Переохолодження — невідкладний стан, який виникає при дії низьких температур та/або несприятливих факторів зовнішнього середовища, що викликає зниження температури тіла постраждалого.

Інші терміни вживаються у значеннях, наведених в Законі України «Основи законодавства України про охорону здоров’я» та інших нормативно-правових актах у сфері охорони здоров’я.

3. Фактори ризику виникнення переохолодження та/або відмороження:

1) вплив низьких температур у поєднанні з вітром, підвищеній вологості;

2) наявність у постраждалого тісного або мокрого взуття/одягу;

3) нерухоме положення постраждалого;

4) наявність у постраждалого супутньої патології (крововтрата тощо);

5) стан алкогольного, наркотичного чи іншого сп’яніння;

6) зневоднення та недостатнє харчування постраждалого.

4. Ознаки загального переохолодження:

1) зниження температури тіла постраждалого нижче 35 °С;

2) тремтіння;

3) повільне дихання;

4) бліда та холодна шкіра;

5) сплутана свідомість.

5. Розрізняють чотири ступені відмороження:

1) I ступінь — шкіра постраждалого блідого кольору, незначно набрякла, чутливість знижена або повністю відсутня;

2) II ступінь — у ділянці відмороження утворюються пухирі, наповнені прозорою або білою рідиною;

3) III ступінь – омертвіння шкіри: зявляються пухирі, наповнені рідиною темно-бурого кольору омертвілої ділянки розвивається запальний вал (демаркаційна лінія).

4) IV ступінь - поява пухирів, наповнених чорною рідиною. У постраждалого присутні ознаки шоку.

6. Послідовність дій при наданні домедичної допомоги постраждалим при загальному переохолодженні та/або відмороженні:

1) перед наданням допомоги переконатися у відсутності небезпеки для себе, оточуючих, постраждалого та тільки за її відсутності перейти до наступного кроку;

2) припинити дію низької температури на постраждалого;

3) заспокоїти постраждалого та пояснити свої подальші дії;

4) здійснити виклик екстреної медичної допомоги та дотримуватись вказівок диспетчера прийому виклику;

5) якщо у постраждалого ознаки загального переохолодження

а) усунути дію несприятливих факторів зовнішнього середовища, перемістити постраждалого у тепле приміщення;

б) зняти з постраждалого холодний, вологий одяг;

в) якщо постраждалий у свідомості, дати безалкогольні теплі напої;

6) якщо у постраждалого ознаки відмороження:

а) усунути дію несприятливих факторів зовнішнього середовища, перемістити постраждалого у тепле приміщення;

б) обережно, без зусиль зняти з постраждалого холодний, вологий одяг/взуття;

в) накласти на уражені ділянки тіла чисті, стерильні, сухі марлеві пов’язки, без здійснення додаткового тиску на тканини;

г) за необхідності знерухомити уражені кінцівки;

г) якщо постраждалий у свідомості, дати безалкогольні теплі напої;

д) не масажувати і не розтирати уражені ділянки, не застосовувати місцево джерела тепла;

е) не пошкоджувати наявні на місці обмороження міхурі;

7) накрити постраждалого термопокривалом/покривалом;

8) забезпечити постійний нагляд за постраждалим до приїзду бригади екстреної (швидкої) медичної допомоги;

9) при погіршенні стану постраждалого до приїзду бригади екстреної (швидкої) медичної допомоги повторно зателефонувати диспетчеру екстреної медичної допомоги;

10) за можливості зібрати у постраждалого чи оточуючих максимально можливу інформацію стосовно обставин виникнення теплового удару. Всю отриману інформацію передати працівникам бригади екстреної (швидкої) медичної допомоги або диспетчеру прийому виклику.

5. Якщо до приїзду бригади екстреної (швидкої) медичної допомоги постраждалий втратив свідомість, слід перейти до Порядку надання домедичної допомоги дорослим при раптовій зупинці кровообігу або Порядку надання домедичної допомоги дітям при раптовій зупинці кровообігу, затверджених

наказом Міністерства охорони здоров’я України від 09 березня 2022 року № 441 [1].

4.3 Санітарно-гігієнічні вимоги до умов праці.

Виробнича санітарія та гігієна праці – це система організаційних, гігієнічних і санітарно-технічних заходів та засобів запобігання впливу на працівників шкідливих виробничих факторів. Державні санітарні норми, правила, гігієнічні нормативи (санітарні норми) — обов’язкові для виконання нормативні документи, що визначають критерії безпеки та/або нешкідливості для людини факторів навколишнього середовища і вимоги щодо забезпечення оптимальних чи допустимих умов життєдіяльності людини.

У приміщеннях на робочих місцях мають забезпечуватись оптимальні значення параметрів мікроклімату: температура повітря повинна становити 22–25°С, відносна вологість повітря — 40–60%, швидкість руху повітря — не більше 0,1 м/с.

Площу приміщень, в яких розташовують персональні комп’ютери, визначають згідно з чинними нормативними документами. Відповідно до ДСанПіН 3.3.2.007-98 з розрахунку на одне робоче місце, обладнане ПК, встановлено такі норми:

- площа – не менше 6,0 кв. м;
- об’єм – не менше 20,0 куб. м.

Заземлені конструкції, що знаходяться у приміщеннях (батареї опалення, водопровідні труби, кабелі із заземленим відкритим екраном тощо), мають бути надійно захищені діелектричними щитками або сітками від випадкового дотику.

Конструкція робочого місця користувача персонального комп’ютера має

забезпечити підтримання оптимальної робочої пози офісного працівника. Конструкція робочого столу має відповідати сучасним вимогам ергономіки і забезпечувати оптимальне розміщення на робочій поверхні використовуваного обладнання (дисплея, клавіатури, принтера) і документів.

Також в цих приміщеннях повинні бути медичні аптечки першої допомоги, системи автоматичної пожежної сигналізації та переносні вуглекислотні вогнегасники. Підходи до засобів пожежогасіння повинні бути вільними.

Робочі місця, згідно з п. 4.3 ДСанПіН 3.3.2.007-98, слід розташовувати відносно світлових прорізів так, щоб природне світло падало переважно з лівого боку. Робоче місце необхідно розміщувати таким чином, щоб уникнути попадання прямого світла в очі. Для забезпечення захисту і досягнення нормованих рівнів комп'ютерних випромінювань необхідне застосування приекранних фільтрів, локальних світлофільтрів (засобів індивідуального захисту очей) та інших засобів захисту, що пройшли випробування в акредитованих лабораторіях і мають щорічний гігієнічний сертифікат (п. 4.19 ДСанПіН 3.3.2.007-98).

Штучне освітлення приміщення має здійснюватись системою загального рівномірного освітлення (п. 3.2.2 ДСанПіН 3.3.2.007-98). У приміщеннях при переважній роботі з документами допускається використання системи комбінованого освітлення, тобто встановлення світильників місцевого освітлення додатково до загального. Як джерела штучного освітлення необхідно використовувати люмінесцентні лампи. Згідно з п. 3.2.5 ДСанПіН 3.3.2.007-98 система загального освітлення має бути у вигляді суцільних або переривчатих ліній світильників, що розташовані збоку від робочих місць (зазвичай ліворуч) паралельно лінії зору працівників.

Допускається застосування ламп розжарювання у світильниках місцевого освітлення та, у разі влаштування відбитого освітлення у виробничих чи адміністративно-громадських приміщеннях, металогалогенних ламп потужністю 250

Вт. Світильники місцевого освітлення слід встановлювати таким чином, щоб не створювати відблисків на поверхні екрана, а освітленість екрана має не перевищувати 300 лк.

Для забезпечення нормованих значень освітленості у приміщеннях відповідно до п. 3.2.15 ДСанПіН 3.3.2.007-98 необхідно мити вікна і світильники не рідше 2 разів на рік, а також своєчасно замінювати лампи, що перегоріли. Правилами встановлюються висота робочої поверхні робочого столу, параметри ширини і глибини для робочих столів, які мають забезпечувати можливість виконання операцій у зоні досяжності моторного поля. Відповідно до п. 4.8 ДСанПіН 3.3.2.007-98 робочий стілець має бути підйомно-поворотним, регульованим за висотою, з кутом нахилу сидіння та спинки, від спинки до переднього краю сидіння поверхня сидіння має бути плоскою, передній край – заокругленим. Регулювання за кожним із параметрів має здійснюватися незалежно, легко і надійно фіксуватися. Поверхня сидіння і спинки стільця має бути напівм’якою з нековзним, повітронепроникним покриттям, що легко чиститься і не електризується (п. 4.12 ДСанПіН 3.3.2.007-98).

Робочий стіл для ПК, як правило, має бути обладнаним підставкою для ніг, вимоги до її розмірів та конструкції також прописані в правилах. Застосування підставки для ніг тими, у кого ноги не дістають до підлоги, є обов’язковим. Приміщення можуть обладнуватись шафами для зберігання документів, магнітних дисків, полицями, стелажами, тумбами тощо з урахуванням вимог до площі приміщень. Поверхня підлоги має бути рівною, неслизькою, з антистатичними властивостями. Забороняється для оздоблення інтер’єру приміщень з персональними комп’ютерами застосовувати полімерні матеріали (деревинно-стружкові плити, шпалери, що миються, рулонні синтетичні матеріали, шаруватий паперовий пластик тощо), що виділяють у повітря шкідливі хімічні речовини.

Дотримання вимог електробезпеки під час роботи. Відповідно до розд. VI Правил № 65 щодня перед початком роботи необхідно очищати монітор від пилу та

інших забруднень. Після закінчення роботи персональний комп’ютер і периферійні пристрої повинні бути відключені від електричної мережі. У разі виникнення аварійної ситуації необхідно негайно відключити персональний комп’ютер і периферійні пристрої від електричної мережі. Персональні комп’ютери, периферійні пристрої повинні підключатися до електромережі тільки з допомогою справних штепсельних з’єднань і електророзеток заводського виготовлення. Штепсельні з’єднання та електророзетки, окрім контактів фазового та нульового робочого провідників, повинні мати спеціальні контакти для підключення нульового захисного провідника. Конструкція їх має бути такою, щоб приєднання нульового захисного провідника відбувалося раніше, ніж приєднання фазового та нульового робочого провідників. Порядок роз’єднання при відключенні має бути зворотним. Необхідно унеможливити з’єднання контактів фазових провідників з контактами нульового захисного провідника. Неприпустимим є підключення комп’ютерів, периферійних пристроїв до звичайної двопровідної електромережі, в тому числі – з використанням перехідних пристроїв.Є неприпустимими:

- експлуатація кабелів та проводів з пошкодженою або такою, що втратила захисні властивості за час експлуатації, ізоляцією;
- застосування саморобних подовжувачів, застосування для опалення приміщення нестандартного (саморобного) електронагрівального обладнання або ламп розжарювання;
- користування пошкодженими розетками, розгалужувальними та з’єднувальними коробками, вимикачами та іншими електровиробами, а також лампами, скло яких має сліди затемнення або випинання;
- використання електроапаратури та приладів в умовах, що не відповідають вказівкам (рекомендаціям) підприємств-виготовлювачів.

Вимоги до вентиляції, опалення, кондиціонування, мікроклімату – приміщення для роботи з персональними комп’ютерами мають бути обладнані системами

опалення, кондиціонування повітря, або припливно-витяжною вентиляцією. У приміщеннях на робочих місцях мають забезпечуватись оптимальні значення параметрів мікроклімату: температури, відносної вологості та рухливості повітря відповідно до норм та правил, а також ДБН В.2.5-67:2013 «Опалення, вентиляція та кондиціонування», затверджених наказом Мінрегіону від 25.01.2013 р. № 24. Під час перевищення припустимих значень робочий день співробітників повинен бути скорочений мінімум на 10 %. Для підтримки допустимих значень мікроклімату та концентрації позитивних і негативних іонів необхідно передбачати установки або прилади зволоження та/або штучної іонізації, кондиціювання повітря. В Україні відсутні затверджені на законодавчому рівні гранично допустимі норми вмісту вуглекислого газу в повітрі для житлових, офісних та громадських споруд. Проте, враховуючи його вплив на працівників, а саме суттєве зниження їх працездатності, роботодавцям варто приділяти цьому питанню увагу та вживати заходи профілактики.

Окрім цього, наслідком сучасного технічного прогресу є зростання з кожним роком енергоспоживання та збільшення навантаження на кабелі, що в свою чергу призводить до збільшення напруги електромагнітних полів, несприятлива дія яких може призвести до погіршення стану здоров’я працівників. Таким чином, роботодавцям варто пам’ятати, що причиною зниження працездатності офісних працівників дуже часто є саме незадовільні параметри мікроклімату.

Рівні шуму та вібрації на робочих місцях осіб, що працюють з ПК, визначаються відповідно до ДСанПіН 3.3.2.007-98. Для забезпечення дотримання допустимих рівнів шуму на робочих місцях застосовуються засоби звукопоглинання, вибір яких обґрунтовується спеціальними інженерно-акустичними розрахунками (п. 3.3.3 ДСанПіН 3.3.2.007-98). Перелік організаційно-технічних заходів щодо обмеження несприятливого впливу шуму та вібрації на працюючих наведено в ДСН 2.3.6.037-99 та ДСН 3.3.6.039-99, серед яких зменшення шуму та вібрації на шляху розповсюдження засобами ізоляції та поглинання, наприклад, за рахунок

використання гумових, поролонових, інших шумо- чи вібропоглинаючих матеріалів, або інших матеріалів аналогічного призначення, що дозволені для оздоблення приміщень органами державного санітарно-епідеміологічного нагляду.

Вимоги щодо рівня неіонізуючих електромагнітних випромінювань, електростатичних та магнітних полів встановлюються відповідно до ДСанПіН 3.3.2.007-98, а також Вимог до роботодавців щодо захисту працівників від шкідливого впливу електромагнітних полів, затверджених наказом Міненергетики від 05.02.2014 р. № 99, ДСанПіН 3.3.6.096-2002. Значення напруженості електростатичного поля на робочих місцях (як у зоні екрана дисплея, так і на поверхнях обладнання, клавіатури, друкувального пристрою) мають не перевищувати гранично допустимих відповідно до встановлених норм [3].

ВИСНОВКИ

Зі стрімким розвитком інформаційних технологій і зростанням популярності мобільних пристроїв адаптивний дизайн набув значної важливості у сучасній веброзробці. Дослідження свідчать, що більша частина сьогоднішнього інтернет-трафіку генерується мобільними пристроями, що зумовлює необхідність створення вебсайтів, які залишаються зручними та функціональними на різних платформах, від смартфонів і планшетів до настільних комп'ютерів і великих екранів.

У ході даної роботи була проведена дослідницька діяльність, спрямована на вивчення історії та розвитку концепцій адаптивного дизайну. Було визначено ключові принципи адаптивного дизайну, такі як гнучкі макети, гнучкі зображення і медійні запити, які є основою для створення вебсайт, здатних автоматично адаптуватися до різних розмірів екранів та орієнтацій пристроїв.

Особлива увага була приділена аналізу можливостей та переваг використання бібліотеки React у контексті адаптивного дизайну. Використання цієї бібліотеки дозволило досягти високої продуктивності та гнучкості при розробці вебдодатків, а також відзначилося легкістю розширення наявних рішень. Практична частина дослідження включала розробку прототипу вебсайт з використанням React, де застосовувалися принципи адаптивного дизайну, з особливим акцентом на оптимізацію зображень та мультимедійного контенту для гнучкої адаптації до різних розмірів екранів.

Для стилізації було використано бібліотеку MUI, яка надала можливість писати CSS прямо в JavaScript, поліпшуючи процес розробки шляхом підтримки динамічних стилів і кращої організації коду. Однак, якщо продуктивність є критично важливою,

рекомендується використовувати CSS Modules, які забезпечують кращу ефективність завдяки локальності стилів і зменшенню навантаження на браузер. Важливо зазначити, що неможливо рекомендувати універсальний підхід до стилізації; натомість, доцільно використовувати змішаний підхід, адаптуючи його під конкретні вимоги проєкту та очікування користувачів.

Проведене тестування вебсайту на різних пристроях і екранах дозволило виявити та виправити недоліки, що в результаті сприяло покращенню користувацького досвіду. У процесі дослідження використовувалися як теоретичні, так і практичні методи, включаючи аналіз наукових та технічних літературних джерел, практичну роботу з бібліотекою React та емпіричні дослідження.

Наукова новизна роботи полягає у комплексному підході до дослідження та реалізації адаптивного дизайну з використанням сучасних методів веброзробка. Визначення нових методів та підходів до створення адаптивних вебсайтів, які враховують властивості й можливості сучасних бібліотек і фреймворків, зокрема React, робить значний внесок у розвиток вебтехнологій. Практичне значення дослідження полягає у створенні ефективного прототипу вебсайт, який може використовуватися для комерційних та некомерційних цілей, а також у наведенні правил і прикладів коду, що будуть корисні для розробників, які прагнуть удосконалити свої навички у створенні адаптивних вебсайтів.

Отже, результати проведеного дослідження підтвердили значущість адаптивного дизайну у сучасній веброзробці та продемонстрували ефективність його реалізації за допомогою бібліотеки React та MUI. Досягнення високого рівня зручності та функціональності вебсайт на різних пристроях підкреслює важливість впровадження адаптивного підходу для відповідності високим стандартам якості в галузі веброзробки. Проте, неможливо рекомендувати універсальний підхід для всіх

проєктів, найкращим підходом є змішаний, адаптований до конкретних вимог і очікувань користувачів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. МІНІСТЕРСТВО ОХОРОНИ ЗДОРОВ'Я УКРАЇНИ [Електронний ресурс]. – 2024. – Режим доступу до ресурсу: https://zakon.rada.gov.ua/laws/show/z0356-22#.

2. БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ Навчальний посібник [Електронний ресурс] / Зацарний В. В, Зацарна О. В, Землянська О. В, Праховнік Н. А. – 2016. – Режим доступу до ресурсу: https://kpdi.edu.ua/biblioteka/%D0%91/%D0%91%D0%96%D0%94%20%D0%97%D0%B0%D1%86%D0%B0%D1%80%D0%BD%D0%B8%D0%B9%20%D0%92.%D0%92..pdf

3. Гігієна праці в деталях: вимоги виробничої санітарії до робочого місця [Електронний ресурс]. – 2024. – Режим доступу до ресурсу: https://pd.dsp.gov.ua/news/hihiiena-pratsi-v-detaliakh-vymohy-vyrobnychoi-sanitarii-do-robochoho-mistsia/

4. Marcotte E. Responsive Web Design [Електронний ресурс] / Ethan Marcotte. – 2024. – Режим доступу до ресурсу: https://alistapart.com/article/responsive-web-design/.

5. CSS Flex [Електронний ресурс]. – 2024. – Режим доступу до ресурсу: https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_flexible_box_layout.

6. Adaptive and Responsive design [Електронний ресурс]. – 2024. – Режим доступу до ресурсу: https://app.uxcel.com/glossary/adaptive-design.

7. Responsive images [Електронний ресурс]. – 2024. – Режим доступу до ресурсу: https://developer.mozilla.org/en-US/docs/Learn/HTML/Multimedia_and_embedding/Responsive_images.

8. Media queries [Електронний ресурс]. – 2024. – Режим доступу до ресурсу: https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_media_queries/Using_media_queries.

9. CSS clamp [Електронний ресурс]. – 2024. – Режим доступу до ресурсу: https://developer.mozilla.org/en-US/docs/Web/CSS/clamp.

10. Mobile first [Електронний ресурс]. – 2024. – Режим доступу до ресурсу: https://www.site2b.ua/ua/web-blog-ua/princip-mobile-first-dlya-rozrobki-dizajnu-sajtiv.html.

11. Adaptive vs Responsive vs Fliud [Електронний ресурс]. – 2024. – Режим доступу до ресурсу: https://learn.onemonth.com/responsive-vs-adaptive-vs-fluid-design/.

12. React documentation [Електронний ресурс]. – 2024. – Режим доступу до ресурсу: https://react.dev/learn.

13. Angular vs React vs Vue: Core Differences [Електронний ресурс]. – 2024. – Режим доступу до ресурсу: https://www.browserstack.com/guide/angular-vs-react-vs-vue.

14. React Ecosystem in 2024 [Електронний ресурс]. – 2024. – Режим доступу до ресурсу: https://dev.to/avinashvagh/react-ecosystem-in-2024-418k.

ДОДАТКИ

ДОДАТОК А

Лістинг коду проєкту

Лістинг коду 1 – файл main.tsx

```
import React from "react";
import ReactDOM from "react-dom/client";
import { RouterProvider } from "react-router-dom";
import "./index.css";
import router from "./router";

ReactDOM.createRoot(document.getElementById("root") as HTMLElement).render(
  <React.StrictMode>
    <RouterProvider router={router} />
  </React.StrictMode>
);
```

Лістинг коду 2 – файл index.html

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8" />
  <link rel="icon" type="image/svg+xml" href="/vite.svg" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <title>Nuegas</title>
  <link rel="preconnect" href="https://fonts.googleapis.com" />
  <link rel="preconnect" href="https://fonts.gstatic.com" crossorigin />
  <link
href="https://fonts.googleapis.com/css2?family=Plus+Jakarta+Sans:wght@400;500
;600&display=swap"
    rel="stylesheet" />
</head>

<body>
  <div id="root"></div>
  <script type="module" src="/src/main.tsx"></script>
</body>

</html>
```

Лістинг коду 3 – файл index.css

```
@tailwind base;
@tailwind components;
@tailwind utilities;
```

```
body {
  background: #ffffff;
}

h1,
h2,
h3,
h4,
h5,
h6,
p,
span,
a,
li,
em {
  font-family: "Plus Jakarta Sans", sans-serif;
}

input[type="range"] {
  -webkit-appearance: none;
  -moz-appearance: none;
  appearance: none;
  background: transparent;
}

input[type="range"]:focus {
  outline: none;
}

/* custom input range style ( track ) */

input[type="range"]::-webkit-slider-runnable-track {
  background: #546fff;
  height: 0.5rem;
  border-radius: 999px;
}

input[type="range"]::-moz-range-track {
  background: #546fff;
  height: 0.5rem;
  border-radius: 999px;
}

/* custom input range style ( thumb ) */
input[type="range"]::-webkit-slider-thumb {
  -webkit-appearance: none;
  margin-top: -4px;
  border: 2px solid #ffffff;
  border-radius: 999px;
  height: 1rem;
  width: 1rem;
  background: #546fff;
}
```

```
input[type="range"]::-moz-range-thumb {
  border: 2px solid #ffffff;
  border-radius: 999px;
  height: 1rem;
  width: 1rem;
  background: #546fff;
}

.mobile-navlist {
  height: calc(100vh - 70px);
  @apply absolute top-full left-0 w-full z-10 py-8 px-6 bg-primary-100;
}

.swiper.mentor-slider .swiper-wrapper,
.swiper.task-slider .swiper-wrapper {
  min-width: 0px;
  width: 0;
}

.primary-button {
  @apply w-full py-3 rounded-xl bg-tertiary-100 text-primary-100 text-sm
font-semibold md:w-2/6;
}

.dropdown {
  @apply cursor-pointer font-medium text-xs flex items-center justify-between
px-5 py-4 rounded-xl border border-primary-300  transition-all duration-300
ease-in-out;
}

.dropdown:hover {
  @apply hover:bg-primary-300;
}

.dropdown-arrow {
  @apply transition-all duration-200 ease-in w-5 h-5 object-cover;
}

.timezone-switch-button {
  @apply cursor-pointer py-4 px-5 rounded-xl border flex items-center gap-4
md:w-full md:gap-0 md:justify-between transition-all duration-300 ease-in-
out;
}

.timezone-switch-circle {
  @apply w-5 h-5 rounded-full transition-all duration-200 ease-in;
}
```

Лістинг коду 4 – файл route.tsx

```
import { createBrowserRouter } from "react-router-dom";
import {
  Mentors,
  Message,
  MessageRoom,
  Overview,
  Root,
  Settings,
  Task,
  TaskDetail,
} from "../pages";

const router = createBrowserRouter([
  {
    path: "/",
    element: <Root />,
    children: [
      {
        index: true,
        element: <Overview />,
      },
      {
        path: "task",
        element: <Task />,
      },
      {
        path: "task/:taskTitle",
        element: <TaskDetail />,
      },
      {
        path: "mentors",
        element: <Mentors />,
      },
      {
        path: "message",
        element: <Message />,
        children: [
          {
            index: true,
            element: (
              <h1 className="hidden md:block">Default Route Message Room</h1>
            ),
          },
          {
            path: "/message/:roomName",
            element: <MessageRoom />,
          },
        ],
      },
      {
        path: "settings",
        element: <Settings />,
      },
```

```
    ],
  },
]);

export default router;
```

Лістинг коду 4 – файл overview.tsx

```
import { ReactElement } from "react";
import { TaskDetail } from "../../components/Card";
import Chart from "../../components/Chart";
import ProfileLayout from "../../components/Layouts/ProfileLayout";
import ProgressBar from "../../components/Progress";
import "../../components/Progress/progressStyle.css";
import MentorSlider from "../../components/Slider/Mentor";
import TaskSlider from "../../components/Slider/Task";
import { mentorSliders } from "../../utils/mentor";
import { taskSliders } from "../../utils/task";

const Dashboard = (): ReactElement => {
  return (
    <div className="flex flex-col md:flex-row">
      <div className="w-full md:w-7/12 lg:w-8/12">
        <ProfileLayout
          titleProfile={`Hi, Masbro`}
          descProfile="Let's finish your task today!"
        />

        <div className="px-6 py-8 bg-primary-100 space-y-8 overflow-hidden
md:bg-primary-200 md:p-8">
          <div className="flex flex-col gap-8 lg:flex-row">
            <ProgressBar />
            <Chart />
          </div>

          <MentorSlider
            heading="Monthly Mentors"
            smSlide={1}
            lgSlide={2}
            sliders={mentorSliders}
          />

          <TaskSlider
            heading="Upcoming Task"
            smSlide={1}
            lgSlide={2}
            sliders={taskSliders}
          />
        </div>
      </div>

      <TaskDetail />
```

```
    </div>
  );
};

export default Dashboard;
```

ДОДАТОК Б

Диск з роботою